Questions on Chapter 4 Answers

1. Describe two applications of barriers?

APP1

If we are timing some part of multi thread program we'd like for all the thread to start the timed

Code from some instance and then report the time taken by last thread to finish (slowest)

Double elapsed_time

/*private */

Double my_start,my_finish,my_elapsed ;

Synchronization threads ;

Store currant time in my_start;

/*Execute time code */

Store currant time in my_finish;

My_elapsed=my_start-my_finish

Elapsed = Maximum of my_elapsed values

APP2

using barriers in debugging
Difficult to determine where an error is occurring in a parallel programe
Each thread print aMSG indicating which point it's reached in the program
Paint on programe we want to reach ;
Barriers;

```
If(my_rank==0 ){
        PrintF("All threads reached this paint /n")
FFlush(stdout);
}
```

2. Give the implementation of a barrier using busy-waiting and a mutex, semaphores, and condition variables ?

1-busy-wating and mutex

Use shared counter ,when counter indicates that every thread has entered the critical section
Thread can leave abusy_await loop

```
Int counter;
Int thread_count;
Pthread_mutex_t basrriers_mutex;
Void* Thread_work(……)
Pthread_mutex_lock(& barriers_mutex);
counter ++;
Pthread_mutex_unlock(& barriers_mutex);
While (counter <thread_count);
```

2-Semaphores

As with busy wait,we have acounter that we use to determine how many thread use to enterd The barrier

Use 2 semaphores

count _sem (protects the counter) ,

barriers_sem (block threads that have entered barrier)

```
Int counter ;
Sem_t count-sem;
Sem_t barriers;
Void* thread_work (….){
Sem-wait (&count –sem)
        If (counter==threadCount-1){
                counter=0;
        sem_Post(count-sem)
        For(j=0;j<thread-count-1;j++) {
                Sem_Post(& barrier_sem);
                        }
        }else {
                Counter ++;
                Sem_post(&cont-sem)
                Sem_wait (&barriers)
              }
```

3-Condition Variable

Allow thread to suspend execution until acertain event or condition occurs

```
Int counter =0;
Pthread_mutex_t mutex Mutext;
Pthread_cond_t Cond_var;
Void* thread-work(…..){
Pthread-mutex_lock(&Mutex);
Counter ++;
If(counter==htread_count){
        Counter =0;
        Pthread_condbrodcast(& cond_var)}
Else{
        While(pthread_cond_wait(Cond_var,&Mutex)!=0);}
Pthread_mutex_unlock(&utex);}
Pthread_cond_wait(….){
Pthread_mutex_unlock(&Mutex-p)
Pthread_wait_on_signal(&Cond_var-p)
Pthread_mutex_lock(&Mutex-p)
```

3. Reusing counter in barrier implementation using busy waiting is problematic, while it is not problematic when semaphores are used, explain why?

2-Reusing counter in barriers implementation using busy_waiting is problematic

Because any attempt to rest counter zero will be failure

1-if the last thread tries to rest it,some other thread may hang in the busy_wait thread this Thread never see that counter==Thread_count

2-if some threads counter after barriers => some other threads enter the second barriers before the counter is reset and its increment to counter will be last this lead to hang this in second barriers

