CSC 139 Operating System Principles
Homework 1

Fall 2021

Posted on October 5, due on October 17 (11:59 pm). Write your own answers. Late submission will be penalized (turn in whatever you have).

Exercise 1. (10%) Consider the following piece of code:

```
int child = fork();
int c = 5;
if (child == 0) {
   c += 5;
} else {
   child = fork();
   c += 10;
   if (child) {
      c += 10;
   }
}
```

1. How many copies of the variable `c` are created by this program? What are their values?
2. Describe the hierarchical process tree that is created from running this program. You can assume that all processes have not yet exited.

Exercise 2. (5%) Consider an environment in which there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads within a process to issue blocking system calls while other threads continue to run. Explain why this model can make multi-threaded programs run faster than their single-threaded counterparts on a uniprocessor computer.

Exercise 3. (5%) Explain why system calls are needed to set up shared memory between two processes. Does sharing memory between multiple threads of the same process also require system calls to be set up?

Exercise 4. (5%) Describe the similarities and differences of doing a context switch between two processes as compared to doing a context switch between two threads in the same process.

Exercise 5. (OSC 3.13) (10%) Using the program below, identify the values of pid at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
pid_t pid, pid1;
    /* fork a child process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (pid == 0) { /* child process */
        pid1 = getpid();
        printf("child: pid = %d",pid); /* A */
        printf("child: pid1 = %d",pid1); /* B */
    }
    else { /* parent process */
        pid1 = getpid();
        printf("parent: pid = %d",pid); /* C */
        printf("parent: pid1 = %d",pid1); /* D */
        wait(NULL);
    }
    return 0;
}
```

Exercise 6. (OSC 3.16) (10%) Using the program shown below, explain what the output will be at lines X and Y.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define SIZE 5
int nums[SIZE] = {0,1,2,3,4};
int main()
{
int i;
pid_t pid;
    pid = fork();
    if (pid == 0) {
        for (i = 0; i < SIZE; i++) {
            nums[i] *= -i;
            printf("CHILD: %d ",nums[i]); /* LINE X */
        }
```

```
    }
    else if (pid > 0) {
        wait(NULL);
        for (i = 0; i < SIZE; i++)
            printf("PARENT: %d ",nums[i]); /* LINE Y */
    }
    return 0;
}
```

Exercise 7. (OSC 4.17) (10%) Consider the following code segment:

```
pid_t pid;
pid = fork();
if (pid == 0) { /* child process */
    fork();
    thread_create( . . .);
}
fork();
```

Answer the following questions and justify your answers.

1. How many unique processes are created?
2. How many unique threads are created?

Exercise 8. (10%) Define turnaround time and waiting time. Give a mathematical equation for calculating both turnaround time and waiting time using a processes start time, S_i, finish time, F_i, and computation time C_i. The computation time, C_i, is the total time the process spends in the running state. Assume that the process starts immediately after its arrival.

Exercise 9. (20%) Five batch jobs A through E, arrive at a computer center at almost the same time. They have estimated running times of 11, 6, 2, 4, and 8 minutes. Their priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the average process turnaround time. Ignore context switching overhead.

- Round-Robin with time quantum = 1 minute
- Priority scheduling
- First come, first served (run in order 11, 6, 2, 4, 8)
- Shortest job first

For Round-Robin scheduling, assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For other scheduling algorithms, assume that only one job runs at a time, until it finishes. All jobs are completely CPU bound.

Exercise 10. (15%) Consider a variant of the Round-Robin algorithm where the entries in the ready queue are pointers to the PCBs.

1. What would be the effect of putting two pointers to the same process in the ready queue?
2. What would be the major advantage of this scheme?
3. How could you modify the basic Round-Robin algorithm to achieve the same effect without the duplicate pointers? (This is an open-ended question.)

Please complete the following survey questions:

1. How much time did you spend on this homework?
2. Rate the overall difficulty of this homework on a scale of 1 to 5 with 5 being the most difficult.
3. Provide your comments on this homework (e.g., amount of work, difficulty, relevance to the lectures, form of questions, etc.)