Florida State University
College of Arts and Sciences
Department of Computer Science

pTools:
Process Information Utilities

Stephen Michael Oberther

Project submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Master of Science

The members of the committee approve the Master's Project of Stephen Oberther defended on April 18, 2002.

Dr. Lois Hawkes
Major Professor

Dr. Theodore Baker
Committee Member

Jeff Bauer
Committee Member

This work is dedicated to my parents, Michael and Joanne. This project would not have been possible without their continued love, support, and understanding. I would also like to thank Jennifer for the support throughout each phase of this project. Without her the pTools may never have been. Also, to J. Gregory Rebholz for providing the idea, desire, and motivation for the pTools project. May I never take on another one of his "fun" projects. And finally, Chuck Williams for allowing me to put school before work during the entire course of this project. This project may not have been completed in a timely manner without his understanding and compassion.

Abstract

Modern operating systems appear to allow multiple processes to execute simultaneously creating a need for advanced process management and control. The operating system provides these functions and also grants the user some management and control of the processes. Many UNIX derived operating systems extend the amount of process information given to the user through the proc filesystem. The proc filesystem provides an interface into the operating system data structures related to the individual processes executing on the system. Additional operating system related information is obtained through the proc filesystem as well, depending on the UNIX variant being examined. The group of process utilities implemented in the Sun Microsystems UNIX variant, Solaris, extends the functionality of the proc filesystem to the user. In this paper, an implementation is presented of a subset of the Solaris process utilities for the Linux operating system. The tools are implemented using the existing information in the Linux proc filesystem where possible. Extensions are added to the Linux proc filesystem to extract additional data that not is available through the current Linux proc implementation. The tools are then compared against the Solaris process utilities to evaluate the implementation.

Contents

1 Introduction

Multiprogramming, or multi-tasking, is the technique of running two or more processes in an operating system at a time using timesharing[4]. Although each process is not simultaneously executing, the operating system provides the illusion that they are. This advancement in operating system technology has created the concept of a task, or process. A process is an operating system concept that captures the idea of a program in execution. A process can hold resources (e.g., open files, address space, protection privileges, etc.) and it can be run by the dispatcher, i.e. scheduler[4]. Management of these processes is the responsibility of both the operating system and the user. The system administrator is also responsible for assuring processes are controlled, by the operating system, in a fair manner by establishing and enforcing policies regarding the use of system resources.

Unfortunately, few tools exist that give the user or system administrator control over these processes. Simple process listings, and the ability to terminate a task do not allow for finer process inspection and management. In the most popular UNIX variants the proc filesystem allows a greater inspection of processes, and also allows for minor tuning of the operating system, or kernel, parameters. The process file system, procfs, is a pseudo file system. Pseudo file systems provide file-like abstractions and file I/O interfaces to something that is not a file in the traditional sense[10]. The increased ability to audit process statistics and control kernel parameters provides more control to the system administrator and allows developers to collect information about the programs they write. Additionally, proc provides general system and process specific information within a single branch of the filesystem, as opposed to the dispersed nature of the typical UNIX filesystem. Unfortunately there is not a proc specification, and each implementation presents different information in a different manner.

The Linux operating system is one of the most popular UNIX variants today, is distributed freely, and is available for modification. The Linux proc filesystem first appeared in the 0.97 kernel release. Linux consistently improves upon the programs and utilities found in other UNIX variants, and the proc filesystem implementation is no different. Lacking a specification for the proc filesystem the Linux developers are free to implement proc however they like. This has led to a user-friendly implementation where the proc files contain ASCII text instead of binary encoded data structures. Tunable kernel parameters are easily changed by writing a new ASCII value/character into the appropriate file.

Sun Microsystems created a UNIX variant which is arguably the most popular commercial UNIX system on the market today. The Sun Microsystems implementation of the proc filesystem first appeared in version 2.5 of the Solaris operating system. The initial implementation consisted of a flat directory structure and is now restructured into a directory hierarchy with a separate directory entry for each process executing on the system. Each directory relates to a Process ID (PID) of a running program, and the files contain information relating to that particular process. The PID refers to a process descriptor in

the operating system. The process descriptor is a data structure in an operating system that contains all the information the operating system needs to record about the process[4]. In the current Solaris proc implementation these files contain the data structures described in the proc(4) man page.

The proc filesystem is largely an untapped resource by end users and developers on Linux systems. The pTools project provides a useful set of tools for users, developers, and systems administrators of Linux based systems. The Solaris system ships with several commands that implement /proc for extracting information and issuing control directives[10]. This project provides for Linux systems, the process utilities commonly located in /usr/proc/bin on Solaris systems. The pTools add to the overall value of the Linux operating system, create a common set of utilities across two popular UNIX variants, and increase the amount of information obtainable from the Linux proc filesystem.

2 Related Work

The Sun Microsystems process toolset, mentioned previously, directly relates to the pTools project. The Solaris process utilities have guided the development of the pTools and represent the initial goal of the project. Projects providing similar functions as the Solaris process tools are pstree, lsof, and pstack (Appendix A). These programs run on the Linux operating system and directly extend the usefulness of the Linux proc filesystem by creating utilities that use the information available via proc. These programs provide functions similar to the appropriate Solaris process utilities. However, an objective of the pTools project to first recreate the Solaris process utilities and then provide extensions to them, as is common in Linux adaptations of common UNIX utilities.

The pstack (Appendix A) tool, in particular, attempts to duplicate the functionality found in the Solaris pstack utility. It is available for 32-bit x86 Linux machines, using ELF formatted executables. The pstack project provides an excellent initial attempt at duplicating the work of the Solaris pstack utility, and best of all requires no modification to the Linux kernel.

The Linux proc filesystem implementation directly relates to the usefulness and overall existence of the pTools. Although no maintainer for the proc filesystem is known at the time of this writing, continued support of the Linux version of proc is essential to the success of the pTools. The pTools project uses the system and process information contained within the Linux proc filesystem and extends the Linux implementation, where needed, to create a compatible implementation of the Solaris process tools.

3 pTools Introduction

The process utilities, referred to as the pTools, show how the proc filesystem interface can be used and extended to provide additional information to the user. The process tools are found in the Sun Microsystems Solaris operating

system, and improve upon the information presented by the ps(1) command. As described in [8], the process tools:

- Provide control over processes, allowing users to stop or resume them.
- Display more information about processes, such as fstat(2) and fcntl(2) information, working directories, and trees of parent and child processes.

The subsequent paragraphs describe each of the process tools found in /usr/proc/bin on the Solaris operating system using information found in the Solaris 2.8 proc(1) man page. The general usefulness of each tool, how the particular tool can be used by different types of users, and security related functions/information provided by the tool are examined.

3.1 pcred

The pcred utility displays the credentials of a process, such as the effective, real, and saved user ID's (UID) and group ID's (GID). This information allows a user or system administrator to verify that a particular process is executing with the proper ownership and group membership. The ability to verify this information on a running process is an important security auditing aspect provided by the proc filesystem. This allows pcred to be used for auditing purposes and can provide debugging information for a developer dealing with a program that changes user identification or group membership during execution.

3.2 pfiles

The function of pfiles is to present information for each of the open file descriptors belonging to a process. The data returned by pfiles is typical of that found by performing an fstat(2) or fcntl(2) system call on the file descriptor. File descriptor information allows developers to debug information relating to open files, sockets, pipes, and doors. The pfiles output allows system administrators to verify what network sockets a program is using, as well as which configuration and other files the process has open. Information relating to file descriptors and the different file types they reference can provide important security information to the administrator as well. Security monitoring tools implemented to watch network connections of processes and other file descriptors could use the output from this tool to perform process auditing on a task as it is executing.

3.3 pflags

The pflags process utility reports the flags related to the process, the pending and held signals, as well as other status information relating to each of the associated threads, or light-weight processes. The process information reported by pflags provides debugging information useful to developers and allows system administrators and users to quickly observe the status of a task executing on the system. If the interrogated task is stopped (see pstop in section 3.9), the pflags

command also allows the user to dump the contents of the processor registers for the task in question. The ability to view machine registers for a process provides a mechanism for later implementations of process check-pointing and process migration systems.

3.4 pldd

Verification of dynamically linked libraries in the operating environment can be performed via the pldd utility. Using pldd a user or developer can view what libraries a program is currently linked to on the system. Data on linked libraries provides useful information to the system administrator by showing which library files should be watched for changes. The pldd tool allows processes to be flagged for auditing because of libraries they are linked to, and specifically can be used for security related auditing of the runtime linker and linked libraries. These aspects of pldd make it a useful tool for any user, and allow for greater auditing of processes as they execute on the system.

3.5 pmap

The pmap utility displays the address space mapping for a process. Information regarding memory region offset, size, permissions, and the associated library or program execution context for that memory region. Output from pmap provides a finer level of detail than the pldd tool. Developers can use pmap to interrogate the memory layout of a program during execution time. Knowing the memory layout of a process allows a process to be targeted for protection based on permissions of memory regions or use of a particular shared library. Unfortunately this tool can also be used by intruders to view memory layouts of processes if proper access rights are not maintained by the proc filesystem and the process tools. Additional security information relating to proc and this project is discussed in section 4.2 below.

3.6 prun

The process control and management operations of the process tools is performed through prun. The prun utility performs the inverse operation of pstop, i.e. the specified task resumes execution from where it was stopped. This allows users and system administrators to restart the execution of tasks which were previously stopped. This utility extends process management to a new level, allowing users to control the execution of their processes and does so without the use of signals, thereby creating a way of controlling a process which cannot be ignored, or caught.

3.7 psig

Signals provide processes with event notification in UNIX and are useful for process communication. The psig tool allows the signal actions of a process to

be inspected. Viewing the processes action to a signal is useful in debugging for developers dealing with signal events, and system administrators can use psig to audit processes as they are executing on the machine. The data psig displays not only shows the action to be taken, but also details any flags associated with the signal, and which signals are blocked while handling a particular signal.

3.8 pstack

The pstack process tool is useful for debugging and system administration. A hexadecimal and symbolic dump of the processes stack is displayed for each thread associated with the process. This information can be used to obtain stack related information when performing a security analysis of a program, for example when looking for stack related buffer overflow exploits. The pstack tool can also be used when developing applications or libraries to perform process migration across systems.

3.9 pstop

The pstop utility compliments the process management operation provided by the prun tool. A processes execution is halted by performing a pstop on the respective PID. This type of process management allows users and system administrators to effectively pause an executing process which can be resumed later via prun. This utility, along with the process information available from pflags allows for basic operating system support for process migration. The benefits of the process control mechanism provided by prun also apply to pstop, because process execution can be controlled in a non-blockable and non-ignorable way.

3.10 ptime

Process execution time can generally be measured via the time(1) command, however the ptime utility allows for execution times to be measured using the microstate accounting feature of the Solaris operating system. Microstate accounting is the fine-grained retrieval of time values taken during one of several possible state changes that can occur during the lifetime of a typical LWP[10]. This allows for the runtime of a process to be measured more precisely and also allows for reproducible precision. Developers will appreciate ptime when testing optimizations to code by using the increased precision of the ptime results.

3.11 ptree

The ptree utility displays the parent-child relationships between processes in a tree format. This makes it possible to see how each process in the system is related to other process. The tree will also display the processes owned by a certain user. This process tool is useful for any user on the system to view what processes are executing as their UID, or for a system administrator to monitor which users are consuming system resources. A security aspect of ptree is the

ability to monitor processes executing as unexpected users, such as: nobody, system, admin, sys, bin, etc.

3.12 pwait

The ability to wait for a process to finish is useful in: scripts, command line execution control, and process monitoring. Using pwait, a user can perform these tasks on one or numerous processes. Even from the command line it is useful to wait for one process to finish before beginning another, or to notify the system administrator when a task is completed. Developers can use the pwait command to execute cleanup, or analysis tools after a larger batch job has executed. The ability to monitor processes allows a systems administrator to be notified if an important system process exits, or a service is terminated unexpectedly. Improper termination of a service can be used to warn the administrator of a possible security breach or other system related problem, making pwait quite useful in a system administrators toolkit.

3.13 pwdx

Viewing the current working directory for a process on the system is accomplished by using pwdx. Examining the working directory allows a system administrator to validate that programs executing in a chroot'ed environment are currently inside the proper directory in the filesystem. This tool can also be used to monitor an intruder as they move throughout a system. This simple tool can also be extended to monitor system daemons, user processes, or used in scripts to automate the monitoring of system services.

Each of the process utilities found in the Solaris operating system operates by using the proc filesystem. The utilities provide valuable information to users, developers, and system administrators and each tool can easily be extended (through scripts and other command line tools) to expand the individual functionals provided. This usefulness to a variety of users provides much of the motivation for the pTools project. The additional motivation, design and implementation of the pTools is discussed in the following section.

4 pTools Design

The primary objective of the pTools project described in this document is to effectively recreate the Solaris proc utilities on the Linux operating system. The pTools process utilities should report the same type of information as the equivalent Solaris tool. The scope, assumptions, requirements analysis and extended objectives of the pTools project are discussed in the following sections.

4.1 Project Scope

The pTools project aims to provide a complete implementation of the Solaris process utilities for the Linux operating system. The scope of this project is to develop a working subset of these utilities, that perform the task of the Solaris version, and are representative of the entire set of process utilities. Multiple tools which report similar information are compared and one tool selected for the initial implementation. The criteria for selecting the implemented tools is discussed below in section 4.3.2. Additionally, the pTools project is restricted by hardware architecture and kernel version. These limitations will be discussed in the following section.

4.2 Assumptions

During the initial design and planning of the pTools project certain assumptions were made regarding the implementation. These assumptions define the scope of the project, and allow the initial implementation to focus on the correct development of the pTools. The assumptions that were made during the planning of the pTools are discussed below.

- The implementation will be developed on the Intel x86 architecture and is not required to work with Linux on any other platform.
- Version 2.4.7 of the Linux kernel will be used as the base kernel for all modifications to the Linux operating system. Porting between releases and into the development tree, currently 2.5.x, is beyond the scope of this project.
- Open-source libraries and code will be examined and used under the rights provided by the license protecting the code.
- The Solaris process utilities are used as a reference for input and output. A comparison of input and output between the Solaris utilities and pTools implementation, with minor differences allowed for library and error message reporting, should be identical. A tool meeting this requirement will be considered a success.
- The pTools that affect program execution or modify program behavior should be tested against the Solaris implementation for similar results. Due to system design differences with input processing, execution speed, and how the operating system implements processes these utilities will not be identical on both operating systems. The pTools implemented that affect program execution, like pstop and prun, shall be considered successful if results are similar to the Solaris implementations.
- The security model for the pTools is handled by the operating system. The current Linux proc filesystem implementation consists of a loose security model. Generally, any proc file can be read by any user on the system.

The pTools shall implement strict filesystem security for any files added to the proc filesystem. In general however, if the necessary information can be obtained for a process the pTools will execute correctly, regardless of any security issues.

These assumptions effectively layout the initial design requirements for the pTools. Choosing one architecture allows the background investigation and design analysis to return greater detail because more time can be spent learning about the Linux kernel code for that particular architecture. The use of existing open-source code eliminates the need to re-develop library routines to extract information from the existing Linux proc filesystem.

The security model is unfortunate because the Linux proc filesystem is currently setup with such an open set of permissions. Defining a better security model for the proc filesystem is something that needs to be determined by the kernel maintainer and is outside the scope of this project. The security model used by any additional files created by the pTools project adhere to the policy that only the owner should be able to examine their processes information. The improved security of the proc filesystem is discussed below as an extended objective of the pTools project in section 4.5.

4.3 Requirements Analysis

Determining the requirements analysis for the pTools involves examining each of the Solaris process utilities, investigating the resources available in the proc filesystem, and looking at other related tools. These processes brought further insight into the project and possible methods to implement the pTools.

4.3.1 Proc Investigation

The project analysis includes examining the proc filesystems in both the Linux and Solaris operating systems. The Linux examination looks at the user-space view of the proc files, and the appropriate kernel functions that create each of the files. The Linux kernel creates text files in the proc directory structure which allow the proc information to easily be read by humans. The pTools must parse the files to extract the process information. The initial examination found what data is available and which process characteristics the project needs to add to the proc files.

The Solaris proc filesystem consists of a directory structure similar to the Linux proc filesystem. Solaris proc files contain data structures and are easily usable by user-space programs. This method is not as user-friendly as the Linux proc filesystem, the Solaris method is desirable from a programming perspective though. The proc(4) man page describes the contents of the files in more depth. This project uses the Solaris method for all new entries to the proc filesystem.

4.3.2 Solaris Examination

The Solaris process tools consist of thirteen separate programs that use information available in files found in the proc filesystem. These files contain the data structures found in the proc(4) Solaris man page. This data represents the current state of the process and is used by the process tools to report information to the user. Each tool in the set examines the files in a particular process directory of the proc filesystem to determine access rights, general process information, and any process specific data that it needs.

The execution of each Solaris tool is examined using two different methods. These methods involve using truss(1) to monitor system calls, and evaluating the execution on test programs. The truss(1) output shows which files in the proc directory the tool uses, order of system call execution, and other information relevant to determining what actions the process is performing. Sample programs allow testing of how different process characteristics affect the process tools.

Examining each tool allows similar design features to be found. Comparing the tools that perform similar functions led to an initial subset of tools to be selected for this project implementation. The subset consists of tools that: use existing Linux proc filesystem features, require kernel modifications, and those needing architecture dependent modifications. Selection of the subset followed the design methodology of the pTools project. The easiest tools are implemented initially to provide a good foundation for the more challenging pTools. The framework of the initial tools is carried through in each of the pTools, making debugging and further improvements easier as well. After the Solaris process tools were examined further inspection of the Solaris administration tools was performed.

The primary administration tools examined were gcore(1) and crash(1M). The Solaris gcore(1) utility creates core images of running processes which can be used with the process tools. Specifying a PID to gcore(1) causes an image of the process to be created as a file. These files can then be examined by some of the ptools. Implementation of a Linux gcore utility is outside the scope of this project but is a possibility for future development.

Additionally, the Solaris crash(1M) utility was examined. Crash(1M) examines the system memory image of a running or crashed system. The crash(1M) utility provides a remarkable amount of information to the system administrator. Although a great tool which would be extremely useful on Linux systems, especially for kernel development and debugging, crash(1M) retrieves data from /dev/kmem. The process tools work through the proc filesystem interface and further research of crash(1M) was postponed.

4.4 Inspecting Related Tools

Creating open-source tools means inspecting the existing open-source utilities that provide similar functions. During the creation of the pTools, numerous programs provided insight into using the existing Linux proc filesystem. Some

of these tools are discussed in section 2 above. Programs like pstree, pstack, and lsof use the proc filesystem to extract process related information.

While examining the procps toolset for Linux the libproc library was encountered. This open-source library handles the parsing of the proc filesystem information for the procps tools. The commonality of these tools on Linux systems implies that libproc will be available for compiling the pTools. Using libproc to parse the proc files essentially eliminates the need for a parser to be written for the project. In the future a parser can be written to extract only the necessary information from the proc files, and to retrieve the data from the additional files added to proc by this project.

Additional tools and programs provided insight into Linux system methodologies. Programs like the binutils package, ld, and ELF provided invaluable information regarding program execution environments and stack information. Each of the open-source programs examined for the project helped in the background research and project design.

4.5 Extended Objectives

The motivation for implementing the Solaris process tools for Linux is driven by many objectives. Each of these extended, or secondary, objectives contributes to the final goal of the project. These objectives guide the design and development of each part in the pTools project. The secondary objectives are not essential to the successful implementation of the pTools, they serve only as a guide to ensure good design, programming practice, and implementation. The extended objectives for this project are:

1. Create a set of library routines and an API for accessing information via the proc filesystem. These routines will standardize how the pTools access the proc files and extract information. The library also allows other user-space tools to access information in proc like the pTools.

2. Any data structures added to the Linux kernel should be contained to allow for easy maintenance, and to protect the integrity of the existing kernel data structures.

3. Modifications to the Linux kernel should enhance the current concept of a task. Kernel extensions should be efficient and localized to a particular region of the kernel. Enhancements to the Linux kernel process descriptor should provide improved task auditing and management capabilities to accommodate the pTools implementations.

4. Improving the current proc filesystem security model is an important aspect of the pTools project. The current security model is loose and any additional files created in the /proc hierarchy by this project will adhere to a strict security policy. This policy allows only the process owner, and root, to inspect or modify any of the added files in the proc filesystem. Eventually the filesystem security on /proc will need to be updated to allow fewer programs to access the files in proc. The Solaris implementation

is a good reference for this, because very few files in proc are examinable by everyone.

5 Implementation of the Linux pTools

This section provides insight into the overall design and implementation of the Linux pTools. A discussion of the design methodology, implementation details, testing design, and validation testing is included. Each component of the development cycle is justified, from library design to tool verification.

5.1 Design Methodology

The pTools design and implementation methodologies tames the difficult learning curve of programming operating system components. Each tool is evaluated along with an inspection of the current Linux proc filesystem, as noted above. Tools that use information already available through the Linux proc filesystem are designed and implemented first. This allows for an easy learning curve as more research into the Linux operating system design and implementation is done.

The Linux data structures and kernel layout become familiar which allows additional information to be extracted and reported through kernel modifications to the proc filesystem. Following an easy learning curve for the kernel is essential to understanding how the subtle kernel optimizations are implemented on the Intel x86 architecture. The modifications to the Linux kernel interrogated different subsections of the process model requiring an intimate understanding of each process data structure. It became essential to understand the interaction between: a user-space program and the kernel implementations of system calls, process memory layouts, virtual memory, and file descriptors. This moderate approach to implementing kernel modifications decreased debugging time of the kernel later in the development cycle.

The pTools design involves user-space programs interrogating the kernel data provided by the proc filesystem. This is imperative, and absolutely necessary in-order for the pTools to function similarly to the Solaris proc tools. During the creation of the pTools user-space programs, it became obvious that the initial program implementations are similar. This led to the design of the pTools library functions and API which is discussed in detail in Appendix B. This library allows each of the pTools to reference proc data structures in exactly the same manor and allows for rapid initial development of the pTools user-space programs.

The rapid development of the pTools user-space framework decreased the initial debugging of the pTools higher level implementation. More information on this and the Linux kernel modifications performed are discussed in the next section.

5.2 Implementation Details

The implementation of the pTools project consists of three major components: user-space code, kernel modifications, and proc interface library. User-space programs, written in C, access the kernel data stored in the proc filesystem. The user-space programs interact with the proc filesystem primarily through a set of library functions which creates a consistent framework between the pTools implementations. The Linux kernel modifications provide additional information to the user-space tools through the addition of new files to the proc filesystem. The implementation details for each component is discussed in the subsequent sections.

5.3 pTools Library

The pTools library allows each pTool, and other user-space programs, to interrogate the proc filesystem through a standard interface. This library provides functions to construct and destruct the libraries internal data structures, parse command line options, and process the PID arguments passed via the command line. The pTools library is implemented in C and is dependent upon libproc (Appendix D) for a substantial part of its functionality.

The library API is discussed in detail in Appendices B and C. Essentially, the pTools library parses the command line passed to the program. Valid options are specified and parsed via the getopt(3) interface. Each valid option is passed to a function specified during the library initialization, which allows the calling program to process the options. Each of the pTools uses this interface to handle argument processing.

The pTools library also parses each PID given on the command line and returns the data through a call to ptools_arg_getnext(). This function returns a proc_t pointer, described in Appendix D. The proc_t pointer contains all information found in the un-modified Linux proc filesystem.

Before exiting, the pTools must call ptools_arg_finish() to clean the data structures used by the pTools argument processing library system. This code essentially cleans up memory used by libproc and can be used to cleanup any other data structures added to the pTools library in the future.

5.4 User-space Programs

The user-space code that contributes to the pTools project is also implemented in C and, as mentioned above, depends on the pTools library for argument processing. Each pTool consists of an identical framework consisting of libptools function calls. After the initial setup is complete the processing of the individual PID arguments commences.

Each PID is processed in the order it appears on the command line. The appropriate data is available via the proc_t structure, or is referenced by calling open(2) and read(2) on the proc files added as a part of this project. Once the

required data is available to the program the necessary checks are performed and the output is constructed.

The pTools that affect program execution operate in a similar manner as that mentioned above. The pstop and pstart commands use the write(2) system call to pass control messages into the Linux kernel space. The control messages direct how the kernel will affect the process. Currently the PCSTOP and PCRUN messages are implemented in this project, which provides the necessary functionality for pstop and prun. The Solaris proc(4) man page describes the other control messages implemented in the Solaris operating system.