While it is not problemtatic when semaphores are used

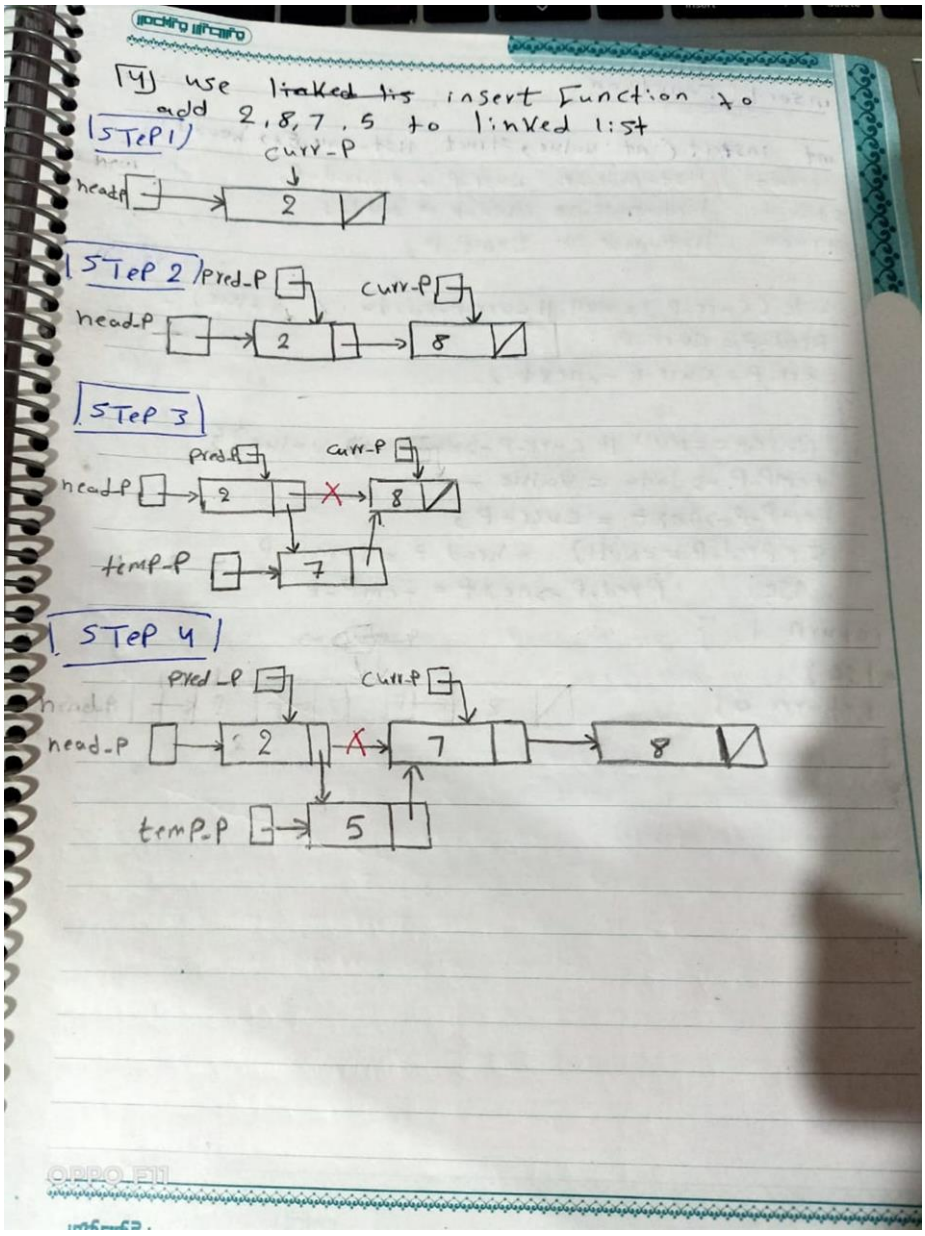
⇨ Counter can be reused because

1-we were careful to reset it before releasing any of thread from barriers

2-count sem also can be reused,since it reset to 1 before any thread can leave it barriers

And If any thread go on the second barriers will execute the else clause and when reaches To sem wait(sem_barrired)will be still 1 and decrement sem_barriers and processed

4. Use insert function to add 2, 8, 7, 5 to a linked list. Then, use member function to find 5 and 9. Finally, delete 5 from the list.



Insert Function

```
Int  insert(int value, struct list node*head_p){
        Struct list_node_s* curr-p=*head_p;
        Struct list_node_s* pred-p=null;
        Struct list_node_s* temp-p;
        While(curr-p != null || curr-p->data <value){
                Pred-p=curr-p;
                Curr-p=curr-p ->next;
                }
        If(curr-p == null || curr-p ->data >value){
                Temp-p -> data =value
                Temp-p->next = curr-p;
                If (pred-p ==null) *head-p=temp-p
                else pred-p->next=temp-p
                return 1;}
        else return 0;
}}
```

Use member function to find 5 and 9

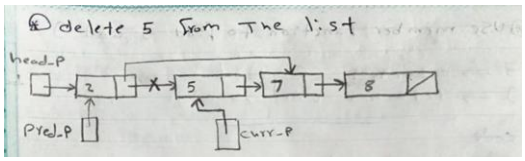
5 => return 1 Found

9 => return 0 not Found

Code

```
Int member(Int value , Struct list_node_s* head-p){
        Struct list_node_s* curr-p =head-p;
        While(curr-p != null && curr-p-> data <value)
        curr-p = curr-p->next;
        if(curr-p == null || curr-p -> data >value) return 0;
        else return 1;
}
```

Delete 5 from the list




```
nt  insert(int value, struct list node*head_p){
        Struct list_node_s* curr-p=*head_p;
        Struct list_node_s* pred-p=null;
        While(curr-p != null || curr-p->data <value){
                Pred-p=curr-p;
                Curr-p=curr-p ->next;
                }
        If(curr-p != null || curr-p ->data ==value){
```

```
                If(pred-p==null){
                        Head-p=curr-p->next;
                        Free(curr-p)}
                else {
                        pred-p->next = curr-p-next;
                        Free(curr-p);}
                return 1;
        else{
                return 0;
        }
}}
```

5. Executing concurrent multiple read and write operations on a linked list can cause problems. What are these problems and how they can be solved?

Insert / Delete write to memory location so there is problem if we try to execute these operation At some time with another operation

Problem1

If

Thread 0 -> execute member(5)

//at the same time

Thread 1 -> delete (5)

Problem -> when thread 0 is executing member(5) it is going to report in the list when it may be deleted before thread return

Problem2

If thread 0 is exciting member(8) thread 1 may free the memory use for the node storing 5 before thread 0 advance to member 8

Problem -> if the memory is reallocated before thread 0 advance

How can we solve this problems?

Solution is simply lock the the list any time that athread attempts to access it

Pthread_mutex_lock(&list_mutex);

Member(value)

Pthread_mutex_unlock(&list_mutex);

OR

Lock individual one mutex per node

For example ,amutex to the list node struct

```
Struct list_node_s[
        Int data
        Struct list_node_s* next ;
        Pthread_mutex mutex; ]
```

OR

Use read-wite lock

Provide 2 lock function

The First lock function -> lock the read – write lock for reading
The second locks it for writing
Multiple thread can simultaneously obtain the lock by calling the read_lock function , while only one thread can obtain the lock by calling the write_lock function

6. How read-write locks can be implemented?

Implementation defines a data structure that uses :-

- ➔ 2 condition one for readers and other for writer
- ➔ Mutax
- ➔ Members that indicates
  - How many reader own lock
  - How many reader are waithing to obtain the lock
  - Whether the writer won the lock and how many writer are waiting to obtain the lock
- ➔ If thread calls one of the function (read 0lock, write, unlock) -> it first locks the mutex, and whenever a thread completes one of the calls it unlock the mutax
- ➔ After acquiring the mutax,the thread checks the appropriate data member to determine how to processed

7. Explain how matrix dimensions can affect the efficiency of matrix-vector multiplication program?

8. Give an example where incorrect program can produce correct result.