**ASSIGNMENT 4**

1. **Assume that the following structure for a mutex lock is available:**

**typedef struct{**

**int available;**

**}**

**Using this, illustrate how the functions acquire and release can be executed using test\_and\_set() and compare\_and\_swap() instructions:**

struct lock{

int locked=0;

}

//Test and Set

void acquire(lock){

while(test\_and\_set(&lock->locked));

}

void release(lock){

lock->locked=0;

}//Compare and swap

void acquire(lock){

value=compare\_and \_swap(&lock->locked,0,1);

if(value==0){

/\*Lock acquired\*/

}

else{

/\*Filed to acquire lock\*/

}

}

void release(lock)

{

lock->locked=0;

}

1. **Assume that a system has multiple processing cores. For each of the following scenarios, explain which is a better locking mechanism; A spin lock or a mutex lock while waiting processes sleep while waiting for the lock to become available:**

**--The lock is to be held for a short duration.**

Spin Lock as the thread trying to acquire the lock keeps spinning continuously which keeps it busy and does not allow other threads to take its place . Hence offers a relatively faster runtime.

**--The lock is to be held for a long duration.**

Mutex lock as it is resource efficient for longer processes.

**--A thread may be put to sleep while holding the lock.**

Mutex lock as Spin lock cannot be used here because spin lock gets released automatically upon putting it to sleep.

1. **A multithreaded web server wishes to keep track of number of requests it services (known as hits). Consider the following strategies to prevent a race condition on the variable hits. The first strategy is to use a basic mutex lock while updating hits:**

**isnt hits;**

**mutex\_lock hit\_lock;**

**hit\_lock.acquire();**

**hits++;**

**hit\_lock.release();**

**A second strategy is to use an atomic integer:**

**atomic\_t hits;**

**atomic\_inc(&hits);**

**Explain which of these strategies is more efficient.**

Atomic integers would be more efficient ,because in case of Mutex lock, multiple threads would have to wait for the lock to become available which decrease efficiency.

Also, the atomic lock will lock the memory bus on the platform . It is impossible to suspend a thread during memory bus lock , but a thread can be suspended during a mutex lock, which is a disadvantage of mutex lock.

1. **Consider the code example for allocating and releasing processes shown:**

**#define MAX\_PROCESSES 255**

**int number\_of\_processes=0;**

**/\*The implementation of fork() calls this function.\*/**

**int allocate\_process(){**

**int new\_pid;**

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

**return -1;**

**else{**

**/\*allocate necessary process resources\*/**

**++number\_of\_processes;**

**return new\_pid;**

**}**

**}**

**/\*The implementation of exit() calls this function\*/**

**void release\_process(){**

**/\*Release process resources\*/**

**--number\_of\_processes;**

**}**

1. **Identify the race condition(s).**

**2. Assume you have mutex lock named mutex with the “acquire” and “release” operations.**

**Indicate where the locking is to be placed to prevent race conditions().**

**3. Could we make the variable number\_of\_processes atomic to prevent the race condition(s)?**

1. There will be a race condition whenever there changes made to the value of number\_of\_processes.
2. We should lock the statements where we are changing the value of number\_of\_processes.
3. Yes. If the variable number\_of\_processes is made atomic, we can prevent the race condition(s).