The output for each pTools command attempts to duplicate the output from the associated Solaris process tool. Error messages from library calls, or for unrecoverable errors cause output to differ, which may make an exact match in all cases impossible. Close examination of the output from different executions of the Solaris process tools helped to format the output for the pTools correctly. The source code for the Solaris proc tools provided a reference as well for the formatting of output.

A majority of the pTools are dependent upon modifications to the Linux operating system. These modifications provide the additional information necessary for the pTools. The Linux kernel changes are discussed in the next section.

5.5 Linux Kernel Modifications

The implementation of the pTools mentioned above is not possible without the kernel modifications discussed here. Although no major subsystem is added to the Linux kernel, the modifications do extract data from multiple areas of the kernel. Additions to the Linux kernel are kept compact and adhere to the Linux kernel coding style, documented in the CodingStyle[1] file. The kernel modifications for this project are available in Appendix F and are in the diff(1) format used for Linux kernel patches.[2]

A complete explanation of each kernel modification is beyond the scope of this document. The functionality added to the proc filesystem by these modifications is discussed instead. This additional functionality consists of three files added to the /proc/(pid)/ directories: ctl, sigact, and stack.

The ctl file duplicates the functionality of the Solaris /proc/(pid)/ctl file, although it currently only supports the PCSTOP and PCRUN control messages discussed previously. The ctl file is implemented as a file writable only by the owning user, and root of course. The write(2) system call implementation for this file performs actions on the appropriate process based on the control message that is sent. This interface is a one way message passing medium, and no status is reported to the process issuing the write(2) command. The current

[1] CodingStyle can be found in the Documentation/ directory of the Linux kernel source code

[2] The command 'diff -urPN un-modified-source/ modified-source/' is used to generate the kernel patch files.

ctl file implementation changes the state variable of the process according to the control message issued.

The sigact file contains a binary representation of the sigaction data structure used by the kernel. This file contains the information relating to the actions performed by a process when a particular signal occurs. This file is implemented through a specific read(2) interface. The stack file is a similar implementation. The stack file contains the contents of the additional ptools_struct data structure added to the Linux task data structure. The ptools_struct includes the additional data necessary for the entire Linux pTools implementation. Changes to the fork(2) system call were necessary to insert this data structure as well.

Additionally, the system call entry point for the Intel x86 architecture was modified. The modifications consist of saving the register values prior to the execution of the system call function. This data is stored in the ptools_struct mentioned previously. The register values are kept to allow the system call name and arguments to be extracted from the kernel by user-space programs.

The interface between these additional proc files and the user-space utilities occur through the use of additional data structures and cross reference tables found in various header files. The programs and scripts necessary to generate these header files and other helper programs can be found in Appendix H.

5.6 Testing Design

Automated testing and data collection is difficult to create for this type of project. This creates testing design problems because simple software testing can not be performed. Differences in operating system installations and program availability make testing difficult as well. These difficulties led to the following testing design for the pTools project.

The internal implementation of the Solaris proc tools and the utilities created by this project are inconsequential. The input parameters and the output format and data must be identical in both format and context. This means that as long as the same relevant data went into the program and appropriate data is output then the tool is a success. The proper format of the input and output must also apply for the above to hold true.

The data must only be relevant because creating identical setups of Linux and Solaris operating environments is difficult, if not impossible. Matching password entries for credentials is relatively straight forward, but verifying exact implementation details of programs for testing is not possible without writing each program from scratch.

The results of the validation testing is discussed next.

5.7 Validation Testing

The pTools generally fall into two categories for testing purposes. The first category consists of the tools that are verified and tested through execution on minor test programs. These tools include pcred, pflags, psig, and pwdx. Test programs are created to change the relevant process characteristics that the

tested pTool inspects. These test programs are then inspected on the Solaris and Linux operating systems.

Other pTools rely on inspecting the entire system or perform process control. The ptree utility is a perfect example because only the output can be verified. The formatting of the tree and the correct parent-child relationships need to be compared to the current system state. No test program exists for these pTools. These pTools are tested by inspecting the output and/or the desired affect on the arguments.

These two testing methods are used depending on the pTool in question. Automated testing across both platforms is outside of the scope of this project and is a possible area for future work on the pTools. Section 8 details the future work and direction for this project, and contains details and speculation on how automated testing is possible.

6 Results

After design, implementation, and testing each of the pTools for the project the implementor reached the following results. Over 50% of the Solaris process tools are implemented for the Linux operating system. The final testing and inspection of each pTool resulted in successful implementations as described in section 4.2. The tools range from those using the existing proc filesystem and needing only user-space code, to those requiring kernel modifications to deal with interrupts and system calls. Additionally, the libptools library successfully implements an API for creating user-space tools that interact with the Linux proc filesystem.

The kernel modifications for the pTools have also been examined for overall performance impact as well. Additional proc features that do not impact the kernel performance are not tested, because additions to the proc directory structure will affect kernel memory sizes but not execution performance. Table 1 details the memory information displayed during kernel initialization. The table compares two Linux kernels containing the same options. The "Base Kernel" is an unmodified Linux kernel, version 2.4.7. The "Project Kernel" contains the modifications made by this project.

Memory (k)	**Base Kernel**	**Project Kernel**
Used	255968	255968
Total	262128	262128
Kernel Code	760	761
Reserved	5776	5776
Data	267	268
Init	180	180
Highmem	0	0

Table 1: System Memory Comparison

Additionally, the performance affects of the pTools kernel modifications were examined. The system call entry point into the kernel contains 16 additional assembly instructions. These instructions store the system call arguments and return values into the ptools_struct data structure inside of the appropriate task descriptor in the kernel. System calls provide access to the operating system and resources on the machine, performance considerations of system call initialization is important to maintain operating system performance. Table 2 shows the average duration to execute a particular number of system call initializations in both the "Base Kernel" and "Project Kernel" described above. The durations are average execution times, the actual data collected can be found in Appendix E. Also computed is the average time to execute the system call initialization code in each kernel. The "Average (minus 1)" row represents the average initialization time without the data collected for the single system call initialization in row 1..

# Calls	**Base Kernel** (μsec)	**Project Kernel** (μsec)
1[a]	11.7	11.3
10	8.0	8.0
20	13.3	14.0
30	19.7	21.0
40	26.0	27.0
50	32.3	33.3
60	38.0	39.3
70	44.7	46.0
80	50.7	52.0
90	57.0	59.0
100	63.7	65.3
1000	625.0	644.7
10000	6264.3	6452.7
Average	1.50	1.49
Average (minus 1)	0.65	0.68

Table 2: Average System Call Initialization Times

[a]The times listed are actual execution times. The extended duration is likely due to caching memory references to the kernel code for the system call code.

Figure 1 shows a graph of the "Base Kernel" and "Project Kernel" values from table 2. The graph clearly shows a distinct increase in system call execution time, as is apparent from the data in table 2 as well. This increase is attributed to the use of two instructions to move data between memory locations in the x86 architecture. The use of memory to memory copy instructions could improve the performance, however the performance is still bound to the memory references.

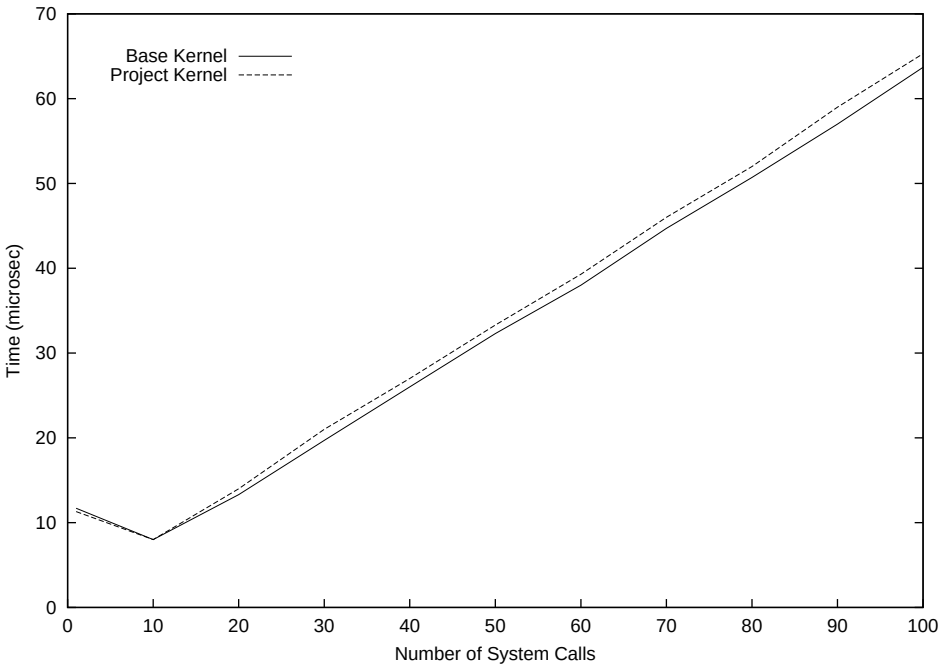

Figure 1: Graph of Average System Call Initialization Times

7 Conclusion

This project has successfully completed the design of the Linux pTools suite. A subset of the Solaris process tools has successfully been implemented for the Linux operating system on the Intel x86 architecture. This implementation extends the current Linux proc filesystem and provides additional tools and resources to developers and users.

The modification of the Linux kernel provides additional inspection of the operating system internals to the user and provides additional functionality to the proc filesystem. The secondary objectives for the project have been met during the implementation as well. Kernel data structure modification is kept to a minimum, while improved process auditing and management capabilities have been added.

The pTools project provides a strong framework for additional process auditing and inspection tools to be implemented. The next section provides insight into the future additions to the pTools project.

8 Future Work

The possibilities for future additions to the pTools project is limitless. Besides creating a framework and providing library functions to create additional user tools for interfacing with the proc filesystem, the following additions are possible:

1. Improve the libptools parsing implementation to move it away from using libproc. This would allow data to be retrieved as is needed by the ptools, instead of parsing the data of every file in the proc directory structure.

2. Implement an automated testing scheme to match output from the Solaris process tools and the Linux pTools. The initial design for this addition has been created during the design and implementation of the pTools. Successful implementation is left for future work on this project.

3. A Graphical User Interface (GUI) for interaction between processes and the pTools would allow processes to be monitored in a dynamic fashion. Selecting a process would allow for additional information and auditing via the pTools.

4. An RPC implementation of the pTools and the Solaris process tools to create a networked system administration program between the two UNIX variants. This is most useful in the heterogeneous network layouts common in the modern network.

5. Porting the remainder of the pTools to Linux using the provided framework from this project also needs to be done.

6. Finally, porting the architecture specific changes to the different architectures supported by Linux needs to be completed. This would make the pTools available on every Linux machine.

Other additions to the pTools project are possible and are not covered here. The hope is to make this project very active in the Linux community and to continue development on all of the above additions to the pTools.

9 Glossary

CHROOT UNIX system call to effectively change the root directory location for a process.

DOOR Interface for processes to issue procedure calls to functions in other processes on the same system. Doors are a Solaris specific procedure call mechanism.

ELF The Executable and Linking Format developed originally by the Unix System Laboratories. Describes the layout of executable files on modern UNIX systems.

GID A unique number associated to each group on the system.

LWP ...a virtual execution environment for each kernel thread within a process. The lightweight process allows each kernel thread within a process to make system calls independently of other kernel threads within the same process[10].

MICROSTATE ACCOUNTING The timing of low-level processing states ...[10] Microstate accounting is the fine-grained retrieval of time values taken during one of several possible state changes that occur during the lifetime of a typical LWP[10].

PID Process ID, unique number referring to an individual process executing on the system.

PROC The process filesystem, commonly mounted as /proc on UNIX operating systems. Proc is a virtual filesystem that does not physically exist on the hard disk, and is only maintained in kernel memory when it is needed via an I/O reference or other system call.

PTOOLS The name given to the Linux implementation of the Solaris process utilities which are generally found in /usr/proc/bin on Solaris machines.

UID A unique number associated with a specific user on the system. If two users have identical UID's they are considered the same user by the operating system.

References

[1] Bowman, Ivan. *Conceptual Architecture of the Linux Kernel.* January 1998. <http://plg.uwaterloo.ca/ itbowman/CS746G/a1/>

[2] Bowman, Ivan et. al. *Concrete Architecture of the Linux Kernel.* February 12, 1998.
<http://plg.uwaterloo.ca/ itbowman/CS746G/a2<

[3] Bovet, Daniel P. and Marco Cesati. *Understanding the Linux Kernel.* O'reilly & Associates, 2001.

[4] Crowley, Charles. *Operating Systems: A Design-Oriented Approach.* Times Mirror Higher Education Group, 1997.

[5] Haendel, Lars. "The Function Pointer Tutorials."
http://www.newty.de/ July, 10, 2001.

[6] Huangs, Michael L. *Extending Sim286 to the Intel386 Architecture with 32-bit processing and Elf Binary input.* September 21, 1998.
<http://www.cs.ucdavis.edu/ haungs/paper/paper.html>

[7] Johnson, Michael K. "proc/readproc.h" procps-2.0.7, 1998.

[8] Klauser, Werner. "The /proc File System." *Sys Admin* vol 07, issue 06.

[9] Mauro, Jim. "Resource utilization and microstate accounting."
Inside Solaris http://www.leto.net/docs/insidesolaris.html
March 14, 2000.

[10] Mauro, Jim and Richard McDougall. *Solaris Internals, Core Kernel Architecture.* Sun Microsystems Press, 2001.

[11] Maxwell, Scott. *Linux Core Kernel Commentary.* The Coriolis Group, 1999.

[12] Plumb, Colin. *A Brief Tutorial on GCC inline asm (x86 biased).* April 20, 1998.
<http://www.uwsg.indiana.edu/hypermail/linux/kernel/9804.2/0953.html>

[13] Pomerantz, Ori. *Linux Kernel Module Programming Guide.* Ver 1.1.0, April 26, 1999.
<http://www.linuxdoc.org/LDP/lkmpg/mpg.html>

[14] Rao, Bharata B. *Inline assembly for x86 in Linux.* March, 2001. <http://www-106.ibm.com/developerworks/linux/library/l-ia.html>

[15] Rubini, Allessandro. and Jonathan Corbet. *Linux Device Drivers.* 2nd ed. O'reilly & Associates, June 2001.

[16] Rusling, David A. *The Linux Kernel.* Ver 0.8-3, 1999.
<http://linuxdoc.org/LDP/tlk/tlk.html>

[17] Russel, Pual R. *Unreliable Guide To Hacking The Linux Kernel.* 2000. <http://www.kernelnewbies.org/documents/kdoc/kernel-hacking/lk-hacking-guide.html>

[18] Russel, Paul R. *Unreliable Guide to Locking.* 2000. <http://www.kernelnewbies.org/documents/kdoc/kernel-locking/lklockingguide.html>

[19] Sun Microsystems. "Solaris 8 Source Code." SUN SOLARIS SOURCE CODE LICENSE,VER 1.1. http://www.sun.com/solaris/source/

[20] Various contributors. *The Linux Kernel API.* January 10, 2002. <http://www.kernelnewbies.org/documents/kdoc/kernel-api/linuxkernelapi.html>

A Reference Software

The following is a list of programs used or referenced for this project:

- **lsof**
  Lsof is a UNIX tool that retrieves information regarding open files on the system. All open files can be examined or particular processes can be queried. More information can be found at:
  http://freshmeat.net/projects/lsof/

- **procps**
  The procps project provides ps, top, vmstat, w, kill, and more for Linux. The included libproc library is used to provide parsing routines for the pTools project. More information regarding the procps project is available at:
  http://procps.sourceforge.net/

- **pstack**
  Pstack is a Linux implementation of the Solaris process tool of the same name. It is available for x86 Linux and is currently not being maintained. This program is available from the following web site:
  http://www.whatsis.com/pstack/

- **pstree**
  Pstree displays process relationships in a tree format similar to the Solaris ptree utility. The format of pstree is slightly different and more options are supported. Pstree is available from:
  http://freshmeat.net/projects/pstree/

- **User Mode Linux (UML)**
  UML is a user-space implementation of the Linux kernel. It is actively used and maintained. UML helps in debugging kernel development, is useful in creating security environments, and supports a multitude of devices including networking from within the UML environment. More information on UML is available from:
  http://user-mode-linux.sourceforge.net/

B Argument Processing API

Appendix B contains the design notes and programming interface for the Argument Processing functions in libptools. Table 3 contains a list of the pTools, the arguments they take, and the default error message description. The "Report Error" type is described as:

Report Error Display error message regarding why the current argument can not be processed.

pTool	Usage	Default Error Message
pcred	pcred pid ...	Report Error
pfiles	pfiles pid ...	Report Error
pflags	pflags [-r] pid ...	Report Error
pldd	pldd pid ...	Report Error
pmap	pmap [-rxl] pid ...	Report Error
prun	prun pid ...	Report Error
psig	psig pid ...	Report Error
pstack	pstack pid ...	Report Error
pstop	pstop pid ...	Report Error
ptime	ptime command [arg ...]	Report "Exec: Failed"
ptree	ptree [-a] [[pid — user] ...]	Ignore Invalid pid/user
pwait	pwait [-v] pid ...	Report Error
pwdx	pwdx pid ...	Report Error

Table 3: Linux pTool Arguments

The following sections contain the design notes, initial design specification, and finally the API for the libptools argument processing functions. It is important to note that the argument and PID processing code currently relies on libproc to parse the proc filesystem information.[3]

[3]libproc may be replaced in future implementations of libptools. Some functions referred to in this Appendix are libproc functions and not found in manual pages present at the time of this writing

B.1 Argument Processing Notes

- Does readproc() loop if we reach the end of the list? *NO*
  - This can happen if we need a low then really high PID then a lower PID again.
  - readproc() loops through the PIDs in the order we give, skipping any it can not stat(2).
- *In our implementation* we go through the argument list and if **!=** to the returned PID we report an error.
  - We do not get permission denied, or invalid PID errors. We only know that the argument was skipped.
  - A stat(2) on each skipped argument if it could be a PID (i.e. a *Positive Integer*) would inform us of why the argument was skipped.

B.2 Implementation for Argument Processing

B.2.1 Command-line Options

- Pass argument-count, argument-vector, string of arguments in getopt(3) format, and a function pointer to the Option processing function.
- A valid argument found by getopt(3) is passed to the function pointer which handles processing the argument
- POSIX compliant argument handling forced by putting a '+' as first element of argument string (optstring)
  - Perhaps a pre-processor directive to specify whether or not POSIX compliant argument handling should be included at compile time.
  - Solaris implementation uses POSIX compliant argument processing

B.2.2 PID Processing

- optind is index of next non-option argument
- The Setup() function should:
  1. Make a linked list of PIDs
     - if argument **!=** INT then *set data value to negative value (ie. INVAL)*
     - Save the initial argument string for error reporting
- Interface Notes – Keep List Management inside the argument processing library
  1. Make PID list static\global in the library
  2. Make PROCTAB* static\global in the library

B.2.3 Library Function Specification

1. Setup() Function

   Arguments argument-count, argument-vector, character array of available options, and a function pointer that processes each argument.

   Return Value 0 on success, a negative error value on failure (possibly just set *errno*)

   Algorithm:

   (a) Process Arguments

   (b) Set up PROCTAB on remainder of arguments (assumed to be PIDs)

      i. Arguments (ie. PIDs) are kept in an internal list, INVALID entries are marked with a negative data (PID) value

      ii. Invalid arguments should not be passed to openproc()

2. GetNext() Function

   Algorithm:

   (a) Call readproc()

   (b) Find returned PID in our internal list of arguments. Any arguments that are skipped should be reported

   (c) Error Handling

      i. Stock Error Messages

      ii. User Passes them with each call to the function (ie. a NULL terminated char**)

      iii. GetNext returns an INVALID code to let the calling function deal with the error

B.3 Argument Processing Function Documentation

B.3.1 ptools_arg_setup()

int ptools_arg_setup(int argcnt, char * const args[], char * const opts, void *fptr(char))

Function Parameters:

argcnt The number of arguments contained in the *args* array. Basically the *argc* parameter of main().

args An array of character strings containing the arguments to be processed by the ptools_args_*() functions. The first element of the array is not processed by default as this array is processed by getopt(3). Essentially this is the *argv* parameter of main().

opts A character string (NULL terminated) of options. This will become the *optstring* parameter that is passed to getopt(3) and will follow all the rules as specified in the getopt(3) man page.

fptr(char) A pointer to a function that takes a character as an argument. This function will be called on each argument found in the *args* array.

Return Values:
0 is returned if the call to ptools_arg_setup() was successful. A value of -1 will be returned if an error occurs. The variable *ptools_errno* will contain an integer value specifying which error occurred. The possible values of *ptools_errno* are:

INVAL_ARG Invalid argument passed to ptools_arg_setup(); Either argcnt or args was invalid.

MEM_ERR Unable to allocate memory via malloc(3), etc.

OPT_ERR Unknown option found in arguments.

NO_PARAM No parameter given in an option requiring one.

LIBPROC_ERR Error during a libproc function call, not our fault.

B.3.2 ptools_arg_getnext()

proc_t *ptools_arg_getnext(void)

Return Values:
A pointer to a valid proc_t structure (defined in readproc.h) is returned. This pointer must be reclaimed via a call to freeproc() or memory leakage will result. NULL is returned and ptools_errno is set if an error occurs. The following are possible values for ptools_errno:

INVAL_PID Invalid PID specified on command line

LIBPROC_ERR Error during a libproc function call, not our fault.

0 No Error, No more processes to return.

C PID Function Documentation

The following functions are members of libptools and are available to programs using the libptools library. These functions process PID parameters for validity and perform actions in the proc filesystem in the appropriate directory for the PID argument. A brief description of each method is provided below.

C.1 ValidPid()

int ValidPid(pid_t pid)
Determine if *pid* is a valid process on the system.

Function Parameters:

pid The Process ID to check

Return Values:
0 is returned if *pid* is a valid process id. A value of -1 is returned if *pid* is not valid.

C.2 AllowExamine()

int AllowExamine(pid_t pid)
AllowExamine determines if a process is allowed to be inspected by the current process.

Function Parameters:

pid The Process ID to check

Return Values:
0 is returned if *pid* can be examined. A value of -1 is returned if access is not allowed.

C.3 OpenPid()

int OpenPid(pid_t pid, char *fname)
Open the /proc file *fname* of *pid*

Function Parameters:

pid The process ID who's /proc file we are opening

fname The name of the /proc file to open

Return Values:
A valid file descriptor is returned on success. -1 is returned and errno is set by open(2) if an error occurs.

C.4 fOpenPid()

FILE *fOpenPid(pid_t pid, char *fname)
Open the /proc file *fname* of *pid*

Function Parameters:

pid The process ID who's /proc file we are opening

fname The name of the /proc file to open

Return Values:
A valid FILE * is returned on success. NULL is returned and errno is set by fopen(3) if an error occurs.

D libproc

The proc information for the pTools is parsed using the libproc library functions that are a part of the procps project. The libproc library is also used in the implementation of the pTools library, libptools. The proc_t data structure returned by the ptools_arg_getnext() function is declared as a type in libproc. The following section describes the relevant library functions used in libptools and by some of the pTools directly. This is followed by a brief discussion of the proc_t data structure.

D.1 Library Functions

The prototypes and descriptions for the functions used by libptools and the pTools utilities are[7]:

void closeproc(PROCTAB *PT)

- Performs cleanup operations on open files and other data structures from the openproc() function.

void freeproc(proc_t *p)

- De-allocate space allocated by readproc()

PROCTAB *openproc(int flags, ... /* pid_t|uid_t|dev_t|char * [, int n] */)

- Initializes the libproc internal data structures.
- The flags argument selects the information to be extracted from proc. The optional list specifies which processes to return information about. Only the pid_t list is used in the pTools.
- The PROCTAB * is used in readproc() and closeproc() and holds data needed by readproc() to return the appropriate information from proc.

proc_t *readproc(PROCTAB *PT, proc_t *return_buf)

- Returns the next process matching the criteria set by openproc().

D.2 proc_t structure

The proc_t data structure contains elements relating to the process information that can be found in a process directory in the proc filesystem hierarchy. A complete specification for proc_t is in the proc/readproc.h header file. A brief summary of the data elements relevant to the pTools is given in the following table.

