Introduction to Operating Systems and SQL for Data Science

Practice 3 – Threads

Threads

- Threads– mini processes sharing their memory (address space)
- Each thread runs a function (in the same code base)
- When 2 threads are running the same code section (approach same resource) the result depends on execution order.
- A section like that is called: critical section.

Question #1

This code sort an unsorted array by using insertion sort algorithm. In each execution of the outer for loop we takes an element from the unsorted array, put it in the last cell of the currently sorted array and then swap it in the sorted array to its correct index.

A. Implement a method that gets a list of unsorted arrays and sort them all in parallel by creating threads and using InsertionSort function.

```
1.  int[] sorted = new int[N];
2.  int lastSorted = 0;
3.  void InsertionSort(int[] unsorted) {
4.      for(int i = 0 ; i < unsorted.length ; i++) {
5.          sorted[lastSorted] = unsorted[i];
6.          int j = lastSorted;
7.          lastSorted++;
8.          bool stop = false;
9.          while (j > 0 && !stop) {
10.             if(sorted[j]<sorted[j-1]) {
11.                 int aux = sorted[j];
12.                 sorted[j] = sorted[j-1];
13.                 sorted[j-1] = aux;
14.             }
15.             else
16.                 stop = true;
17.             j--;
18.         }
19.     }
20. }
```

swap

Reichman University

Solution 1.A

```
void ParallelInsertionSort(List<int[]> unsortedArrays){
    List<Thread> lThread = new List<Thread>();
    foreach( int[] unsortedArray in unsortedArrays)
    {
        Thread t = new Thread( () => InsertionSort(unsortedArray));
        t.Start();
        lThread.Add(t);
    }
    foreach( Thread t in lThread)
    {
        t.Join();
    }

}
```

Thread's input is the function it runs

Thread t starts running

The program waits for thread t to finish

Reichman University

Question #1

B. Describe 2 different races in InsertionSort

```
1.  int[] sorted = new int[N];
2.  int lastSorted = 0;
3.  void InsertionSort(int[] unsorted) {
4.      for(int i = 0 ; i < unsorted.length ; i++)
    {
5.          sorted[lastSorted] = unsorted[i];
6.          int j = lastSorted;
7.          lastSorted++;
8.          bool stop = false;
9.          while (j > 0 && !stop) {
10.             if(sorted[j]<sorted[j-1]) {
11.                 int aux = sorted[j];
12.                 sorted[j] = sorted[j-1];
13.                 sorted[j-1] = aux;
14.             }
15.             else
16.                 stop = true;
17.             j--;
18.         }
19.     }
20. }
```

Solution

- **Race #1:**
- **Initial state:**
  - sorted={}
  - t1: unSorted1={1}
  - t2: unSorted2={2}
- **Execution:**
  - t1 starts and put the value 1 in index 0 of sorted (execute code until line 6) and then sleeps.
    sorted={1}
  - t2 starts and put the value 2 in index 0 of sorted
    sorted={2}
- **Result:**
  - **Information lost!** The value 1 (unSorted1[0]) is overwritten.

```
1.  int[] sorted = new int[N];
2.  int lastSorted = 0;
3.  void InsertionSort(int[] unsorted) {
4.      for(int i = 0 ; i < unsorted.length ; i++)
    {
5.          sorted[lastSorted] = unsorted[i];
6.          int j = lastSorted;
7.          lastSorted++;
8.          bool stop = false;
9.          while (j > 0 && !stop) {
10.             if(sorted[j]<sorted[j-1]) {
11.                 int aux = sorted[j];
12.                 sorted[j] = sorted[j-1];
13.                 sorted[j-1] = aux;
14.             }
15.             else
16.                 stop = true;
17.             j--;
18.         }
19.     }
20. }
```

