Home Assignment- Thread

27/1/2015

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);

     }

   }

}
```

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");
}
```

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

Submission Dead line : one week (from date of posting)