Data Element	Data Type	Description
pid	int	Process ID
ppid	int	Parent Process ID
ruid	int	Real UID
rgid	int	Real GID
euid	int	Effective UID
egid	int	Effective GID
suid	int	Saved UID
sgid	int	Saved GID
fuid	int	Filesystem UID (Linux Only)
fgid	int	Filesystem GID (Linux Only)
environ	char **	Environment Settings
cmdline	char **	Command line
cmd	char *	Base executable name
flags	unsigned long	Kernel process flags
signal	char *	Mask of pending signals
blocked	char *	Mask of blocked signals
sigignore	char *	Mask of ignored signals
sigcatch	char *	Mask of caught signals
l	proc_t *	Pointer for building linked lists
r	proc_t *	Pointer for building linked lists

Table 4: proc_t elements used in pTools

E Kernel System Call Data

The following tables contain the collected data from the tests performed on the "Base" Linux 2.4.7 kernel, and the "Project" kernel containing the pTools modifications.

# Calls	Test #1	Test #2	Test #3	Average
1	12	12	11	11.7
10	8	8	8	8.0
20	13	14	13	13.3
30	20	19	20	19.7
40	26	26	26	26.0
50	33	32	32	32.3
60	38	38	38	38.0
70	44	45	45	44.7
80	51	51	50	50.7
90	57	57	57	57.0
100	63	64	64	63.7
1000	625	625	625	625.0
10000	6260	6261	6272	6264.3

Table 5: Base Kernel System Call Initialization Data

# Calls	Test #1	Test #2	Test #3	Average
1	12	11	11	11.3
10	8	8	8	8.0
20	14	14	14	14.0
30	21	21	21	21.0
40	27	27	27	27.0
50	33	33	34	33.3
60	39	39	40	39.3
70	46	46	46	46.0
80	52	52	52	52.0
90	59	59	59	59.0
100	65	66	65	65.3
1000	644	645	645	644.7
10000	6464	6437	6457	6452.7

Table 6: Project Kernel System Call Initialization Data

F Kernel Modification Code

The following is the kernel modifications made for the pTools project. This output is from the GNU diff(1) command using the u, r, P, and N options. The format is known as the unified output format and is how patches are accepted into the Linux kernel. The patch can be applied against a clean copy of the Linux 2.4.7 kernel and will add support for the pTools. Executing the following command in the kernel source directory will patch the kernel:

patch -p1 < patchfile

F.1 Kernel Diff

```
diff −urPN linux−2.4.7/arch/i386/kernel/entry.S linux/arch/i386/kernel/entry.S
−−− linux−2.4.7/arch/i386/kernel/entry.S Tue Jun 12 14:47:28 2001
+++ linux/arch/i386/kernel/entry.S Fri Mar 29 16:21:16 2002
@@ −47,6 +47,9 @@
 #define ASSEMBLY
 #include <asm/smp.h>

+/* Added by SOberther (oberther@cs.fsu.edu) for pTools */
+#include <linux/ptools.h>
+                                                                    10
 EBX            = 0x00
 ECX            = 0x04
 EDX            = 0x08
@@ −198,10 +201,12 @@
        GET_CURRENT(%ebx)
        cmpl $(NR_syscalls),%eax
        jae badsys
+       PTOOLS_SYSCALL
        testb $0x02,tsk_ptrace(%ebx) # PT_TRACESYS
        jne tracesys                                                 20
        call *SYMBOL_NAME(sys_call_table)(,%eax,4)
        movl %eax,EAX(%esp)       # save the return value
+       PTOOLS_RET_SYSCALL
 ENTRY(ret_from_sys_call)
        cli                             # need_resched and signals atomic test
        cmpl $0,need_resched(%ebx)
diff −urPN linux−2.4.7/fs/proc/base.c linux/fs/proc/base.c
−−− linux−2.4.7/fs/proc/base.c Fri Jul 20 15:39:56 2001
+++ linux/fs/proc/base.c    Fri Mar 29 16:21:16 2002
@@ −499,6 +499,168 @@                                                    30
        follow_link:    proc_pid_follow_link
 };

+
+/**********************************
+ * Added by SOberther 12-10-2001 *
+ **********************************/
+static int ptools_open(struct inode *inode, struct file *file) {
+
+       /* We shouldn't care if we are opened */                     40
+       return 0;
+
```

```
+}
+
+
+/* Special case file only available if task->state is STOPPPED */
+static int ptools_regs_open(struct inode *inode, struct file *file) {
+
+       struct task_struct *task = file->f_dentry->d_inode->u.proc_i.task;
+                                                                                50
+       if (task->state != TASK_STOPPED)
+               return -EACCES;
+
+       return 0;
+}
+
+
+/*
+ * /proc/pid/ctl:
+ */                                                                             60
+
+#include <linux/procfs_ctl.h>
+
+static ssize_t ctl_write(struct file *file, const char *buf,
+                        size_t count, loff_t *ppos) {
+
+       ctl_t msg = -1;
+       struct task_struct *task = file->f_dentry->d_inode->u.proc_i.task;
+
+       printk("ctl_write(): called by PID %d\n", current->pid);                 70
+       printk("ctl_write(): called on ctl file of PID %d\n", task->pid);
+
+       /* We expect a long value */
+       if (count != sizeof(ctl_t)) return -EINVAL;
+
+       /*
+       for (cnt = 0; cnt < sizeof(ctl_t); cnt++) {
+       msg += ((int)buf[cnt]<< ( (sizeof(long int)-1) - cnt) );
+       }
+       */                                                                       80
+
+       msg = (ctl_t)(*buf);
+
+
+
+       printk("ctl_write(): buf contained %ld\n", msg);
+
+       switch(msg) {
+
+               case PCSTOP:                                                    90
+                       printk("ctl_write(): %d: PCSTOP read from buf\n", current->pid);
+
+                       if (!(task->state & TASK_STOPPED)) {
+                               printk("ctl_write(): %d: Changing state of PID %d to TASK_STOPPED\n",
+                                       current->pid, task->pid);
+                               task->ptools.pr_prev_state = task->state;
+                               set_task_state(task, TASK_STOPPED);
+                               schedule();
+                       }
```

```
+                                                                                100
+                       break;
+               case PCRUN:
+                       printk("ctl_write(): %d: PCRUN read from buf\n", current->pid);
+
+                       if (task->state & TASK_STOPPED) {
+                               printk("ctl_write(): %d: Changing state of PID %d\n",
+                                       current->pid, task->pid);
+                               set_task_state(task, task->ptools.pr_prev_state);
+                               task->ptools.pr_prev_state = TASK_STOPPED;
+                               schedule();                                      110
+                       }
+
+                       break;
+               default:
+                       printk("ctl_write(): %d: Unknown Control Message read\n", current->pid);
+
+       };
+
+       return sizeof(ctl_t);
+                                                                                120
+}
+
+/* Going by example on this one :) */
+static struct inode_operations proc_ctl_inode_operations = {
+       permission:     proc_permission,
+};
+
+
+static struct file_operations proc_ctl_operations = {
+       write:          ctl_write,                                               130
+       open:           ptools_open,
+};
+
+static struct file_operations proc_regs_operations = {
+       open:           ptools_regs_open,
+       read:           proc_info_read,
+};
+
+
+/*************************************************************************/     140
+
+
+static ssize_t proc_pid_sigread(struct task_struct *task, char *buffer) {
+
+       struct signal_struct *signals = task->sig;
+       const int sa_sz = sizeof(struct sigaction);
+       int cnt;
+       char *loc = buffer;
+
+       for (cnt = 0; cnt <= _NSIG; cnt++) {                                     150
+               struct sigaction *cur_sa = &(signals->action[cnt].sa);
+
+               memcpy(loc, (char *)cur_sa, sa_sz);
+               loc += sa_sz;
+       }
+
```

```
+       return (loc - buffer);
+}
+
+/*************************
+ * Start stack section  *
+ *************************/
+static int proc_pid_stack(struct task_struct *task, char *buffer) {
+
+       char *loc = buffer;
+       struct ptools_struct *p = &(task->ptools);
+       const int pt_sz = sizeof(struct ptools_struct);
+
+       memcpy(loc, (char *)p, pt_sz);
+       loc += pt_sz;
+
+       return (loc - buffer);
+}
+
+
+
+/*************************
+ * Start regs section   *
+ *************************/
+static int proc_pid_regs(struct task_struct *task, char *buffer) {
+       char *loc = buffer;
+       struct vm86_regs *r = &(task->thread.vm86_info->regs);
+       const int r_sz = sizeof(struct vm86_regs);
+
+       memcpy(loc, (char *)r, r_sz);
+       loc += r_sz;
+
+       return (loc - buffer);
+}
+
+/*********************************************
+ * End Section Added by SOberther for ptools *
+ *********************************************/
+
 struct pid_entry {
        int type;
        int len;
@@ -520,6 +682,18 @@
        PROC_PID_STATM,
        PROC_PID_MAPS,
        PROC_PID_CPU,
+
+       /*****************************************
+        *       Added by SOberther 12-10-2001   *
+        *****************************************/
+       PROC_PID_CTL,
+       PROC_PID_SIG,
+       PROC_PID_STK,
+       PROC_PID_REGS,
+       /*****************************************
+        *         End SOberther Additions       *
+        *****************************************/
```

```
+
        PROC_PID_FD_DIR = 0x8000, /* 0x8000-0xffff */
 };

@@ -539,6 +713,14 @@
   E(PROC_PID_CWD, "cwd",      S_IFLNK|S_IRWXUGO),
   E(PROC_PID_ROOT, "root",    S_IFLNK|S_IRWXUGO),
   E(PROC_PID_EXE, "exe",      S_IFLNK|S_IRWXUGO),
+
+ /* Added by SOberther */
+ E(PROC_PID_CTL, "ctl",      S_IFREG|S_IWUSR),
+ E(PROC_PID_SIG, "sigact",   S_IFREG|S_IRUSR),
+ E(PROC_PID_STK, "stack",    S_IFREG|S_IRUSR),
+ E(PROC_PID_REGS, "regs",    S_IFREG|S_IRUSR),
+ /*********************/
+
   {0,0,NULL,0}
 };
 #undef E
@@ -898,6 +1080,30 @@
                        inode->i_op = &proc_mem_inode_operations;
                        inode->i_fop = &proc_mem_operations;
                        break;
+               /**********************************
+               * Added by SOberther 12-10-2001 *
+               **********************************/
+               case PROC_PID_CTL:
+                       inode->i_op = &proc_ctl_inode_operations;
+                       inode->i_fop = &proc_ctl_operations;
+                       break;
+               case PROC_PID_SIG:
+                       inode->i_fop = &proc_info_file_operations;
+                       inode->u.proc_i.op.proc_read = proc_pid_sigread;
+                       break;
+
+               case PROC_PID_STK:
+                       inode->i_fop = &proc_info_file_operations;
+                       inode->u.proc_i.op.proc_read = proc_pid_stack;
+                       break;
+               case PROC_PID_REGS:
+                       inode->u.proc_i.op.proc_read = proc_pid_regs;
+                       inode->i_fop = &proc_regs_operations;
+                       break;
+               /**********************************
+               *      End SOberther Section    *
+               **********************************/
+
                default:
                        printk("procfs: impossible type (%d)",p->type);
                        iput(inode);
diff -urPN linux-2.4.7/include/linux/procfs_ctl.h linux/include/linux/procfs_ctl.h
--- linux-2.4.7/include/linux/procfs_ctl.h Wed Dec 31 19:00:00 1969
+++ linux/include/linux/procfs_ctl.h Fri Mar 29 16:21:16 2002
@@ -0,0 +1,21 @@
+#ifndef _LINUX_PROC_FS_CTL_H
+#define _LINUX_PROC_FS_CTL_H
+
```

```
+/*
+ * 12-10-2001: Stephen Oberther
+ *  Created for /proc/<pid>/ctl file, based off Solaris sys/procfs.h
+ *
+ */
+
+
+typedef unsigned long ctl_t;
+
+/*                                                                    280
+ * Control codes for messages written to ctl file
+ *  (These constants are found in sys/procfs.h in Solaris 8)
+ */
+#define PCSTOP 1L /* Tell process to stop */
+#define PCRUN 5L /* Make process runnable */
+
+
+#endif /* _LINUX_PROC_FS_CTL_H */
diff -urPN linux-2.4.7/include/linux/ptools.h linux/include/linux/ptools.h
--- linux-2.4.7/include/linux/ptools.h Wed Dec 31 19:00:00 1969          290
+++ linux/include/linux/ptools.h  Fri Mar 29 16:21:16 2002
@@ -0,0 +1,68 @@
+#ifndef _LINUX_PTOOLS_H
+#define _LINUX_PTOOLS_H
+
+
+#ifdef ASSEMBLY
+
+#define PTS_SYSCALL 88
+#define PTS_NSYSARG 90                                                300
+#define PTS_ERRNO 92
+#define PTS_SYSARG1 96
+#define PTS_SYSARG2 100
+#define PTS_SYSARG3 104
+#define PTS_SYSARG4 108
+#define PTS_SYSARG5 112
+#define PTS_SYSARG6 116
+
+
+#define PTOOLS_SYSCALL        \                                       310
+        movl EAX(%esp),%ecx;      \
+        movl %ecx,PTS_SYSCALL(%ebx); \
+        movl EBX(%esp),%ecx;      \
+        movl %ecx,PTS_SYSARG1(%ebx); \
+        movl EDX(%esp),%ecx;      \
+        movl %ecx,PTS_SYSARG3(%ebx); \
+        movl ESI(%esp),%ecx;      \
+        movl %ecx,PTS_SYSARG4(%ebx); \
+        movl EDI(%esp),%ecx;      \
+        movl %ecx,PTS_SYSARG5(%ebx); \                                320
+       movl EBP(%esp),%ecx;      \
+       movl %ecx,PTS_SYSARG6(%ebx); \
+        movl ECX(%esp),%ecx;      \
+        movl %ecx,PTS_SYSARG2(%ebx);
+
+#define PTOOLS_RET_SYSCALL \
+        movl $-1,PTS_SYSCALL(%ebx); \
```

```
+       movl %eax,PTS_ERRNO(%ebx);
+
+#endif /* ASSEMBLY */                                                  330
+
+
+#ifndef __ASSEMBLY__
+
+#define PTS_NR_SYSARGS 6
+
+struct ptools_struct {
+
+ /* Used for pstop/prun */
+ volatile long pr_prev_state;                                          340
+
+ short pr_syscall;
+ short pr_nsysarg;
+ int pr_errno;
+ long pr_sysarg[PTS_NR_SYSARGS];
+};
+
+
+#define INIT_PTOOLS                             \
+{                                               \                      350
+       pr_prev_state: -1,                       \
+       pr_syscall:    -1,                       \
+       pr_nsysarg:    -1,                       \
+       pr_errno:      -1,                       \
+       pr_sysarg:     { -1, -1, -1, -1, -1 } \
+}
+
+#endif
+
+#endif                                                                 360
diff -urPN linux-2.4.7/include/linux/sched.h linux/include/linux/sched.h
--- linux-2.4.7/include/linux/sched.h Fri Jul 20 15:52:18 2001
+++ linux/include/linux/sched.h Fri Mar 29 16:21:16 2002
@@ -282,6 +282,10 @@
 extern struct user_struct root_user;
 #define INIT_USER (&root_user)

+/* Added by SOberther (oberther@cs.fsu.edu) for pTools */
+#include <linux/ptools.h>
+                                                                       370
+
 struct task_struct {
        /*
         * offsets of these are hardcoded elsewhere - touch with care
@@ -320,6 +324,11 @@
        struct task_struct *next_task, *prev_task;
        struct mm_struct *active_mm;

+ /* Added 2-26-2002 by SOberther (oberther@cs.fsu.edu)
+  * for pTools implementation                                           380
+  */
+       struct ptools_struct ptools;
+
 /* task state */
```

```
        struct linux_binfmt *binfmt;
        int exit_code, exit_signal;
@@ -441,6 +450,7 @@
  *  INIT_TASK is used to set up the first task table, touch at
  * your own risk!. Base=0, limit=0x1fffff (=2MB)
  */                                                                    390
+/* Modified by SOberther for struct_ptools init */
 #define INIT_TASK(tsk) \
 {                                                                      \
     state:              0,                                             \
@@ -454,6 +464,7 @@
     policy:             SCHED_OTHER,                                   \
     mm:                         NULL,                                          \
     active_mm:          &init_mm,                                      \
+    ptools:              INIT_PTOOLS,                                   \
     cpus_allowed:       -1,                                            \    400
     run_list:           LIST_HEAD_INIT(tsk.run_list),                  \
     next_task:          &tsk,                                          \
diff -urPN linux-2.4.7/kernel/fork.c linux/kernel/fork.c
--- linux-2.4.7/kernel/fork.c Tue Jul 17 21:23:28 2001
+++ linux/kernel/fork.c Fri Mar 29 16:21:16 2002
@@ -279,6 +279,22 @@
        }
 }

+/* Added by SOberther */                                               410
+static int copy_ptools(struct task_struct *tsk) {
+
+ tsk->ptools.pr_prev_state = -1;
+ tsk->ptools.pr_syscall = -1;
+ tsk->ptools.pr_nsysarg = -1;
+ tsk->ptools.pr_errno = -1;
+ tsk->ptools.pr_sysarg[0] = -1;
+ tsk->ptools.pr_sysarg[1] = -1;
+ tsk->ptools.pr_sysarg[2] = -1;
+ tsk->ptools.pr_sysarg[3] = -1;                                        420
+ tsk->ptools.pr_sysarg[4] = -1;
+
+ return 0;
+}
+
 static int copy_mm(unsigned long clone_flags, struct task_struct * tsk)
 {
        struct mm_struct * mm, *oldmm;
@@ -654,6 +670,11 @@
                goto bad_fork_cleanup_fs;                              430
        if (copy_mm(clone_flags, p))
                goto bad_fork_cleanup_sighand;
+
+       /* Added by SOberther */
+       if (copy_ptools(p))
+         goto bad_fork_cleanup_mm;
+
        retval = copy_thread(0, clone_flags, stack_start, stack_size, p, regs);
        if (retval)
                goto bad_fork_cleanup_mm;                              440
@@ -758,6 +779,11 @@
```

```
 /* SLAB cache for mm_struct structures (tsk->mm) */
 kmem_cache_t *mm_cachep;

+/* Added by SOberther
+ * SLAB cache for ptools structures (tsk->ptools)
+ */
+/* kmem_cache_t *ptools_cachep; */
+
 void __init proc_caches_init(void)                                    450
 {
        sigact_cachep = kmem_cache_create("signal_act",
@@ -789,4 +815,13 @@
                        SLAB_HWCACHE_ALIGN, NULL, NULL);
        if(!mm_cachep)
                panic("vma_init: Cannot alloc mm_struct SLAB cache");
+
+       /* Added by SOberther */
+       /*
+         ptools_cachep = kmem_cache_create("ptools_cache",             460
+         sizeof(struct ptools_struct), 0,
+         SLAB_HWCACHE_ALIGN, NULL, NULL);
+         if (!ptools_cachep)
+         panic("Cannot create ptools SLAB cache");
+       */
 }
diff -urPN linux-2.4.7/kernel/signal.c linux/kernel/signal.c
--- linux-2.4.7/kernel/signal.c Wed Jan 3 23:45:26 2001
+++ linux/kernel/signal.c   Fri Mar 29 16:21:16 2002
@@ -1029,6 +1029,11 @@                                                  470
                sigdelsetmask(&k->sa.sa_mask, sigmask(SIGKILL) | sigmask(SIGSTOP));

                /*
+                printk("%d: sys_sigaction(): flags %lx\n",
+                     current->pid, k->sa.sa_flags);
+                */
+
+               /*
                 * POSIX 3.3.1.3:
                 *  "Setting a signal action to SIG_IGN for a signal that is   480
                 *   pending shall cause the pending signal to be discarded,
```

G pTools Source Code

The following is the README files, Makefile, and source code for each of the pTool implementations. This code can be placed in $PROJDIR/src/<ptools_name>/ and compiled by typing make in $PROJDIR/

G.1 pcred

README

---

pTools Process Credential reporting utility (pcred)

Based off the Sun Microsystems pcred utility. Reports process credentials of the process given on the command line.

Usage:
pcred { pid } ...

NOTES:
Group listing is based off ruid of process (ie the user who actually ran the proccess), not the euid of the process.

– This info is in /proc/pid/status as well (9−16−01)

Consistant with Sun implementation, if uids or gids are different they are enumerated in the output

All data reported can be found in /proc/pid/status file

Security is handled by the OS

If an open() succeeds then report the info we need to report

TODO:
– Fix / for() */* comment at end of while loop

Makefile

```
#
# pcred Makefile
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

LIBS += -lsl_string

all: pcred
```

```
pcred: pcred.c
	$(CC) $(CFLAGS) -o pcred pcred.c $(LIBS)

clean:
	rm -f pcred *.o *~
```

---

pcred.c

```
/*
 * pcred.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * PTools Process Credential Reporting Tool
 *
 * Compile:
 *   gcc -o pcred pcred.c -lptools
 *                                                                    10
 * Usage:
 *   pcred { pid } ...
 *
 * Reference (given as MARKER:LineNumber in source code)
 *    ARR     linux/fs/proc/array.c
 */

#include <config.h>
#include <SMOLabs_string.h>
                                                                    20


#define GRPSEP " " /* Space seperated list of groups (ARR:172) */
#define GRPSEP_C ' '
/*******************************
 * Global data
 *******************************/
int *grps, numgrps;

/*                                                                  30
 * ParseGroups(FILE *):
 *   Parse the Groups information from the
 * file pointed to by fd
 */
void ParseGroups(FILE *fd) {
  char buf[BUFLN];

  while (!feof(fd)) {
    if (!fgets(buf, BUFLN, fd))
      break;                                                        40

    chomp(buf);

    if (!strncmp(buf, "Groups", 6)) {
      char *grouplist, *tok;
      int cnt;
```

```
      numgrps = 0; /* should already be initialized */
      grps = NULL;
                                                                          50
      grouplist = strdup(buf+8);

      if (grouplist) {
        chomp_space(grouplist); /* remove trailing spaces */

        /* Break-up buf to store gids in grps array */
        for (cnt = 0; cnt < strlen(grouplist); cnt++) {
          if (grouplist[cnt] == GRPSEP_C) {
            numgrps++;
          }                                                               60
        }
        numgrps++;

        grps = (int *)malloc(numgrps * sizeof(int));
        if (!grps) { free(grouplist); numgrps = 0; grps = NULL; return; }

        cnt = 0;
        tok = strtok(grouplist, GRPSEP);
        while ( tok && (cnt < numgrps) ) {
          grps[cnt] = atoi(tok);                                          70
          cnt++;
          tok = strtok(NULL, GRPSEP);
        }

        free(grouplist);
      }
    } /* else if */

  }
  return;                                                                 80
}


/*
 * PrintCred(proc_t):
 *   Print process credentials
 */
void PrintCred(proc_t *p) {

  printf("%d:\t", p->pid);                                                90

  if ( (p->euid == p->ruid) &&
       (p->ruid == p->suid) &&
       (p->suid == p->fuid) )
    printf("e/r/s/fsuid=%d", p->euid);
  else
    printf("euid=%d ruid=%d suid=%d fsuid=%d",
           p->euid, p->ruid, p->suid, p->fuid);

  if ( (p->egid == p->rgid) &&                                            100
       (p->rgid == p->sgid) &&
       (p->sgid == p->fgid) )
    printf(" e/r/s/fsgid=%d", p->egid);
```

```
  else
    printf(" egid=%d rgid=%d sgid=%d fsgid=%d",
           p->egid, p->rgid, p->sgid, p->fgid);

  if (numgrps > 1) {
    int cnt;
    printf("\n\tgroups:");
    for (cnt = 0; cnt < numgrps; cnt++)
      printf(" %d", grps[cnt]);
  }
  printf("\n");
}


/*
 * ResetGlobalData:
 *   Initilization function
 */
void ResetGlobalData(void) {

  numgrps = 0;
  free(grps);
  grps = NULL;
}

void PrintUsage(char *prog) {

  fprintf(stderr, "usage: %s { pid } ...\n (report process credentials)\n",
          prog);
  exit(-1);
}


int main(int argc, char **argv) {

  int ret;
  FILE *statfd;
  proc_t *curproc = NULL;

  if (argc < 2) PrintUsage(argv[0]);


  /* Initialize ptools_arg library routines */
  ret = ptools_arg_setup(argc, argv, NULL, NULL);

  if (ret < 0) {
    fprintf(stderr, "Error, ptools_arg_setup(): %d\n", ptools_errno);
    return -1;
  }

  /* Loop through all pid's given on command line */
  while (1) {

    ResetGlobalData();

    curproc = ptools_arg_getnext();
```

```
    if (!curproc) {
      if (ptools_errno == 0) break;

      else if (ptools_errno == INVAL_PID)
        fprintf(stderr, "%s: cannot examine %s: no such process\n",
                argv[0], ptools_arg_str);
      else
        fprintf(stderr, "Error: %d with arg: %s\n",
                ptools_errno, ptools_arg_str);
                                                                    170
      continue;
    }

    /* Open status file of cur_pid */
    statfd = fOpenPid(curproc->pid, "status");
    if (!statfd) {
      printf("%s: cannot examine %d: permission denied\n",
             argv[0], curproc->pid);
      continue;
    }                                                               180

    ParseGroups(statfd);

    fclose(statfd);

    PrintCred(curproc);

    freeproc(curproc);
    curproc = NULL;
  } /* for() */                                                     190

  ptools_arg_finish();

  return 0;
}
```

G.2 pflags

README

```
pTools Process flag reporting utility (pflags)

Based off the Sun Microsystems pflags utility. Reports various
flags for each process (light weight process in the solaris case).
Information such as data_model, global process flags, current system
call waiting in, signal mask etc.

Usage:
        pflags pid ...
                                                                    10
NOTES:
        -r flags could be implemented by re-evaluating the pstop command.

        The signal and fault tracing parts (ala PCSTRACE & PCSFAULT) of
                the Solaris pflags command are not implemented because
```

this _feature_ does not exist yet in the Linux kernel yet.

