CSCI 4061: Introduction to Operating System

Fall 2017 (Instructor: Anand Tripathi)

Assignment 3: Process Management

Due: November 9

This assignment can be done either individually or in a team of up to two students.

PART - 1 : Theory (20 points)

Question 1: (5 points) What are the important data items related to a process that are maintained by the kernel for process management?

Question 2: (5 points) For each of the following four cases, identify the conditions under which the scheduler will change the status of a process:

a. Running to Ready
b. Swapped to Running
c. Running to Waiting (Blocked)
d. Ready or Waiting to Swapped

Question 3: (5 points) When a UNIX process executes fork(), does the child process inherit

a. any pending signals of the parent?
b. the signal handlers of the parent process?
c. the signal mask of the parent?

Question 4: (5 points) Why is a separate stack in the kernel memory space used for handling system call functions and interrupt handlers for a process, instead of using the process stack?

.

PART – 2 : Programming Project (100 points)

1. General information

This document has 3 pages. Please make sure you have all the pages.

2. Objective

In this assignment you will learn system calls related to Process Management. You will learn system calls to create processes to execute programs, create processes to execute a pipe command, wait for termination of a process, and signal handling.

The objective of this assignment is to learn UNIX system calls for process creation and program execution. Specifically, you will learn various system calls: fork, exec, wait, waitpid, dup, dup2, pipe, kill, getpid, strtok, signal, etc.
You should study the various example programs posted on the course webpage, in the "Process Management" section. These examples illustrate the use of various system calls and functions mentioned above.
You MUST use the given piper_skeleton.c program code as the starting point and fill the missing code for various functions.

3. Group Size

This assignment is to be completed either individually or in a group of two. Students must use the provided template for implementation.

4. Assignment Description

In this assignment you are required to write a program called 'piper' which will perform piped execution of multiple commands. It will create separate process for each command which will execute that command. Your program will perform the piped execution by creating pipes between each pair of process.
The program will take a sequence of commands from the console to be executed in pipe fashion. Commands are separated by "|".
You will execute you program as follows:

```
./piper
```

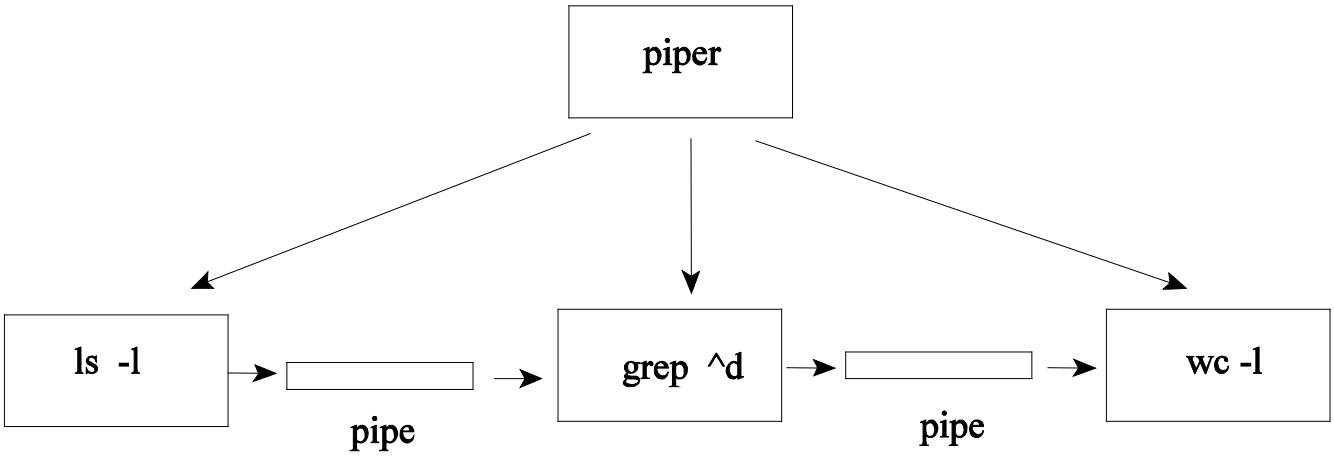
The input from console would be a pipe command such as the one shown below:

```
ls -l | grep ^d | wc -l
```

Your program will need to parse the given command string and identify each command. It will then create one process for each command which will execute the corresponding command along with the specified arguments, and it will set up a pipeline as shown in the example figure.

All the processes will be executed simultaneously. The piper program should create pipe between adjacent commands using the 'pipe' system call. The standard-in of 'piper' will become the standard-in of first command, the standard-out of the first command will become the standard-in of the next command, and so on. The standard out of the last command will be the standard out of the 'piper' program.

For example, in the above case the stdout of the process executing 'ls -l' will be connected as the stdin of the process executing the command "grep ^d".



The piper process will 'wait' for the completion for all of its child process. When a child process terminates, it will print the process id and exit status of the child process.

If any of the child processes terminates due to some failure, the piper would terminate (kill) all the process in the pipeline

Maximum Limits:

(i) Assume that your program is expected to handle at least up to 8 commands, i.e. the input may contain up to 8 commands.
(ii) Assume that the input commands may have up to 4096 characters in total. A command will have a command name (i.e. a program name) and any number of its arguments within this length limit.

Requirement of Printing Status Messages to LOGFILE:

(a) After parsing the console input, the piper should print to the LOGFILE the parsing results, as well as the number of commands.

For example, the expected output to LOGFILE for the above example is:
Command 0 info: ls -l
Command 1 info: grep ^d
Command 2 info: wc -l
Number of commands from the input: 3

(b) After creating all processes in the pipeline, as shown above, the piper should print to the LOGFILE the list of all processes created, indicating their process-ids, the command they are executing. If a failure occurs in creating some process, then it should print an error message such as “Terminating the pipeline process” and indicate which process creation failed and which succeeded (with their command names and process-ids)

Example output for the above example, after creating the processes, it will print to the LOGFILE:

PID	COMMAND
32110	ls –l
32111	grep ^d
32112	wc –l

(c) During the execution of processes, you can print the intermediate steps to the LOGFILE, such as creating processes, waiting processes to finish, and so on.

For example, you may print the following information to the LOGFILE:

```
waiting...Process id 32110 finished
Process id 32110 finished  with exit status 0
waiting...Process id 32111 finished
Process id 32111 finished  with exit status 0
waiting...Process id 32112 finished
Process id 32112 finished  with exit status 0
```

(d) After all processes are completed, it will print to the LOGFILE, the exit status of all processes, along with their process-ids.

After the completion of all pipeline processes, it will print to the LOGFILE:

PID	COMMAND	EXIT STATUS
32110	ls –l	0
32111	grep ^d	0
32112	wc –l	0

(e) The standard output stream used only for printing command execution results.

5. Grading Criteria:

1. Parsing input line containing a pipe command 10%
2. Create processes in the pipeline 20%
3. Create pipes between process in the pipeline 20%
4. Terminate all processes in the pipeline if any process in the pipeline fails to execute 10%
5. Wait for termination of processes in the pipeline 10%
6. Signal Handling 15%
   - Terminate all processes in the pipeline
   - Go back to main loop if user types Ctrl-C
   - Cntrl-C should terminate the “piper” only when no pipeline is currently in execution
7. LOGFILE writing 10%
8. Documentation 5%

You **must** include a Makefile and Readme file as part of the project code submission.

You should frequently check the class forum page on Moodle for FAQs and any clarification and corrections.