Operating Systems, Spring 2009

Midterm

2:10pm $\sim$ 4:10pm, Tuesday, April 21, 2009

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. (10%) A small computer has four page frames. At the first clock tick, the R bits are 0111 (page 0 is 0, the rest are 1). At subsequent clock ticks, the values are 1011, 1010, 1101, 0010, 1010, 1100, 0001, 0101, 1011, and 1001. If the aging algorithm is used with a 4-bit counter, give the values of the four counters after the last ticks.

2. (10%) Assume a page reference string for a process with m frames (initially all empty). The page reference string has length p with n distinct page numbers occurring in it. For any page-replacement algorithms,

   (a) What is a lower bound on the number of page faults?

   (b) What is an upper bound on the number of page faults?

3. (20%) Consider the two-dimensional array `A`:

   ```
   int A[][] = new int[100][100];
   ```

   where each integer occupies 4 bytes and `A[0][0]` is at location 200, in a paged system with pages of size 200 bytes. A small process is in page 0 (locations 0 to 199) for manipulating the matrix; thus, every instruction fetch will be from page 0. For three page frames, how many page faults are generated by the following array initialization loops, using LRU replacement, and assuming page frame 1 has the process in it, and the other two are initially empty:

   (a)
   ```
   for (int i = 0; i < 100; i++)
       for (int j = 0; j < 100; j++)
           A[i][j] = 0;
   ```

   (b)
   ```
   for (int j = 0; j < 100; j++)
       for (int i = 0; i < 100; i++)
           A[i][j] = 0;
   ```

   Hint: Make sure you take into account how arrays are stored in memory.

4. (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 without from being blocked by each other?

5. (15%) Consider the following C program that uses the Pthreads API. What would be the output of the program?

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

int value = 5;

static void *runner(void *param);

int main(int argc, char **argv)
{
    pid_t pid = fork();
    if (pid > 0) {
        printf("A = %d\n", ++value);
    }
    else if (pid == 0) {
        pid_t pid = fork();
        if (pid > 0) {
            printf("B = %d\n", ++value);
        }
        else if (pid == 0) {
            pid_t pid = fork();
            pthread_t tid;
            pthread_attr_t attr;
            pthread_attr_init(&attr);
            pthread_create(&tid, &attr, runner, NULL);
            pthread_join(tid, NULL);
            if (pid > 0)
                printf("C = %d\n", ++value);
            else
                printf("D = %d\n", ++value);
        }
        else {
            exit(1);
        }
    }
    else {
        exit(1);
    }
    return 0;
}

static void *runner(void *param)
{
    value++;
    pthread_exit(0);
}
```

6. (10%) Suppose that two processes, P_1 and P_2, are running in a uniprocessor system. P_1 has three threads. P_2 has two 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?

7. (20%) Suppose that a disk drive has 1000 cylinders, numbered from 0 to 999. The drive is currently serving a request at cylinder 200, and the previous request was at cylinder 125. The queue of pending

requests, in FIFO order, is

50, 500, 250, 800, 350, 550, 400, 600, 100.

Starting from the current head position, what is the *total distance* (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk scheduling algorithms?

(a) SSTF

(b) SCAN

(c) LOOK

(d) C-SCAN

(e) C-LOOK