The system call entry and exit parts (ala PCSENTRY & PCSEXIT) of the Solaris pflags command are not implemented either because they do not exist in the Linux kernel yet.

Makefile

```
#
# pflags Makefile
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

all: pflags

pflags: pflags.c
	$(CC) $(CFLAGS) -o pflags pflags.c $(LIBS)

clean:
	rm -f pflags *.o *~
```

pflags.c

```
/*
 * pflags.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * pTools: Process Information Utilities
 *
 * Reports various flags and information for a process
 *
 * Compile:
 *    gcc -o -I../../include -I. -I.. -I../../lib -I../../libsl pflags pflags.c -lproc -lptools
 *
 * Usage:
 *    pflags <pid> ...
 */

#include <config.h>
#include <sig_list.h>
#include <syscall_list.h>
#include <pflags.h>

char *flags2str(unsigned long flags) {
  static char ret[BUFLN];
  char *p = ret;
```

```
  if (!flags) return "0";

  *p = '\0';

  if (flags & PF_ALIGNWARN)                                          30
    strcat(p, "|PF_ALIGNWARN");
  if (flags & PF_STARTING)
    strcat(p, "|PF_STARTING");
  if (flags & PF_EXITING)
    strcat(p, "|PF_EXITING");
  if (flags & PF_FORKNOEXEC)
    strcat(p, "|PF_FORKNOEXEC");
  if (flags & PF_SUPERPRIV)
    strcat(p, "|PF_SUPERPRIV");
  if (flags & PF_DUMPCORE)                                           40
    strcat(p, "|PF_DUMPCORE");
  if (flags & PF_SIGNALED)
    strcat(p, "|PF_SIGNALED");
  if (flags & PF_MEMALLOC)
    strcat(p, "|PF_MEMALLOC");
  if (flags & PF_USEDFPU)
    strcat(p, "|PF_USEDFPU");

  if (*p == '|') p++;
                                                                     50
  return p;
}


/* We do the open()/read() first becuase if it fails we
 * shouldn't return any information about the process.
 * This is close to the way Solaris does it and could
 * be changed
 */
int do_pflags(proc_t *p) {                                           60
  int fd;
  char buf[BUFLN];
  unsigned long long sigpend;
  u_int32_t pend1, pend2;

  struct ptools_struct pt_info;


  /* Solaris has multiple LWP's but we don't (really),
   * this support will be added later                                70
   */
  snprintf(buf, BUFLN, "/proc/%d/stack", p->pid);
  fd = open(buf, O_RDONLY);
  if (fd < 0) {
#ifdef _PTOOLS_DEBUG
    fprintf(stderr, "Error on open() of %s\n", buf);
#endif
    return -1;
  }
                                                                     80
  if (read(fd, &pt_info, sizeof(struct ptools_struct)) <
      sizeof(struct ptools_struct)) {
```

```
#ifdef _PTOOLS_DEBUG
    fprintf(stderr, "Error on read() of %s\n", buf);
#endif
    return -1;
  }



                                                                    90
  printf("%d:\t%s\n", p->pid, *(p->cmdline));
  /* NOTE: This is hard-coded for the i386 architecture
   * We should change this by accessing the binary with the
   * elf libraries if possible
   */
  printf("\tdata model = _ILP32");

  if (p->flags) {
    printf(" flags = %s", flags2str(p->flags));
  }                                                                 100
  printf("\n");


  sigpend = ConvertMask(p->signal);
  if (sigpend < 0) {
#ifdef _PTOOLS_DEBUG
    fprintf(stderr, "ConvertMask(%s) failed.\n", p->signal);
#endif
    return -1;
  }                                                                 110

  pend1 = *((u_int32_t*)(&sigpend));
  pend2 = *( ((u_int32_t*)&sigpend) + 1);

  /* Gotta switch the order of printing due to endian diff
   * between SPARC and Intel
   */
  if (pend1 || pend2)
    printf(" sigpend = 0x%.8x,0x%.8x\n", pend2, pend1);
                                                                    120
  /* We would have individual flags if we had LWP's
   * printf(" /1:\tflags: = %s", );
   */
  printf(" /1:");
  if (pt_info.pr_syscall >= FIRST_SYSCALL) {
    int n;
    /* skip over sys_ in name returned from syscallnum2str() */
    printf("\t[ %s(", syscallnum2str(pt_info.pr_syscall, buf, BUFLN)+4);

    for(n = 0; n < syscall_numargs(pt_info.pr_syscall); n++) {      130
      if (n) printf(",");
      printf("0x%lx", pt_info.pr_sysarg[n]);
    }

    printf(") ]");
  }
  printf("\n");
```

```
                                                                    140
  return 0;
}


void PrintUsage(const char *name) {
  const char *ptr;

  if ( (ptr = strrchr(name, '/')) != NULL)
    ptr += 1;                                                       150
  else
    ptr = name;

  fprintf(stderr, "usage: %s pid . . .\n", ptr);
  fprintf(stderr, "  (report process status flags)\n");
  /*
    fprintf(stderr, " -r : report registers\n");
  */

  exit(-1);                                                         160
}


int main(int argc, char **argv) {

  proc_t *curproc;

  if (argc < 2) PrintUsage(argv[0]);

  if (ptools_arg_setup(argc, argv, NULL, NULL) < 0) {               170
    fprintf(stderr, "Error, ptools_arg_setup(): %d\n",
            ptools_errno);
    return -1;
  }

  while (1) {
    curproc = ptools_arg_getnext();
    if (!curproc) {
      if (ptools_errno == 0) break;
                                                                    180
      else if (ptools_errno == INVAL_PID)
        fprintf(stderr, "%s: cannot examine %s: no such process\n",
                argv[0], ptools_arg_str);
      else
        fprintf(stderr, "Error: %d with arg: %s\n",
                ptools_errno, ptools_arg_str);

      continue;
    }
                                                                    190
    if (do_pflags(curproc) < 0)
      printf("Error processing PID %d, skipping.\n", curproc->pid);

    freeproc(curproc);
    curproc = NULL;
  } /* while() */
```

```
  ptools_arg_finish();

  return 0;                                                          200
}
```

G.3 pldd

README

```
pTools Process Dynamic Library Listing Utility (pldd)

Based on Sun Microsystems pldd utility. Displays linked library
information about a process, including those loaded with dlopen(3).

Usage:
        pldd { pid } ...

NOTES:
        Library path information is included in the /proc/pid/maps          10
        file, this info can be used for initial reporting of linked
        library information. Even better would be to use the device
        and inode information in the maps file to determine
        the type of file and if it is a library to determine the path
        (somehow) and then report it. The kernel must do this somehow
        to report it in the maps file???
          - Retrieved from inode information somehow, see
          fs/proc/array.c around line 600

        Shared Memory areas obtained through shmget() are not reported     20
        by pldd. These areas appear in the maps file with the ---s
        permission setting. These are not reported by pldd in Solaris
        either.
```

Makefile

```
#
# pldd Makefile
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

LIBS += -lfl                                                         10

LEX = /usr/bin/lex
CO = /usr/bin/co

all: pldd
```

```
pldd: pldd.c lex.yy.o $(PTOOLS_INC)/ptools_maps.h
	$(CC) $(CFLAGS) -o pldd pldd.c lex.yy.o $(LIBS)

lex.yy.o: maps.l $(PTOOLS_INC)/ptools_maps.h
	$(LEX) maps.l; \
	$(CC) $(CFLAGS) -c lex.yy.c

clean:
	rm -f *.o *~ pldd lex.yy.c
```

---

pldd.c

```
/*
 * pldd.c
 * Stphen Oberther
 * oberther@cs.fsu.edu
 *
 * pTools Process Library Reporting Tool
 *
 * Reports the dynamic libraries linked to a process
 *
 * Usage:
 *   pldd [ pid ] . . .
 *
 * NOTES:
 *   Check on how -F flag in Solaris version is implemented,
 * has to do with process locking I think.
 *
 */

#include <config.h>

#include <ptools_maps.h>


extern FILE *yyin;
int yylex(void);

extern struct maps_ent *head;

/* Linked-List of library names */
struct liblist {
  char *name;
  struct liblist *next;
};



/*
 * void CleanupMapsList(struct maps_ent *):
 *   Reclaims dynamic memory of the single-linked
 * list of map file entries parsed by lex
 */
```

```
void CleanupMapsList(struct maps_ent *p) {
  struct maps_ent *cur;

  cur = p;

  while (cur) {
    struct maps_ent *tmp;

    tmp = cur;                                                    50
    cur = cur->next;
    free(tmp);
  }

}


/*
 * void DumpMapsList(struct maps_ent *):
 *   Prints out maps_ent structures to stdout                    60
 */
void DumpMapsList(struct maps_ent *p) {

  if (p->next) DumpMapsList(p->next);

  printf("%08lx-%08lx %s %08lx %02d:%02d %lu\t%s\n", p->start, p->end,
        p->perms, p->offset, p->maj, p->min, p->inode, p->path);
}

                                                                  70
/*
 * void FreeLibrarList(struct liblist *):
 *   Reclaims dynamic memory used in creating
 * the linked list of library names. The name data
 * element is not reclaimed because it points to the
 * static array "path" in the maps_ent structure.
 */
void FreeLibraryList(struct liblist *p) {
  struct liblist *cur;
                                                                  80
  cur = p;
  while (cur) {
    struct liblist *tmp;

    tmp = cur;
    cur = cur->next;
    free(tmp);
  }

}                                                                 90


/*
 * void DumpLibraryList(struct liblist *):
 *   Prints the linked list of library
 * names to stdout.
 */
void DumpLibraryList(struct liblist *p) {
```

```
  struct liblist *cur;

  cur = p;

  while (cur) {
    printf("%s\n", cur->name);
    cur = cur->next;
  }

}

/*
 * IsShared(const char *):
 *   Return true if the shared flag is set
 * in p, false if not
 */
int IsShared(const char *p) {

  if (p[3] == 's') return TRUE;

  return FALSE;

}


/*
 * BuildLibraryList(struct maps_ent *):
 *
 *   Build a unique list of libraries, excluding shared
 * memory regions, from the mapping information in p
 * Return the list, or NULL on error
 */
struct liblist *BuildLibraryList(struct maps_ent *p) {

  struct liblist *ret, *ptr;
  struct maps_ent *cur;

  ret = NULL;
  cur = p;

  while (cur) {
    if (*cur->path) {

      /*
        printf("%s\t", cur->path);
      */

      if (IsShared(cur->perms)) {
        cur = cur->next; continue;
      }

      if (ret) {
        /* Based on 2.4.x, x <= 13, maps file is layed out so that
         * only need to compare current against most recent added to
         * list */
        if ( (strlen(ret->name) == strlen(cur->path)) &&
```

```
            (strncmp(ret->name, cur->path, strlen(ret->name)) == 0) ) {
          /* Already added cur->path to library list */

          /*
             printf("SKIPPED\n");
          */

          cur = cur->next;
          continue;
        }
      }

      ptr = (struct liblist *)malloc(sizeof(struct liblist));
      if (!ptr) {
        fprintf(stderr, "Memory allocation error\n");
        FreeLibraryList(ret);
        return NULL;
      }
      /* Latch ptr onto list */
      ptr->name = cur->path;
      ptr->next = ret;
      ret = ptr;

      /*
        printf("ADDED\n");
      */

    }
    cur = cur->next;
  }

  return ret;
}


/*
 * void PrintUsage(const char *):
 *   Prints the program usage.
 */
void PrintUsage(const char *name) {
  fprintf(stderr,
          "usage: %s { pid } ...\n (reports process dynamic libraries)\n",
          name);
  exit(-1);
}



/* No Comment */
int main(int argc, char **argv) {
  int ret;
  char buf[BUFLN];
  FILE *ifile;

  struct liblist *libhead = NULL;

  proc_t *curproc;
```

```
if (argc < 2) PrintUsage(argv[0]);

ret = ptools_arg_setup(argc, argv, NULL, NULL);

if (ret < 0) {
  fprintf(stderr, "Error, ptools_arg_setup(): %d\n", ptools_errno);
  return -1;
}

/* Go through each returned value and parse
 * the maps file for each pid returned from readproc
 */
while (1) {
  curproc = ptools_arg_getnext();
  if (!curproc) {
    if (ptools_errno == 0) break;

    else if (ptools_errno == INVAL_PID)
      fprintf(stderr, "%s: cannot examine %s: no such process\n",
              argv[0], ptools_arg_str);
    else
      fprintf(stderr, "Error: %d with arg: %s\n",
              ptools_errno, ptools_arg_str);

    continue;
  }

  printf("%d:\t", curproc->pid);

  snprintf(buf, BUFLN, "/proc/%d/maps", curproc->pid);

  /* Open the input file */
  ifile = fopen(buf, "r");
  if (!ifile) {
    fprintf(stderr, "Error opening: %s\n", buf);
    continue;
  }

  /* Parse the maps file */
  yyin = ifile;
  yylex();
  fclose(ifile);

  /*
    printf("\n*******************************\n");
    DumpMapsList(head);
    printf("\n********** Building Library List ***********\n");
  */

  /* Build the list of libraries */
  libhead = BuildLibraryList(head);
  if (!libhead) {
    printf("\n");
    continue;
  }
```

```
    /* Print the list of libraries and cleanup */
    DumpLibraryList(libhead);

    CleanupMapsList(head);
    FreeLibraryList(libhead);
    head = NULL;
    libhead = NULL;
    freeproc(curproc);
    curproc = NULL;
  }

  /* Final cleanup */
  CleanupMapsList(head);
  FreeLibraryList(libhead);
  head = NULL;
  libhead = NULL;
  freeproc(curproc);
  curproc = NULL;

  ptools_arg_finish();

  return 0;
}
```

maps.l

```
%{
/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Lex file to parse contents of the /proc/pid/maps file.
 */

#include <config.h>
#include <ptools_maps.h>


struct maps_ent *current, *head = NULL;


void CleanupMapsList(struct maps_ent *p);

%}

loc  [0-9a-f]{8}
perm [-rwxsp]{4}
dev [0-9]{2}:[0-9]{2}
inode [0-9]+

 /* does not cover case where filename containes \n,
  * should be modified so that file names only valid if they
  * contain characters from the "portable filename character set"
```

```
  */
file (\/[^\/\n]+)+

%%

[ \t]+      ;

\n          { /* printf("**********\n"); */ }

 /* Memory Location */
{loc}-{loc} {
              current = (struct maps_ent *)malloc(sizeof(struct maps_ent));
              if (!current) {
                 fprintf(stderr, "malloc() error\n");
                 CleanupMapsList(head);
                 exit(-1);
              }
              memset(current, 0, sizeof(struct maps_ent));

              current->next = head;
              head = current;
              sscanf(yytext, "%lx-%lx", &(current->start),
                             &(current->end));
              /*
               printf("Memory Loc: %lx-%lx\n", current->start, current->end);
              */
            }

 /* Address Space Permissions */
{perm}    {
              strncpy(current->perms, yytext, 5);
              /*
                printf("Address Space perms: %s\n", current->perms);
              */
            }

 /* Offset into the file/device etc */
{loc}       {
              sscanf(yytext, "%lx", &(current->offset));
              /*
                printf("Offset: %lx\n", current->offset);
              */
            }

 /* The device maj:min numbers */
{dev}       {
              sscanf(yytext, "%d:%d", &(current->maj),
                             &(current->min));
              /*
                printf("Device (maj:min): %d:%d\n", current->maj,
                     current->min);
              */
            }

 /* Inode on the device */
{inode}     {
              sscanf(yytext, "%lu", &(current->inode));
```

```
            /*
              printf("File INode: %lu\n", current->inode);
            */
          }

 /* File name where appropriate */
{file}      {
              strncpy(current->path, yytext, BUFLN);
            /*
              printf("File Name: %s\n", current->path);
            */
          }

%%
```

G.4 prun

README

pTools Process Execution Modification Utility (prun)

Based off the Sun Microsystems prun utility. Set each process running (inverse of pstop).

Usage:
prun { pid } ...

NOTES:
- Solaris accomplishes this through the use of the /proc/pid/ctl file.
- Our implementation will initially use the kill() system call to send a SIGCONT, which cannot be caught or ignored.

- 11-28-01 -
SIGCONT can be ignored or handled. Even if stopped with a SIGSTOP the handeler will be executed and the program will not begin execution again.

A new means of implementing prun must be devised. A new signal (duplicating SIGCONT) that can't be caught or ignored seems like the most obvious solution

- 12-21-01 -
/proc/<pid>/ctl file has been added into fs/proc/base.c and is now being used to implement prun and pstop. The ctl file implementation is still undergoing development however.

Makefile

```
#
# make rules for prun
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

all: prun

prun: prun.c
	$(CC) $(CFLAGS) -o prun prun.c $(LIBS)

clean:
	rm -f *~ *.o prun
```

prun.c

```
/*
 * prun.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * pTools: Process Information Utilities
 *
 * Restarts a process that was previously stopped by pstop
 *
 * Usage:
 *    prun ¡pid¿ . . .
 *
 * Compile:
 *    gcc -o prun prun.c -lproc -lptools
 *
 */

#include <config.h>
#include <procfs˙ctl.h>


void PrintUsage(char *name) {

  fprintf(stderr, "usage: %s pid . . .\n", name);
  fprintf(stderr, "  (set stopped processes running)\n");
}


int main(int argc, char **argv) {

  int ret, ctlfd;
  proc_t *curproc;
  char buf[BUFLN];
```

```
ctl_t msg;


if (argc == 1) {
  PrintUsage(argv[0]);
  return −1;
}
else if (argc < 1) {
  PrintUsage("prun");
  return −1;
}

/* Initialize ptools_arg library routines */
ret = ptools_arg_setup(argc, argv, NULL, NULL);
if (ret < 0) {
  fprintf(stderr, "Error, ptools_arg_setup(): %d\n", ptools_errno);
  return −1;
}

/* Call getnext() repeatedly until no more PID's are returned */
while(1) {
  curproc = ptools_arg_getnext();

  if (!curproc) {
    if (ptools_errno == 0) break;

    fprintf(stderr, "Error: %d with arg: %s\n",
            ptools_errno, ptools_arg_str);
    continue;
  }

  fprintf(stderr, "Processing PID: %d\tRunning\n", curproc−>pid);

  snprintf(buf, BUFLN, "/proc/%d/ctl", curproc−>pid);
  ctlfd = open(buf, O_WRONLY);

  if (ctlfd < 0) {
    switch(errno) {
    case EACCES:
      fprintf(stderr, "%d: Permission denied\n", curproc−>pid);
      freeproc(curproc); continue;
      break;

    case ENOENT:
      fprintf(stderr, "%s: cannot control %d: no such process\n",
              argv[0], curproc−>pid);
      freeproc(curproc); continue;
      break;
    };
  }

  msg = PCRUN;

  if (write(ctlfd, (char *)(&msg), sizeof(ctl_t)) < 0) {
    fprintf(stderr, "%s: error during write() to ctl file of %d\n",
            argv[0], curproc−>pid);
  }
```

```
    close(ctlfd);

    freeproc(curproc);
  }

  ptools_arg_finish();


  return 0;                                                    100

}
```

G.5 psig

README

```
pTools Process Signal utility (psig)

Based off the Sun Microsystems psig utility. Displays information relating
to how the process handles signals. For each signal the set of flags
for each, the action, and what processes are blocked while handling the
current signal.

Usage:
        psig pid ...
                                                               10
NOTES:
        Output format is derived from Sun Microsystems implementation after
          reviewing thier psig implementation

        ** Check return value from do_psig function in main() incase an error
        occurs
```

Makefile

```
#
# make rules for psig
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

all: psig                                                      10

psig: psig.c
        $(CC) $(CFLAGS) -o psig psig.c $(LIBS)

clean:
        rm -f psig *.o *~
```

psig.c

```
/*
 * psig.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * pTools: Process Credential Reporting Tool
 *
 * Reports information regarding signal handeling for a process.
 *
 * Usage:                                                              10
 *    psig <pid> ...
 */

#define _PTOOLS_DEBUG
#include <config.h>
#include <signal.h>
#include <sigact.h>
#include <sig_list.h>

                                                                       20

char *SignalFlags(int sig, int flags, char *buf, size_t len) {

  int flagmask = (SA_ONSTACK|SA_RESTART|SA_NODEFER|SA_RESETHAND|SA_SIGINFO);

  if (sig == SIGCLD)
    flagmask |= (SA_NOCLDSTOP|SA_NOCLDWAIT);

  *buf = '\0';
                                                                       30
  if (flags & ~flagmask) {
    snprintf(buf, len, ",0x%x,", flags & ~flagmask);
    len -= strlen(buf) - 1;
  }
  else if (flags == 0)
    return(buf);

  if (len < 1) goto out;

  if (flags & SA_RESTART) {                                            40
    strcat(buf, ",RESTART");
    len -= 8; if (len < 1) goto out;
  }
  if (flags & SA_RESETHAND) {
    strcat(buf, ",RESETHAND");
    len -= 10; if (len < 1) goto out;
  }
  if (flags & SA_ONSTACK) {
    strcat(buf, ",ONSTACK");
    len -= 8; if (len < 1) goto out;                                   50
  }
```

```
  if (flags & SA_SIGINFO) {
    strcat(buf, ",SIGINFO");
    len -= 8; if (len < 1) goto out;
  }
  if (flags & SA_NODEFER) {
    strcat(buf, ",NODEFER");
    len -= 8; if (len < 1) goto out;
  }

  if (sig == SIGCLD) {
    if (flags & SA_NOCLDWAIT) {
      strcat(buf, ",NOCLDWAIT");
      len -= 10; if (len < 1) goto out;
    }
    if (flags & SA_NOCLDSTOP) {
      strcat(buf, ",NOCLDSTOP");
      len -= 10;
    }
  } /* if (sig == SIGCLD) */

 out:

  *buf = '\t';
  return(buf);
}

/*
 * inmask(int, unsigned long long):
 *   Checks to see if sig is contained within
 * mask. Returns 1 iff sig is found in mask.
 */
int inmask(int sig, unsigned long long mask) {

  mask = mask >> (sig - 1);
  if (0x1 & mask) return 1;

  return 0;
}


/*
 * do_sigact(proc_t):
 *
 */
int do_sigact(proc_t *p) {
  unsigned long long blockmask;
  char buf[BUFLN];
  int tmp, fd, sig;
  struct kern_sigaction signals[NSIG], *cursig;

  cursig = NULL;
  memset(signals, 0, NSIG*sizeof(struct kern_sigaction));

  /* Setup mask of blocked signals */
  if ( (blockmask = ConvertMask(p->blocked)) < 0) {
#ifdef _PTOOLS_DEBUG
```

```
    fprintf(stderr, "ConvertMask(%s) failed.\n", p->blocked);
#endif
    return -1;
  }

#ifdef _PTOOLS_DEBUG
  printf("blockmask : 0x%016Lx\n", blockmask);
#endif


  /* Open sigact file */
  snprintf(buf, BUFLN, "/proc/%d/sigact", p->pid);
  if ( (fd = open(buf, O_RDONLY)) < 0) {
#ifdef _PTOOLS_DEBUG
    fprintf(stderr, "open(%s, O_RDONLY): failed, %d\n", buf, errno);
#endif
    return -1;
  }

  tmp = NSIG * sizeof(struct kern_sigaction);
  if (read(fd, (void *)signals, tmp) != tmp) {
    close(fd);
#ifdef _PTOOLS_DEBUG
    fprintf(stderr, "read(%d, %p, %d): failed, %d\n", fd, signals, tmp, errno);
#endif
    return -1;
  }
  close(fd);


  /* Print PID and command line */
  printf("%d:\t%s\n", p->pid, p->cmdline[0]);

  /* Go through each signal */
  for(sig = 1; sig < SIGRTMIN; sig++) {
    char str[100];

    tmp = 0;
    cursig = &(signals[sig-1]);
    cursig->flags &= ~SA_RESTORER; /* SA_RESTORER is set by
                                    * libc, during the library call
                                    * so we remove it here
                                    */
    printf("%s\t", signum2str(sig, buf, BUFLN)+3); /* Skip the initial
                                                    * SIG */
    /* Print status */
    if (inmask(sig, blockmask))
      printf("blocked,");

    if (cursig->handler == SIG_DFL)
      printf("default");

    else if (cursig->handler == SIG_IGN)
      printf("ignored");

    else {
      printf("caught");
```

```
      tmp = 1;
    }

    /* If caught, go through and print the signals that are blocked
     * while processing the current signal */
    if (tmp) {
      int cnt;
      SignalFlags(sig, cursig->flags, str, 100);

      printf("%s", (*str != '\0')?str : "\t0");
      tmp = 0;
      for (cnt = 1; cnt < NSIG; cnt++) {
        if (inmask(cnt, cursig->mask)) {
          printf(tmp++? "," : "\t");
          printf("%s", signum2str(cnt, str, 100)+3);
        }

      } /* for() */

    }

    else if (sig == SIGCLD) {
      printf("%s", SignalFlags(sig, cursig->flags & (SA_NOCLDSTOP),
                               str, 100));
    }

    printf("\n");
  }
  return 0;
}



void PrintUsage(char *prog) {

  fprintf(stderr, "usage:\t%s pid . . .\n", prog);
  fprintf(stderr, "  (report process signal actions)\n");

  exit(-1);
}



int main(int argc, char **argv) {
  int ret;
  proc_t *curproc = NULL;

  if (argc <= 1) PrintUsage(argv[0]);

  /* Initialize ptools_arg library routines */
  ret = ptools_arg_setup(argc, argv, NULL, NULL);

  if (ret < 0) {
    fprintf(stderr, "Error, ptools_arg_setup(): %d\n", ptools_errno);
    return -1;
  }
```

```
  /* Loop through all pid's given on command line */
  while (1) {
    curproc = ptools_arg_getnext();

    /* Deal with errors */
    if (!curproc) {
      if (ptools_errno == 0) break;                                    230
      else if (ptools_errno == INVAL_PID)
        fprintf(stderr, "%s: cannot examine %s: no such process\n",
                argv[0], ptools_arg_str);
      else
        fprintf(stderr, "Error: %d with arg: %s\n",
                ptools_errno, ptools_arg_str);

      continue;
    }
                                                                       240
    /* Perform the actual psig operation on the current process
     * we are investigating */
    do_sigact(curproc);
    freeproc(curproc);
    curproc = NULL;
  }

  ptools_arg_finish();

  return 0;                                                            250
}
```

