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:

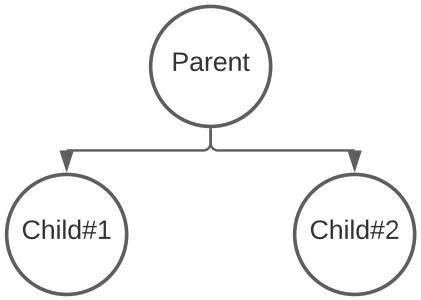
```
int child = fork();
int c = 5;
if (child == 0) {
    c += 5;
} else {
child = fork();
c += 10;
if (child) {
        c += 10;
    }
}
```

1. How many copies of the variable c are created by this program? What are their values?

   - 3 processes are created each having their own copy of variable c.
   - Process 0(Parent): c = 15 (goes up by 10 due to else statement but doesn't get second if)
   - Process 1(Child#1): c = 10 (Only goes up due to the first if statement)
   - Process 2(Child#2): c = 25 (runs through else and is a child so goes up by 20)

2. Describe the hierarchical process tree that is created from running this program. You can assume that all processes have not yet exited.

   - The hierarchical tree is one where the parent process creates a child by the first fork() and run through the first if statement. Then the parent will create another child through the fork() inside of the else statement as their child variable does not equal 0 but instead Child#1's pid.



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.

- This will allow the multi-threaded one to run faster as in the single threaded one when blocked everything stops. So, when a blocking call is done no other process can run even though they are available. Whereas with multi-thread when the blocking happens other programs still run. Making completion time faster for multi-threaded.

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?

- System calls are needed between two processes for shared memory due to the OS handling allocation space. This space is a shared zone that both processes use to read and write to memory.
- Yes, even though each thread of a process has their own stack they share memory. Especially for open files. Therefore, a system call for the allocation of memory is needed to make it work.

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.

Similarities	Differences
- Both hand control over to OS	- TCS: occur due to CPU

	- PCS: occur due to OS - TCS: helps handle multi-threading - TCS: faster - PCS: slower

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;
}
```

- Line A PID = 0 (due to successful fork)
- Line B PID = 2600
- Line C PID = 2600
- Line D PID = 2603 (actual value of child that was returned to parent)

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;
int 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;
}
```

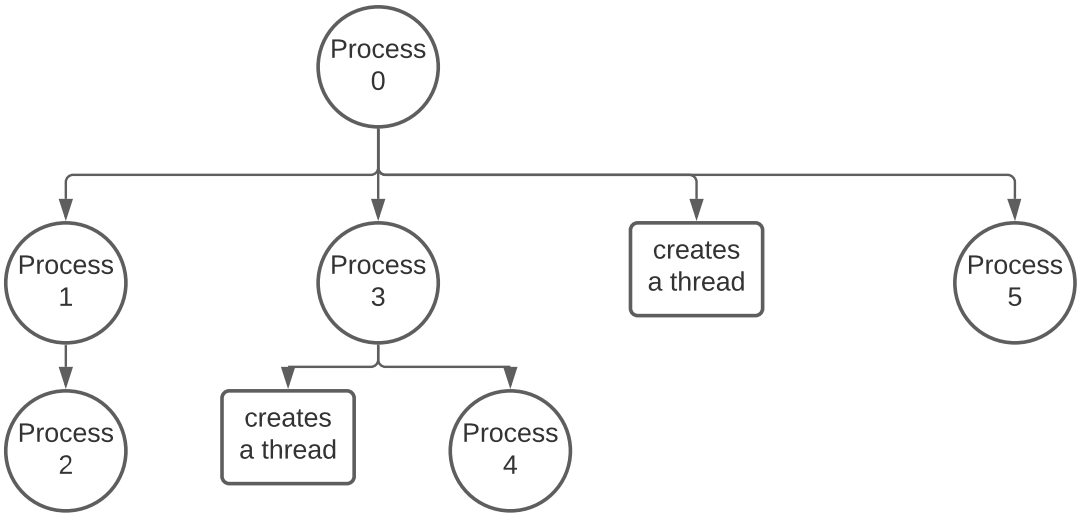
- Line X = CHILD: 0
  CHILD: -1
  CHILD: -4
  CHILD: -9
  CHILD: -16
- Line Y = PARENT: 0
  PARENT: 1
  PARENT: 2
  PARENT: 3
  PARENT: 4
- Child runs first while the parents for the completion of its child. These values come as child multiplies the value by itself and flips the sign. While the parent only prints the values as they are.

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

```
Int pid_t pid;
```

```
Int 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?

   - There are 5 unique processes created. The first one is created by the initial fork call. Second is created by the fork call at the end of the code for process 1. Third is created by the fork call withing the if statement. Fourth is created by the fork call at the end of the code for the process 3. Fifth is created by the last fork call at the end of the code of the Parent process 0.

2. How many unique threads are created?

   - There are 3 unique threads from this call. The first one is the original thread. The second one is created after the parent process forks in the if statement. When this happens the child process 3 creates a thread. The last happens when the parent process 0 creates a thread after the fork call in the if statement.

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.

- Turnaround time function= $F_i - S_i$
- Waiting time= $(F_i - S_i) - C_i$

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
    - Process order: 12345123451245124512512515151111
    - Turnaround time = (31+22+5+13+23)/5 = 18.8
- Priority scheduling
    - Process order: 25134
    - Turnaround time = (6+14+25+27+31)/5 = 20.6
- First come, first served (run in order 11, 6, 2, 4, 8)
    - Process order: 12345
    - Turnaround time = (11+17+19+23+31)/5 = 20.2
- Shortest job first
    - Process order: 34251
    - Turnaround time = (2+6+12+20+31)/5 = 14.2

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?

   - When there are two pointer to the same process it's the same as it taking up double the time or having the same process run twice in a succession. As each pointer will have its own time slot of the same round robin cycle.

2. What would be the major advantage of this scheme?

   - An advantage could be using this technique when you want processes to all finish at the same time and each processes time are off by multiples of 2. Another reason could be to have a system where they follow round robin but there is still a priority system that will allow high impact processes to finish sooner rather than later.

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.)

   - To do this you could have a value like priority that would be used to say how many times it should take over. Before it passes off to the next process. Let's say there is a time quantum of 1 and processes follow this chart:

	Burst time	Run Amount
P0	4	2
P1	2	1

   Then with my proposed system the process order would be: P0 P0 P1 P0 P0 P1. This is due to P0 having a run amount of 2 and P1 having a run amount of 1. Therefore with this system you would be able to keep the time quantum the same across processes but implement the ability to have multiple pointers increase the amount of CPU time a process takes per cycle.

Please complete the following survey questions:

1. How much time did you spend on this homework?

   - 3 hours

2. Rate the overall difficulty of this homework on a scale of 1 to 5 with 5 being the most difficult.

   - 3

3. Provide your comments on this homework (e.g., amount of work, difficulty, relevance to the lectures, form of questions, etc.)