CS471: Operating System Concepts
Spring 2016
Module 1: Homework #1
Solution
Points: 20

Question 1 [Points 5] Including the initial parent process, how many processes are created by the program shown below?

```
int main()
{
        /* fork a child process */
        fork();  /* First Fork */
        /* fork another child process */
        fork();  /*Second Fork */
        /* and fork another */
        fork();   /* Third Fork  */
        int i;
        for (i=0; i <5; i++)
          fork();/* Fourth fork */ */
        return 0;
}
```

Solution:

- Main starts first process. After the **first fork**, we get **2 processes** which both execute the remaining code.
- Second fork is executed by the above two processes resulting in a total of **4 processes** after the **second fork**. All 4 processes execute the remaining code.
- Third fork is executed by the above processes resulting is **8 processes** at the end of the **third fork**.
- The last for loop is executed by all the above 8 processes. Here is where it gets confusing. Let us just look at what happens with just one of these 8 processes. The same thing happens with the rest. Let P1 be this process.
    - When P1 executes, with i=0, it creates a child process P11. Now both P1 and P11 have a value of i=0; Both execute the next iteration of for loop.
    - In the second iteration of for loop, both P1 and P11 create two other processes. So at teh end of this for loop we have 4 processes with i=1.
    - In the third iteration of the for loop, all the 4 processes created in 2nd iteration, i is incremented to 2 and tested. Since it is less than 5, all 4 execute the for loop resulting in 8 processes at the end of this loop with i=2.
    - In the fourth iteration, in all the 8 processes, I is incremented to 3, and all execute fork resulting in 16 processes at the end of this loop with i=3.
    - In the 5th iteration, I is incremented to 4 for all 16 processes, and all execute fork resulting in 32 processes with i=4.
    - When the 32 go back to for loop, I is incremented to 5 and the for loop terminates and all terminate.
    - In other words, P1 resulted in 31 additional processes, with a total of 32 including P1.

- Thus, each of the 8 result in 32 processes, with a total of 256 processes.

Answer: 256 processes including the initial parent process.

Moral of the example: When a child process is created, it inherits all the parent code with the variables having the values that existed at the time of child creation. It then continues execution along with its parent.

Question 2 [Points 8] Write a program using the fork() system call that computes the factorial of a given integer in the child process. The integer whose factorial is to be computed will be provided in the command line. For example, if 5 is provided, the child will compute 5! and output 120. Because the parent and child processes have their own copies of the data it will be necessary for the child to output the factorial. Have the parent invoke the wait () call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a non-negative number is passed on the command line.

Solution: There are several variations for this program. Here is one version. Main idea is to fork a process and let the child process compute the factorial and print it.

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

int main(int argc, char* argv[])
 { int numb = atoi(argv[1]);
   int fact(int);
   pid_t child_pid;
   int temp;
   child_pid = fork();
   if(child_pid == -1) {return 1;}
   else if (pid==0) {
   temp =fact (numb);
   printf("Factorial of %d is %d ",numb, temp);}
   else {wait (NULL); printf ("Child Complete");}
   return(0);
}
int fact(int i){if (i == 0 ) return 1; return i*fact(i - 1);}
```

Question 3 [Points 7] Modify and run the program shown below in the following way. There is an array of 20 elements defined in the program. The elements of the array are:
[20 18 16 14 12 10 8 6 4 2 -10 -20 -30 -40 15 23 25 75 45 33]. Thread 1 adds the first four elements (i.e., 20, 18, 16, 14), Thread 2 adds the next four elements (i.e., 12, 10, 8, 6), …, Thread 5 adds the last four elements (25, 75, 45, 33). Finally, the sum of all the 20 elements is printed by the program.

```
include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 3
int counter=1;
```

```
void *PrintHello(void *threadid)
{
   counter = 2*counter+ (int) threadid;
   printf("\n Thread Id: %d  Counter:  %d\n", threadid, counter);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
pthread_t threads[NUM_THREADS];
int rc, t;
for(t=0;t<NUM_THREADS;t++){
  printf("Creating thread %d\n", t);
  rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
  if (rc){
    printf("ERROR; return code from pthread_create() is %d\n", rc);
    exit(-1);
    }
  }
   printf("\n Counter:  %d\n", counter);
pthread_exit(NULL);
```

Solution:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS  5

int inArray[20]={ 20, 18, 16, 14, 12, 10, 8, 6, 4, 2, -10, -20, -30, -40, 15, 23, 25, 75, 45, 33 };

int threadSum[NUM_THREADS];

void *addFourNums(void *threadId)
{
    int tId = (int)threadId;
    int start = tId*4;
    threadSum[tId] = inArray[start] + inArray[start + 1] + inArray[start + 2] +
inArray[start + 3];
    printf(" Thread Id: %d  sum: %d \n", tId, threadSum[tId]);
    pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t;
    int sum = 0;
    for(t=0;t<NUM_THREADS;t++)
    {
```

```
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL,addFourNums, (void *)t);
        if (rc)
        {
           printf("ERROR; return code from pthread_create() is %d\n",  rc);
           exit(-1);
        }
    }

    // wait for all the threads to complete
    for (t=0; t< NUM_THREADS; t++)
          pthread_join(threads[t], NULL);

    for (t=0; t< NUM_THREADS; t++)
           sum = sum + threadSum[t];

    printf("\n Sum:  %d\n", sum);
    pthread_exit(NULL);
}
}
```