# CS 470 - Operating Systems
# Spring 2019 - Process Management Project
**40 points**
**Out: March 4, 2019**
**Due: March 27, 2019 (Wednesday, week and a half after spring break)**

The purpose of this project is to gain an appreciation of the work involved in managing processes. The assignment is to write a simple process scheduling simulation.

## Problem Statement

The program should simulate the process state structure shown in Figure 3.2 on page 108 of the textbook and also shown in the Lecture 16 slides (Process Scheduling). Input to drive the simulation must be read in from a file, and output must be written to a file. The format of the input file will be discussed below.

In the simulation, each process is identified by a unique process ID (PID). For our purposes, the process control block (PCB) of each process will keep track of at least the parent of each process, all of the children of each process, and the remaining burst time for the process (i.e., what is left of the total amount of time requested), and the remaining quantum time for the current process' quantum.

All processes are pre-emptible. When any interrupt occurs, the running process is "pre-empted," and its remaining burst time and remaining quantum time are decremented by 1. When the remaining burst time of a process reaches 0, it terminates. **Whenever a process terminates, all its children also must be terminated**. I.e., this simulation has cascading termination semantics, which is different than the UNIX termination semantics. After an interrupt, if the current process is not continuing, it is placed on the end of the Ready Queue, and the first process on the Ready Queue becomes the running process with a new time quantum. If there are no processes on the Ready Queue, assume that an "init" process with PID 0 runs, but this process is never put on the Ready queue. (Thus when the simulation first begins, the only thing in the system is the init process 0, and it is the parent of the first "real" process.)

Input consists of a series of actions . The actions are:

| | |
|---|---|
| **C n b** | Create a process with PID *n* with initial burst size *b* and add it to the Ready Queue. The parent is the current running process. |
| **D n** | Destroy the process with PID *n* and any children processes that it has. Only the parent of process *n* or process *n* itself can destroy process *n*. Other requests are ignored. You may assume that there will not be any requests to destroy PID 0. |
| **I** | Timer interrupt. This represents the system clock tick. |
| **W n** | Current running process has initiated a wait for event with event ID (EID) *n* and moves to the Wait Queue. |
| **E n** | Event with EID *n* has taken place. The process waiting for this event moves to the Ready Queue. |
| **X** | Exit the program. The program should output the current state of the simulation before exiting. |

For the purposes of this simulation, each action acts as a timer interrupt and consumes one unit of time. (I.e., **all** actions interrupt the current running process and consume one unit of time for the current running process.) Note that actions C, D, I, and W apply to the current running process. For example, for action C, a child process is

created with the current running process as its parent and then the current running process' remaining burst time is decremented by 1 (as is its remaining quantum time). If the current running processes' remaining quantum time becomes 0, it is moved back to the Ready Queue as well. For actions, D, I and W, if there is no current running process (i.e. PID 0 is running), the action is ignored. For action E, if there is no process waiting for the EID event, then the action is ignored.

Output should consist of a logfile of actions performed (i.e., an echo of the input and any state transitions that processes make) and the state of the Ready Queue and Wait Queue as shown in the example run below. The simulator must output enough logging information so that the instructor can follow the execution of processes in the simulation. An example is given below.

## Assignment

(20 points) **This project may be done individually or in pairs**. Each student/pair must implement a simulation meeting the specifications above for the RR scheduling algorithm for a quantum size **that can be set for each run of the program**.

The simulator may be written in any language for either Linux (projects must run on csserver) or Windows (projects using any IDE available in CS Lab are acceptable). However, the instructor can provide assistance only for C/C++ projects, and maybe Java projects. Provide a makefile that will make your project if it needs to be compiled. The file names and quantum size value may be provided to the simulation either as command-line arguments or interactively. Hardcoding the test file names or quantum sizes into the program is not acceptable.

(10 points) Provide a high-level discussion of the program describing the functionality of the major components of the program and how they interact with each other, including a more detailed discussion for the data structures and algorithms used in implementing the Ready and Wait Queues. If the program does not meet all of the project requirements, describe which parts are missing and what you think should be done to implement them.

