Assigment-5

1. We define a semaphore as follows:

typedef struct {

int value;

struct process \*list;

} semaphore;

Each semaphore has an integer value and a list of processes list. When a process must wait on a semaphore, it is added to the list of processes. A signal() operation removes one process from the list of waiting processes and awakens that process.

Now, the wait() semaphore operation can be defined as

wait(semaphore \*S) {

S->value--;

if (S->value < 0) {

add this process to S->list;

block();

}

}

And the signal() semaphore operation can be defined as

signal(semaphore \*S) {

S->value++;

if (S->value <= 0) {

remove a process P from S->list;

wakeup(P);

}

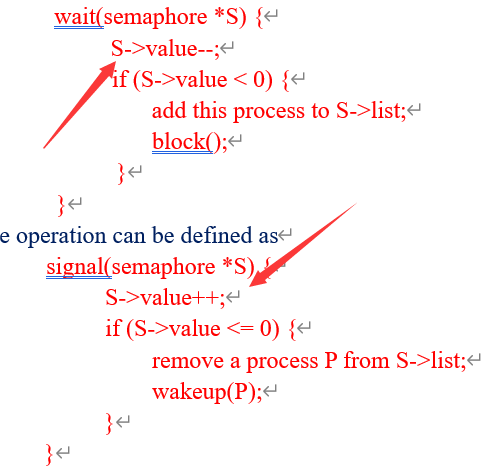
}

The block() operation suspends the process that invokes it. The wakeup(P) operation resumes the execution of a blocked process P. These two operations are provided by the operating system as basic system calls.

Identify race conditions in the above implementation of wait() and signal(), and illustrate how to resolve the problem using the mutex lock mechanism (recall Assignment-4).

Answer:

Race condition:



Mutex lock:

Acquire(Boolean \*lock) at the begin of wait() and signal()

release(Boolean \*lock) at the end of wait() and signal()