

Recitation Week #5

More Parsing, More Process Control

Alejandro Cabrera
Operating Systems
COP4610 / CGS5765

Today's Recitation

- Parsing
 - Corner cases
- C String Manipulation
- Process Control
 - I/O Re-direction (external slide set)
 - Background Processing
 - Handling Multiple Pipes
 - Zombies

Parsing Corner Cases

- Special symbol attached to command line token:
 - # ls -l&
 - # grep 'other'< input_file
 - # grep 'other' <input_file
 - # ls> ls.log
 - # ls >ls.log
 - # ls| more
 - # ls |more
 - # ls |more&

Parsing Corner Cases

- \$VAR inside of quotation marks:
 - # ls "\$HOME" → ls "/home/cabrera"
 - # ls '\$HOME' → ls '\$HOME'
 - # ls "\$dont_exist" → error
 - # ls '\$dont_exist' → ls '\$dont_exist'
- Important to consider.
- However, in this project, we'll assume that no variable declaration occurs within quotation marks.
 - Parsing is hard enough as is.

Parsing Corner Cases

- More arguments after I/O redirection symbol:
 - # ls > ls.log /usr/include /home/
 - # grep 'strcmp' < input_file /usr/include/

How TCSH Handles Additional Args After I/O Redirect: ls

```
# ls > ls.log /usr/include
```

How TCSH Handles Additional Args After I/O Redirect: ls

```
# ls > ls.log /usr/include
```

```
#
```

How TCSH Handles Additional Args After I/O Redirect: ls

```
# ls > ls.log /usr/include  
#  
# more ls.log
```

How TCSH Handles Additional Args After I/O Redirect: ls

```
# ls > ls.log /usr/include  
#  
# more ls.log  
aio.h  
aliases.h  
alloca.h  
a.out.h  
...
```

How TCSH Handles Additional Args After I/O Redirect: grep(1)

```
# grep 'aio.h' < ls.log /usr/include
```

How TCSH Handles Additional Args After I/O Redirect: grep(1)

```
# grep 'aio.h' < ls.log /usr/include  
#
```

How TCSH Handles Additional Args After I/O Redirect: grep(2)

```
# grep 'aio.h' < ls.log
```

How TCSH Handles Additional Args After I/O Redirect: grep(2)

```
# grep 'aio.h' < ls.log  
aio.h
```

How TCSH Handles Additional Args After I/O Redirect: grep(2)

```
# grep 'aio.h' < ls.log  
aio.h  
#
```

How TCSH Handles Additional Args After I/O Redirect: grep(3)

```
# grep 'aio.h' /usr/include/* <ls.log
```

How TCSH Handles Additional Args After I/O Redirect: grep(3)

```
# grep 'aio.h' /usr/include/* <ls.log  
/usr/include/aio.h:#endif /* aio.h */
```

How TCSH Handles Additional Args After I/O Redirect: grep(3)

```
# grep 'aio.h' /usr/include/* <ls.log  
/usr/include/aio.h:#endif /* aio.h */  
#
```

How TCSH Handles Additional Args After I/O Redirect: grep(4)

```
# grep 'aio.h' <ls.log /usr/include/*
```

How TCSH Handles Additional Args After I/O Redirect: grep(4)

```
# grep 'aio.h' <ls.log /usr/include/*  
/usr/include/aio.h:#endif /* aio.h */
```

How TCSH Handles Additional Args After I/O Redirect: grep(4)

```
# grep 'aio.h' <ls.log /usr/include/*  
/usr/include/aio.h:#endif /* aio.h */  
#
```

Parsing Corner Cases

- In all cases, refer to the TCSH implementation with the project specification in mind. There are some corner cases you won't need to handle.

C String Manipulation: A Useful Trick

- Understanding *const*
- Pointer Arithmetic
 - Obtaining the Index of a Character

Understanding *const*

- Definitions:
 - Constant – A read-only memory location. It cannot be assigned to. The compiler will issue warnings if it sees you assigning to one.
 - Mutable – A read-write memory location. Anything goes!
- Liberal use of *const* can save a great deal of debugging time, since we're asking the compiler to warn us if we're changing something we don't want to change.

Understanding *const*

- *const* is *left-associative*
- This means that what lies to the left of the *const* declaration is what is protected.
- Examples follow.

Understanding *const*

`char *str`

→ Mutable pointer to mutable char

`const char *str`

→ Mutable pointer to constant char

`char const *str`

→ Mutable pointer to constant char

Understanding *const*

`char *const str`

→ Constant pointer to mutable char

`const char *const str`

→ Constant pointer to constant char

`char const **const str`

→ Constant pointer to constant char

`char const **const str`

→ Constant pointer to mutable pointer to constant char

Pointer Arithmetic

- Pointer arithmetic refers to the act of adding a value to or subtracting a value from a pointer to generate a new address.
- Compiler knows the size of all pointers, so can perform correct additions for you.

Pointer Arithmetic: Obtaining the Index of a Character in a String

```
char *str = "ls -l | more";  
char *target_location = strchr(str, '|');  
  
/* Subtract the memory location of the target from  
   the memory location of the start of string. */  
size_t index = target_location - str;  
printf("%c \n", str[index]);
```

I/O Redirection

- Refer to second slide set.

Multiple Pipes

- Although in this project, you won't be tested against more than 3 '|' in a single command line, the code to correctly handle any number of pipes is the same.
- A recursive definition of the algorithm follows.

Recursively Parsing Pipes

```
Procedure Execute_Pipe(lhs,rhs)
  Input: command lhs, command rhs
Begin
  Parse pipe from end of string;
  If there are no pipes remaining
    return;
  Else If a pipe is encountered
    lhs = new_lhs; rhs = new_rhs
    set up I/O redirection
    Execute_Pipe(lhs,rhs)
    execute(lhs), execute(rhs)
  Endif
End
```

Zombies

- A zombie is a child process that has no parent to report to.
 - The parent terminated before the child without cleaning up.
- This child processes stays allocated on the system until reboot, using up memory and scheduler resources.

The init Process

- All processes have as a common (super)parent the init process.
- Unlike typical parent processes, the init process kills all processes that have ceased executing.
- In other words, the init processes cleans up all zombie children.

Avoiding Zombies (Easily)

- Fork() twice:
 - fork() once from the shell to create a child.
 - fork() again from the child to create a child of the child.
 - Terminate the child process.
 - The child of the child is no longer connected to the parent.
 - The child of the child is therefore inherited by the init process.
 - It will be reaped when it finishes executing.

Useful Functions for Shell Project

```
int dup(int fd);
int open(const char* pth, int flags);
int close(int fd);
pid_t waitpid(pid_t pid,
              int *stat_loc,
              int options);
int execv(const char *pth,
          char *const argv[]);
pid_t fork(void);
int access(const char *pth, int mode);
int pipe(int fd[2]);
```

Next Time:

- Introduction to the Linux kernel sources
- Introduction to Linux Cross Reference
- Kernel compilation – how to
- Introduction to the basement CS Lab and the machines you'll be using for project 3 & 4.
- (Possibly) Using Linked Lists in C

Any Questions?