Simon Fraser University
School of Computing Science
CMPT 300: Assignment #3

PCB's and Process Scheduling Simulation

Reminder: The rules of academic conduct apply as described in the course outline. All coding is to be done alone. We will be using electronic software to compare all assignments handed in by the class to each other, as well as to assignments handed in for previous terms that this, or similar, assignment may have been given in.

Be sure that this assignment compiles on the Linux computers in the CSIL lab in room Surrey 4080 using the gcc compiler. You can access the Linux server remotely using ssh to *csil-cpu.cs.surrey.sfu.ca*.

What to Hand In: Include your source code files (including .h files and the list implementation), makefile, and any documentation files you would like to include. Please ensure that all files are in plain ASCII format.

Introduction

For this assignment you are going to create an interactive operating system simulation which supports a few basic O/S functions. The simulation will be keyboard driven, and will make reports to the screen.

First of all, what do I mean by a simulation? Well, there will be no real processes doing any real work. There will, however, be data structures which represent processes and the state of the system. For example, when the simulation user issues the command to create a process, a PCB for the new process will be created, initialized and placed in the appropriate O/S queue. When the user issues the command to signal the expiry of a time quantum, the internal state of the simulation will react accordingly and the effects of the expiry will be reported to the screen. The commands issued by the simulation user will be on behalf of the system or the running process.

User Interaction

As mentioned, there will be a simulation user typing at the keyboard to issue commands to your simulation. Each command will consist of a single character. Most commands will require extra information (parameters). These must be prompted for by the simulation after the command character has been read and interpreted.

Please make the screen reports and menu fairly descriptive. When the simulation takes some action, make a full (though concise) report to the screen indicating the actions taken so that the operation of your simulation can easily be followed.

Commands

implment priority

The following commands should be implemented by your simulation:

Command	Character	Parameters	Action	Reports
1 Create	C	int priority (0 = high, 1 = norm, 2 = low)	create a process and put it on the appropriate ready Q.	success or failure, the pid of the created process on success.
7 Fork	F	None	Copy the currently running process and put it on the ready Q corresponding to the original process' priority. Attempting to Fork the "init" process (see below) should fail.	success or failure, the pid of the resulting (new) process on success.
3 Kill	K	int pid (pid of process to be killed)	kill the named process and remove it from the system.	action taken as well as success or failure.
4 Exit	E	None	kill the currently running process.	process scheduling information (eg. which process now gets control of the CPU)
6 Quantum	Q	None	time quantum of running process expires.	action taken (eg. process scheduling information)
Send	S	pid (pid of process to send message to), char *msg (null-terminated message string, 40 char max)	send a message to another process - block until reply	success or failure, scheduling information, and reply source and text (once reply arrives)
Receive	R	None	receive a message - block until one arrives	scheduling information and (once msg is received) the message text and source of message
Reply	Y	int pid (pid of process to make the reply to), char *msg (null-terminated reply string, 40 char max)	unblocks sender and delivers reply	success or failure

New Semaphore	N	int semaphore (semphore ID), initial value (0 or higher)	Initialize the named semaphore with the value given. ID's can take a value from 0 to 4. This can only be done once for a semaphore - subsequent attempts result in error.	action taken as well as success or failure.
Sempahore P	P	int semaphore (semphore ID)	execute the semaphore P operation on behalf of the running process. You can assume sempahores IDs numbered 0 through 4.	action taken (blocked or not) as well as success or failure.
Sempahore V	V	int semaphore (semphore ID)	execute the semaphore V operation on behalf of the running process. You can assume sempahores IDs numbered 0 through 4.	action taken (whether/ which process was readied) as well as success or failure.
Procinfo	I	int pid (pid of process information is to be returned for)	dump complete state information of process to screen (this includes process status and anything else you can think of)	see Action
Totalinfo	T	None	display all process queues and their contents	see Action

Extra Information

You will need to have one process (the "init" process) automatically created and running at simulation startup. It is assumed that this process will always be available to run (will never block). It must only run if ALL other processes are blocked or if there are no other processes. This process can only be killed (exit) if there are no other processes. Once all processes (including init) are gone from the system, the simulation must terminate.

You are expected to make heavy use of the list datatype and routines you created for assignment one. You may use the provided list implementation from the course web-site, however, this implementation is not guaranteed to be free from errors and is strictly use-at-your-own-risk. You are also expected to do all reasonable error checking.

For full marks, the scheduling algorithm you are to use is preemptive round robin with three levels of priority.

You will likely find it handy to have a "proc_message" field in your pcb. This field can contain a message that will be printed out the next time the process is scheduled. For example, when a sender is readied, you may want to put something like the following text

in that field: *reply received, text is: blah blah blah* This field should be displayed and cleared the next time the process runs.

There may be details which will come to mind as you start to think about this assignment. Please feel free to bring them during office hours or in the discussion forum.

Epilogue

Hopefully this assignment will give you a good understanding of the issues that arise when designing the core of an operating system kernel. Best of luck.