Lab #2: COMP 3000A (Operating Systems)
October 2, 2007

Please answer all questions below. There are **80** points total.

Part A is designed to be done in the lab while part B is designed to be done on your own time. Many of the questions require you to compile and run sample programs. These programs can be compiled using either the Corel Lab computers or your SCS Linux account.

For all questions asking you to modify source code, you should always start with a clean version of the source code and modify that (unless otherwise directed in the question). Don't continue modifying your solution to a previous question in order to answer the next question.

All programs given in this assignment are written in C. A makefile is provided to compile the core programs given by the assignment—type "make" to compile. If you wish to rename files, you will need to either edit the makefile or run GCC to compile the programs on your own.

1 Part A

[8] Have the instructor mark down that you were present and attempted part A during lab hours.

1.1 Processes and Threads

1. **[2]** The program `threads.c` is a multithreaded producer/consumer program. Unfortunately it consumes faster than it produces, resulting in an error. Why does it not print the same number every time?

2. **[2]** The program `passstr.c` is a multithreaded program using the *clone* function call. What is wrong with the way this program blocks, waiting for the string to arrive in the buffer?

1.2 Fork & Exec

[2] What is the difference between the clone and the fork function call?

1.3 IPC

Examine the program given in wait-signal.c. It multiplies two matrices together using the standard trivial algorithm (which also happens to be a n^3 algorithm). It spawns off a child process to compute the value of each element in the resulting matrix. The program has a problem, however, in that it fails to pass the resulting values back to the parent process in order to give the right result. In this section, we will examine various methods for passing data between processes.

1.3.1 Signals

Signals can be sent to each process running on the system. Signals, however, don't allow the passing of any data along with the signal. Therefore, they are most useful for triggering actions.

1. **[1]** The `kill` command actually sends signals to processes. What signal does the `kill` command by default send to a process?

2. **[2]** Modify the *wait-signal-1.c* file to use the `signal` function to install the signal handler instead of the `sigaction` function call. You can have it install the `child_handler_alt` signal handler instead of the `child_handler` signal handler. What line did you add to install the signal handler to `child_handler_alt`?

3. **[2]** Modify the *wait-signal.c* file to ignore the abort signal. What line did you have to add to do this?

1.3.2 Pipes

Pipes (also called FIFO's) allow two processes to communicate through a file handle. One process writes data into a file handle and the other process can then read that data out through a related but different file handle.

1. **[2]** What happens to file descriptors across an *exec* call? Write a small program that tests this behavior, i.e. that opens a file, calls execve, and then the new program attempts to read from the previously opened file descriptor. Explain how this program behaves.

2. **[2]** Compile and run *pipe.c*. Notice how data is sent through the pipe by writing to one end of the pipe in the child and reading from the other end of the pipe in the parent. Also notice how the message *Finished writing the data!* is never displayed on the screen. The problem has to do with the SIGPIPE signal. What is the problem?

1.3.3 Shared Memory

1. **[1]** Shared memory regions are controlled by the kernel to prevent other processes from accessing the memory without permission. Like files in Unix, the shared memory regions are given read, write and execute permission. These permissions are specified in the call to `shmget`. Where in the arguments to `shmget` are the permissions specified?

2. **[1]** The permissions must be specified as a value. By reading the manpage of *chmod*, determine what the permission 0760 means.

3. **[2]** What number is going to be required in order for two processes owned by the same user to be able to read and write to the shared memory?

2 Part B

2.1 Processes

1. **[3]** From class, you know that the process descriptor contains numerous information about a running process on the system. The task structure in Linux is called `struct task_struct`. By examining the source of the Linux kernel, determine what source file this structure is defined in. The `grep` command may be useful in locating the correct file.

2. **[7]** Figure 6.3 (page 213) in your textbook contains a list of common elements found in a process table. Determine at least one variable in the Linux task structure which is related to each element listed in Figure 6.3. You may omit address space and stack.

2.2 Fork & Exec

1. **[10]** Examining the flags that can be passed to the clone function call. Choose 5 flags and describe a situation in which each of them would be useful.

   [5] Find the portion of the Linux kernel that implements the fork, clone, and vfork system calls for i386 systems. Based upon this code, could Linux instead just have one of these system calls?

   If so, which one, and how would you implement userspace "wrappers" that would provide identical functionality for the other two calls?

   If not, why are all three necessary? Explain.

   (For this question, ignore issues of binary compatibility.)

2. **[3]** File descriptors 0, 1, and 2 are special in Linux, in that they refer to standard in, standard out, and standard error. Does the Linux kernel know they are special? Explain, referring to appropriate parts of the Linux kernel source.

2.3 IPC

In this section, you will be modifying the program wait-signal to correctly compute the value of the matrix multiplication.

2.3.1 Signals

[5] Describe in words how you might modify the *wait-signal* program to correctly pass back the value computed in the child to the parent using only signals. Remember that signals do not allow data to be passed back and forth. Also keep in mind that there are only around 32 signals that can be sent to a process. You do not have to implement your answer, only describe what you would do.

2.3.2 Pipes

[10] Modify the *wait-signal.c* program to pass the appropriate matrix data back to the parent via a pipe. Remember that you will also have to pass back the x and y locations that the data should be put in. What is your updated main function?

2.3.3 Shared Memory

[10] Modify *wait-signal.c* to send data back to the main process using shared memory. You will need to use the functions `shmget` and `shmat`.