CS 4/53201 # Homework #2 **Operating Systems**

Due in class on Friday 20 October 2000

1. **(Exercise 6.1 from OSC) What is the meaning of the term busy waiting? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.**

   Busy-waiting refers to waiting in a loop, essentially doing nothing useful, until whatever is being waited on occurs (for example, until a semaphore variable changes value).

   A thread could also be waiting while blocked (e.g., while waiting on a semaphore), but in this situation it sits in a queue off to the side and does not consume CPU cycles while waiting.

   As the book points out, since disabling interrupts is not a good solution in a multiprocessor environment, some busy waiting is required in most implementations of semaphores. However, this is a small amount of busy waiting in comparison to waiting for a prolonged period in the critical section of a program.

2. **(Exercise 6.11a from OSC, slightly rewritten) Consider a bounded-buffer monitor in which the buffers are embedded within the monitor itself. The strict mutual exclusion within the monitor makes it mainly suitable for small buffers. Explain.**

   A monitor enforces strict mutual exclusion by allowing only one of the member functions at a time to access the data. In the case of data such as would be in a database, where it might be permissible for multiple readers to access the data simultaneously, the monitor would be overly restrictive. Furthermore, if the buffer is large, the access time might also be large, in which case each thread using the monitor might require access to one of its functions for an inordinately long amount of time.

3. **(Exercise 5.10 from OSC) Explain the differences in the degree to which the following scheduling algorithms discriminate in favor of short processes. (a) FCFS, (b) RR, and (c) multilevel feedback queues.**

   FCFS can allow a long process to keep the CPU for long time; it gives short processes no special priority at all.

   RR gives each process a time slice in turn, in effect giving short processes a chance to finish soon while delaying the execution of long processes. However, a short process may have to wait until every other process has a chance to run for a time slice before it gets a chance to run.

   Using multilevel feedback queues, if the short processes are in the higher priority queue (which is likely) they will run first, possibly even non-preemptively. This algorithm is the best for short processes.

4. **(counts double) Suppose there are five processes in the ready list with expected CPU burst times as follows (P0 is at the head of the list and P4 is at the tail): P0 350, P1 125, P2, 475, P3 250, P4 75. Draw a Gantt chart showing the resulting schedule for a round robin scheduler with a time slice of 100, give the wait time for each process, and compute the average wait time.**

Gantt chart should show:
P0 for 100, P1 for 100, P2 for 100, P3 for 100, P4 for 75
P0 for 100, P1 for 25, P2 for 100, P3 for 100
P0 for 100, P2 for 100, P3 for 50
P0 for 50, P2 for 100, P2 for 75

P0 waits for 0 + (475–100) + (800–575) + (1050–900) = 750
P1 waits for 100 + (575–200) = 475
P2 waits for 200 + (600–300) + (900–700) + (1100–1000) = 800
P3 waits for 300 + (700–400) + (1000–800) = 800
P4 waits for 400

Average wait time = (750 + 475 + 800 + 800 + 400) / 5 = 645