G.6 pstop

README

```
pTools Process Execution Modification Utility (pstop)

Based off the Sun Microsystems pstop utility. Stops each process
specified on the command line.

Usage:
        pstop { pid } ...

NOTES:
        - Solaris accomplishes this through the use of the /proc/pid/ctl   10
        file.
        * Our implementation will initially use the kill() system call to
        send a SIGSTOP, which cannot be caught or ignored.

        - Revision 2.1 now uses the /proc/<pid>/ctl file method of
        stopping the process.
```

Makefile

```
#
# make rules for pstop
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

all: pstop

pstop: pstop.c
	$(CC) $(CFLAGS) -o pstop pstop.c $(LIBS)

clean:
	rm -f *~ *.o pstop
```

pstop.c

```
/*
 * pstop.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * pTools: Process Information Utilities
 *
 * Stops the execution of a process
 *
 * Usage:
 *    pstop <pid> ...
 */

#include <config.h>
#include <procfs_ctl.h>

void PrintUsage(char *name) {

  fprintf(stderr, "usage: %s pid ...\n", name);
  fprintf(stderr, "  (stop processes with /proc request)\n");
}


int main(int argc, char **argv) {

  int ret, ctlfd;
  proc_t *curproc = NULL;
  char buf[BUFLN];
  ctl_t msg;

  if (argc == 1) {
    PrintUsage(argv[0]);
```

```
  return −1;
}
else if (argc < 1) {
  PrintUsage("pstop");
  return −1;
}

/* Initialize ptools_arg library routines */
ret = ptools_arg_setup(argc, argv, NULL, NULL);
if (ret < 0) {
  fprintf(stderr, "Error, ptools_arg_setup(): %d\n", ptools_errno);
  return −1;
}

/* Call getnext() repeatedly until no more PID's are returned */
while(1) {
  curproc = ptools_arg_getnext();

  if (!curproc) {
    if (ptools_errno == 0) break;

    fprintf(stderr, "Error: %d with arg: %s\n",
            ptools_errno, ptools_arg_str);
    continue;
  }

  fprintf(stderr, "Processing PID: %d\tStopping\n", curproc−>pid);

  snprintf(buf, BUFLN, "/proc/%d/ctl", curproc−>pid);
  ctlfd = open(buf, O_WRONLY);

  if (ctlfd < 0) {
    switch(errno) {
    case EACCES:
      fprintf(stderr, "%d: Permission denied\n", curproc−>pid);
      freeproc(curproc); continue;
      break;

    case ENOENT:
      fprintf(stderr, "%s: cannot control %d: no such process\n",
              argv[0], curproc−>pid);
      freeproc(curproc); continue;
      break;
    };
  }

  msg = PCSTOP;

  if (write(ctlfd, (char *)(&msg), sizeof(ctl_t)) < 0) {
    fprintf(stderr, "%s: error during write() to ctl file of %d\n",
            argv[0], curproc−>pid);
  }

  close(ctlfd);

  freeproc(curproc);
  curproc = NULL;
```

```
  }

  ptools_arg_finish();


  return 0;

}
```

G.7 ptree

README

```
pTools Process Tree utility (ptree)

Based off the Sun Microsystems ptree utitlity. Prints a tree view of the
process running on the system in relation to the command line arguments. If
a user is given all processes owned by that user are displayed. If a PID is
given the processes in direct relation are displayed. An argument of all
digits is taken to be a PID. The default action is all processes on the
system.

Usage:                                                                  10
        ptree < pid | username >

NOTES:
   Initial Implementation Notes
        - Exact duplicate of ptree under Solaris

        - Emphasis on data structures and output functions
                - Easy to add functionality later if concentration
                on these parts
                                                                        20
   Redhat Notes
        - procps-2.0.7 RPM and SRPM will not allow ptree to be compiled.
          These do not provide the necessary header files
        - Install procps-2.0.7.tar.gz by hand in order for ptree to
          compile and execute correctly
```

Makefile

```
#
# ptree Makefile
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

all: ptree                                                              10
```

```
ptree: ptree.c
	$(CC) $(CFLAGS) -o ptree ptree.c $(LIBS)

clean:
	rm -f ptree *.o *~
```

---

ptree.c

```
/*
 * ptree.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * pTools: Process Information Utilities
 *
 * Displays process relationships in a tree format
 *
 * Compile:                                                          10
 *   gcc -Wall -g -o ptree ptree.c -lproc
 * Requires:
 *   libproc (Available in procps package
 *            ftp://people.redhat.com/johnsonm/procps/)
 */

#include <config.h>

#define INIT 1
                                                                     20
enum type { ProcessID, UserID };

enum bool_t { false = 0, true = 1};
typedef enum bool_t bool;

/* Tree node structure */
struct tnode_t {
  proc_t *pent;
  bool visited;
  struct tnode_t *l, *r;                                             30
};
typedef struct tnode_t tnode;


/* Link List node */
struct llnode_t {
  tnode *data;
  struct llnode_t *next;
};
typedef struct llnode_t llnode;                                      40


int PrintInit = 0;
const char *PROGNAME;
tnode *tree;
```

```
/*
 * tnode *MkTnode(proc_t, tnode *, tnode *):
 *   Makes a tree node and initializes appropriate
 * fields.  Returns tree node or NULL on error.
 */
tnode *MkTnode(proc_t *data, tnode *l, tnode *r) {
  tnode *ret;

  ret = (tnode *)calloc(1, sizeof(tnode));
  if (!ret)
    return NULL;

  ret->pent = data;
  ret->l = l; ret->r = r;
  ret->visited = false;

  return ret;
}

/**********************************
 * Modified from readproc.c in
 * procps 2.0.7
 **********************************/
/* tnode *LookupPID(tnode *, int):
 *   Traverse tree 't' breadth-first looking
 * for a process with pid p. Return pointer
 * to tree entry or NULL on error
 */
tnode *LookupPID(tnode* t, int p) {

  tnode* tmp = NULL;

  if (!t)
    return NULL;

  /* look here/terminate recursion */
  if ( (t->pent) && (t->pent->pid == p) )
    return t;

  /* recurse over children */
  if ((tmp = LookupPID(t->l, p)))
    return tmp;

  /* recurse over siblings */

  if ((tmp = LookupPID(t->r, p)))
    return tmp;

  return NULL;
}


/*
 * tnode *MakeTree(PROCTAB *):
 *    Make a tree from the proc entries
 * found in p and return it. Builds a
 * Left child, Right sibling tree.
```

```
 * NULL returned on error.
 */
tnode *MakeTree(PROCTAB *p) {

  tnode *t = NULL;
  tnode *srch = NULL;
  tnode *tmp = NULL;                                                    110
  proc_t *cur;

  if (!p)
    return NULL;

  cur = readproc(p, NULL);

  if (cur)   t = MkTnode(cur, NULL, NULL);
  if (!t) return NULL;
                                                                        120
  if (cur->pid != INIT) t = MkTnode(NULL, NULL, t);
  if (!t) return NULL;

  /* Pass one over /proc, generates initial tree */
  while ( (cur = readproc(p, NULL)) ) {

    if (cur->pid == INIT) {
      t->pent = cur;
      continue;
    }                                                                   130

    srch = LookupPID(t, cur->ppid);

    if (srch) {
      /* Latch child(cur) under right side of parent(srch) */
      tmp = MkTnode(cur, NULL, srch->l);
      if (!tmp) return NULL;
      srch->l = tmp;
    }
                                                                        140
    else {
      /* Place cur on right side of tree to be scanned next */
      /* THIS SHOULD NEVER HAPPEN WITH LIBPROC */
      tmp = MkTnode(cur, NULL, t->r);
      if (!tmp) return NULL;
      t->r = tmp;
    }

  }
                                                                        150
  /* Pass two: Look at tree.r only */
  /* Not needed with libproc-2.0.7, processes are returned in a sorted
   * manner, apparently by process creation time
   */

  return t;
}


/*                                                                      160
```

```
 * ClearVisited(tnode *t):
 *    Sets visited to false for t and all child nodes
 * (ie t->l and t->r) of t.
 */
void ClearVisited(tnode *t) {
  if (!t) return;

  ClearVisited(t->r);
  ClearVisited(t->l);

  t->visited = false;
}

/*
 * FreeTree(tnode *):
 *   Reclaim the memory used by t
 */
void FreeTree(tnode *t) {

  if (!t) return;

  FreeTree(t->r);
  FreeTree(t->l);

  freeproc(t->pent);
  t->r = NULL; t->l = NULL;
  free(t);
}


/*
 * PrintProc(proc_t *, int):
 *   Actually prints out a single process line
 * of a process tree.
 */
void PrintProc(proc_t *p, int t) {

  if (!p) return;

  if (t)
    printf("%*c", t, ' '); /* Print 'tab' spaces */

  printf("%-6d", p->pid);

  if (p->cmdline)
    printf("%s\n", p->cmdline[0]);
  else
    printf("%s\n", p->cmd);

}


/*
 * PrintTree(proc_t *):
 *   Prints the tree t in a manner similiar
 * to ptree on Solaris.
 */
```

```
void PrintTree(tnode *t, int tab) {

  if (!t) return;

  PrintTree(t->r, tab);

  if (t->visited == false) {
    PrintProc(t->pent, tab);
    t->visited = true;
  }

  if (t->l) {
    tab += 2;
    PrintTree(t->l, tab);
    tab -= 2;
  }

}



/*
 * PrintList(llnode *):
 *   Handles outputing the elements of 'list' in
 * a tree format
 */
void PrintList(llnode *list) {
  llnode *tmp;
  int tab;


  if (!list) return;
  /* Print out each list element and then the rest of the tree
   * from the last item
   */
  tab = 0;
  for (tmp=list; tmp; tmp=tmp->next) {
    /* continue should be just fine */
    if ( (!tmp->data) || (!tmp->data->pent) ) continue;

    if ( (tmp->data->pent->pid == 1) && (!PrintInit) ) continue;

    if (tmp->data->visited) { tab+=2; continue; }

    PrintProc(tmp->data->pent, tab);
    tmp->data->visited = true;
    tab +=2;

    if (!(tmp->next)) break;
  }

  if (tmp) /* Need to check incase we continue out of loop */
    PrintTree(tmp->data->l, tab);

}
```

```
/*
 * FreeList(llnode *):
 *   Cleans up the dynamic memory
 * used by 'list'
 */
void FreeList(llnode *list) {
  llnode *tmp, *holder;

  /* Clean up the list */
  tmp = list;
  while (tmp) {

    holder = tmp;
    tmp = tmp->next;
    holder->data = NULL;
    holder->next = NULL;
    free(holder);
  }
}

/*
 * PrintPID(tnode *, int):
 *   Print out parents and children of process 'pid'
 * Searches starting at 't'. If 't' is NULL search
 * from global var 'tree'
 */
void PrintPID(tnode *t, int pid) {

  llnode *list = NULL;
  llnode *tmp;
  tnode *n;
  int id;

  id = pid;

  if (!t)
    t = tree;

  /* Build list of the nodes to be printed */
  /* Could be optimized to skip adding n if it has already
   * been visited
   */
  while ( (n = LookupPID(t, id)) ) {

    tmp = (llnode *)malloc(sizeof(llnode));
    if (!tmp) {
      printf("Internal error: malloc()\n");
      return;
    }

    tmp->next = list;
    tmp->data = n;

    list = tmp;

    if (!n->pent) break; /* can't get parent id */
    id = n->pent->ppid;
```

```
  }

  PrintList(list);
  FreeList(list);
}


/*
 * PrintUIDTree(tnode *, int, int):                                    340
 *   Prints out all the process owned by user 'id'
 */
void PrintUIDTree(tnode *t, int tab, int id) {

  if (!t) return;

  PrintUIDTree(t->r, tab, id);

  if ( (t->visited == false) && (t->pent) && (t->pent->ruid == id) ) {
    /* Build up a list of parents */                                   350
    PrintPID(NULL, t->pent->pid);
  }


  if (t->l) {
    tab += 2;
    PrintUIDTree(t->l, tab, id);
    tab -= 2;
  }
                                                                       360
}


/*
 * PrintUID(tnode *, const char *):
 *   Print all processes owned by 'id'
 * Looks up 'id' in the passwd file and calls
 * PrintUIDTree() which does all the work
 */
void PrintUID(tnode *t, const char *id) {                              370
  struct passwd *pwent;
  uid_t uid;

  /* Lookup id */
  pwent = getpwnam(id);
  if (!pwent) {
    printf("%s: cannot find %s passwd entry\n", PROGNAME, id);
    return;
  }
  uid = pwent->pw_uid;                                                 380

  /* Print the actual processes in tree format
   * that are owned by id */
  PrintUIDTree(t, 0, (int)uid);

}

/*
```

```
 * PrintUsage():
 *   Print program usage                                                    390
 */
void PrintUsage(void) {

  fprintf(stderr, "usage: %s [-a] [ {pid|user} . . . ]\n", PROGNAME);
  fprintf(stderr, "  (show process trees)\n");
  fprintf(stderr, "  list can include process-ids and user names\n");
  fprintf(stderr, "  -a : include children of process 0\n");
}

                                                                           400
/*
 * ProcessArgs(tnode *, int, char **):
 *   Processes arguments sent to the program
 */
void ProcessArgs(tnode *t, int argc, char **argv) {
  int cnt;
  char c;
  extern char *optarg;
  extern int optind, opterr, optopt;
                                                                           410
  while (1) {
    /* '+' forces POSIX compliance */
    c = getopt(argc, argv, "+a");

    if (c == -1)
      break;

    switch(c) {
    case 'a':
      PrintInit = 1;                                                       420
      break;

    case '?':
      PrintUsage();
      return;
    };

  }

                                                                           430
  cnt = optind; /* first pid/uid is at argv[optind] */

  if (cnt == argc) {
    if (PrintInit) PrintTree(t, 0);
    else PrintTree(t->l, 0);
  }
  else
    for (; cnt < argc; cnt++) {
      if (isdigit(argv[cnt][0]))
        PrintPID(t, atoi(argv[cnt]));                                      440
      else
        PrintUID(t, argv[cnt]);
    }

}
```

```
int main(int argc, char **argv) {

  PROCTAB *PT = NULL;
                                                                          450
  PROGNAME = argv[0];

  PT = openproc(PROC_FILLCMD|PROC_FILLSTAT|PROC_FILLSTATUS, 0);
  if (!PT) {
    fprintf(stderr, "Internal openproc() error\n");
    return -1;
  }

  tree = MakeTree(PT);
                                                                          460
  if (!tree) {
    closeproc(PT);
    return -1;
  }

  ProcessArgs(tree, argc, argv);

  closeproc(PT);
  FreeTree(tree);
                                                                          470
  return 0;
}
```

G.8 pwdx

README

```
pTools Process Working Directory reporting utility (pwdx)

Based off the Sun Microsystems pwdx utility. Reports current working
directory of the process given on the command line.

usage: pwdx pid ...
  (show process working directory)


NOTES:                                                                    10
      Due to strange handeling of pwd link in /proc/pid some processes
           owned by the user running pwdx will not have the pwd file
           readable by the user. An 'Eterm' process is the first
           notable example. 'Eterm' is not SUID but it SGID, however
           the user who owns the 'Eterm' process cannot read the pwd
           link in /proc becuase it is owned by root.
```

Makefile

```
#
# pwdx Makefile
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ../../Config

all: pwdx

pwdx: pwdx.c
	$(CC) $(CFLAGS) $(INCDIR) $(LIBDIR) -o pwdx pwdx.c $(LIBS)

clean:
	rm -f pwdx *.o *~
```

pwdx.c

```
/
 * pwdx.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * pTools: Process Information Utilities
 *
 * Reports the current working directory of a process
 *
 * Compile:
 *   gcc -g -Wall -o pwdx pwdx.c
 *
 */

#include <config.h>


void PrintUsage(char *prog) {

  fprintf(stderr, "usage: %s pid ...\n (show process working directory)\n",
          prog);
  exit(-1);

}

int main(int argc, char **argv) {

  int cnt, ret, pid;
  char buf[BUFLN], proc[BUFLN];

  if (argc == 1) PrintUsage(argv[0]);

  for (cnt = 1; cnt < argc; cnt++) {
```

```
    pid = atoi(argv[cnt]);

    if ( ValidPid((pid_t)pid) == 0 ) {
      snprintf(proc, BUFLN, "/proc/%d/cwd", pid);

      ret = readlink(proc, buf, BUFLN);
                                                                          40
      if (ret < 0) {
        /* Determine Error message to print */
        switch(errno) {
        case EACCES:
          printf("pwdx: cannot resolve cwd for %d: Permission denied\n", pid);
          break;
        default:
          perror("readlink()");
        };
      }                                                                   50
      else {
        buf[ret] = '\0';
        printf("%d: %s\n", pid, buf);
      }

    } /* ValidPid */
    else
      printf("pwdx: cannot examine %d: no such process\n", pid);

  }                                                                       60

  return 0;

}
```

H Libraries, Headers, Scripts

The following files are the build, header, library, and script files for generating the additional tools needed to compile the ptools. They should be placed in the lib/, libsl/, include/, and scripts/, and project root directories where the pTools will be compiled. The scripts create the pflags.h and syscall_list.h header files. The instructions to build the entire pTools project are:

Untar project distribution from archive file, OR

Setup Project directory by hand:

- Place the source code for the individual ptools into $PROJDIR/src/ in seperate directories based on the ptool name.
- Place include files in $PROJDIR/include/
- Place libsl files in $PROJDIR/libsl/
- Place lib files in $PROJDIR/lib/
- Place top level Makefile and configure script in $PROJDIR
- Place scripts in $PROJDIR/scripts/

After getting the project directory ready, the remainder of the pTools can be built by:

1. Run ./configure in $PROJDIR
2. Run ‘make all‘ in $PROJDIR
3. Patch Linux kernel as described above and compile the Linux kernel
4. Reboot machine for kernel changes to take affect

Obtaining the project as a tar archive file is the preferred method of creating the project build directories.

