Homework 2

due Tuesday, September 24, 2002, 12:01 AM

Your answers should be submitted as a plain text file `hw02.txt`, a postscript file `hw02.ps`, or a PDF file `hw02.pdf`.

1. You need only submit your answers to *3 of the 5 parts of this question.* You should make sure you can answer all of them, but you need only submit 3. (3 points)

   (a) MS-DOS provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system. (1 point)

   (b) A CPU-scheduling algorithm determines an order of execution of its scheduled processes. Given n processes to be scheduled on one processor, how many different schedules are possible? Give a formula in terms of n. (1 point)

   (c) Define the difference between preemptive and nonpreemptive scheduling. State why strict nonpreemptive scheduling is unlikely to be used in a computer center. (1 point)

   (d) What advantage is there in having different time-quantum sizes on different levels of a multilevel queueing system? (1 point)

   (e) Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs? (1 point)

2. Consider the following set of processes, with their CPU-burst times:

Process	Burst Time
P_1	10
P_2	1
P_3	2
P_4	1
P_5	5

   The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, and all at time 0. (2 points)

   (a) Draw three Gantt charts illustrating the execution of these processes using FCFS, SJF, and RR ($q = 1$) scheduling.

   (b) What are the waiting and turnaround times of each process for each of the scheduling algorithms in part (a)?

   (c) Which of the schedules in part (a) results in the minimal average waiting time (over all processes)?

3. Read the paper by Petrou, *et al.*, that is linked from the online version of this document, which describes a FreeBSD implementation of lottery scheduling. How you think this kind of scheduling would work for our FreeBSD lab, given the usage patterns you would expect there? How about a fair-share scheduler, as described by Tanenbaum on p. 148? What assumptions are you making about the usage patterns in the lab, and how might your answer change if those patterns changed? (5 points)

   Note: there is also a recent FreeBSD implementaion of fair-share scheduling. See `http://www.kerneltrap.org/node.php?id=349` for more information.

4. The Petrou paper also gives us some hints as to what parts of the FreeBSD kernel are concerned with CPU scheduling. The functions `setrunqueue()` and `cpu_switch()` are two important ones.

   The FreeBSD kernel source code is found under the `/sys` subdirectory on the systems in the lab. Since it's always nice to see how things are done in a real system, I'd like you to check out several key files.

   Traditionally, process state ("process control block") in Unix has been divided between two data structures: the *proc* structure and the *user* structure.

   The `proc` structure is located in `/sys/sys/proc.h`. The main definition is `struct proc`. You will notice several familiar (and even more unfamiliar) fields in this structure. Find several fields related to CPU scheduling. Do the same for the `user` structure in `/sys/sys/user.h`. Where are the actual CPU registers stored? Check out `/sys/i386/include/pcb.h` to see what registers are saved on the Intel 386 architecture.

   Another thing to notice are a few global variables. The global variable `curproc` always points to the `proc` structure for the currently executing process.

   Other related global variables:

   - `allproc`: list of all processes
   - `zombproc`: list of zombie processes
   - `initproc`: points to "init" process

   CPU scheduling-related code can be found in the following files:

   - `/sys/kern/kern_synch.c`:
     - `roundrobin()`
     - `schedcpu()`
     - `updatepri()`
     - `resetpriority()`
     - `mi_switch()`
   - `/sys/kern/kern_clock.c`:
     - `hardclock()`

  - `statclock()`
- `/sys/kern/kern_timeout.c`:
  - `softclock()`
- `/sys/i386/i386/swtch.s`:
  - `cpu_switch()` Assembly code!

4.3/4.4BSD uses a *multilevel feedback queue* scheduling algorithm:

- Runnable processes are assigned a *scheduling priority* (range 0-127) that determines in which *run queue* they are placed.
- Dispatcher (`cpu_switch()` in `/sys/i386/i386/swtch.s`) chooses first process on highest priority queue.
- Process is allowed to run in user mode for up to one *quantum* before it is preempted.
  - Default quantum: 100ms.
- Processes that use a lot of CPU have their priorities lowered.
- Processes blocked for a long time for I/O have their priorities raised, so that they will run soon after I/O is finished.

There is nothing to turn in for this part, but look over the code.