1.

semaphore = sem\_open("/semaphore", O\_CREAT, 0644 ,1);

void lock(){

sem\_wait(semaphore);

}

void unlock(){

sem\_post(semaphore);

}

2.

Basically, it refers to the case that waiting thread miss its chance to be waked because the one who is supposed to wake it up send the wake-up signal before the waiting thread is ready for taking wake-up signal.

3.

Basically, if we do not consider the case that some threads perform I/O operation, it should be 8, for each core can utilize hyper-threading technology to allow two thread execute in parallel fashion.

But, if we consider that some threads perform I/O, therefore leading them to sleep while waiting I/O to finish, we should spawn more than 8 in order to keep CPU busy.

4.

//static variable will be shared by every thread and therefore need to resolve the synchronization issue.

static double sum\_stat\_a = 0;

static double sum\_stat\_b = 0;

static double sum\_stat\_c = 1000;

int aggregateStats(double stat\_a, double stat\_b, double stat\_c) {

pthread\_mutex\_lock(lock) ;

sum\_stat\_a += stat\_a ;

sum\_stat\_b -= stat\_b; //w

sum\_stat\_c -= stat\_c; //w

pthread\_mutex\_unlock(lock) ;

return sum\_stat\_a + sum\_stat\_b + sum\_stat\_c;

}

void init(void) {

pthread\_mutex\_t lock;

}

5.

static double sum\_stat\_a = 0;

static double sum\_stat\_b = 0;

static double sum\_stat\_c = 1000;

int aggregateStats(double stat\_a, double stat\_b, double stat\_c) {

if(static\_a != 0){

pthread\_mutex\_lock(lock\_a) ;

sum\_stat\_a += stat\_a ;

pthread\_mutex\_unlock(lock\_a) ;

}

if(static\_b != 0){

pthread\_mutex\_lock(lock\_b) ;

sum\_stat\_b -= stat\_b; //w

pthread\_mutex\_unlock(lock\_b) ;

}

if(static\_c != 0){

pthread\_mutex\_lock(lock\_c) ;

sum\_stat\_c -= stat\_c; //w

pthread\_mutex\_unlock(lock\_c) ;

}

return sum\_stat\_a + sum\_stat\_b + sum\_stat\_c;

}

void init(void) {

pthread\_mutex\_t lock\_a ;

pthread\_mutex\_t lock\_b ;

pthread\_mutex\_t lock\_c ;

}

6.

a.

The lock and unlock functions should be atomic and only one thread can acquire a specific lock.

Help avoid race condition

Help avoid deadlock

Fairness

b.

The lock and unlock functions should be

Help avoid race condition

Help avoid deadlock

Fairness

7.

Busy wait is a process that program keep checking whether it fulfills certain conditions by looping.

Pro: as soon as the condition is met, it will take the corresponding actions.

Con: keep consuming CPU resource just for waiting the conditions to be met.