

CS471: Operating System Concepts
Fall 2009
Course Project
Due: November 3, 2009
Points: 100

You may work in teams of 1, 2, or 3 members.
Choose your own team members.

The project deals with three problems related to operating systems:

1. CPU Scheduling (50 points)
2. Process Synchronization (50 points)

The background material required to solve these problems has been covered in chapters 3-6 of the textbook. You can choose either Java or C++ as a programming language for the project. Solutions to each of these problems will be graded separately. You are required to submit the following by the deadline using the submit command in Unix ***submit cs471*** command. You will submit a single directory that has two subdirectories for each of the problems. The directory you submit will have the following structure:

CS471PROJECT (main dir)

----- **CPUSCHED** (*subdir 1*)

- README file with directions (make sure to include actual commands that can be copied and pasted to run)
- SOURCE code (well documented)
- Executable code
- Sample input data files
- Sample output file(s)

----- **SLEEPING-BARBER PROBLEM** (*subdir 2*)

- README file with directions (make sure to include actual commands that can be copied and pasted to run)
- SOURCE code (well documented)
- Executable code
- Sample input data files
- Sample output file(s)

Details of each of these problems are in the appendix.

Appendix

Problem 1: CPU Scheduling

This part of the projects simulates a CPU scheduler. Since it is a simulation, there are no real processes/threads to be scheduled. Instead, you simulate the arrival of new processes and threads. Whenever a new process/thread arrives (in simulation) into the ready queue, the CPU scheduler is invoked. Each simulated process (or thread) has the following parameters: <Process ID, Arrival time, Priority, CPU burst units>. Each CPU unit is equivalent to CPU time needed to execute the following loop:

```
for (int i=0, int temp =0; i < 10000; i++)  
    if (i mod 2 ==0) temp= i/2 + 1;  
    else temp=2*i;}
```

At the time of invocation of the scheduler, the user indicates the type of scheduling to be enforced. You are required to implement the following scheduling types:

1. FIFO
2. SJF with preemption
3. RR (with specified time quantum)
4. Priority with preemption

Each run will handle scheduling of 10,000 (simulated) processes. In other words, as soon as the number of processes that have completed CPU execution reaches 10,000, you can stop running the program and print the following statistics.

Hint: Since the processes to be scheduled are not real but simulated, you need a generator to generate the process requests to the CPU scheduler. The generator generates and stores the process requests in a file. Your scheduler program reads the file and simulates real process arrives. In order to control the process arrivals and the CPU burst time for each process, your generator requires two inputs: Average number of process arrivals per unit time and average number of burst units. You need to vary these two parameters and run different cases. A sample generator function is available for you under the project directory.

Statistics for the Run

Number of processes: 10,000

Total elapsed time (for the scheduler):

Throughput:

CPU utilization:

Average waiting time:

Average turnaround time:

Average response time:

All times in the statistics are measured in milliseconds (in decimal notation).

Problem 2: Producer-Consumer Problem

This follows the description in Exercise 6.40 (page 274) of 8th Edition. Develop the producer-consumer problem using Pthreads or Winn32 API. Test the program with several inputs (the three parameters to the main program are shown in page 275). Definitely, test the following parameters. Try with different sleep times. Measure the performance of the time in terms of overall turnaround time. Finally, summarize your results and give an explanation for the results.

Test case	Number of producers	Number of consumers
1	1	1
2	4	1
3	16	1
4	1	2
5	4	2
6	16	2
7	1	4
8	4	4
9	16	4
10	1	16
11	4	16
12	16	16
13	1	32
14	4	32
15	16	32