Problem Set 10

CS 411

Due at the beginning of class on the first class day of the following week.
Sections 5.4–6

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).

2. Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single processor system if the synchronization primitives are to be used in user-level programs.

3. Consider how to implement a mutex lock using an 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:

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

Be sure to include any initialization that may be necessary.