CS 323 Exercises
Week 3
08/03/2019

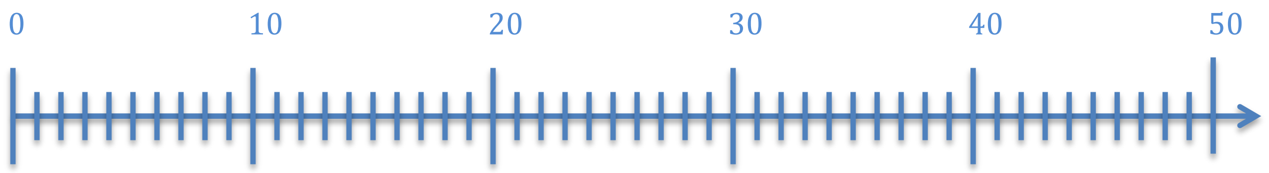
1) Scheduling Algorithms

Assume a preemptive scheduler that implements a combination of two round-robin schedulers, the first (the *short queue*) with a time quantum of 2 seconds, and the second (the *long queue*) with a time quantum of 10 seconds. A newly arriving process is put at the tail of the short queue. A process is allowed a maximum of 8 seconds of execution in the short queue, and then it is put in the long queue. Processes in the short queue have priority over processes in the long queue. If a process in the long queue has to give up the CPU during its quantum because of the arrival of a new process, it is put back at the *head* of the long queue. The next time it is executing it will be given what remained of its time quantum at the time it had to give up the CPU, i.e., if it had been executing for 3 seconds of its 10-second quantum, it would get 7 seconds the next time it gets to run.

Assume the machine is initially idle, and that you have the following process arrivals (all times in seconds):

Process identifier	Arrives at time	Length of execution
P0	1	20
P1	2	1
P2	5	5
P3	9	20
P4	25	3

Indicate on the timeline below which process will be running at which time.



2) Scheduling Algorithms

Assume that the machine is initially idle, and that you have the following process arrivals (all times in seconds):

Process identifier	Arrives at time	Length of execution
P0	1	20
P1	2	1
P2	5	5
P3	9	15
P4	25	3

All processes have equal priority.

A – Design a scheduler for which P0 ends before P1.

B – Design a scheduler for which P2 ends no later than at time 12 and P3 ends no later then at time 32.

C – Design a scheduler for which P2 ends no later than at time 10.

3) IPC and signals

From Wikipedia:
"Signals are a limited form of inter-process communication (IPC), typically used in Unix-like operating systems. A signal is an asynchronous notification sent to a process or to a specific thread within the same process in order to notify it of an event that occurred. When a signal is sent, the operating system interrupts the target process' normal flow of execution to deliver the signal. If the process has previously registered a signal handler, that routine is executed. Otherwise, the default signal handler is executed."

In the following code snippet explain what happens when all processes reach their steady state. Describe the process tree and identify the line of code each process has stopped. Feel free to make any assumptions for the values you can't know, e.g. PIDs.

Ideally, you should be able to solve this exercise without running the code. If you can't do so, describe the process and tools you used to solve the exercise.

Note that signals interrupt blocking system calls.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>


int n, pd[2];
char c;
pid_t p;

void Fn(int fd1, int fd2)
{ while(1); }

void h(int s)
{ write(pd[1], &p, 2); read(pd[0], &p, 2); Fn(pd[0], pd[1]); }

int main(void)
{
      pipe(pd);
      signal(SIGUSR1, h);
      n = pd[1];

      p = fork();
      if (p == 0) {
            pipe(pd);
            pd[1] = n;
            p = fork();
      } else
            p = 0;

      if (p)
            kill(p, SIGUSR1);

      read(pd[0], &c, 1);
      Fn(p, pd[0]);
      wait(NULL);
      Fn(4, 2);
      return 0;
}
```

4) Remote procedure calls

We have a server that implements a single remotely callable procedure

```
long Position( char c, char * s)
```

It takes as arguments a single character and a null-terminated character string, and returns the position of the first occurrence of the character in the string, or -1 if the character does not occur in the string.

Write pseudo-code for a client stub and a server stub for this remote procedure call. Describe clearly the format of the messages sent between client and server. You can assume the existence of helper functions to allocate and de-allocate memory for these messages.