H.1 Build Files

Top-Level Makefile

```
#
# Top level pTools Makefile
# Stephen Oberther
# oberther@cs.fsu.edu
#

include ./Config

DIRS = lib libsl src
```

```
                                                                    10
all clean:
        @for subdir in $(DIRS); do                  \
          make -C $$subdir -f Makefile $@;          \
        done

distclean: clean
        rm -f Config
```

configure script

```
#!/bin/sh
#
# Configure script for pTools
# Stephen Oberther
# oberther@cs.fsu.edu
#

echo "checking for system dependencies"
rm -rf Config
touch Config                                                        10

PTOOLS_ROOT=`pwd`
PTOOLS_INC="$PTOOLS_ROOT/include"
INCDIRS="-I. -I.. -I$PTOOLS_INC"
LIBDIRS="-L$PTOOLS_ROOT/lib -L$PTOOLS_ROOT/libsl"

# define C compiler and options

CC=gcc
CFLAGS="-Wall -g $INCDIRS $LIBDIRS"                                 20
LIBS="-lproc -lptools"

echo "PTOOLS_ROOT = $PTOOLS_ROOT" >> Config
echo "PTOOLS_INC = $PTOOLS_INC" >> Config
echo "INCDIRS = $INCDIRS" >> Config
echo "LIBDIRS = $LIBDIRS" >> Config
echo "CC = $CC" >> Config
echo "CFLAGS = $CFLAGS" >> Config
echo "LIBS = $LIBS" >> Config
                                                                    30
```

H.2 Header Files

config.h

```
#ifndef _CONFIG_H
#define _CONFIG_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
```

```
 *
 * pTools global configuration header file.
 * Included by all pTools to centralize header include
 * lines.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <fcntl.h>
#include <errno.h>
#include <pwd.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <proc/readproc.h>

#include <ptools.h>

#define BUFLN 1024
#define TRUE 1
#define FALSE 0
#endif
```

---

ptools.h

```
#ifndef _PTOOLS_H
#define _PTOOLS_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * General include file for all things general to each of
 * the pTools.
 */

#include "./ptools-pid.h"
#include "./ptools-args.h"

#define MAX_NUM_ARGS 6

/* Found in linux/include/linux/ptools.h */
struct ptools_struct {

  /* Used for pstop/prun */
  volatile long pr_prev_state;

  short pr_syscall;
  short pr_nsysarg;
  int pr_errno;
  long pr_sysarg[MAX_NUM_ARGS];
```

```
};

/*
 * Error value set by ptools library functions to pass
 * error information to the caller. Same idea as errno
 */
extern int ptools_errno;

/*
 * String containing the original argument processed
 * by ptools_arg_setup(). If an error is returned by the
 * argument processing code (ie a call to ptools_arg_next())
 * this will point to the argument that caused the error.
 */
extern char *ptools_arg_str; /* NEVER free() this pointer */
#endif
```

ptools-pid.h

```
#ifndef _PTOOLS_PID_H
#define _PTOOLS_PID_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * PID Processing functions available through libptools
 */
#include <sys/types.h>

/*
 * AllowExamine(pid_t):
 *    See if we should allow (pid) to be
 * examined. This is a security function.
 *
 * NOTE: This may not be used in any of the
 * ptools but has been left in just incase.
 *
 * Return 0 on success (allowed), -1 on error,
 * 1 if access is not allowed
 */
int AllowExamine(pid_t pid);

/*
 * ValidPid(pid_t):
 *    See if (pid) is a valid proccess
 * on the system
 *
 * Return 0 if valid, -1 otherwise
 */
int ValidPid(pid_t pid);
```

```
/*
 * OpenPID(pid_t, char *):
 *    Open the specified proc file of pid
 * Return open result, ie -1 on error, or a valid
 * file descriptor on success. errno set on error,
 * see open(2) for more info on error codes.
 */
int OpenPid(pid_t pid, char *fname);

/*
 * fOpenPid(pid_t, char *):
 *   Same as above using the fopen(3) call
 * instead.
 */
FILE *fOpenPid(pid_t pid, char *fname);

#endif
```

ptools-args.h

```
#ifndef _PTOOLS_ARGS_H
#define _PTOOLS_ARGS_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Header file for argument processing functions found
 * in libptools.
 */

#include <proc/readproc.h>

/* Error Values returned from ptools_arg_setup() */
#define NO_ERROR 0
#define INVAL_PID -1 /* Invalid PID specified on command line */
#define MEM_ERR -2 /* Unable to allocate memory via malloc() etc */
#define OPT_ERR -3 /* Unknown option found in arguments */
#define NO_PARAM -4 /* No parameter given in an option requiring one */
#define LIBPROC_ERR -5 /* Error during a libproc function call,
                        * not our fault */
#define INVAL_ARG -6 /* Invalid argument passed to ptools_arg_setup()
                        * Either argcnt or args was invalid
                        */
/*
 * int ptools_arg_setup(int, const char **,
 *                  const char *, (int *)(char)):
 *
 * Initializes internal data structures for all future calls
 *  to ptools_arg_getnext(). Parses argument from command line
 *  by using getopt() and any valid argument is sent to fptr.
 *  All other non-arguments are taken as PID values and stored internally.
 *  These PID's are sent to the libproc setup function so that libproc
```

```
 *  can handle returning the proc related information.
 *
 *  Return -1 on error and sets ptools_errno to an appropriate
 * error value. Returns 0 on success.
 */
int ptools_arg_setup(int argcnt, char * const args[],
                     char * const opts, void *fptr(char));



/*
 * proc_t *ptools_arg_getnext(void):
 *  Returns a pointer to a valid proc_t entry, or
 * NULL on error.
 *
 *  NOTE: It is the callers responsibility to free()
 * the memory region used by the proc_t * by calling
 * freeproc(proc_t *)
 *
 *      This function is non-reentrant.
 */
proc_t *ptools_arg_getnext(void);



/*
 * void ptools_arg_finish(void):
 *   Cleanup the memory used maintaining the linked-list
 * of PID's. No return value.
 */
void ptools_arg_finish(void);

#endif
```

---

ptools_maps.h

```
#ifndef _PTOOLS_MAPS_H
#define _PTOOLS_MAPS_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Contains information needed for implementation of
 * pldd.
 */

#include <config.h>

struct maps_ent {

  long int start, end;
  char perms[5]; /* NULL terminated */
  long int offset;
```

```
  int maj, min;
  long unsigned int inode;
  char path[BUFLN];

  struct maps_ent *next;
};

#endif
```

pflags.h

```
#ifndef _PFLAGS_H
#define _PFLAGS_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * THIS IS A GENERATED FILE, DO NOT EDIT!
 *
 * This file contains the process flag definitions that are
 * extracted from the linux kernel.
 */


#define PF_ALIGNWARN 0x00000001 /* Print alignment warning msgs */
#define PF_STARTING 0x00000002 /* being created */
#define PF_EXITING 0x00000004  /* getting shut down */
#define PF_FORKNOEXEC 0x00000040 /* forked but didn't exec */
#define PF_SUPERPRIV 0x00000100 /* used super-user privileges */
#define PF_DUMPCORE 0x00000200 /* dumped core */
#define PF_SIGNALED 0x00000400 /* killed by a signal */
#define PF_MEMALLOC 0x00000800 /* Allocating memory */
#define PF_FREE_PAGES 0x00002000 /* per process page freeing */
#define PF_USEDFPU 0x00100000 /* task used FPU this quantum (SMP) */


#endif
```

procfs_ctl.h

```
#ifndef _PROCFS_CTL_H
#define _PROCFS_CTL_H


/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
```

```
 * Essentially a userspace version of the new
 * include/linux/procfs_ctl.h from the kernel (added by SOberther)
 */

typedef unsigned long   ctl_t; /* should this be in sys/types.h?? */

#define PCSTOP 1L /* Tell process to stop */
#define PCRUN 5L  /* Make process runnable */


#endif
```

sigact.h

```
#ifndef _SIGACT_H
#define _SIGACT_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Sets up data structures needed for reading the proc/pid/sigact
 * file added for psig.
 */

#define SA_RESTORER 0x04000000
#define WP_SIG_MASK  2  /* Words Per Signal Mask */

/* Based off i386 include/asm-i386/signal.h
 * kernel sigaction structure...need to work
 * around this :(
 *
 * 3-20-2002: Can include asm/signal.h and #define kernel
 * in source code so we don't need to declare and update
 * this structure.
 */

struct kern_sigaction {
  __sighandler_t handler;
  unsigned long flags;
  void (*restorer) (void);
  /* unsigned long mask[WP_SIG_MASK]; */
  unsigned long long mask;
};

#endif
```

```
#ifndef _SIG_LIST_H
#define _SIG_LIST_H

/*
 * pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * sig_list.h
 *
 * Structure to provide signal number to signal name mapping.
 *
 * Based off information in libc implementation by Sun Microsystems, Inc.
 */

#include <signal.h>

/* Convert sig into a string representation. String
 * representation is stored in buf and will not be
 * longer than bufln. The value stored is also returned
 *
 * Null is returned on error.
 */
char *signum2str(int sig, char *buf, size_t bufln);

/* Convert str into the numerical equivalent. On error
 * -1 is returned, a valid signal number is returned on
 * success
 */
int str2signum(char *str);


/*
 * unsigned long long ConvertMask(char *):
 *   Convert the string containing the signal
 * mask into an unsigned long long. Return
 * values or -1 on error.
 */
unsigned long long ConvertMask(char *str);

#ifdef _USE_SIGNAL_LIST_

struct sigmap {
  const char *sigstr;
  const int signum;
};

static struct sigmap signal_list[] = {
#ifdef SIGHUP
  { "HUP",      SIGHUP },
#endif
#ifdef SIGINT
  { "INT",      SIGINT },
#endif
```

```
#ifdef SIGQUIT
  { "QUIT",     SIGQUIT },
#endif
#ifdef SIGILL
  { "ILL",      SIGILL },                                    60
#endif
#ifdef SIGTRAP
  { "TRAP",     SIGTRAP },
#endif
#ifdef SIGABRT
  { "ABRT",     SIGABRT },
#endif
#ifdef SIGIOT
  { "IOT",      SIGIOT },
#endif                                                       70
#ifdef SIGBUS
  { "BUS",      SIGBUS },
#endif
#ifdef SIGFPE
  { "FPE",      SIGFPE },
#endif
#ifdef SIGKILL
  { "KILL",     SIGKILL },
#endif
#ifdef SIGUSR1                                               80
  { "USR1",     SIGUSR1 },
#endif
#ifdef SIGSEGV
  { "SEGV",     SIGSEGV },
#endif
#ifdef SIGUSR2
  { "USR2",     SIGUSR2 },
#endif
#ifdef SIGPIPE
  { "PIPE",     SIGPIPE },                                   90
#endif
#ifdef SIGALRM
  { "ALRM",     SIGALRM },
#endif
#ifdef SIGTERM
  { "TERM",     SIGTERM },
#endif
#ifdef SIGSTKFLT
  { "STKFLT",   SIGSTKFLT },
#endif                                                       100
#ifdef SIGCHLD
  { "CHLD",     SIGCHLD },
#endif
#ifdef SIGCONT
  { "CONT",     SIGCONT },
#endif
#ifdef SIGSTOP
  { "STOP",     SIGSTOP },
#endif
#ifdef SIGTSTP                                               110
  { "TSTP",     SIGTSTP },
#endif
```

```
#ifdef SIGTTIN
  { "TTIN",     SIGTTIN },
#endif
#ifdef SIGTTOU
  { "TTOU",     SIGTTOU },
#endif
#ifdef SIGURG
  { "URG",      SIGURG },                                          120
#endif
#ifdef SIGXCPU
  { "XCPU",     SIGXCPU },
#endif
#ifdef SIGXFSZ
  { "XFSZ",     SIGXFSZ },
#endif
#ifdef SIGVTALRM
  { "VTALRM",   SIGVTALRM },
#endif                                                           130
#ifdef SIGPROF
  { "PROF",     SIGPROF },
#endif
#ifdef SIGWINCH
  { "WINCH",    SIGWINCH },
#endif
  /* Same as POLL, and SUN uses POLL
    #ifdef SIGIO
    { "IO",       SIGIO },
    #endif                                                       140
  */
#ifdef SIGPOLL
  { "POLL",     SIGPOLL },
#endif
#ifdef SIGPWR
  { "PWR",      SIGPWR },
#endif
#ifdef SIGSYS
  { "SYS",      SIGSYS },
#endif                                                           150
#ifdef SIGUNUSED
  { "UNUSED",   SIGUNUSED },
#endif
  /* These are found in the Solaris OS by SUN Microsystems, Inc.
   * and are not included in Linux
   *
    #ifdef SIGCANCEL
    { "CANCEL", SIGCANCEL },
    #endif
    #ifdef SIGCLD                                                160
    { "CLD",    SIGCLD },
    #endif
    #ifdef SIGEMT
    { "EMT",    SIGEMT },
    #endif
    #ifdef SIGFREEZE
    { "FREEZE", SIGFREEZE },
    #endif
    #ifdef SIGLOST
```

```
    { "LOST",    SIGLOST },
    #endif
    #ifdef SIGLWP
    { "LWP",     SIGLWP },
    #endif
    #ifdef SIGTHAW
    { "THAW",  SIGTHAW },
    #endif
    #ifdef SIGWAITING
    { "WAITING", SIGWAITING },
    #endif
  */
  { NULL, 0 }
};

#endif /* _USE_SIGNAL_LIST_ */

#endif
```

---

syscall_list.h

```
#ifndef _SYSCALL_LIST_H
#define _SYSCALL_LIST_H

/* WARNING: DO NOT EDIT THIS FILE BY HAND */
/* THIS FILE IS GENERATED BY ./gen_syscall_info.pl */
/* IF MAX_NUM_ARGS changes in ptools.h the script */
/* must be updated                                */


/* pTools: Process Information Utilities
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Functions to handle signal number to name and name to number conversions.
 */#include "./ptools.h"

/* Convert syscall number into a string representation
 * String will be stored in buf and will not be longer
 * than buflen. The value stored is also returned.
 *
 * NULL is returned on error.
 */
char *syscallnum2str(int syscall, char *buf, size_t buflen);


/* Convert str into the numerical equivalent. On erro
 * -1 is returned, a valid syscall number is returned
 * on success
 */
int str2syscallnum(char *str);

/* Return the number of arguments that the system call has */
int syscall_numargs(int sc_num);
```

```
#define FIRST_SYSCALL 1


struct arg_info {
  const char *arg_type;
  const char *arg_name;
};

struct syscall_map {
  const char *syscall_str;
  const int syscall_num;
  const char *syscall_ret;
  const int syscall_numargs;
  const struct arg_info syscall_args[MAX_NUM_ARGS];
};

static struct syscall_map syscall_list[] = {
  {"sys_ni_syscall", 0, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_exit", 1, "long", 1, { {"int", "error_code"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_fork", 2, "int", 1, { {"struct pt_regs", "regs"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_read", 3, "ssize_t", 3, { {"unsigned int", "fd"},
                {"char *", "buf"}, {"size_t", "count"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_write", 4, "ssize_t", 3, { {"unsigned int", "fd"},
                {"const char *", "buf"}, {"size_t", "count"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_open", 5, "long", 3, { {"const char *", "filename"},
                {"int", "flags"}, {"int", "mode"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_close", 6, "long", 1, { {"unsigned int", "fd"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_waitpid", 7, "long", 3, { {"pid_t", "pid"},
                {"unsigned int *", "stat_addr"}, {"int", "options"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_creat", 8, "long", 2, { {"const char *", "pathname"},
                {"int", "mode"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
  {"sys_link", 9, "long", 2, { {"const char *", "oldname"},
                {"const char *", "newname"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
```

```
{"sys_unlink", 10, "long", 1, { {"const char *", "pathname"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_execve", 11, "int", 1, { {"struct pt_regs", "regs"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_chdir", 12, "long", 1, { {"const char *", "filename"},                    100
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_time", 13, "long", 1, { {"int *", "tloc"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mknod", 14, "long", 3, { {"const char *", "filename"},
                {"int", "mode"}, {"dev_t", "dev"},
                {NULL, NULL}, {NULL, NULL},                                     110
                {NULL, NULL} } },
{"sys_chmod", 15, "long", 2, { {"const char *", "filename"},
                {"mode_t", "mode"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_lchown16", 16, "long", 3, { {"const char *", "filename"},
                {"old_uid_t", "user"}, {"old_gid_t", "group"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 17, "long", 0, { {NULL, NULL},                               120
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_stat", 18, "long", 2, { {"char *", "filename"},
                {"struct __old_kernel_stat *", "statbuf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_lseek", 19, "off_t", 3, { {"unsigned int", "fd"},
                {"off_t", "offset"}, {"unsigned int", "origin"},
                {NULL, NULL}, {NULL, NULL},                                     130
                {NULL, NULL} } },
{"sys_getpid", 20, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mount", 21, "long", 5, { {"char *", "dev_name"},
                {"char *", "dir_name"}, {"char *", "type"},
                {"unsigned long", "flags"}, {"void *", "data"},
                {NULL, NULL} } },
{"sys_oldumount", 22, "long", 1, { {"char *", "name"},                          140
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setuid16", 23, "long", 1, { {"old_uid_t", "uid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getuid16", 24, "long", 0, { {NULL, NULL},
```

```
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                    150
                {NULL, NULL} } },
{"sys_stime", 25, "long", 1, { {"int *", "tptr"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ptrace", 26, "int", 4, { {"long", "request"},
                {"long", "pid"}, {"long", "addr"},
                {"long", "data"}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_alarm", 27, "unsignedlong", 1, { {"unsigned int", "seconds"},            160
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_fstat", 28, "long", 2, { {"unsigned int", "fd"},
                {"struct __old_kernel_stat *", "statbuf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_pause", 29, "int", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                    170
                {NULL, NULL} } },
{"sys_utime", 30, "long", 2, { {"char *", "filename"},
                {"struct utimbuf *", "times"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 31, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 32, "long", 0, { {NULL, NULL},                              180
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_access", 33, "long", 2, { {"const char *", "filename"},
                {"int", "mode"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_nice", 34, "long", 1, { {"int", "increment"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                    190
                {NULL, NULL} } },
{"sys_ni_syscall", 35, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sync", 36, "void", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_kill", 37, "long", 2, { {"int", "pid"},                                 200
                {"int", "sig"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_rename", 38, "long", 2, { {"const char *", "oldname"},
                {"const char *", "newname"}, {NULL, NULL},
```

```
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mkdir", 39, "long", 2, { {"const char *", "pathname"},
                {"int", "mode"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                  210
                {NULL, NULL} } },
{"sys_rmdir", 40, "long", 1, { {"const char *", "pathname"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_dup", 41, "long", 1, { {"unsigned int", "fildes"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_pipe", 42, "int", 1, { {"unsigned long *", "fildes"},                  220
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_times", 43, "long", 1, { {"struct tms *", "tbuf"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 44, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                  230
                {NULL, NULL} } },
{"sys_brk", 45, "unsignedlong", 1, { {"unsigned long", "brk"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setgid16", 46, "long", 1, { {"old_gid_t", "gid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getgid16", 47, "long", 0, { {NULL, NULL},                              240
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_signal", 48, "unsignedlong", 2, { {"int", "sig"},
                {"__sighandler_t", "handler"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_geteuid16", 49, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                  250
                {NULL, NULL} } },
{"sys_getegid16", 50, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_acct", 51, "long", 1, { {"const char *", "filename"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_umount", 52, "long", 2, { {"char *", "name"},                         260
                {"int", "flags"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
```

```
                {NULL, NULL} } },
{"sys_ni_syscall", 53, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ioctl", 54, "long", 3, { {"unsigned int", "fd"},
                {"unsigned int", "cmd"}, {"unsigned long", "arg"},
                {NULL, NULL}, {NULL, NULL},                                  270
                {NULL, NULL} } },
{"sys_fcntl", 55, "long", 3, { {"unsigned int", "fd"},
                {"unsigned int", "cmd"}, {"unsigned long", "arg"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 56, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setpgid", 57, "long", 2, { {"pid_t", "pid"},                           280
                {"pid_t", "pgid"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 58, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_olduname", 59, "int", 1, { {"struct oldold_utsname *", "name"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                  290
                {NULL, NULL} } },
{"sys_umask", 60, "long", 1, { {"int", "mask"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_chroot", 61, "long", 1, { {"const char *", "filename"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ustat", 62, "long", 2, { {"dev_t", "dev"},                            300
                {"struct ustat *", "ubuf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_dup2", 63, "long", 2, { {"unsigned int", "oldfd"},
                {"unsigned int", "newfd"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getppid", 64, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                  310
                {NULL, NULL} } },
{"sys_getpgrp", 65, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setsid", 66, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
```

```
{"sys_sigaction", 67, "int", 3, { {"int", "sig"},                                    320
              {"const struct old_sigaction", "*act"}, {"struct old_sigaction", "*oact"},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_sgetmask", 68, "long", 0, { {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_ssetmask", 69, "long", 1, { {"int", "newmask"},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},                                            330
              {NULL, NULL} } },
{"sys_setreuid16", 70, "long", 2, { {"old_uid_t", "ruid"},
              {"old_uid_t", "euid"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_setregid16", 71, "long", 2, { {"old_gid_t", "rgid"},
              {"old_gid_t", "egid"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_sigsuspend", 72, "int", 3, { {"int", "history0"},                              340
              {"int", "history1"}, {"old_sigset_t", "mask"},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_sigpending", 73, "long", 1, { {"old_sigset_t", "*set"},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_sethostname", 74, "long", 2, { {"char", "*name"},
              {"int", "len"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},                                            350
              {NULL, NULL} } },
{"sys_setrlimit", 75, "long", 2, { {"unsigned int", "resource"},
              {"struct rlimit", "*rlim"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_old_getrlimit", 76, "long", 2, { {"unsigned int", "resource"},
              {"struct rlimit", "*rlim"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_getrusage", 77, "long", 2, { {"int", "who"},                                   360
              {"struct rusage", "*ru"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_gettimeofday", 78, "long", 2, { {"struct timeval", "*tv"},
              {"struct timezone", "*tz"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_settimeofday", 79, "long", 2, { {"struct timeval", "*tv"},
              {"struct timezone", "*tz"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},                                            370
              {NULL, NULL} } },
{"sys_getgroups16", 80, "long", 2, { {"int", "gidsetsize"},
              {"old_gid_t", "*grouplist"}, {NULL, NULL},
              {NULL, NULL}, {NULL, NULL},
              {NULL, NULL} } },
{"sys_setgroups16", 81, "long", 2, { {"int", "gidsetsize"},
```

```
                {"old_gid_t", "*grouplist"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"old_select", 82, "int", 1, { {"struct sel_arg_struct", "*arg"},                    380
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_symlink", 83, "long", 2, { {"const char *", "oldname"},
                {"const char *", "newname"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_lstat", 84, "long", 2, { {"char *", "filename"},
                {"struct __old_kernel_stat *", "statbuf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                          390
                {NULL, NULL} } },
{"sys_readlink", 85, "long", 3, { {"const char *", "path"},
                {"char *", "buf"}, {"int", "bufsiz"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_uselib", 86, "long", 1, { {"const char *", "library"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_swapon", 87, "long", 2, { {"const char *", "specialfile"},                    400
                {"int", "swap_flags"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_reboot", 88, "long", 4, { {"int", "magic1"},
                {"int", "magic2"}, {"unsigned int", "cmd"},
                {"void *", "arg"}, {NULL, NULL},
                {NULL, NULL} } },
{"old_readdir", 89, "int", 3, { {"unsigned int", "fd"},
                {"void *", "dirent"}, {"unsigned int", "count"},
                {NULL, NULL}, {NULL, NULL},                                          410
                {NULL, NULL} } },
{"old_mmap", 90, "int", 1, { {"struct mmap_arg_struct", "*arg"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_munmap", 91, "long", 2, { {"unsigned long", "addr"},
                {"size_t", "len"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_truncate", 92, "long", 2, { {"const char *", "path"},                         420
                {"unsigned long", "length"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ftruncate", 93, "long", 2, { {"unsigned int", "fd"},
                {"unsigned long", "length"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_fchmod", 94, "long", 2, { {"unsigned int", "fd"},
                {"mode_t", "mode"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                          430
                {NULL, NULL} } },
{"sys_fchown16", 95, "long", 3, { {"unsigned int", "fd"},
                {"old_uid_t", "user"}, {"old_gid_t", "group"},
```

```
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getpriority", 96, "long", 2, { {"int", "which"},
                {"int", "who"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setpriority", 97, "long", 3, { {"int", "which"},                                  440
                {"int", "who"}, {"int", "niceval"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 98, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_statfs", 99, "long", 2, { {"const char *", "path"},
                {"struct statfs *", "buf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                             450
                {NULL, NULL} } },
{"sys_fstatfs", 100, "long", 2, { {"unsigned int", "fd"},
                {"struct statfs *", "buf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ioperm", 101, "int", 3, { {"unsigned long", "from"},
                {"unsigned long", "num"}, {"int", "turn_on"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_socketcall", 102, "long", 2, { {"int", "call"},                                   460
                {"unsigned long", "*args"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_syslog", 103, "long", 3, { {"int", "type"},
                {"char *", "buf"}, {"int", "len"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setitimer", 104, "long", 3, { {"int", "which"},
                {"struct itimerval", "*value"}, {"struct itimerval", "*ovalue"},
                {NULL, NULL}, {NULL, NULL},                                             470
                {NULL, NULL} } },
{"sys_getitimer", 105, "long", 2, { {"int", "which"},
                {"struct itimerval", "*value"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_newstat", 106, "long", 2, { {"char *", "filename"},
                {"struct stat *", "statbuf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_newlstat", 107, "long", 2, { {"char *", "filename"},                              480
                {"struct stat *", "statbuf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_newfstat", 108, "long", 2, { {"unsigned int", "fd"},
                {"struct stat *", "statbuf"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_uname", 109, "int", 1, { {"struct old_utsname *", "name"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                             490
```

```
                {NULL, NULL} } },
{"sys_iopl", 110, "int", 1, { {"unsigned long", "unused"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_vhangup", 111, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 112, "long", 0, { {NULL, NULL},                                  500
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_vm86old", 113, "int", 1, { {"struct vm86_struct *", "v86"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_wait4", 114, "long", 4, { {"pid_t", "pid"},
                {"unsigned int *", "stat_addr"}, {"int", "options"},
                {"struct rusage *", "ru"}, {NULL, NULL},                            510
                {NULL, NULL} } },
{"sys_swapoff", 115, "long", 1, { {"const char *", "specialfile"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sysinfo", 116, "long", 1, { {"struct sysinfo", "*info"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ipc", 117, "int", 6, { {"uint", "call"},                                     520
                {"int", "first"}, {"int", "second"},
                {"int", "third"}, {"void", "*ptr"},
                {"long", "fifth"} } },
{"sys_fsync", 118, "long", 1, { {"unsigned int", "fd"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sigreturn", 119, "int", 1, { {"unsigned long", "__unused"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                         530
                {NULL, NULL} } },
{"sys_clone", 120, "int", 1, { {"struct pt_regs", "regs"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setdomainname", 121, "long", 2, { {"char", "*name"},
                {"int", "len"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_newuname", 122, "long", 1, { {"struct new_utsname *", "name"},               540
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_modify_ldt", 123, "int", 3, { {"int", "func"},
                {"void", "*ptr"}, {"unsigned long", "bytecount"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
```

```
{"sys_adjtimex", 124, "long", 1, { {"struct timex", "*txc_p"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mprotect", 125, "long", 3, { {"unsigned long", "start"},
                {"size_t", "len"}, {"unsigned long", "prot"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sigprocmask", 126, "long", 3, { {"int", "how"},
                {"old_sigset_t", "*set"}, {"old_sigset_t", "*oset"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_create_module", 127, "unsignedlong", 2, { {"const char", "*name_user"},
                {"size_t", "size"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_init_module", 128, "long", 2, { {"const char", "*name_user"},
                {"struct module", "*mod_user"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_delete_module", 129, "long", 1, { {"const char", "*name_user"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_get_kernel_syms", 130, "long", 1, { {"struct kernel_sym", "*table"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_quotactl", 131, "long", 4, { {"int", "cmd"},
                {"const char", "*special"}, {"int", "id"},
                {"caddr_t", "addr"}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getpgid", 132, "long", 1, { {"pid_t", "pid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_fchdir", 133, "long", 1, { {"unsigned int", "fd"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_bdflush", 134, "long", 2, { {"int", "func"},
                {"long", "data"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sysfs", 135, "long", 3, { {"int", "option"},
                {"unsigned long", "arg1"}, {"unsigned long", "arg2"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_personality", 136, "long", 1, { {"unsigned long", "personality"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 137, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setfsuid16", 138, "long", 1, { {"old_uid_t", "uid"},
```

```
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setfsgid16", 139, "long", 1, { {"old_gid_t", "gid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                    610
                {NULL, NULL} } },
{"sys_llseek", 140, "long", 5, { {"unsigned int", "fd"},
                {"unsigned long", "offset_high"}, {"unsigned long", "offset_low"},
                {"loff_t *", "result"}, {"unsigned int", "origin"},
                {NULL, NULL} } },
{"sys_getdents", 141, "long", 3, { {"unsigned int", "fd"},
                {"void *", "dirent"}, {"unsigned int", "count"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_select", 142, "long", 5, { {"int", "n"},                                 620
                {"fd_set", "*inp"}, {"fd_set", "*outp"},
                {"fd_set", "*exp"}, {"struct timeval", "*tvp"},
                {NULL, NULL} } },
{"sys_flock", 143, "long", 2, { {"unsigned int", "fd"},
                {"unsigned int", "cmd"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_msync", 144, "long", 3, { {"unsigned long", "start"},
                {"size_t", "len"}, {"int", "flags"},
                {NULL, NULL}, {NULL, NULL},                                    630
                {NULL, NULL} } },
{"sys_readv", 145, "ssize_t", 3, { {"unsigned long", "fd"},
                {"const struct iovec *", "vector"}, {"unsigned long", "count"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_writev", 146, "ssize_t", 3, { {"unsigned long", "fd"},
                {"const struct iovec *", "vector"}, {"unsigned long", "count"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getsid", 147, "long", 1, { {"pid_t", "pid"},                             640
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_fdatasync", 148, "long", 1, { {"unsigned int", "fd"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sysctl", 149, NULL, -1, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                    650
                {NULL, NULL} } },
{"sys_mlock", 150, "long", 2, { {"unsigned long", "start"},
                {"size_t", "len"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_munlock", 151, "long", 2, { {"unsigned long", "start"},
                {"size_t", "len"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mlockall", 152, "long", 1, { {"int", "flags"},                           660
                {NULL, NULL}, {NULL, NULL},
```

```
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_munlockall", 153, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sched_setparam", 154, "long", 2, { {"pid_t", "pid"},
                {"struct sched_param", "*param"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                    670
                {NULL, NULL} } },
{"sys_sched_getparam", 155, "long", 2, { {"pid_t", "pid"},
                {"struct sched_param", "*param"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sched_setscheduler", 156, "long", 3, { {"pid_t", "pid"},
                {"int", "policy"}, {"struct sched_param", "*param"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sched_getscheduler", 157, "long", 1, { {"pid_t", "pid"},                  680
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sched_yield", 158, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sched_get_priority_max", 159, "long", 1, { {"int", "policy"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                    690
                {NULL, NULL} } },
{"sys_sched_get_priority_min", 160, "long", 1, { {"int", "policy"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sched_rr_get_interval", 161, "long", 2, { {"pid_t", "pid"},
                {"struct timespec", "*interval"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_nanosleep", 162, "long", 2, { {"struct timespec", "*rqtp"},              700
                {"struct timespec", "*rmtp"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mremap", 163, "unsignedlong", 5, { {"unsigned long", "addr"},
                {"unsigned long", "old_len"}, {"unsigned long", "new_len"},
                {"unsigned long", "flags"}, {"unsigned long", "new_addr"},
                {NULL, NULL} } },
{"sys_setresuid16", 164, "long", 3, { {"old_uid_t", "ruid"},
                {"old_uid_t", "euid"}, {"old_uid_t", "suid"},
                {NULL, NULL}, {NULL, NULL},                                    710
                {NULL, NULL} } },
{"sys_getresuid16", 165, "long", 3, { {"old_uid_t", "*ruid"},
                {"old_uid_t", "*euid"}, {"old_uid_t", "*suid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_vm86", 166, "int", 2, { {"unsigned long", "subfunction"},
                {"struct vm86plus_struct *", "v86"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
```

```
                {NULL, NULL} } },
{"sys_query_module", 167, "long", 5, { {"const char", "*name_user"},
                {"int", "which"}, {"char", "*buf"},
                {"size_t", "bufsize"}, {"size_t", "*ret"},
                {NULL, NULL} } },
{"sys_poll", 168, "long", 3, { {"struct pollfd *", "ufds"},
                {"unsigned int", "nfds"}, {"long", "timeout"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_nfsservctl", 169, "int", 3, { {"int", "cmd"},
                {"void", "*argp"}, {"void", "*resp"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setresgid16", 170, "long", 3, { {"old_gid_t", "rgid"},
                {"old_gid_t", "egid"}, {"old_gid_t", "sgid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getresgid16", 171, "long", 3, { {"old_gid_t", "*rgid"},
                {"old_gid_t", "*egid"}, {"old_gid_t", "*sgid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_prctl", 172, "long", 5, { {"int", "option"},
                {"unsigned long", "arg2"}, {"unsigned long", "arg3"},
                {"unsigned long", "arg4"}, {"unsigned long", "arg5"},
                {NULL, NULL} } },
{"sys_rt_sigreturn", 173, "int", 1, { {"unsigned long", "__unused"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_rt_sigaction", 174, "long", 4, { {"int", "sig"},
                {"const struct sigaction", "*act"}, {"struct sigaction", "*oact"},
                {"size_t", "sigsetsize"}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_rt_sigprocmask", 175, "long", 4, { {"int", "how"},
                {"sigset_t", "*set"}, {"sigset_t", "*oset"},
                {"size_t", "sigsetsize"}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_rt_sigpending", 176, "long", 2, { {"sigset_t", "*set"},
                {"size_t", "sigsetsize"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_rt_sigtimedwait", 177, "long", 4, { {"const sigset_t", "*uthese"},
                {"siginfo_t", "*uinfo"}, {"const struct timespec", "*uts"},
                {"size_t", "sigsetsize"}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_rt_sigqueueinfo", 178, "long", 3, { {"int", "pid"},
                {"int", "sig"}, {"siginfo_t", "*uinfo"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_rt_sigsuspend", 179, "int", 2, { {"sigset_t", "*unewset"},
                {"size_t", "sigsetsize"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_pread", 180, "ssize_t", 4, { {"unsigned int", "fd"},
                {"char *", "buf"}, {"size_t", "count"},
                {"loff_t", "pos"}, {NULL, NULL},
                {NULL, NULL} } },
```

```
{"sys_pwrite", 181, "ssize_t", 4, { {"unsigned int", "fd"},
                {"const char *", "buf"}, {"size_t", "count"},
                {"loff_t", "pos"}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_chown16", 182, "long", 3, { {"const char *", "filename"},
                {"old_uid_t", "user"}, {"old_gid_t", "group"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getcwd", 183, "long", 2, { {"char", "*buf"},
                {"unsigned long", "size"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_capget", 184, "long", 2, { {"cap_user_header_t", "header"},
                {"cap_user_data_t", "dataptr"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_capset", 185, "long", 2, { {"cap_user_header_t", "header"},
                {"const cap_user_data_t", "data"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sigaltstack", 186, "int", 2, { {"const stack_t", "*uss"},
                {"stack_t", "*uoss"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_sendfile", 187, "ssize_t", 4, { {"int", "out_fd"},
                {"int", "in_fd"}, {"off_t", "*offset"},
                {"size_t", "count"}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 188, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 189, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_vfork", 190, "int", 1, { {"struct pt_regs", "regs"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getrlimit", 191, "long", 2, { {"unsigned int", "resource"},
                {"struct rlimit", "*rlim"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mmap2", 192, "long", 6, { {"unsigned long", "addr"},
                {"unsigned long", "len"}, {"unsigned long", "prot"},
                {"unsigned long", "flags"}, {"unsigned long", "fd"},
                {"unsigned long", "pgoff"} } },
{"sys_truncate64", 193, "long", 2, { {"const char *", "path"},
                {"loff_t", "length"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ftruncate64", 194, "long", 2, { {"unsigned int", "fd"},
                {"loff_t", "length"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_stat64", 195, "long", 3, { {"char *", "filename"},
```

```
                {"struct stat64 *", "statbuf"}, {"long", "flags"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_lstat64", 196, "long", 3, { {"char *", "filename"},
                {"struct stat64 *", "statbuf"}, {"long", "flags"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_fstat64", 197, "long", 3, { {"unsigned long", "fd"},                  840
                {"struct stat64 *", "statbuf"}, {"long", "flags"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_lchown", 198, "long", 3, { {"const char *", "filename"},
                {"uid_t", "user"}, {"gid_t", "group"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getuid", 199, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                  850
                {NULL, NULL} } },
{"sys_getgid", 200, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_geteuid", 201, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getegid", 202, "long", 0, { {NULL, NULL},                              860
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setreuid", 203, "long", 2, { {"uid_t", "ruid"},
                {"uid_t", "euid"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setregid", 204, "long", 2, { {"gid_t", "rgid"},
                {"gid_t", "egid"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                  870
                {NULL, NULL} } },
{"sys_getgroups", 205, "long", 2, { {"int", "gidsetsize"},
                {"gid_t", "*grouplist"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setgroups", 206, "long", 2, { {"int", "gidsetsize"},
                {"gid_t", "*grouplist"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_fchown", 207, "long", 3, { {"unsigned int", "fd"},                     880
                {"uid_t", "user"}, {"gid_t", "group"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setresuid", 208, "long", 3, { {"uid_t", "ruid"},
                {"uid_t", "euid"}, {"uid_t", "suid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getresuid", 209, "long", 3, { {"uid_t", "*ruid"},
                {"uid_t", "*euid"}, {"uid_t", "*suid"},
```

```
                {NULL, NULL}, {NULL, NULL},                                     890
                {NULL, NULL} } },
{"sys_setresgid", 210, "long", 3, { {"gid_t", "rgid"},
                {"gid_t", "egid"}, {"gid_t", "sgid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_getresgid", 211, "long", 3, { {"gid_t", "*rgid"},
                {"gid_t", "*egid"}, {"gid_t", "*sgid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_chown", 212, "long", 3, { {"const char *", "filename"},                   900
                {"uid_t", "user"}, {"gid_t", "group"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setuid", 213, "long", 1, { {"uid_t", "uid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setgid", 214, "long", 1, { {"gid_t", "gid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},                                     910
                {NULL, NULL} } },
{"sys_setfsuid", 215, "long", 1, { {"uid_t", "uid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_setfsgid", 216, "long", 1, { {"gid_t", "gid"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_pivot_root", 217, "long", 2, { {"const char", "*new_root"},               920
                {"const char", "*put_old"}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_mincore", 218, "long", 3, { {"unsigned long", "start"},
                {"size_t", "len"}, {"unsigned char *", "vec"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_madvise", 219, "long", 3, { {"unsigned long", "start"},
                {"size_t", "len"}, {"int", "behavior"},
                {NULL, NULL}, {NULL, NULL},                                     930
                {NULL, NULL} } },
{"sys_getdents64", 220, "long", 3, { {"unsigned int", "fd"},
                {"void *", "dirent"}, {"unsigned int", "count"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_fcntl64", 221, "long", 3, { {"unsigned int", "fd"},
                {"unsigned int", "cmd"}, {"unsigned long", "arg"},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 222, "long", 0, { {NULL, NULL},                              940
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } },
{"sys_ni_syscall", 223, "long", 0, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
```

```
                {NULL, NULL} } },
  {NULL, -1, NULL, -1, { {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL}, {NULL, NULL},
                {NULL, NULL} } }
};

#endif
```

