Assigment-5

1. Consider how to implement a mutex lock using an atomic hardware instruction. Assume that a mutex lock is defined as

boolean lock

(lock == 0) indicates that the lock is available, and a value of 1 indicates that the lock is unavailable. Illustrate how the following functions

void acquire(boolean \*lock)

void release(boolean \*lock)

can be implemented using the atomic test\_and\_set() instruction. The meaning of test\_and\_set() can be explained as:

boolean test and set(boolean \*target) {

boolean res = \*target;

\*target = true;

return res;

}

Be sure to include any initialization that may be necessary.

Answer:

void acquire(boolean \*lock)

{

while(test\_and\_set(lock))

;

}

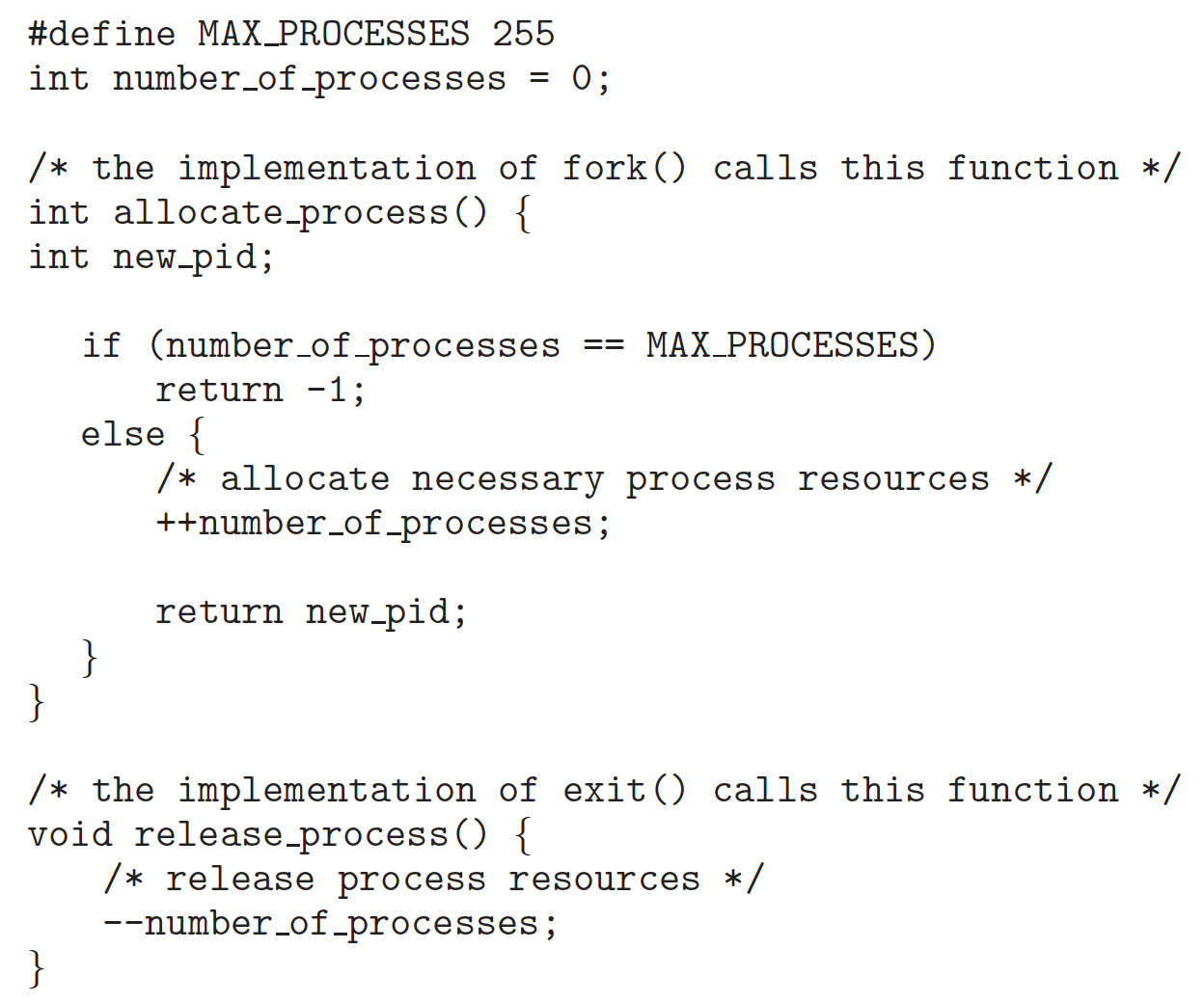
void release(boolean \*lock)

{

\*lock = false;

}

1. Consider the code example for allocating and releasing processes shown in the following figure:



a. Identify the race condition(s).

b. Assume you have a mutex lock named **mutex\_lock** with the operations acquire() and release(). Indicate where the locking needs to be placed to prevent the race condition(s).

Answer:

1. When functions are executed two times at the same time(e.g. in two CPU core). For the allocate\_process(), they may check the ‘number\_of\_processes’ at the same time and both comes out that ‘number\_of\_processes’ is not MAX, although the variable may originally be 254. As a result, there may be 256 processes keeping running with number\_of\_processes equals to 255.

Also, it is similar in release\_process(). As the pid may be allocated incorrectly when the functions’ statements are executed in real time in this order:

++number\_of\_processes; //allocate\_process()

--number\_of\_processes; //release\_process()

Return new\_pid; //allocate\_process()

1. In both allocate and release functions, the acquire() and release() should be just before and after the statement with the ‘number\_of\_processes’ variable respectively. Example:

int allocate\_process(){

int new\_pid;

acquire(&mutex\_lock);

if (number\_of\_processes == MAX\_PROCESSES)

return -1;

else{

++number\_of\_processes;

}

release(&mutex\_lock);

return new\_pid;

}

void release\_process(){

acquire(&mutex\_lock);

--number\_of\_processes;

release(&mutex\_lock);

}