

Operating Systems Assignment - 7

NAME: SOURABH PATEL

ADMISS NO: U19CS082

1. Consider this code example for allocating and releasing processes:

```
#define MAX_PROCESSES 255
int numberOfProcesses = 0;
/* the implementation of fork() calls this
function */ int allocateProcess() { int newPid;
if (numberOfProcesses == MAX_PROCESSES)
    return -1;
    else {
        /* allocate necessary process resources */
        ++ numberOfProcesses;
        return newPid;
    }
}
/* the implementation of exit() calls this
function */ void releaseProcess() {
    /* release process resources */
    --numberOfProcesses;
}
```

- a) Identify the race condition(s). Be specific — refer to the code.
- b) Assume that you have a mutex lock named mutex with the operations acquire() and release(). Indicate where in the code above that the locking/unlocking needs to be placed to prevent the race condition(s).

Solution:

- a) Race conditions:

- 1) ++numberOfProcesses \Rightarrow in allocateProcess()
- 2) --numberOfProcesses \Rightarrow in releaseProcess()

- b) The Lock should be placed at ++numberOfProcesses & the Release should be placed at --numberOfProcesses.

2) Consider how to implement a mutex lock using atomic hardware instruction. Assume that the following structure defining the mutex lock is available:

```
typedef struct { int
    unavailable;

} lock;
```

(unavailable == 0) indicates that the lock is available, and a value of 1 indicates that the lock is unavailable. Using this struct, illustrate how the following functions can be implemented using the test_and_set() instruction and compare_and_swap() instructions:

- void acquire(lock *mutex)
- void release(lock *mutex)

Be sure to include any initialization that may be necessary.

Solution:

We have a struct defining the availability of mutex lock.

```
//initialization
```

```
mutex->unavailable = 0;
```

```
//acquire using compare_and_swap()
```

```
void acquire(lock *mutex){
while(compare_and_swap(&mutex->unavailable,0,1) != 0);
return;
}
```

```
//acquire using test_and_set()
```

```
Void acquire(lock *mutex){
while(test_and_set(&mutec->unavailable) !=0); return;
}
```

```
Void release(lock *mutex){  
    mutec->unavailable = 0; return;  
}
```