SMOLabs_string.h

```
#ifndef _SMOLABS_STRING_H
#define _SMOLABS_STRING_H

/*
 * String related functions not implemented in the standard
 * C string library.
 *
 * Stephen Oberther
 * oberther@cs.fsu.edu
 */

#define chomp(x) if(x[strlen(x)-1] == '\n') x[strlen(x)-1] = '\0'

void chomp_space(char *str);
#endif
```

SMOLabs_color.h

```
#ifndef _SMOLABS_COLOR_H
#define _SMOLABS_COLOR_H

/*
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Color definitions useful for providing color
 * output
 */

#define S_NM    "\033[0m"
#define S_BD    "\033[1m"
#define S_GYBG "\033[5m"
#define S_WTBG "\033[7m"
#define S_WT    "\033[8m"
#define S_RD    "\033[31m"
#define S_GR    "\033[32m"
#define S_YW    "\033[33m"
#define S_BL    "\033[34m"
#define S_PP    "\033[35m"
#define S_CN    "\033[36m"
#define S_GY    "\033[37m"
#define S_RDBG "\033[41m"
```

```
#define S_GRBG "\033[42m"
#define S_YWBG "\033[43m"
#define S_BLBG "\033[44m"
#define S_PPBG "\033[45m"
#define S_CNBG "\033[46m"
#define S_BGBG "\033[47m"

#endif
```

H.3 Library Files

libptools Makefile

```
#
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#
# make rules for building the ptools
# library and object files
#

include ../Config

COPTS = -D_FORCE_POSIX_

CFLAGS += $(COPTS)
AR = ar

FILES = pid.o args.o sig_list.o syscall_list.o

.c.o:
	$(CC) $(CFLAGS) -c $<

all: libptools.a

libptools.a: Makefile $(FILES)
	$(AR) rcs libptools.a $(FILES)

sig_list.o: sig_list.c ../include/sig_list.h
	$(CC) $(CFLAGS) -c sig_list.c

syscall_list.o: syscall_list.c ../include/syscall_list.h
	$(CC) $(CFLAGS) -c syscall_list.c

# Clean up the source directory
clean: clean-tex
	rm -f *.o libptools.a *~

clean-tex:
	rm -f *.aux *.dvi *.log
```

```
/*
 * args.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Argument processing functions used in pTools
 * implementation. The three main interface functions
 * are:
 *
 * ptools_arg_setup: Parses arguments from command line and
 *   passes them to the function given as fptr. Handles creation
 *   of internal lists of PID's and setting up libproc data structures.
 *
 * ptools_arg_getnext: Returns a proc_t structure of the next PID to
 *   be processed.
 *
 * ptools_arg_finish: Cleans up internal data structures and handles
 *   cleanup of libproc data structures.
 *
 * Compile:
 *   gcc -c args.c
 *
 * Options:
 *   If _FORCE_POSIX_ is defined arguments are processed in a POSIX way by
 *      getopt()
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <ctype.h>
#include <errno.h>
#include <sys/types.h>
#include <proc/readproc.h>

#include <ptools-args.h>

#define TRUE 1
#define FALSE 0

#define BUFLN 1024

/* Used for Error reporting */
int ptools_errno;
char *ptools_arg_str;

/* List of PIDs passed to openproc() */
static pid_t *PIDS;

typedef struct pid_list_s {
  pid_t dataN; /* This is an int in LINUX..so neg values are ok */
  int visited;
  long int dataS_len;
```

```
  char *dataS;
  struct pid_list_s *next;
} pidlist;

/* single-linked list of PID's. If (>dataN < 0) this argument is not
 * valid and an error should be reported.
 */
static struct ArgData {
  PROCTAB *PT;
  pidlist *head;
} proc_ents;

/* Linked-list of buffered proc_t structures */
static struct proc_list {
  proc_t *data;
  struct proc_list *next;
} *plist;


/*
 * int AttachPID(pidlist *):
 *   Attach p onto the list of arguments maintained
 * in proc_ents. Return 0 on success, negative value
 * on error.
 */
int AttachPID(pidlist *p) {
  pidlist *cur;

  if (!proc_ents.head) {
    proc_ents.head = p;
    return 0;
  }

  cur = proc_ents.head;
  while (cur->next) { cur = cur->next; }

  cur->next = p;

  return 0;
}

/*
 * int AddPID(pid_t *, const char *):
 *   Validate the argument passed and attach it
 * onto the list of pids found in the proc_ents
 * structure. The converted value is also stored in
 * dst.  Return 0 on success. Set ptools_errno and
 * and return -1 on error.
 */
int AddPID(pid_t *dst, const char *pid) {
  pidlist *ptr;
  long int tmp = 0;
  int cnt;

  ptools_errno = NO_ERROR; /* Reset ptools_errno */

  /* Allocate space for the list object and to maintain the argument
```

```
   * passed to this function
   */
  ptr = (pidlist *) malloc(sizeof(pidlist));
  if (!ptr) { ptools_errno = MEM_ERR; return -1; }
  memset(ptr, 0, sizeof(pidlist));

  ptr->dataS_len = strlen(pid) + 1;
                                                                        120
  ptr->dataS = (char *)malloc(sizeof(char) * ptr->dataS_len);
  if (!(ptr->dataS)) { ptools_errno = MEM_ERR; return -1; }
  memset(ptr->dataS, 0, (ptr->dataS_len * sizeof(char)));

  /* Is PID Valid */
  for (cnt = 0; cnt < strlen(pid); cnt++) {
    if (!isdigit(pid[cnt])) {
      tmp = INVAL_PID; break;
    }
  }                                                                     130

  if (!tmp) {
    errno = 0;
    tmp = strtol(pid, (char **)NULL, 10);
    if (errno == ERANGE) {
      errno = 0;
      fprintf(stderr, "Possible Overflow/Underflow error, using %ld for %s\n",
              tmp, pid);
    }
  }                                                                     140

  /* Fill in list member structure */
  ptr->dataN = (pid_t)tmp;
  ptr->visited = FALSE;
  strncpy(ptr->dataS, pid, ptr->dataS_len);
  ptr->next = NULL;

  (*dst) = ptr->dataN;
  /* Attach to list and return */
  return AttachPID(ptr);                                                150
}


/*
 * void Dump_Arg_List(void):
 *    Dump the data portion of proc_ents.head
 * to stderr. No return value.
 */
void Dump_Arg_List(void) {
                                                                        160
  pidlist *cur;

  cur = proc_ents.head;

  while (cur) {
    fprintf(stderr, "PID: %s(%d)\n", cur->dataS, cur->dataN);
    cur = cur->next;
  }
}
```

```
                                                                    170
/*
 * void ptools_arg_finish(void):
 *   Cleanup the memory used maintaining the linked-list
 * of PID's. No return value.
 */
void ptools_arg_finish(void) {

  pidlist *cur, *tmp;

  if (PIDS)                                                         180
    free(PIDS); /* make sure we free the list of pids we
                 * passed to openproc()
                 */
  closeproc(proc_ents.PT);
  cur = proc_ents.head;

  while (cur) {
    tmp = cur;
    cur = cur->next;
                                                                    190
    tmp->next = NULL;
    free(tmp->dataS);
    free(tmp);
  }
}


/*
 * int ptools_arg_setup(int, const char **,
 *                  const char *, (int *)(char)):                  200
 *
 * Initializes internal data structures for all future calls
 *  to ptools_arg_getnext(). Parses argument from command line
 *  by using getopt() and any valid argument is sent to fptr.
 *  All other non-arguments are taken as PID values and stored internally.
 *  These PID's are sent to the libproc setup function so that libproc
 *  can handle returning the proc related information.
 *
 *  Return -1 on error and sets ptools_errno to an appropriate
 * error value. Returns 0 on success.                              210
 */
int ptools_arg_setup(int argcnt, char * const args[],
                     char * const opts, void *fptr(char)) {
  int cnt, ret, num_pids, pidcntr;
  extern char *optarg;
  extern int optind, opterr, optopt;
  char *options, c;


  /* Dump args */                                                   220
  /* fprintf(stderr, "DEBUG:\tname = %s\n\targcnt = %d\n\targs[0] = %s\n",
   * name, argcnt, args[0]);
   */

  /* Initialize global data values */
  PIDS = NULL;
```

```
  ptools_errno = NO_ERROR; /* Reset ptools_errno */
  ptools_arg_str = NULL;
  plist = NULL;
  proc_ents.PT = NULL;                                                    230
  proc_ents.head = NULL;

  optind = 1; /* Just incase we don't do a call to getopt */

  if ( (argcnt < 0) || (!args) ) {
    ptools_errno = INVAL_ARG;
    return -1;
  }

  /* Look at options passed */                                            240
  if (opts && fptr) {
#ifdef _FORCE_POSIX_
    if (opts[0] != '+') {
      options = (char *)malloc(sizeof(char) * (strlen(opts) + 2));
      if (!options) { ptools_errno = MEM_ERR; return -1; }

      options[0] = '+';
      strncpy(options+1, opts, strlen(opts));
      /*
       * fprintf(stderr, "DEBUG: arg_setup(), options: %s\n", options);     250
       */
    }
#else
    options = opts;
#endif

    while(1) {
      c = getopt(argcnt, args, options);

      if (c == -1) break;                                                 260

      if (c == '?') { ptools_errno = OPT_ERR; return -1; }
      if (c == ':') { ptools_errno = NO_PARAM; return -1; }

      fptr(c);
    }
  } /* if (opts) */

  /*
    fprintf(stderr, "DEBUG: Allocating room for %d pid_t's\n",            270
    argcnt-optind+1);
  */

  /* Allocate space for array of pid_t that is passed to openproc() */
  num_pids = argcnt-optind+1;
  PIDS = (pid_t *)malloc(sizeof(pid_t) * num_pids);
  if (!PIDS) { ptools_errno = MEM_ERR; return -1; }
  memset(PIDS, 0, (num_pids * sizeof(pid_t)));

                                                                          280
  /* We keep our own counter for the pids array */
  pidcntr = 0;
```

```
  /* Build a list of all non-processed args */
  for (cnt = optind; cnt < argcnt; cnt++) {
    /*
      fprintf(stderr, "DEBUG: Processing %s, pidcntr: %d\n", args[cnt],
      pidcntr);
     */
    ret = AddPID(PIDS+pidcntr, args[cnt]);
    /* Advance if we assigned a valid PID during the last AddPID() *
     * otherwise we reuse the same array slot (This prevents us *
     * from passing illegal values to openproc() below          *
     */
    if (*(PIDS+pidcntr) > 0) pidcntr++;
    else *(PIDS+pidcntr) = 0;

    if (ret < 0) {
      /* AddPID() Error */
      fprintf(stderr, "AddPID() error: %d returned\n", ret);
      free(PIDS);
      return ret;
    }
  }

  /*
    fprintf(stderr, "\nDumping Internal Argument List\n");
    Dump_Arg_List();
    fprintf(stderr, "\nList of PID's being sent to openproc() is:\n");
    for (cnt = 0; cnt < num_pids; cnt++)
    fprintf(stderr, "%d\n", PIDS[cnt]);
  */

  /* Call openproc() */
  proc_ents.PT = openproc(PROC_FILLBUG|PROC_PID, PIDS);

  if (!proc_ents.PT) { ptools_errno = LIBPROC_ERR; free(PIDS); return -1; }

  return 0;
}


/*
 * int BufferProcT(proc_t *):
 *   Handles buffering of proc_t items if libproc
 * returns a PID we are not expecting.
 *
 * Returns 0 on success, on error ptools_errno is set and -1
 * is returned.
 */
int BufferProcT(proc_t *p) {
  struct proc_list *cur, *tmp;

  ptools_errno = NO_ERROR; /* Reset ptools_errno */
  /* Build struct to store p in */
  tmp = (struct proc_list *)malloc(sizeof(struct proc_list));
  if (!tmp) { ptools_errno = MEM_ERR; return -1; }
  memset(tmp, 0, sizeof(struct proc_list));
```

```
  tmp->data = p;
  tmp->next = NULL;

  /* start list if plist is NULL */
  if (!plist) {
    plist = tmp;
    return 0;
  }

  cur = plist;                                                    350

  while (cur->next) cur = cur->next;

  cur->next = tmp;
  return 0;
}


/*
 * int Visited(pid_t):                                            360
 *   See if p is in our list of PID's to process
 * and return the visited status when found. If p
 * is not in our list of PID's return -1.
 */
int Visited(pid_t p) {
  pidlist *cur;

  cur = proc_ents.head;

  while (cur) {                                                   370
    if (cur->dataN == p)
      return cur->visited;

    cur = cur->next;
  }

  return -1;
}

                                                                  380
/*
 * proc_t *LookInBuffer(pid_t):
 *    Looks for a PID in the internal buffer, returning
 * a pointer to the proc_t structure if it is. NULL is
 * returned if the PID is not in our buffer.
 */
proc_t *LookInBuffer(pid_t p) {

  struct proc_list *cur, *prev;
  proc_t *ret;                                                    390


  prev = NULL;
  cur = plist;

  while (cur) {
    if (cur->data->pid == p) {
```

```
      /* found a match */
      ret = cur->data;
      if (cur == plist) plist = cur->next;
      else prev->next = cur->next;
      free(cur);
      return ret;
    }

    /* move down the list */
    prev = cur;
    cur = cur->next;
  }

  return NULL;

}


/*
 * proc_t *ptools_arg_getnext(void):
 *  Returns a pointer to a valid proc_t entry, or
 * NULL on error.
 *
 *  NOTE: It is the callers responsibility to free()
 * the memory region used by the proc_t * by calling
 * freeproc(proc_t *)
 *
 *      This function is non-reentrant.
 */
proc_t *ptools_arg_getnext(void) {

  pidlist *cur = NULL;
  proc_t *ret = NULL;

  ptools_errno = NO_ERROR; /* Reset ptools_errno */
  ptools_arg_str = NULL;

  cur = proc_ents.head;

  /* Search for a PID that hasn't been processed yet */
  while (cur) {
    if (!cur->visited) {
      cur->visited = TRUE;
      ptools_arg_str = cur->dataS;

      /* Return NULL if this argument appears next and was invalid */
      if (cur->dataN == INVAL_PID) {
        ptools_errno = INVAL_PID;
        return NULL;
      }

      if (plist) { /* Search the buffer first */
        ret = LookInBuffer(cur->dataN);
        if (ret) return ret;
      }
```

```
    GetAnotherEntry:
      /* Get next proc_t from libproc */
      ret = readproc(proc_ents.PT, NULL);

      if (!ret) { /* Reached the end of the PID list */
        /*                                                                  460
          ptools_errno = LIBPROC_ERR;
         */
        ptools_errno = INVAL_PID;
        return NULL;
      }

      /* If not what was expected buffer it if we havn't processed
       * it already
       */
      if (cur->dataN != ret->pid) {                                        470
        int tmp;
        tmp = Visited(ret->pid);
        if (tmp == TRUE) goto GetAnotherEntry;
        else if (tmp == -1) return ret; /* We don't know about this one
                                         * but we return it anyway.
                                         * This SHOULD never happen!!!
                                         */
        else
          BufferProcT(ret); goto GetAnotherEntry;
      }                                                                    480
      else
        return ret;
    } /* if (!cur->visited) */
    cur = cur->next;
  }

  return NULL;
}
```

