CS330: Operating Systems
Quiz#2

Name:
Roll No.:

1(a). Write down the possible outputs of the following program. **(4 points)**

```
#define _GNU_SOURCE
#include <stdio.h>
#include <signal.h>
#include <sched.h>
#include <stdlib.h>
#include <sys/syscall.h>

int x=6, y=-1, z=-1;

int f(void *arg)
{
   if (z == getpid()) {
      x++;
      printf("%d\n", x); fflush(stdout); // Created thread prints 4
   }
   else {
      x += 3;
      if (y == syscall(SYS_gettid)) {
         printf("%d\n", x); fflush(stdout);
      }
      else {
         x--;
         printf("%d\n", x); fflush(stdout);  // Parent process prints 8
      }
   }
   return x;
}

int main(void)
{
   void *child_stack;
   y = fork();
   if (y==0) {
      z = getpid();
      x=x/2;
      child_stack = malloc(16384);
      clone(f, child_stack+16384, CLONE_VM | CLONE_THREAD | CLONE_SIGHAND, NULL);
      while(wait(NULL)==-1);
      printf("%d\n", x); fflush(stdout);
   }
   else {
      wait(NULL);
      x = f(NULL);
```

```
        printf("%d\n", x); fflush(stdout);  // Parent process prints 8
    }
    x -= 4;
    printf("%d\n", x); fflush(stdout);  // Parent process prints 4
    return 0;
}
```

Solution: The fork() call creates a child process. The child process creates a thread which inherits the pid. Therefore, the value of `z` in the created thread will be same as the return value of its getpid() call. Since the wait() call in the child process will return -1 (because it has no child), the child process will remain stuck in the while loop. The parent process will remain stuck in its wait() call until the child process terminates. So, the created thread first prints 4 from function `f` and exits. This will terminate the thread group (as discussed in class as per ubuntu 14.04 behavior). Therefore, the child process gets terminated inside the while loop. This, in turn, wakes up the parent process, which calls the function `f`. The value of `z` in the parent process is -1 which doesn't match its pid. The value of `y` in the parent process is the pid of the child process which doesn't match the tid of the parent process (tid of a process is same as its pid by default). So, the parent process prints 8 from function `f` and returns 8. This is again printed by the parent process right after returning from `f` and finally, it prints 4 before terminating. The print statements that execute are shown in comments in the code. The output is as follows.

```
4
8
8
4
```

Grading policy: One point for each correct answer in the correct order; otherwise no point.

1.(b) What will be the output of the program in 1.(a) if the third argument of the `clone` call is changed to `CLONE_VM | SIGCHLD` leaving everything else unchanged? **(3 points)**

Solution: The fork() call creates a child process, say, C. The child process C creates another child process, say, C' which executes the function `f`. The value of `z` in C' is same as the pid of C. So, it will not match its own pid. The value of `y` in C' is zero. So, that will not match its tid (tid of a process is same as its pid by default). The process C will remain stuck in its wait() call and the parent process will remain stuck in its wait() call. So, the first print will come from C' which will print 5 (=6/2+3-1) and terminate. This will wake up C which will get the pid of C' as the return value of wait() and come out of the while loop. It then prints the value of `x` which is now 5 (since `CLONE_VM` is used, the changes made by C' to `x` are visible to C). Then C goes on to decrement `x` by 4 and prints 1 and terminates. This will wake up the parent process, which calls function `f`. The subsequent execution of the parent process is same as in part (a). The overall output is given below.

```
5
5
1
8
8
4
```

Grading policy: Half point for each correct answer in the correct order; otherwise no point.

2.(a) Consider a program that creates ten children. Each of these eleven processes computes a function `y=f(x)` on a private variable `x`. Then each process sends its result `y` to all other processes. If every pair of processes uses

a separate message queue for this communication, compute the minimum number of system calls invoked by this program for creation of message queues, sending/receiving messages, and deletion of message queues. **(2 points)**

Solution: There are $\binom{11}{2} = 55$ message queues. So, the parent process will make 55 msgget calls before forking the children. Each process makes ten msgsnd and ten msgrcv calls. Finally, the parent process needs 55 msgctl calls to delete the queues. So, the total number of system calls is 55+220+55=330.
Grading policy: Half mark each for correct numbers of msgget, msgsnd, msgrcv, and msgctl.

2.(b) Suppose the aforementioned problem is solved using a single shared memory region. The number of processes is unchanged. Compute the minimum number of system calls needed to create the shared memory region, use it, detach from it, and delete it. **(1 point)**

Solution: One shmget call by the parent process before forking the children for creating the shared memory region. One shmat call by each process. One shmdt call by each process. One shmctl call by the parent process. So, the total number of system calls = 1+11+11+1 = 24.
Grading policy: No partial points.