# EECE.4810/EECE.5730: Operating Systems

Spring 2017

Midterm Exam
March 8, 2017

**Name:** _____

**Section:** **EECE.4810 (undergraduate)** **EECE.5730 (graduate)**

For this exam, you may use two 8.5" x 11" double-sided page of notes. All electronic devices (e.g., cell phones, calculators, laptops) are prohibited. If you have a cell phone, please turn it off prior to the start of the exam to avoid distracting other students.

The exam contains 6 questions for a total of 100 points. Please answer the questions in the spaces provided. If you need additional space, use the back of the page on which the question is written and clearly indicate that you have done so.

Please note that students enrolled in EECE.5730 must complete three extra problems, which are worth a total of 20 points:

- Question 1d, on page 4
- Question 3c, on page 7
- Question 6d, on page 15

You will have three hours to complete this exam.

| | |
|---|---|
| Q1: Process management | / 19 + 6 |
| Q2: Inter-process communication | / 10 |
| Q3: Multithreading | / 11 + 6 |
| Q4: Synchronization | / 21 |
| Q5: Scheduling | / 20 |
| Q6: Memory management | / 19 + 8 |
| **TOTAL SCORE** | / 100 + 20 |

1. (19 + **6** points) ***Process management***
Parts (a) and (b) of this problem refer to the following program:

```
int main() {
   pid_t pid1, pid2;

   pid1 = fork();
   if (pid1 == 0) {
      pid2 = fork();
      if (pid2 > 0) {
         wait(NULL);
         printf("1\n");
      }
      else
         execlp("/bin/ls", "ls", NULL);
   }
   else if (pid1 > 0) {
      printf("2\n");
   }
   return 0;
}
```

a. (9 points) How many unique processes does this program create, including the initial process? Draw a process tree to support your answer.

1 (continued)
Program to which part (b) refers:

```
int main() {
   pid_t pid1, pid2;

   pid1 = fork();
   if (pid1 == 0) {
      pid2 = fork();
      if (pid2 > 0) {
         wait(NULL);
         printf("1\n");
      }
      else
         execlp("/bin/ls", "ls", NULL);
   }
   else if (pid1 > 0) {
      printf("2\n");
   }
   return 0;
}
```

b. (5 points) Will any of the created processes become zombie processes after their termination? If so, explain which one(s) and why; if not, explain why not.

1 (continued)

c. (5 points) How does the operating system differentiate between processes in the ready and waiting states?

d. (6 points, ***EECE.5730 only***) Suppose a single parent process creates five different child processes. When each child terminates, the parent process should handle each one in a different, specific way. Describe how the parent process can determine which child terminates so it can handle each one appropriately.

2. (10 points) ***Inter-process communication***

a. (6 points) Describe how the two major classes of inter-process communication use memory. In your answer, describe (i) what types of data or data structures are used to exchange information, and (ii) in which address space these data are stored.

b. (4 points) Explain why indirect communication is often favored over direct communication for message passing inter-process communication.

3. (11 **+ 6** points) ***Multithreading***

a. (6 points) Explain why each thread in a multithreaded process needs its own (i) program counter, (ii) register copies, and (iii) stack.

b. (5 points) Given the two threads below, is it possible for x to be equal to 4 and y to be equal to 2 when both threads are done? If so, explain how; if not, explain why not.

| Thread 1 | Thread 2 |
|---|---|
| x = 4 | y = 5 |
| x = y + 2 | y = x - 2 |

3 (continued)

c. (6 points, ***EECE.5730 only***) Explain how (i) two threads in the same process could easily share data, and (ii) how each of those two threads can protect its data from the other thread. Your answer should not require any special mechanisms for sharing—placing the data appropriately should be sufficient.

4. (21 points) ***Synchronization***

a. (5 points) Explain what busy waiting is and how it is typically avoided when processes need to wait for access to a synchronization primitive (lock, condition variable, etc.)

4 (continued)

b. (8 points) The example code below shows a pair of processes (one reader, one writer) that use semaphores to allow (i) multiple reader processes to run simultaneously, while also ensuring (ii) the writer process can only enter its critical section if no readers are executing. Assuming the two semaphores `rwSem` and `rcSem` are initialized to 1, and `rdCnt` is initialized to 0, explain how the two semaphores enable conditions (i) and (ii) above.

