CS 471-001: Operating Systems
Spring 2017

Homework 1
Due date: Tuesday, February 21, 11:59 pm

Grading, submission and late policy:

- You are expected to complete this homework **on your own** (not with a partner or in a group)
- The homework accounts for **4 %** of your final grade
- Standard late policy applies - Late penalty for this homework will be 15% for each day. Submissions that are late by 3 days or more will not be accepted - You will submit your assignment via Blackboard

Part I: Multiple choice - circle the correct answer [20 points]

1. Which of the following cause(s) an interrupt?
   a. Division by zero
   b. System call
   c. I/O instruction
   d. All of the above

2. Which of the following is NOT part of an process's address space?
   a. Data segment
   b. Global variables
   c. Heap
   d. The value of the stack pointer

3. A process can leave the "Running" state when
   a. It adds zero to the contents of first general register
   b. Disk read I/O on which it was waiting completes
   c. It is dispatched by the scheduler for execution
   d. Timer expires

4. Which of the following gives an accurate access time ordering for the involved storage-device levels? (From slowest to the fastest)
   a. Magnetic Disk, RAM, Optical Disk, SSD
   b. Cache Memory, RAM, SSD, Optical Disk
   c. Optical disk, SSD, RAM, cache, registers

d. Registers, Optical Disk, RAM, Magnetic Disk

5. When a user process attempts to execute the TRAP instruction,
   a. This will cause a software interrupt
   b. The system will switch to kernel mode
   c. This means that the user process is about to invoke a system call
   d. All of the above

6. Which of the following is NOT shared between the threads of a process?
   a. Process's data segment
   b. Program counter
   c. Files open by the process
   d. Process's heap segment

7. Which of the following can happen when a user-level thread blocks?
   a. All other user level threads of that process can also block
   b. Run time procedure can schedule another user level thread of that process
   c. All of the above

8. Which of the following is NOT true?
   a. Kernel threads are faster to switch compared to user threads
   b. The operating system does not need to know about user threads
   c. User level threads can have any customized thread scheduling algorithm
   d. Blocking calls can be implemented with system calls in kernel threads

9. Which of the following is true? All the computers in a clustered system share
   a. All computers in a clustered system share main memory
   b. All computers in a distributed system share one disk storage
   c. All computers in a clustered system share a storage network
   d. All computers in a distributed system are identical (non-heterogeneous)

10. Assume that in Unix, a process Q creates a child process W. Then, Q exits (without executing a wait system call). In this scenario:
   a. Q becomes a zombie
   b. W will be eventually adopted by the "init" process
   c. W is immediately aborted by the operating system
   d. Q becomes an orphan

Part II: True or False - mark the following as True or False [20 points]

11. In symmetric multiprocessing, all CPUs will be executing the exact same code at all times

True False

12. DMA (Direct Memory Access) I/O results in faster data transfer compared to programmed I/O

True False

13. Device controller is an operating system software that interacts with the device

True False

14. Under certain conditions, a given process can migrate directly from one device queue to another one, by-passing the ready queue

True False

15. The short-term (CPU) scheduler determines the number of processes in the main memory

True False

16. Thread context switching (kernel level threads) is faster than process context switching

True False

17. Jacket code is primarily used to avoid kernel threads from calling blocking system calls

True False

18. Default POSIX behavior for fork() call within a thread is to duplicate all threads of the parent process in the child process

True False

19. Linux uses the term "task" when referring to either a process or a thread

True False

20. Thread cancellation can be used to terminate a thread when the task associated with the thread is no longer needed to be performed

True False

Part III: Answer the following questions

21. [5 points] What is the difference between concurrent and parallel execution? Is it possible to have concurrent and parallel execution on a single CPU system? Explain your answer.

Concurrent execution takes turns executing instructions for different tasks. Parallel execution is where more than one task's instructions execute simultaneously. Concurrent execution can take

place on a single CPU system because it alternates between tasks. If the time of execution is small enough then it may seem as though the tasks are in parallel but to have true parallel execution you need multiple CPUs working together.

