UNIX System Programming
Lecture 11: Processes

Outline
- Introduction to Processes
- Creating New Processes

Reference
- BLP: Chapter 11
- man pages: getpid, getppid, system, wait, waitpid, fork, exec

Lecture 11: Processes Introduction

- Suppose we are writing an application that requires several matrix operations, or we want to produce a nice graph, or convert an image from GIF to PNG formats, etc.
- We could either: (1) write the required routines from scratch, (2) find and use __________ routines that do the work, or (3) use existing programs such as octave, gnuplot or the PNM utilities.

Lecture 11: Processes Introduction

- The third approach can be very powerful. It is often the easiest method as well. Why not use an existing utility that does what I need and I know to be efficient and robust?
- UNIX allows us to run other programs from our program and communicate with these programs using a number of different interprocess communication (______) methods.

Lecture 11: Processes What Is A Process?

- A program is a collection of instructions and data kept in an file.
- A ____________ is a running program. It consists of instruction, user data, and system data segments.
- The instruction and user data segments are initialized from the program.
- System data includes the current directory, open file descriptors, total CPU time, etc.

Lecture 11: Processes Parent and Child Processes

- A new _____ process is created by the kernel on behalf of a parent process (via fork()).
- A child inherits most of the system data from the parent. For example, files that are open by the parent, will also be open to the child.
- Each process has a unique process-ID (a positive int). getpid() and getppid() can be used to find the current and parent process IDs. The init process has ID 1.
Lecture 11: Processes  
Process Groups

- Each process is a member of a unique process group with its own process group ID (PGID). When a process is created it is a member of its parent's process group.
- One process is the process group leader. The PGID is the same as the leader PID.
- Only one process group is the ____________ process group. The terminal device driver sends tty signals (interrupt, quit, etc) to each process in the foreground process group.

Lecture 11: Processes  
Additional Process Info

- At the command line use "ps ajx" to list the PIDs, PPIDs and PGIDs of all processes.
- A parent may wait for a child to terminate by using the wait() or waitpid() routines.
- If a child ends before the parent calls wait(), then the child lives on as a "______" process until the parent calls wait() or ends.
- If a parent ends, a child's parent process ID is set to 1. (The child is adopted by init.)

Lecture 11: Processes  
Process Creation: system()

- The system() function can be used to run another _______ (and thereby create a new process) from an existing process:
  system("program_name");
- The parent process waits until the program completes. (See sys_xmpl.cpp)
- The program is run just as if the following had been entered at the command prompt:
  sh -c program_name

Lecture 11: Processes  
Process Creation: fork/exec

- The system() function is built upon the fork(), exec() and wait() routines.
- fork() creates a child process that is a _____ of the parent (identical instruction, user-data, and system-data segments).
- The exec() routine reinitializes a process from a designated program (file on disk).
- fork() & exec() are usually used together.

Lecture 11: Processes  
The exec() System Calls

- There are six exec*() calls. First exec():
  int exec(char *path, char *arg0, ..., char *argv, (char *)0);
- Here is an example:
  execut("/bin/ls", "ls", "-al", "/tmp", 0);
- Recall that a new process is not created by exec(). The code and user-data segments are reinitialized from the indicated program. The ______ data segment is not overwritten.

Lecture 11: Processes  
The exec() System Calls

- exec*() is the only way to execute a program under UNIX. (exec_xmpl.cpp)
- Other exec*() calls: execclp(), execle(), execv(), execvp() and execl():
  - execv() routines expect the arguments to be passed in an array.
  - execvp() routines look for the program in the _________.
  - exec*e() routines pass a pointer to a new environment array (instead of a copy).
Lecture 11: Processes
The fork() System Call

• With only exec*() we'd have a single process on the system. We could run different programs by having each program call exec*() to start a new program.
• The __________() routine is the only way to create a new process. The new (child) process's instruction, user-data, and system-data segments are almost exact copies of the old (parent) process segments. (The PIDs and PPIDs will, of course, be different.)

Lecture 11: Processes
The fork() System Call

• The child gets copies of the parents open file descriptors. The file pointer is in the system file table and is shared by both processes.
• When fork() returns, both processes (parent and child) get different _______ values. The child receives 0 while the parent receives the PID of the child. Usually the child will do an exec() while the parent either waits for the child or goes off to do something else.

Lecture 11: Processes
The fork() System Call

• For example, the following code:
  cout << “Start of test.” << endl;
  pid = fork();
  cout << “fork() returned ” << pid << endl;

  would display
  Start of test.
  fork() returned 0
  fork() returned 17625

• See frk_xmpl1.cpp, frk_xmpl2.cpp, and vssh_xmpl.cpp.