National University of Computer & Emerging Sciences (NUCES),

Islamabad Department of Computer Science

Semester Project

Course: Operating Systems **Semester: Spring 2018**
Project Title: Kernel Simulator **Due Date: 29-April-2018**

Submission Guidelines: Submit your code using Slate account within due date/time only.

Allowed No. Of Students in group: One or Max Two

Project Objective:

The main objective of this project is to simulate process management module of OS kernel. We will achieve this using our existing knowledge that we have learned in class as well as in labs. For this purpose Linux system calls such as fork ( ) , pipe( ), wait( ), signal( ) and alarm( ) can be used.

Project Statement:

You have to simulate following functions of Process management module:

- Process Creation
- Provide Environment to simulate execution of created processes.
- Simulate exec ( ) system call. i.e. replace the current process instructions and related data with new process and file from disk.
- Process State transition
- Process Scheduling (Round Robin Only).
- Context Switching.
- Mechanism for Inter-Process communication and process synchronization.

Explanation:

To simulate above mentioned functions our project will consist of three types of processes:

1. In_charge Process

   This will be your main process that will start the simulation. It will create a process using fork() named as " controller process" and communicate with it using pipe().

2. Controller process

This process will act according to commands sent by in_charge Process. This is core process to handle process management functions, like creating processes, scheduling , context switching etc.

3. Broad caster process

This process will be created by controller process and will be responsible to show/Display current status of simulation.

In charge Process:

Our simulation will start by executing the in charge process. This process will create a pipe and then fork ().The child process will be known as "Controller process" in this project onwards. There will be only one in charge process and one controller process throughout the simulation.

We can only interact with in charge program though predefined set of commands and it will repeatedly reads commands from terminal after every 2 seconds of time. You have to implements this using alarm signal (sleep is not allowed). The responsibility of this process is to pass on commands to controller process through pipe. Whenever in charge process sends a command it will generate a signal for controller process and then controller will read from pipe, add it to commands queue and execute the next instruction in the pipeline.

Following types of commands can be passed to in charge process:

Command	action
CRT <*filename*>	Create a simulated process and load instructions from *filename***.** Add it to process queue.
UNB	Unblock the first process placed in blocked queue and move it to ready queue.
INC	Increment 1 time unit. Execute 1 instruction of running process in queue.
PRT	Print the current state of the system (for deatails see Broad caster process section)
END	Print the average turnaround time and terminate the system.END command appears exactly once, being the last command.

Controller process:

This process will act according to commands sent by in_charge Process. This is core process to handle process management functions, like creating processes, scheduling, context switching etc .The controller process will maintain three process queues, named as Running, Blocked and Ready queue. All process transitions /context switching will be moving the process state/object between these three queues. Controller will also define some data structure to store relevant information of process like its PID, IR, PC,STATe,start time and cpu time used so far, Instruction set, and a Integer value.

The controller process will also have data structure for Time, Cpu.Time is an integer variable initialized to zero. Cpu is used to simulate the execution of a simulated process that is in running state. It should include data members to store a pointer to the program array, current program counter value, integer value, and time slice of that simulated process. In addition, it should store the number of time units used so far in the current time slice.

1.Processing input commands

After creating the first process and initializing all its data structures, the process manager

repeatedly receives and processes one command at a time from the Incharge process

(read via the pipe). On receiving a INC command, the controller process executes the next

instruction of the currently running simulated process, increments program counter value , increments Time, and then performs round robin scheduling with time slice 3 . Note

that scheduling may involve performing context switching.

On receiving a UNB command, the controller process moves the first simulated process in the

blocked queue to the ready state queue array. On receiving a PRT command, the controller process spawns a new Broadcaster process. On receiving a END command, the controller first spawns a braodcaster process and then terminates after termination of the broadcaster process. The process manager ensures that no more than one broadcaster process is running at any moment.

2.Executing simulated processes

The controller process executes the next instruction of the currently running simulated

process on receiving a INC command from the in charge process.

Instructions S, A and D update the integer value stored in PCB of the process. Instruction B moves the currently running simulated process to the blocked state and moves a process from the ready state to the running state. This will result in a context switch. Instruction E terminates the currently running simulated process, frees up all memory (e.g. program array) associated with that process and updates the PcbTable. A simulated process from the ready state is moved to running state. This also results in a context switch. .

Finally, the R instruction results in replacing the process image of the currently running simulated process. Its program array is overwritten by the code in file filename, program counter value is set to 0, and integer value is undefined. Process id, parent process id, start time, CPU time used so far, and state remain unchanged.

3.Scheduling

The process manager also implements a scheduling policy. Round robin with time slice 3 will be used for all simulated processes.

4.Context Switching

Context switching involves copying the state of the currently running simulated process from Cpu to PcbTable(unless this process has completed its execution), and copying the state of the newly scheduled simulated process from PcbTable to Cpu.

Broad caster Process

The Broad caster process prints the current state of the system on the standard output and then terminates. The output should appears as follows:

```
****************************************************************
The current system state is as follows:
****************************************************************\\
CURRENT TIME: time
RUNNING PROCESS:
pid, ppid, value, start time, CPU time used so far
BLOCKED PROCESSES:
```

Queue of blocked processes:

pid, ppid,, value, start time, CPU time used so far

…

pid, ppid,, value, start time, CPU time used so far

PROCESSES READY TO EXECUTE:

Queue of processes:

pid, ppid, value, start time, CPU time used so far

pid, ppid, value, start time, CPU time used so far

…

…

TOTAL TIME PASSED: ……

Simulated Processes:

These are thes process created by controller process upon reciving CRT command from incharge process.you donot need to fork() for simulated process.only create a new object for a process and load file containing instruction code.

Controller process manages the execution of simulated processes. Each simulated process is comprised of a program (input from file) that manipulates (sets/updates) the value of a single integer variable. Thus the state of a simulated process at any instant is comprised of the value of its integer variable and the value of its program counter. A simulated process' program consists of a sequence of instructions. There are seven types of

instructions as follows:

1. S n: Set the value of the integer variable to n, where nis an integer.
2. A n: Add nto the value of the integer variable, where nis an integer.
3. D n: Subtract nfrom the value of the integer variable, where nis an integer.
4. B: Block this simulated process.
5. E: Terminate this simulated process.

6. R filename: Replace the program of the simulated process with the program in the file filename, and set program counter to the first instruction of this new program.

An example of a program for a simulated is as follows:

```
S 1000
A 19
A 20
D 53
A 55
R file_a
A 21
A 0
D 3
```

You may store the program of a simulated process in an array, with one array entry for each instruction.