Home Assignment-1
-Shailee Mehta (121048)

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 (vfork() == 0) {  //to give the child process the first priority
printf("Child sees i = %d\n", i);
exit(1);
} else {
printf("Parent sees i = %d\n", i);
}
}
}
```

ANSWER:
Using vfork() //priority to child process

```
Child sees i = 0
Parent sees i = 0
Child sees i = 1
Parent sees i = 1
Child sees i = 2
Parent sees i = 2
```

Using fork()

```
 Parent sees i = 0
Parent sees i = 1
Child sees i = 0
Parent sees i = 2
Child sees i = 1
Child sees i = 2
```

2. 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!

```
int a = 0;
void *print_fn(void *ptr)
{
int tid = *(int *)ptr;
int b = 0;
a++; b++;
printf("id: %d a: %d b: %d\n", tid, a, b);
while (1); // Spin-wait here forever
}
int main()
{
```

```
pthread_t t1, t2;
int tid1 = 1;
int tid2 = 2;
int ret1, ret2;
a++;
printf("Parent says a: %d\n", a);
ret1 = pthread_create(&t1, NULL, print_fn, (void *)&tid1);
ret2 = pthread_create(&t2, NULL, print_fn, (void *)&tid2);
if (ret1 || ret2) {
fprintf(stderr, "ERROR: pthread_create failed\n");
exit(1);
}
if (pthread_join(t1, NULL)) {
perror("join of t1");
exit(1);
}
if (pthread_join(t2, NULL)) {
perror("join of t2");
exit(1);
}
printf("Thread 1 and 2 complete\n");
}
```

ANSWER:

```
Parent says a: 1
id: 1 a: 2 b: 1
id: 2 a: 3 b: 1
```

3.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:

Using pure User levels threads or kernel-level threads has a few disadvantages which slow down the processing, like pure ULTs cannot use the power of multi-processing or that with multi-threaded KLTs to transfer control from one thread to another within the same process requires a mode switch to the kernel. So sometimes the best of both approaches is combined, where m user level threads of a process are mapped to n kernel-level threads. They are known as the combined approaches.

Here, thread creation done in the user space. Bulk of scheduling and synchronization of threads done in the user space and the programmer may adjust the number of KLTs.

m<n is the correct choice as per my view because in the applications most of the thread management is done by the programmer, are created faster and are portable whereas the dependence on the kernel level threads is just enough to have the feel of multi-processing. Also, kernel will not have to bother about the user level complications hence keeping the kernel light.