Quinn Roemer

Dr. Yuan Cheng

CSC 139

15 October 2020

Exercise 1. 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?
    - This program creates two children each spawned from the main process. Therefore, there are **three copies of the variable "c"**, with **values of 10, 15, and 25,** at the end of execution.

2. Describe the hierarchical process tree that is created from running this program. You can assume that all process have not yet exited.
    - At the beginning of execution, the main process spawns a single child. This child goes on to execute the code inside the first "if" statement. The main process (or parent) executes the code inside the "else" statement. Inside this statement another child is spawned from the parent. The original parent goes on to execute the code inside the second "if" statement. The following tree is formed:

Main
/ \
Main **Child1**
/ \
Main Child2

Exercise 2. What is the purpose of system calls? How do system calls differ from procedure calls?

- System calls are designed to be an interface for certain OS services. They are generally accessed through an API and provide abstraction, portability, and encapsulation for common OS functions. In addition to this, they raise the HW privilege level to allow for execution in kernel mode. In contrast, procedure calls always occur in user mode and merely change the execution location of a program.

Exercise 3. 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 set up?

- System calls are necessary to set up shared memory between two processes because the memory of a process is generally off-limits to another process executing on the CPU. To remove this barrier, a system call executing in kernel mode is necessary to allow access to the same memory location from multiple processes. In contrast, system calls are not necessary to share memory between threads of the same process. This is because threads share the same address space, and therefore share a multitude of resources.

Exercise 4. 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.

- During a context switch the CPU must perform a save and load operation. This saves the state of a process and loads the state of another process. This state takes the form of a PCB (Process Control Block) for a process and a TCB (Thread Control Block) for a thread. Both data structures (when a context switch is performed) must be either saved or loaded. The difference between these two is the size of the data structure used. A TCB is smaller than a PCB, therefore allowing for quicker context switches with less overhead.

Exercise 5. Using the program below, identify the values of "pid" at lines A, B, C, D. (Assume that the actuals "pids" of the parent and child are 2600 and 2603.

```
#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;
}
```

- At line A: **pid = 0**
- At line B: **pid1 = 2603**
- At line C: **pid = 2603**
- At line D: **pid1 = 2600**

Exercise 6. 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;
}
```

- At line X, the values of the array will be printed after being multiplied by the negative index. This produces the following output:

  CHILD: 0 CHILD: -1 CHILD: -4, CHILD: -9 CHILD: -16

- At the line Y, the values of the array are printed out unmodified. This produces the following output:
  PARENT: 0 PARENT: 1 PARENT: 2 PARENT: 3 PARENT: 4

Exercise 7. Consider the following code segment:

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

1. How many unique processes are created?
    - **Five unique processes are created (not including the parent).** Initially, the parent splits off a single child, that child splits off to create a child of its own. Two threads are created, and then the parent and the two children each create another child, totaling six.

2. How many unique threads are created?
    - **Two unique threads are created.** The child process that enters the "if" statement creates a child before executing the "thread_create" statement.

Exercise 8. Define turnaround time and waiting time. Give a mathematical equation for calculating both turnaround time and waiting time using a process 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.

- **Turnaround time** is the amount of time it takes for a process to complete execution from submission.

  Turnaround time = $F_i - S_i$

- **Waiting time** is the total time a process spends waiting in the ready queue.

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

Exercise 9. 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 the context switch overhead.

- Round Robin with time quantum = 1 minute

A	B	C	D	E	A	B	C	D	E	A	B	D	E	A	B	D	E	A	B	E	A	B	E	A	E	A	E	A	A	A

Turnaround A: 31min
Turnaround B: 23min
Turnaround C: 8min **Average Turnaround: 21.4min**
Turnaround D: 17min
Turnaround E: 28min

- Priority Scheduling

Turnaround A: 25min
Turnaround B: 6min
Turnaround C: 27min **Average Turnaround: 20.6min**
Turnaround D: 31min
Turnaround E: 14min

- FCFS (Assuming order 11, 6, 2, 4, 8)

Turnaround A: 11min
Turnaround B: 17min
Turnaround C: 19min **Average Turnaround: 20.2min**
Turnaround D: 23min
Turnaround E: 31min

- SJF

Turnaround A: 31min
Turnaround B: 12min
Turnaround C: 2min **Average Turnaround: 14.2min**
Turnaround D: 6min
Turnaround E: 20min

Survey Questions:

1. How much time di you spend on this homework?
    - About 1 hour

2. Rate the overall difficulty of this homework on a scale of 1 to 5, with 5 being the most difficult.
    - 3 – Moderate

3. Provide your comments on this homework (amount of work, difficulty, relevance to the lectures, form of questions, etc.).
    - This homework was perfectly adequate. It was relevant to the lectures, asking questions about topics that were discussed in the lecture, allowing me to refer to my notes where my knowledge faltered. It was not too difficult, and I enjoyed the programming questions.