**Home Assignment(1)-Thread**

**-URJA PATEL (121058)**

1. Consider the following C code that calls fork(). If you assume that the child process is always scheduled before the parent process, what will be the output?

int main()

{

int i;

for (i = 0; i < 3; i++)

{

if (fork() == 0)

{

printf("Child sees i = %d\n", i); exit(1);

}

Else

{

printf("Parent sees i = %d\n", i);

}

}

}

ANSWER:

* child sees i=0  
  parent sees i=0  
  child sees i=1  
  parent sees i=1  
  child sees i=2  
  parent sees i=2

1. Consider the following C code that creates and joins with two threads. Assuming that the threads are scheduled completely before the parent process (i.e., have a higher priority), what will be the output from running this program? Be careful! There is a significant trick!

ANSWER: pthread.h header file should be included

While(1) in the program makes the program run infinitely should be removed

Intead of exit replace it by return

Output:

Parent says a:1

Id : 1a:2 b:1

Id : 2a:3 b:1

Thread 1 and 2 complete

1. In some multi-threaded applications, m user-level threads are mapped to n kernel-level threads. Why can this be a good idea (compared to using only user-level or only kernel-level threads)?

For what relative values of m and n is this mapping a possibility (or at all reasonable)? For which relative values is this the best choice?

m >> n

m > n

m (approx) = n

m < n

m << n

ANSWER: From the kernel's perpective, each process has its own address space, file descriptors, etc and **one** thread of execution**.** To support multiple threads at user-level,  the process contains code to create, destroy, schedule and synchronise user-level threads - which can be thought of as mulitplexing many user-level threads onto the single kernel thread, all managed within the process. The scheduler can run any arbitrary scheduling algorithms, and is independent of the kernel's scheduler.

* M user level threads should be greater than n kernel level threads i.e m>>n

because users level thread programs need not require extra checking around blocking i/o to avoid it - i.e. other ready threads within the process can be scheduled if the running thread blocks on a system call or page fault.