22. [5 points] What are two differences between user-level threads and kernel-level threads? Under what circumstances is one type better than the other?

User-level threads: 1) managed by user-level library, 2) kernel only handles execution.
Kernel-level threads: 1) managed by the OS, 2) thread operations implemented in kernel code.

User-level threads are ideal when a task doesn't use blocking because thread management takes a procedural call. Kernel-level threads are slow because of management overhead and lack of being portable due to OS dependency.

23. [5 points] Briefly explain the function of the medium-term scheduler. How can it improve the system performance? Explain.

The medium-term scheduler manages swapping tasks in and out of memory. When there is limited memory the medium-term scheduler improves system performance by swapping out lower priority jobs while higher priority jobs are waiting in the ready queue.

24. [25 points] Consider the following C program. Assume that all fork() calls complete successfully.

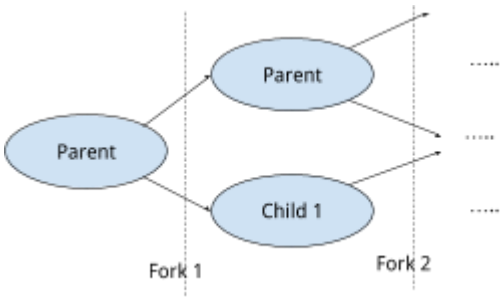
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int count = 0;
main()
{       int
i;
      for (i=0; i<3; i++)
      {
            fork();
count=count+i;
printf("Inside:%d\n", count);
//Inside Print Statement
      }
      printf("Outside:%d\n", count); //Outside Print Statement       exit(0);
}
```

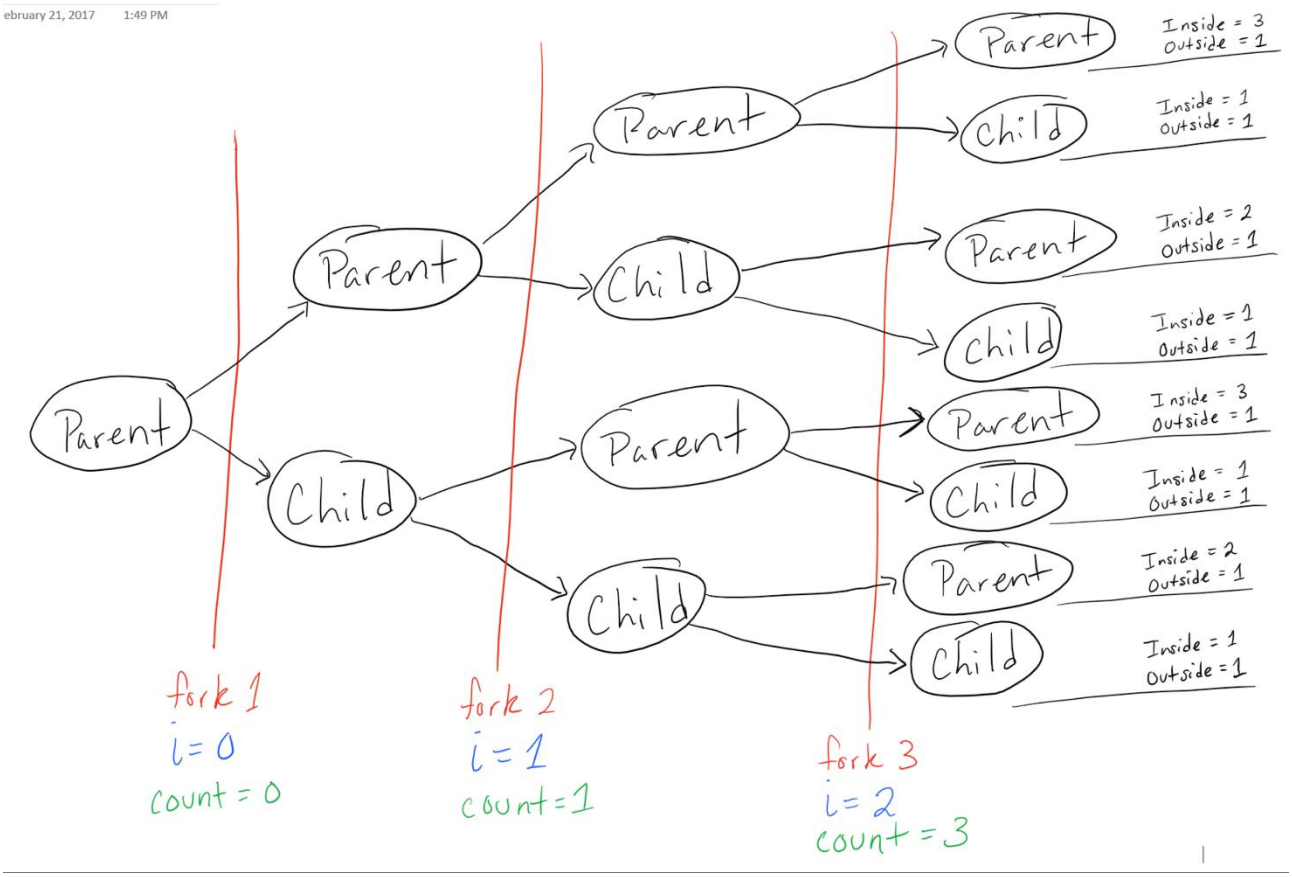
Answer the following questions-

a. How many processes are created during the execution of the program (including the "parent" process)?

8 processes.

b. Draw the process hierarchy tree for this program. An example process hierarchy tree is shown below.





c. How many times “Inside print statement” is executed in this program?

14 times.

d. For every process you showed in the process hierarchy tree, indicate the number of times it executes the “Inside print statement”.

Check tree for indications.

e. What are the minimum and maximum values of “count” printed by the execution of the “Inside print statement”?

Minimum value is 0, Maximum value is 3.

f. How many times “Outside print statement” is executed in this program?

8 times

g. For every process you showed in the process hierarchy tree, indicate the number of times it executes the “Outside print statement”.

