# Operating Systems, Spring 2014

## Midterm

2:10pm $\sim$ 3:50pm, Tuesday, April 22, 2014

---

**INSTRUCTIONS:**

1. This is a *closed-book* exam.
2. Try to solve all of the problems.
3. Try to give short answers. (Hint: An answer need not always be longer than the question.)
4. No cheating.
5. Please hand in both the exam sheet and the answer sheet.
6. Please note that unless otherwise stated, all the line numbers for the program listings are for reference only.

---

1. (20%) Consider the following preemptive priority-scheduling algorithm based on dynamically changing priority. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate $\alpha_1$; when it is running, its priority changes at a rate $\alpha_2$. All processes are given a priority of 0 when they enter the ready queue. The parameters $\alpha_1$ and $\alpha_2$ can be set to give many different scheduling algorithms.

   (a) What is the algorithm that results from $\alpha_1 < \alpha_2 < 0$?

   (b) What is the algorithm that results from $\alpha_2 > \alpha_1 > 0$?

2. (20%) Suppose that two processes, $P_1$ and $P_2$, are running in a uniprocessor system. $P_1$ has two threads. $P_2$ has three threads. All threads in both processes are CPU-intensive; that is, they never block for I/O. The operating system uses simple round-robin scheduling.

   (a) Suppose that all of the threads are user-level threads, and that user-level threads are implemented using a single kernel thread per process. What percentage of the processor's time will be spent running $P_1$'s threads?

   (b) Suppose instead that all of the threads are kernel threads. What percentage of the processor's time will be spent running $P_1$'s threads?

3. (15%) For the `fork` system call on Unix or its variants, answer the following questions.

   (a) What is the return value of the `fork` system call in the parent process on success?

   (b) What is the return value of the `fork` system call in the child process on success?

   (c) What is the return value of the `fork` system call in the parent process on failure?

4. (15%) Measurements of a certain system have shown that the average process runs for a time $T$ before blocking on I/O. A process switch requires a time $S$, which is effectively wasted (overhead). For round-robin scheduling with quantum $Q$, give a formula for the CPU efficiency (i.e., the useful CPU time divided by the total CPU time) for each of the following:

   (a) $Q > T$

   (b) $S < Q < T$

   (c) $Q = S$

   *To simplify the answers, you may assume $Q$ divides $T$ evenly.*

5. (15%) Consider the interprocess-communication scheme where mailboxes are used. Suppose a process $P$ wants to wait for two messages, one from mailbox $A$ and one from mailbox $B$. What sequence of `send` and `receive` should it execute so that the messages can be received in any order?

6. (15%) What would be the output of the following C program that uses the Pthreads API? (*Note that the line numbers are for references only.*)

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <unistd.h>
 4 #include <pthread.h>
 5 #include <sys/types.h>
 6 #include <sys/wait.h>
 7
 8 static void *runner(void *param)
 9 {
10     (* (int*) param)--;
11     pthread_exit(0);
12 }
13
14 int main(int argc, char **argv)
15 {
16     int status;
17     int value = 101;
18     pid_t pid = fork();
19     if (pid > 0) {
20         printf("A = %d\n", --value);
21         waitpid(-1, &status, 0);
22     }
23     else if (pid == 0) {
24         pid_t pid = fork();
25         if (pid > 0) {
26             printf("B = %d\n", --value);
27             waitpid(-1, &status, 0);
28         }
29         else if (pid == 0) {
30             pid_t pid = fork();
31             pthread_t tid;
32             pthread_attr_t attr;
33             pthread_attr_init(&attr);
34             pthread_create(&tid, &attr, runner, &value);
35             pthread_join(tid, NULL);
36             if (pid > 0) {
37                 printf("C = %d\n", --value);
38                 waitpid(-1, &status, 0);
39             }
40             else
41                 printf("D = %d\n", --value);
42         }
43         else {
44             return 1;
45         }
46     }
47     else {
48         return 1;
49     }
50     return 0;
51 }
```