COMP 530
Introduction to Operating Systems

HW2
Building a Simple Linux Shell

Kevin Jeffay
Department of Computer Science
University of North Carolina at Chapel Hill
jeffay@cs.unc.edu
September 6, 2017

http://www.cs.unc.edu/~jeffay/courses/comp530

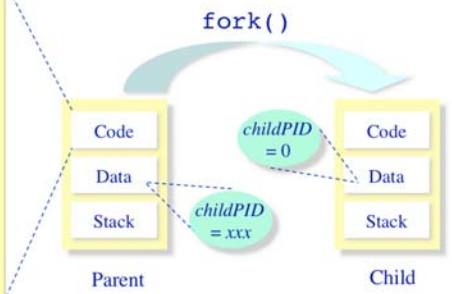
A Simple Linux Shell
More fun with processes

- Goal: Do something substantive with Linux processes
  - » Build a simple shell for Linux
- Write a program that reads a command line from *stdin*
  - » Print a prompt ('% ')
  - » Read a command
  - » Create a child process to parse and execute the command
  - » Parent process just waits for the child to terminate

Process Creation Paradigms
fork/join in Linux

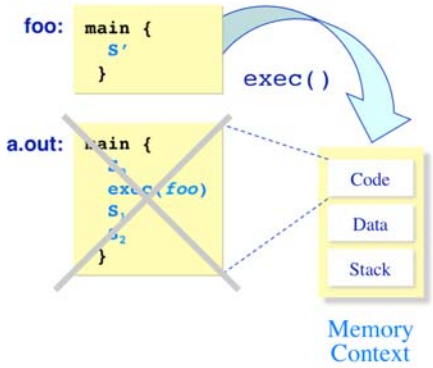
- The execution context for the child process is a *copy* of the parent's context at the time of the call

```
main {
  int childPID;
  S1;

  childPID = fork();

  if(childPID == 0)
    <code for child process>
  else {
     <code for parent process>
     wait();
   }

  S2;
}
```



Process Creation Paradigms
exec in Linux

- *exec* allows a process to replace itself with another program
  - » (The contents of another binary file)

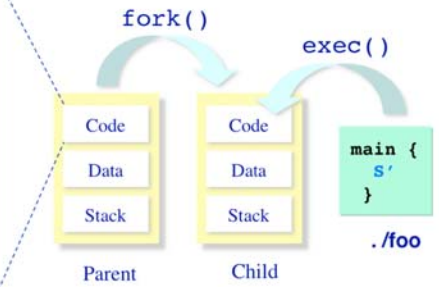


Process Creation Paradigms

Abstract *fork* in Linux

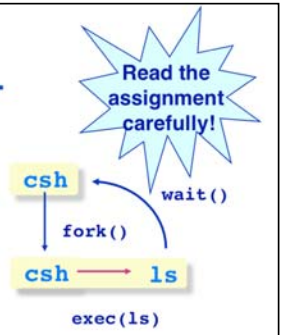
- The original *fork* semantics can be realized in Linux via a (UNIX) *fork* followed by an *exec*

```
main {
  int childPID;
  S1;

  childPID = fork();

  if(childPID == 0)
    exec(filename)
  else {
     <code for parent process>
     wait();
   }

  S2;
}
```



HW2

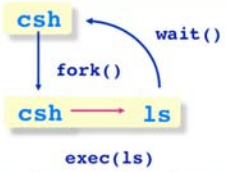
A simple Linux shell



- Write a program that reads commands from *stdin* and forks a process to parse and execute each command
- Commands are interpreted as filenames
- Extra credit:
  - » Check to see if the file exists before calling *exec*
  - » Read the PATH environment variable and find the appropriate directory prefix for the command file
  - » Read up on *signals* and make your program "catch" the cntrl-C and cntrl-Z signals

HW2

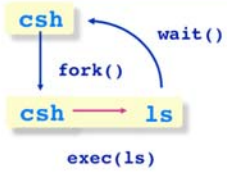
A not-so-simple Linux shell!



- You're going to be creating lots of processes in this assignment
- If you fork a process and it never terminates…
- You've just created a ZOMBIE PROCESS!!
  - » Zombie's will fill up the process table in the Linux kernel
  - » Nobody can create a new process
  - » This means no one can launch a shell to kill the zombies
  - » You've just launched a denial-of-service attack on your classmates!

HW2

A not-so-simple Linux shell!



- Be safe! Limit the number of processes you can create
  - » add the command "*limit maxproc 10*" to the file ~/.cshrc
  - » (remember to delete this line at the end of the course!)
- Periodically check for and KILL! zombie processes
  - » *ps -ef | egrep -e PID -e YOUR-LOGIN-NAME*
  - » *kill pid-number*
- Read the HW handout carefully for zombie-hunting details!

HW2 Building a Simple Linux Shell

%man execvp

EXEC(3) Linux Programmer's Manual EXEC(3)

NAME
execl, execlp, execle, execv, execvp - execute a file

SYNOPSIS
#include <unistd.h>

extern char **environ;

int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg , ..., char * const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);

DESCRIPTION
The exec family of functions replaces the current process image with a new process image. The functions described in this manual page are front-ends for the function execve(2). (See the manual page for execve for detailed information about the replacement of the current process.)

HW2 Building a Simple Linux Shell

%man execvp

DESCRIPTION
The exec family of functions replaces the current process image with a new process image. The functions described in this manual page are front-ends for the function execve(2). (See the manual page for execve for detailed information about the replacement of the current process.)

The initial argument for these functions is the pathname of a file which is to be executed.

The const char *arg and subsequent ellipses in the execl, execlp, and execle functions can be thought of as arg0, arg1, ..., argn. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program. The first argument, by convention, should point to the file name associated with the file being executed. The list of arguments must be terminated by a NULL pointer.

The execv and execvp functions provide an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the file name associated with the file being executed. The array of pointers must be terminated by a NULL pointer.

Important Reminder!

COMP 530 Policy on collaboration

- Working in groups on homeworks is OK but…
  - » You can only collaborate with other students in the course
  - » Every student must craft their own final solution
  - » Every student must fully write up their solution
  - » All collaborators must be acknowledged in writing
- Programming assignments must be completed by each student individually
  - » You have to write every line of code yourself
  - » Code may *never* be shared
  - » Using code from the Internet is not allowed

Not following these rules is an Honor Code violation