Building blocks for Unix power tools

Now that we have given a good overview of a lot of the better Unix tools, I want to take some time to talk about our toolset for building Unix programs.

The most important of these are the system calls.

Building blocks for Unix power tools

A Unix system call is a direct request to the kernel regarding a system resource. It might be a request for a file descriptor to manipulate a file, it might be a request to write to a file descriptor, or any of hundreds of possible operations.

These are exactly the tools that every Unix program is built upon.

File descriptor and file descriptor operations

In some sense, the mainstay operations are those on the file system.

File descriptor and file descriptor operations

Unlike many other resources which are just artifacts of the operating system and disappear at each reboot, changing a file system generally is an operation that has some permanence (although of course it is possible and even common to have “RAM” disk filesystems since they are quite fast, and for items that are meant to be temporary anyway, they are quite acceptable.)

Important file descriptor calls

A file descriptor is an int. It provides stateful access to an i/o resource such as a file on a filesystem, a pseudo-terminal, or a socket to a tcp session.

```
open()    -- create a new file descriptor to access a file
close()   -- deallocate a file descriptor
```

Important file descriptor calls

```
dup()     -- duplicate a file descriptor
dup2()    -- duplicate a file descriptor
```

Important file descriptor calls

```
fchmod()  -- change the permissions of a file associated with a file
          -- descriptor
fchown()  -- change the ownership of a file assocaited with a file
```

Important file descriptor calls

```
fcntl()   -- miscellaneous manipulation of file descriptors: dup(), set
          -- close on exec(), set to non-blocking, set to asynchronous
          -- mode, locks, signals
ioctl()   -- manipulate the underlying ‘‘device’’ parameters for
```

Important file descriptor calls

```
flock()   -- lock a file associated with a file descriptor
```

Important file descriptor calls

```
pipe()    -- create a one-way association between two file
          -- descriptors so that output from
          -- one goes to the input of the other
```

Important file descriptor calls

```
select()  -- multiplex on pending i/o to or from a set of file descriptor
```

Important file descriptor calls

```
read()    -- send data to a file descriptor
write()   -- take data from a file descriptor
```

Important file descriptor calls

```
readdir() -- raw read of directory entry from a file descriptor
```

Important file descriptor calls

```
fstat()   -- return information about a file associated with a fd: inode,
             perms, hard links, uid, gid, size, modtimes
fstatfs() -- return the mount information for the filesystem that the fil
          -- descriptor is associated with
```

Important filesystem operations

In addition to using the indirect means of file descriptors, Unix also offers a number of direct functions on files.

```
access()  -- returns a value indicating if a file is accessible
chmod()   -- changes the permissions on a file in a filesystem
chown()   -- changes the ownership of a file in a filesystem
```

Important filesystem operations

```
link()    -- create a hard link to a file
symlink() -- create a soft link to a file
```

Important filesystem operations

```
mkdir()   -- create a new directory
rmdir()   -- remove a directory
```

Important filesystem operations

```
stat()    -- return information about a file associated with a fd: inode,
             perms, hard links, uid, gid, size, modtimes
statfs()  -- return the mount information for the filesystem that the fil
          -- descriptor is associated with
```

Signals

```
alarm       -- set an alarm clock for a SIGALRM to be sent to a process
            -- time measured in seconds
getitimer   -- set an alarm clock in fractions of a second to deliver eit
            -- SIGALRM, SIGVTALRM, SIGPROF
```

Signals

```
kill        -- send an arbitrary signal to an arbitrary process
killpg      -- send an arbitrary signal to all processes in a process gro
```

Signals

```
sigaction   -- interpose a signal handler (can include special ‘‘default’
            -- ‘‘ignore’’ handlers)
sigprocmask -- change the list of blocked signals
```

Signals

```
wait        -- check for a signal (can be blocking or non-blocking) or ch
waitpid     -- check for a signal from a child process (can be general or
```

Modifying the current process's state

```
chdir       -- change the working directory for a process to dirname
fchdir      -- change the working directory for a process via fd
chroot      -- change the root filesystem for a process
```

Modifying the current process's state

```
execve      -- execute another binary in this current process
fork        -- create a new child process running the same binary
clone       -- allows the child to share execution context (unlike fork(2
exit        -- terminate the current process
```

Modifying the current process's state

```
getdtablesize  -- report how many file descriptors this process can have
               -- active simultaneously
```

Modifying the current process's state

```
getgid      -- return the group id of this process
getuid      -- return the user id of this process
getpgid     -- return process group id of this process
getpgrp     -- return process group's group of this process
```

Modifying the current process's state

```
getpid      -- return the process id of this process
getppid     -- return parent process id of this process
getrlimit   -- set a resource limit on this process (core size, cpu time,
            -- data size, stack size, and others)
getrusage   -- find amount of resource usage by this process
```

Modifying the current process's state

```
nice        -- change the process's priority
```

Networking

```
socket      -- create a file descriptor

bind        -- bind a file descriptor to an address, such a tcp port
listen      -- specify willingness for some number of connections to be
            -- blocked waiting on accept()
accept      -- tell a file descriptor block until there is a new connecti

connect     -- actively connect to listen()ing socket

setsockopt  -- set options on a given socket associated with fd, such out
            -- data, keep-alive information, congestion notification, fin
            -- and so forth (see man tcp(7))
getsockopt  -- retrieve information about options enabled for a given con

getpeername -- retrieve information about other side of a connection from
getsockname -- retrieve information this side of a connection from fd
```