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.

A：

boolean lock = false;

void acquiry(boolean \*lock)

{

while(!test\_and\_set(\*lock))

; //do nothing

}

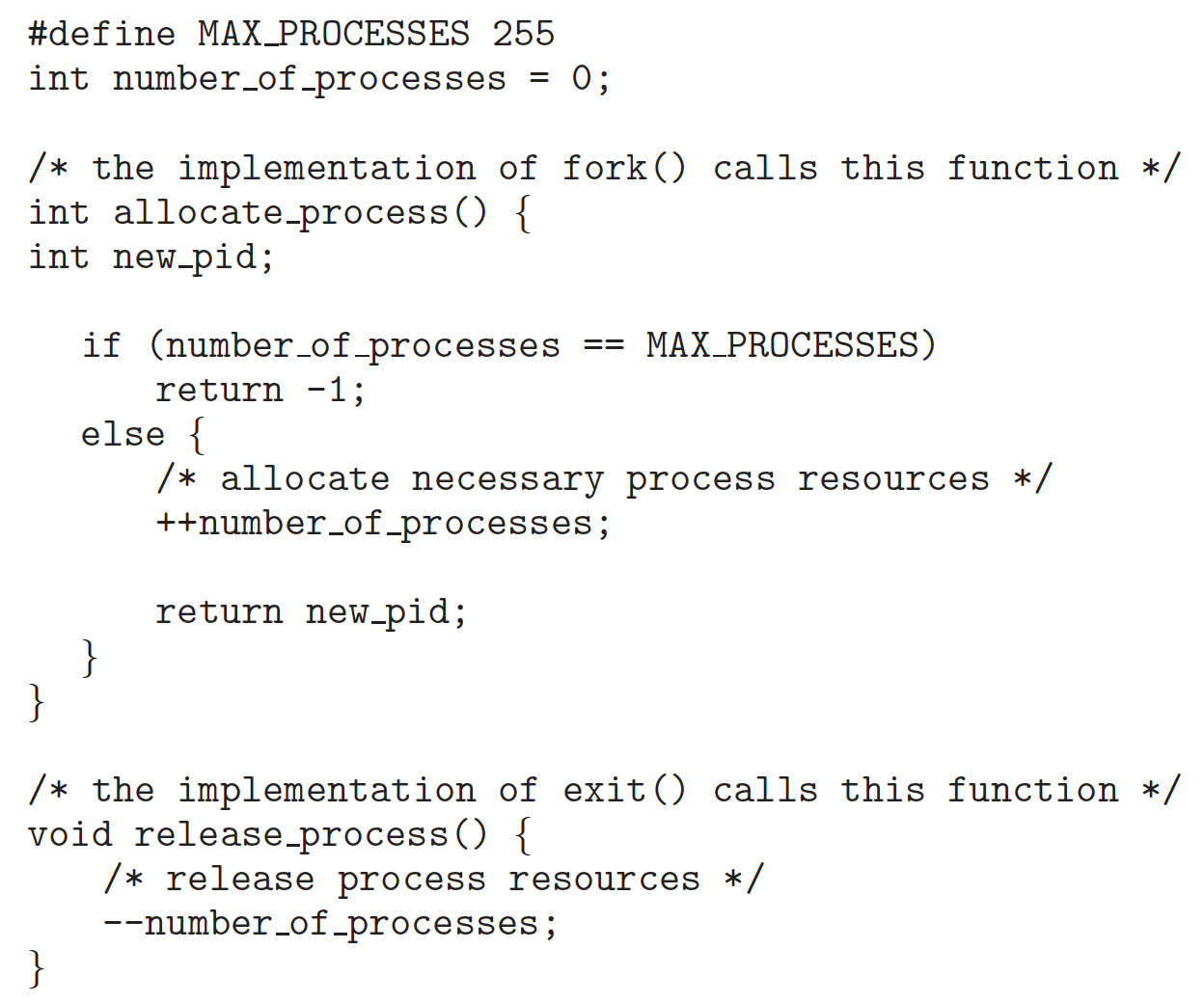
void release(boolean \*lock)

{

lock = false;

}

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

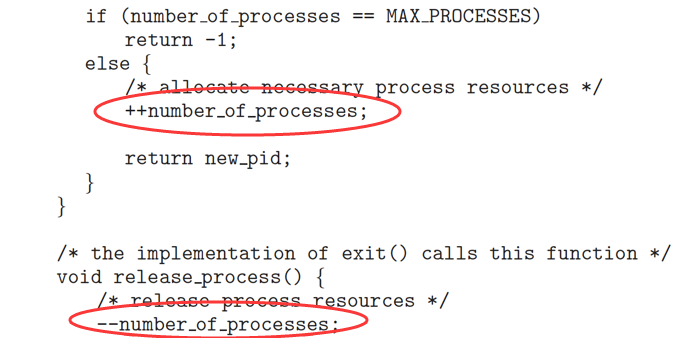


a. Identify the race condition(s).

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).

A：

a. As shown in the figure



b. As shown in the figure