**Reader process**

```
do {
   down(rcSem);
   rdCnt++;
   if (rdCnt == 1)
      down(rwSem);
   up(rcSem);

   /* read shared data */

   down(rcSem);
   rdCnt--;

   if (read_count == 0)
      up(rwSem);
   up(rcSem);
} while (true);
```

**Writer process**

```
do {
   down(rwSem);

   /* write shared data

   up(rw_mutex);

} while (true);
```

4 (continued)

c. (8 points) Consider a multithreaded bank software package in which the following function is used to transfer money. Assume that the program contains a global array of locks, `locks[]`, with one entry per account number, such that `locks[i]` is the lock for account number `i`.

```
void transfer_money(int src_acct, int dest_acct, int amt) {
     locks[src_acct].lock();   // Lock account sending money
     locks[dest_acct].lock();  // Lock account receiving money
     <transfer money>
     locks[dest_acct].unlock();
     locks[src_acct].unlock();
}
```

Explain how this function can deadlock if called in multiple threads, and rewrite (in pseudo-code or actual code) the function to remove the deadlock condition.

5. (20 points) ***Scheduling***

Consider the following set of processes, with the length of the CPU burst time given in milliseconds. Processes may begin executing 1 ms after they arrive (i.e., a process arriving at time 5 could start executing at time 6). Any process arriving at time 0 is in the ready queue when the scheduler makes a decision about the first process to run.

| Process | Burst | Priority | Arrival time |
|---|---|---|---|
| P1 | 10 | 1 | 0 |
| P2 | 3 | 4 | 0 |
| P3 | 7 | 2 | 2 |
| P4 | 1 | 2 | 4 |
| P5 | 5 | 3 | 6 |

Determine the turnaround time for each process using each of the following scheduling algorithms: (i) first-come, first-serve (FCFS), (ii) shortest job first (SJF), (iii) shortest time to completion first (STCF), (iv) round-robin (RR) with time quantum = 1 ms, and (v) a non-preemptive priority scheme in which lower numbers indicate higher priority.

To break ties between processes (same burst time/priority), use first-come, first-serve ordering.

You may use space on both this page and the following page to solve this problem. The list of processes is copied on the next page for your reference.

5 (continued)
List of processes:

| Process | Burst | Priority | Arrival time |
|---|---|---|---|
| P1 | 10 | 1 | 0 |
| P2 | 3 | 4 | 0 |
| P3 | 7 | 2 | 2 |
| P4 | 1 | 2 | 4 |
| P5 | 5 | 3 | 6 |

6. (19 points) ***Memory management***

a. (4 points) Describe one benefit of base and bounds memory management over segmentation, and one benefit of segmentation over base and bounds.

b. (5 points) Say the currently running process has 16 active pages, P0-P15, all of which have their reference bits set to 1. If the operating system uses the clock algorithm for page replacement, the pages are ordered numerically around the "clock" (P0 is first, P1 is second, etc.), and the "clock hand" currently points to P6, which page will be replaced if a frame is needed to bring in a new page? **Explain your answer for full credit.**

6 (continued)

c. (10 points) A portion of the currently running process's page table is shown below:

| Virtual page # | Valid bit | Reference bit | Dirty bit | Frame # |
|---|---|---|---|---|
| 7 | 1 | 1 | 1 | 3 |
| 8 | 0 | 0 | 0 | -- |
| 9 | 1 | 0 | 1 | 4 |
| 10 | 0 | 0 | 0 | -- |
| 11 | 1 | 1 | 0 | 0 |
| 12 | 1 | 1 | 0 | 1 |

Assume the system uses 20-bit addresses and 16 KB pages. The process accesses four addresses: 0x25FEE, 0x2B149, 0x30ABC, and 0x3170F.

Determine (i) which address would cause a page to be evicted if there were no free physical frames, (ii) which one would mark a previously clean page as modified, if the access were a write, and (iii) which one accesses a page that has not been referenced for many cycles. **For full credit, show all work.**

6 (continued)

d. (8 points, ***EECE.5730 only***) Consider a system with 48-bit virtual addresses and 32-bit physical addresses. The system uses 4 KB pages. Assume each page table entry requires 4 bytes to store. Also, assume the currently running process is using only 16 KB of physical memory—the lowest-addressed 8 KB of its virtual address space, and the highest-addressed 8 KB of its virtual address space.

If this process uses a two-level page table, and the first- and second-level page tables each use the same number of entries, how much space will that page table require given all of the information above? **Show all of your work for full credit.**