(10 points) Students must provide the result files of running their simulation using the test files provided on csserver in `/home/hwang/cs470/project3` (to be released on or before March 11) for $q$= 1, 5, and 10. In addition, **each** student should answer the following questions (i.e., provide **two sets of answers** if doing the project in a pair):

1. Examine the test files and their resulting output. What can be said about the RR scheduling algorithm? What can you say about the effect of different quantum sizes?
2. What aspect of process management did you find most difficult to implement?
3. What aspect of process management did you find easiest to implement?
4. What, if anything, would you change in your current design?
5. What, if anything, did you find interesting or surprising about process management that you did not know before doing this project?

## Example Program Output

For RR ($q$=3) scheduling, you might have the following output log (shown in three columns to conserve space), where the lines in italics are the echoes of the input file lines, "PID *n b*" means process *n* with burst time *b* remaining, and "with *q* left" means *q* time left in the current quantum. In the Wait Queue, the third number is the event ID being waited upon. (The file `example.txt` will have this example in it.)

PID 0 running
Ready Queue:
Wait Queue:
*C 1 7*
PID 1 7 placed on Ready Queue
PID 1 7 running with 3 left
Ready Queue:
Wait Queue:
*I*
PID 1 6 running with 2 left
Ready Queue:
Wait Queue:
*C 2 3*
PID 2 3 placed on Ready Queue
PID 1 5 running with 1 left
Ready Queue: PID 2 3
Wait Queue:
*I*
PID 1 4 placed on Ready Queue
PID 2 3 running with 3 left
Ready Queue: PID 1 4
Wait Queue:
*W 4*
PID 2 2 placed on Wait Queue
PID 1 4 running with 3 left
Ready Queue:
Wait Queue: PID 2 2 4

*C 3 10*
PID 3 10 placed on Ready Queue
PID 1 3 running with 2 left
Ready Queue: PID 3 10
Wait Queue: PID 2 2 4
*I*
PID 1 2 running with 1 left
Ready Queue: PID 3 10
Wait Queue: PID 2 2 4
*E 4*
PID 1 1 placed on Ready Queue
PID 2 2 placed on Ready Queue
PID 3 10 running with 3 left
Ready Queue: PID 1 1 PID 2 2
Wait Queue:
*I*
PID 3 9 running with 2 left
Ready Queue: PID 1 1 PID 2 2
Wait Queue:
*I*
PID 3 8 running with 1 left
Ready Queue: PID 1 1 PID 2 2
Wait Queue:
*I*
PID 3 7 placed on Ready Queue
PID 1 1 running with 3 left
Ready Queue: PID 2 2 PID 3 7
Wait Queue:

*I*
PID 1 0 terminated
PID 2 2 terminated
PID 3 7 terminated
PID 0 running
Ready Queue:
Wait Queue:
*C 5 5*
PID 5 5 placed on Ready Queue
PID 5 5 running with 3 left
Ready Queue:
Wait Queue:
*C 6 3*
PID 6 3 placed on Ready Queue
PID 5 4 running with 2 left
Ready Queue: PID 6 3
Wait Queue:
*W 7*
PID 5 3 placed on Wait Queue
PID 6 3 running with 3 left
Ready Queue:
Wait Queue: 5 3 7
*X*
Current state of simulation:
PID 6 3 running with 3 left
Ready Queue:
Wait Queue: PID 5 3 7

## What to Submit

Create a **tarfile** or **zipfile** archive containing the following items:

- The well-documented code source code for your project. If you are submitting a Windows project, submit the entire project folder.
- A makefile to make your project, if needed
- All output files from simulation runs (total of nine output files)
- **A single document in PDF format** containing:
  - instructions on how to build and/or run the program
  - the discussion of the functional design of your project
  - answers to the questions above.

Submit your archive using the submission system (http://submission.evansville.edu) on the due date (**to only one of a pair**). The grading script only will accept submissions. It will not run anything.