Check tree for indications.

h. What are the minimum and maximum values of “count” printed by the execution of the “Outside print statement”?

Minimum value is 3, Maximum value is 3.

25. [20 points] Consider the following C program. Assume that all fork() calls complete successfully.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

int count=1;
int MAX=500;

void *runner(void *param) {
      int temp,i;
for(i=0; i<MAX; i++) {
```

```
        temp=count;
        temp++;
                count=temp;
        }
}

int main()
{
        pid_t pid_1;

        pid_1=fork();
if (pid_1==0) {
                pthread_t tid1; pthread_attr_t attr1;
pthread_attr_init(&attr1);
                pthread_create(&tid1,&attr1,runner,NULL);
                pthread_join(tid1,NULL);
                  printf("Child count=%d\n", count);              /* LINE CHILD COUNT */
        }
else {
                 wait(NULL);                                    /* LINE A */
                pthread_t tid2; pthread_attr_t attr2;
pthread_attr_init(&attr2);
                pthread_create(&tid2,&attr2,runner,NULL);
                pthread_join(tid2,NULL);
                  printf("Parent count=%d\n", count);           /* LINE PARENT COUNT */
        }
}
```

a. How many processes are created during the execution of the program (including the "parent" process)?

2 processes

b. How many threads are created during the execution of the program?

2 threads

c. What is value of "count" at LINE CHILD COUNT? Explain your answer.

501 because count starts at 1 so on the 499th iteration of the for loop the count is at 500 and then you increment it 1 more time so you're left at 501.

d. What is value of "count" at LINE PARENT COUNT? Explain your answer.

501 because count starts at 1 so on the 499th iteration of the for loop the count is at 500 and then you increment it 1 more time so you're left at 501.

e. If you disable the wait(NULL) call at LINE A, what will be the value of "count" at LINE CHILD COUNT and LINE PARENT COUNT? Explain your answer.

501 because nothing changes. Either process can finish first now whereas the parent process was waiting for the child process to end first before.