pid.c

```
/*
 * pid.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Implementation of functions that process PID related
 * requests in /proc.
 *
 * Compile:
 * gcc -c pid.c                                                          10
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
```

```
#define BUFLEN 512
#define ROOT 0  /* Root uid */


/*
 * ValidPid(pid_t):
 *    See if (pid) is a valid proccess
 * on the system
 *
 * Return 0 if valid, -1 otherwise
 */
int ValidPid(pid_t pid) {
  struct stat stats;
  char buf[BUFLEN];

  snprintf(buf, BUFLEN, "/proc/%d", pid);
  if (stat(buf, &stats) < 0) return -1;

  return 0;

}


/*
 * AllowExamine(pid_t):
 *    See if we should allow (pid) to be
 * examined. This is a security function.
 *
 * NOTE: This may not be used in any of the
 * ptools but has been left in just incase.
 *
 * Return 0 on success (allowed), -1 on error,
 * 1 if access is not allowed
 */
int AllowExamine(pid_t pid) {

  struct stat stats;
  char buf[BUFLEN];
  uid_t myuid;

  myuid = geteuid();

  if (myuid == ROOT) return 0; /* Root can examine anything */

  snprintf(buf, BUFLEN, "/proc/%d", pid);
  if (stat(buf, &stats) < 0) return -1;

  /* DEBUG
     printf("Owner UID: %d\n", stats.st_uid);
  */

  if (myuid == stats.st_uid) return 0;
  return 1;
}
```

```
/*
 * OpenPid(pid_t, char *):
 *   Open /proc/pid/fname and return the file descriptor, the caller must
 * close the fd when finished with it.
 *
 * Return a valid (ie > 0) file descriptor on success or -1 on error. errno
 * will be set by open(2) if an error occurs.
 */
int OpenPid(pid_t pid, char *fname) {

  int fd;
  char buf[BUFLEN];

  snprintf(buf, BUFLEN, "/proc/%d/%s", pid, fname);

  fd = open(buf, O_RDONLY|O_EXCL);

  /* Try open again but not exclusive (Follow from truss of Sun pcred
   * routine)
   */
  if (fd < 0) {
    fd = open(buf, O_RDONLY);
  }

  return fd;

}



/*
 * fOpenPid(pid_t, char *):
 *   Open /proc/pid/fname and return the FILE pointer. The caller must close
 * the FILE object with fclose(2) when finished.
 *
 * Return a valid (ie != NULL) FILE pointer on success or NULL on error. errno
 * will be set by fopen(3) if an error occurs.
 */
FILE *fOpenPid(pid_t pid, char *fname) {

  FILE *fd;
  char buf[BUFLEN];

  snprintf(buf, BUFLEN, "/proc/%d/%s", pid, fname);

  fd = fopen(buf, "r");

  return fd;

}
```

```
/*
 * sig_list.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Signal List helper functions
 *
 * Compile:
 *   gcc -c sig_list.c
 */

#define _USE_SIGNAL_LIST_

#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <sig_list.h>


char *signum2str(int sig, char *buf, size_t bufln) {
  int cnt, len;

  if ((bufln <= 0) || (!buf)) return NULL;
  for (cnt = 0;; cnt++) {
    if (signal_list[cnt].sigstr == NULL) break;
    if (signal_list[cnt].signum == sig) {
      len = snprintf(buf, bufln, "SIG%s", signal_list[cnt].sigstr);
/* Removed 2-20-2002 by SMO
      if (len >= bufln) buf[bufln-1] = '\0';
*/
      return buf;
    }
  }
  return NULL;
}



int str2signum(char *str) {
  int cnt;

  if (!str) return -1;

  for (cnt = 0;; cnt++) {
    if (signal_list[cnt].sigstr == NULL) break;
    if ( (strlen(signal_list[cnt].sigstr) == strlen(str)) &&
         (strncmp(str, signal_list[cnt].sigstr, strlen(str)) == 0) ) {
      return signal_list[cnt].signum;
    }
  }

  return -1;
}
```

```
/*
 * unsigned long long ConvertMask(char *):
 *   Convert the string containing the signal
 * mask into an unsigned long long. Return
 * values or -1 on error.                                    60
 */
unsigned long long ConvertMask(char *str) {
  int cnt, shift;
  char c;
  unsigned long long tmp, ret = 0;

  for(cnt = 0; cnt < strlen(str); cnt++) {
    c = tolower(str[cnt]);
    if (isdigit(c))
      tmp = c - '0';                                         70

    else if (isalpha(c)) {
      if ( (c < 'a') || (c > 'f') )
        return -1;
      tmp = c - 'a' + 10;
    }
    shift = 60 - (4 * cnt);

    ret |= tmp << shift;
  }                                                          80

  return ret;
}
```

---

syscall_list.c

```
/*
 * syscall_list.c
 *
 * Stephen Oberther
 * pTools: Process Information Utilities
 * oberther@cs.fsu.edu
 *
 * System call related function implementations. These functions
 * map syscall number to names and names to number through
 * the syscall_list data structure.                           10
 *
 * Compile:
 *   gcc -c syscall_list.c
 */

#include <string.h>
#include <stdio.h>
#include <syscall_list.h>

/* Convert syscall number into a string representation.       20
 * String will be stored in buf and will not be longer
 * than bufln. The value stored is also returned.
```

```
 *
 * NULL is returned on error.
 */
char *syscallnum2str(int syscall, char *buf, size_t bufln) {
  int cnt;

  if ((bufln <= 0) || (!buf)) return NULL;
  for (cnt = 0;; cnt++) {                                                    30
    if (syscall_list[cnt].syscall_str == NULL) break;
    if (syscall_list[cnt].syscall_num == syscall) {
      snprintf(buf, bufln, "%s", syscall_list[cnt].syscall_str);
      return buf;
    }
  }
  return NULL;
}

                                                                             40
/* Convert str into the numerical equivalent. On erro
 * -1 is returned, a valid syscall number is returned
 * on success
 */
int str2syscallnum(char *str) {
  int cnt;

  if (!str) return -1;

  for (cnt = 0;; cnt++) {                                                    50
    if (syscall_list[cnt].syscall_str == NULL) break;
    if ( (strlen(syscall_list[cnt].syscall_str) == strlen(str)) &&
         (strncmp(str, syscall_list[cnt].syscall_str, strlen(str)) == 0) ) {
      return syscall_list[cnt].syscall_num;
    }
  }
  return -1;
}

                                                                             60
/* Return the number of arguments that the system call has */
int syscall_numargs(int sc_num) {

  if (sc_num < 0) return -1;

  return (syscall_list[sc_num].syscall_numargs);
}
```

libsl Makefile

```
#
# Stephen Oberther
# oberther@cs.fsu.edu
#
# make rules for libsl_string
#

CC=gcc
CFLAGS=-Wall -g

INC=../include

AR=ar

all: libsl_string

libsl_string: SMOLabs_string.o
	$(AR) rcs libsl_string.a SMOLabs_string.o

SMOLabs_string.o: SMOLabs_string.c $(INC)/SMOLabs_string.h
	$(CC) $(CFLAGS) -c SMOLabs_string.c

clean:
	rm -f *.o *~ libsl_string.a
```

SMOLabs_string.c

```
/*
 * SMOLabs_string.c
 * Stephen Oberther
 * oberther@cs.fsu.edu
 *
 * Implementation of string related functions
 * that are prototyped in SMOLabs_string.h
 */

#include <string.h>

void chomp_space(char *str) {
  while (str[strlen(str)-1] == ' ') str[strlen(str)-1] = '\0';
}
```

H.4 Other Tools

gen_pflags_h.sh

```
#!/bin/bash

#
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#
# Generates pflags.h file for use with the pflags
# utility.  Scans the linux kernel and extracts the
# process flags from include/linux/sched.h.
#

RM=/bin/rm
GREP=/bin/grep
HFILE=/usr/include/linux/sched.h
OFILE=../../include/pflags.h


$RM -f $OFILE

echo "#ifndef _PFLAGS_H" >> $OFILE
echo "#define _PFLAGS_H" >> $OFILE
echo >> $OFILE

echo "/*" >> $OFILE
echo " * pTools: Process Information Utilities" >> $OFILE
echo " * Stephen Oberther" >> $OFILE
echo " * oberther@cs.fsu.edu" >> $OFILE
echo " *" >> $OFILE
echo " * THIS IS A GENERATED FILE, DO NOT EDIT!" >> $OFILE
echo " *" >> $OFILE
echo " * This file contains the process flag definitions that are" >> $OFILE
echo " * extracted from the linux kernel." >> $OFILE
echo " */" >> $OFILE

echo >> $OFILE
echo >> $OFILE

$GREP '#define PF_.*' $HFILE >> $OFILE

echo >> $OFILE
echo >> $OFILE
echo "#endif" >> $OFILE
echo >> $OFILE
```

gen_syscall_info.pl

```
#!/usr/bin/perl

#
# pTools: Process Information Utilities
# Stephen Oberther
# oberther@cs.fsu.edu
#
# This script generates syscall_list.h for use with the pTools.
# Information relating to system call function arguments, syscall
# numbers, and return values are extracted from the kernel and added
# to a data structure usable for retrieving information about the
# system calls.
#

use Getopt::Std;

# Syscall Function Info Record Layout
#{
#   RETTYPE => Type Returned by system call
#      NAME => Function name without the leading sys_
#       NUM => System Call number
#   NUMARGS => The number of arguments to the system call
#      ARGS => Array of argument information
#          {
#            TYPE => Argument Type
#            NAME => Argument Name
#          }
#}
*ARG_TYPE = \0;
*ARG_NAME = \1;
$MAX_NUM_ARGS = 6;


# Programs
$FIND = "/usr/bin/find";
$GREP = "/bin/grep";

# Directory Information
$LNX_SRC = "/home/oberther/ptools/linux-2.4.7";
$ARCH = "i386";
$ENTRY_S = "$LNX_SRC/arch/$ARCH/kernel/entry.S";
$SYSCALL_FILE = "./syscall_func";
$HEADER_FILE = "../../include/syscall_list.h";


getopt('');

if ( defined $opt_h ) {
  PrintUsage();
  exit 0;
}


print "Starting...\n";
gen_syscall_array();
```

```
get_syscall_info();
create_header_file();
if ( defined $opt_V ) {
  Verify_Syscalls();
}
print "Finished. . .\n";

#####################################################################

sub PrintUsage() {

  print "\nUsage: $0 -V -h\n\t-V Verify system calls found,";
  print " reports which system calls if any were not located.\n";
  print "\t-h Print this message.\n\n";

}

sub get_syscall_info() {
  my($date, $answer);

  parse_syscall_file();
  return;

  if (( -e $SYSCALL_FILE ) && ( -f $SYSCALL_FILE ) &&
      ( -r $SYSCALL_FILE ) && ( -s $SYSCALL_FILE )) {
    $time = (stat($SYSCALL_FILE))[9];
    ($sec, $min, $hr, $dom, $mon, $year,
     $wday, $yday, $isdst) = localtime($time);

    print "$SYSCALL_FILE exists and was created/modified on ";
    printf("%02d:%02d:%02d (%02d/%02d/%04d)\n",
           $hr, $min, $sec, $mon+1, $dom, $year+1900);
    print "Would you like to use this file for the system call ";
    print "information (y/n)? ";

    $answer = <STDIN>;
    chomp $answer;

    $answer = lc($answer);
    if ($answer ne "y") {
      create_syscall_file();
    }
  }
  else {
    create_syscall_file();
  }

  parse_syscall_file();
}


# Create the array of hashes with initial
# info based off system call functions
# listed in the $ENTRY_S file
sub gen_syscall_array() {
  my(@a, $cnt, $cnt2, $name, $call);
```

```
  @a = `$GREP -E '.long SYMBOL_NAME\(.+\)' $ENTRY_S`;

  $cnt = 0;
  foreach $call (@a) {
    chomp $call;

    $call =~ /\s*\.long\sSYMBOL_NAME\((\w+)\).*/;
    $name = $1;                                                    120

#    $name =~s/^old_//; # strip old_ first
#    $name =~ s/^sys_//; # strip sys_

    $SYSCALLS[$cnt]{"NAME"} = $name;
    $SYSCALLS[$cnt]{"NUM"} = $cnt;
    $SYSCALLS[$cnt]{"RETTYPE"} = undef;
    $SYSCALLS[$cnt]{"NUMARGS"} = undef;
    for ($cnt2 = 0; $cnt2 < $MAX_NUM_ARGS; $cnt2++) {
      $SYSCALLS[$cnt]{"ARGS"}[$cnt2][$ARG_TYPE] = "NULL";          130
      $SYSCALLS[$cnt]{"ARGS"}[$cnt2][$ARG_NAME] = "NULL";
    }

    $cnt++;
  }
}


#
sub parse_syscall_file() {                                         140

  my($line, $cnt, $cnt2, $arg, $name, $ret, $args,
     @arguments, $n, $arg_type, $arg_name, @a);

  open(FH, $SYSCALL_FILE) || die "Unable to open $SYSCALL_FILE";

  foreach $line (<FH>) {
    chomp $line;

    ($ret, $name, $args) = split(/:/, $line, 3);                   150

    $args =~ s/[()]//g;

    for ($cnt=0; $cnt <= $#SYSCALLS; $cnt++) {
      # Fill in info for this function
      if ($name ne $SYSCALLS[$cnt]{"NAME"}) {
        next;
      }

      $SYSCALLS[$cnt]{"RETTYPE"} = $ret;                           160

      @arguments = split(/,/, $args);


      if ($#arguments == 0) {
        # remove starting/trailing whitespace
        $arguments[0] =~ s/^\s+//;
        $arguments[0] =~ s/\s+$//;
        if ( (index($arguments[0]," ") == -1) &&
```

```
        ($arguments[0] eq "void") ) {
      $SYSCALLS[$cnt]{"NUMARGS"} = 0;
    }
    else {
      $arg = $arguments[0];
      $SYSCALLS[$cnt]{"NUMARGS"} = 1;
      $n = rindex($arg, " ");
      $arg_type = substr($arg, 0, $n);
      $arg_name = substr($arg, $n+1);
      $arg_type =~ s/^\s+//;
      $arg_name =~ s/^\s+//;

      # Fill in the ARGS part of SYSCALLS
      # @a = @SYSCALLS[$cnt]{"ARGS"};
      $SYSCALLS[$cnt]{"ARGS"}[0][$ARG_TYPE] = $arg_type;
      $SYSCALLS[$cnt]{"ARGS"}[0][$ARG_NAME] = $arg_name;

      # $a[0][$ARG_TYPE] = $arg_type;
      # $a[0][$ARG_TYPE] = $arg_name;

    }
  }
  else {
    $SYSCALLS[$cnt]{"NUMARGS"} = $#arguments+1;
    # break apart the arguments
    # @a = @SYSCALLS[$cnt]{"ARGS"};
    $cnt2 = 0;
    foreach $arg (@arguments) {
      $n = rindex($arg, " ");
      $arg_type = substr($arg, 0, $n);
      $arg_name = substr($arg, $n+1);
      $arg_type =~ s/^\s+//;
      $arg_name =~ s/^\s+//;

      # $a[$cnt2][$ARG_TYPE] = $arg_type;
      # $a[$cnt2][$ARG_NAME] = $arg_name;
      $SYSCALLS[$cnt]{"ARGS"}[$cnt2][$ARG_TYPE] = $arg_type;
      $SYSCALLS[$cnt]{"ARGS"}[$cnt2][$ARG_NAME] = $arg_name;
      $cnt2++;
    }
  }
  $n = $SYSCALLS[$cnt]{"NUMARGS"};

 } # for()
} # foreach
}


# Search through source code for
# system call function implementations (ie all .c files are searched)
# Parse out the arguments, name, return value and dump
# to argument
sub create_syscall_file() {
  my(@a, $ret, $massive_string, $line, $arg);

  print "Generating $SYSCALL_FILE file\n";
  open(SCFILE, "+>".$SYSCALL_FILE) ||
```

```
    die "Unable to open $SYSCALL_FILE";

  @a = `$FIND $LNX_SRC -type f -iname \"*.c\" -print`;

  foreach $ret (@a) {
    $massive_string = undef;

    # Skip if file is in arch/ directory that isn't the $ARCH
    # we are looking for
    if ($ret =~ /\/arch\/(?!$ARCH)/) {
      next;
    }

    open(FH, $ret) || die "Unable to open $ret";
    @contents = <FH>;

    foreach $line (@contents) {
      $massive_string .= $line;
    }

    $_ = $massive_string;
    # Split up each system call function we find
    #                    *Return *    *Name* *   Arguments    *
    while (/^\s*asmlinkage\s+([\w\s]+)\s+((sys|old)_\w+)\s*(\([\w\s,\*]*\))/gm) {
      $ret_type = $1;
      $func_name = $2;
      $func_args = $4;

      $ret_type =~ s/\s+//g;
      $func_name =~ s/\s+//g;
      $func_args =~ s/\s*,\s*/, /g;

      print SCFILE "$ret_type:$func_name:$func_args\n";
    }

    close(FH);
  }

  close(SCFILE);
  print "$SYSCALL_FILE generated.\n";
}


sub create_header_file() {

  my($cnt, $cnt2, $sname, $snum, $sret, $snumargs);


  open(HFILE, "+>$HEADER_FILE") || die "Unable to open $HEADER_FILE";

  # Print Header and function prototype info
  print HFILE "#ifndef _SYSCALL_LIST_H\n#define _SYSCALL_LIST_H\n\n";
  print HFILE "/* WARNING: DO NOT EDIT THIS FILE BY HAND */\n";
  print HFILE "/* THIS FILE IS GENERATED BY $0 */\n";
  print HFILE "/* IF MAX_NUM_ARGS changes in ptools.h the script */\n";
  print HFILE "/* must be updated                              */\n\n\n";
  print HFILE "/* pTools: Process Information Utilities\n * Stephen Oberther\n";
```

```
print HFILE " * oberther\@cs.fsu.edu\n";
print HFILE " *\n * Functions to handle signal number to name and name";
print HFILE " to number conversions.\n */";

print HFILE "#include \"./ptools.h\"\n\n";

print HFILE "/* Convert syscall number into a string representation\n";
print HFILE " * String will be stored in buf and will not be longer\n";
print HFILE " * than bufln. The value stored is also returned.\n *\n";
print HFILE " * NULL is returned on error.\n */\n";
print HFILE "char *syscallnum2str(int syscall, char *buf, size_t bufln);\n\n\n";

print HFILE "/* Convert str into the numerical equivalent. On erro\n";
print HFILE " * -1 is returned, a valid syscall number is returned\n";
print HFILE " * on success\n */\n";
print HFILE "int str2syscallnum(char *str);\n\n";

print HFILE "/* Return the number of arguments that the system call has */\n";
print HFILE "int syscall_numargs(int sc_num);\n\n";

# print arg_info and syscall_map structure definitions
print HFILE "#define FIRST_SYSCALL 1\n\n";
print HFILE "\nstruct arg_info {\n";
print HFILE " const char *arg_type;\n";
print HFILE " const char *arg_name;\n";
print HFILE "};\n";

print HFILE "\nstruct syscall_map {\n";
print HFILE " const char *syscall_str;\n";
print HFILE " const int syscall_num;\n";
print HFILE " const char *syscall_ret;\n";
print HFILE " const int syscall_numargs;\n";
print HFILE " const struct arg_info syscall_args[MAX_NUM_ARGS];\n";
print HFILE "};\n\n";

print HFILE "static struct syscall_map syscall_list[] = {\n";

for ($cnt=0; $cnt <= $#SYSCALLS; $cnt++) {
  # Print initialization info for each syscall
  $sname = $SYSCALLS[$cnt]{"NAME"};
  $snum = $SYSCALLS[$cnt]{"NUM"};
  $sret = $SYSCALLS[$cnt]{"RETTYPE"};
  $snumargs = $SYSCALLS[$cnt]{"NUMARGS"};

  if (! defined $SYSCALLS[$cnt]{"RETTYPE"}) {
    print HFILE " {\"$sname\", $snum, NULL, -1";
  }
  else {
    print HFILE " {\"$sname\", $snum, \"$sret\", $snumargs";
  }

  print HFILE ", { ";
  for ($cnt2=0; $cnt2 < $MAX_NUM_ARGS; $cnt2++) {
    $argtype = $SYSCALLS[$cnt]{"ARGS"}[$cnt2][$ARG_TYPE];
    $argname = $SYSCALLS[$cnt]{"ARGS"}[$cnt2][$ARG_NAME];
    if ($argtype ne "NULL") {
```

```
      print HFILE "{\"$argtype\", ";
    }
    else {
      print HFILE "{$argtype, ";
    }
    if ($argname ne "NULL") {
      print HFILE "\"$argname\"}";
    }
    else {
      print HFILE "$argname}";
    }

    if ($cnt2+1 < $MAX_NUM_ARGS) { print HFILE ", "; }
    if (! ($cnt2 % 2)) { print HFILE "\n              "; }
  }

  print HFILE " } },\n";

}

# Terminate initialization
print HFILE " {NULL, -1, NULL, -1, { ";
for ($cnt2 = 0; $cnt2 < $MAX_NUM_ARGS; $cnt2++) {
  print HFILE "{NULL, NULL}";
  if ($cnt2+1 < $MAX_NUM_ARGS) { print HFILE ", "; }
  if (! ($cnt2 % 2)) { print HFILE "\n              "; }
}

print HFILE " } }\n};\n\n";

print HFILE "#endif\n";
close(HFILE);
}



sub Verify_Syscalls() {

  my($cnt, $name);

  for ($cnt=0; $cnt <= $#SYSCALLS; $cnt++) {
    if (! defined $SYSCALLS[$cnt]{"RETTYPE"}) {
      $name = $SYSCALLS[$cnt]{"NAME"};
      print "($cnt)$name not located\n";
    }
  }

}
```