Solution

- **Race #2:**
- **Initial state:**
  - sorted={3}
  - t1: unSorted1={1}
  - t2: unSorted2={2}
- **Execution:**
  - t1 is starting to run and puts value 1 in index 1 of sorted (go to sleep after line 8)
    sorted={3,1}
  - t2 starts to run and puts the value 2 in index 2 of sorted, doesn't swap at all and finishes

    sorted={3,1,2}
  - t1: wake up and swap 1 and 3
    sorted={1,3,2}
- **Result:**
  - **We got an unsorted array!**

```
1.  int[] sorted = new int[N];
2.  int lastSorted = 0;
3.  void InsertionSort(int[] unsorted) {
4.      for(int i = 0 ; i < unsorted.length ; i++)
    {
5.          sorted[lastSorted] = unsorted[i];
6.          int j = lastSorted;
7.          lastSorted++;
8.          bool stop = false;
9.          while (j > 0 && !stop) {
10.             if(sorted[j]<sorted[j-1]) {
11.                 int aux = sorted[j];
12.                 sorted[j] = sorted[j-1];
13.                 sorted[j-1] = aux;
14.             }
15.             else
16.                 stop = true;
17.             j--;
18.         }
19.     }
20. }
```

Question #2

Given an insert function (insert to a binary search tree)

1. Give an example of initial not empty tree an insertion attempt by 2 different thread that creates wrong result.

```
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
5.                                  if(current.right == null){
6.                                                  current.right = new node(key,data)
7.                                                  done = true
8.                                  }
9.                                  else{
10.                                                 current = current.right
11.                                 }
12.                         }
13.         else{
14.                                 if(current.left == null){
15.                                                 current.left = new node(key,data)
16.                                                 done = true
17.                                 }
18.                                 else{
19.                                                 current = current.left
20.                                 }
21.         }
22. }
```

Solution

- **Initial state:**
  - One node with the value 3.
  - Thread A tries to insert the key 6 and thread B tries to insert the key 7.
- **Execution:**
  - Thread A starts and execute code until line 6
  - Thread B starts, insert key 7 and finished
  - Thread A return to run from line 6 and insert 6 and override 7
- **Result:**
  - Information lost!

```
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
5.                                  if(current.right == null){
6.                                                  current.right = new node(key,data)
7.                                                  done = true
8.                                  }
9.                                  else{
10.                                                 current = current.right
11.                                  }
12.                         }
13.         else{
14.                                 if(current.left == null){
15.                                                 current.left = new node(key,data)
16.                                                 done = true
17.                                 }
18.                                 else{
19.                                                 current = current.left
20.                                 }
21.         }
22. }
```

Solution

What is the critical section?

- Lines 5-6
- Lines 14-15

```
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
5.                                  if(current.right == null){
6.                                                  current.right = new node(key,data)
7.                                                  done = true
8.                                  }
9.                                  else{
10.                                                 current = current.right
11.                                 }
12.                         }
13.         else{
14.                                 if(current.left == null){
15.                                                 current.left = new node(key,data)
16.                                                 done = true
17.                                 }
18.                                 else{
19.                                                 current = current.left
20.                                 }
21.         }
22. }
```

TSL = Test Set & Lock

- **TSL(lock) – an atomic function to protect a CS and avoid multiple entrance to the CS** .

```
bool lock=false;
.
.
.
Public F()
{
  While( TSL(lock) );

   // CS…

  lock =false
}
```

```
bool TSL(ref  bool  lock)
{
    if( lock == true )
     {
       return true;
     }
     else
     {
       lock = true;
       return false;
     {
}
```

Question 2.1

Use TSL to protect the critical section.

```
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
5.                                  if(current.right == null){
6.                                                  current.right = new node(key,data)
7.                                                  done = true
8.                                  }
9.                                  else{
10.                                                 current = current.right
11.                                  }
12.                         }
13.         else{
14.                                 if(current.left == null){
15.                                                 current.left = new node(key,data)
16.                                                 done = true
17.                                 }
18.                                 else{
19.                                                 current = current.left
20.                                 }
21.         }
22. }
```

Solution

```
bool lock = false;
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
                                                    while(TSL(lock));
5.                                                  if(current.right == null){
6.                                                          current.right = new node(key,data)
                                                            lock = false;
7.                                                          done = true
8.                                                  }
9.                                                  else{
                                                            lock = false;
10.                                                         current = current.right
11.                                                 }
12.                         }
13.         else{
                                                    while(TSL(lock));
14.                                                 if(current.left == null){
15.                                                         current.left = new node(key,data)
                                                            lock = false;
16.                                                         done = true
17.                                                 }
18.                                                 else{
                                                            lock = false;
19.                                                         current = current.left
20.                                                 }
21.         }
22. }
```

Question 2.3

Use 2 Booleans to increase parallelism.

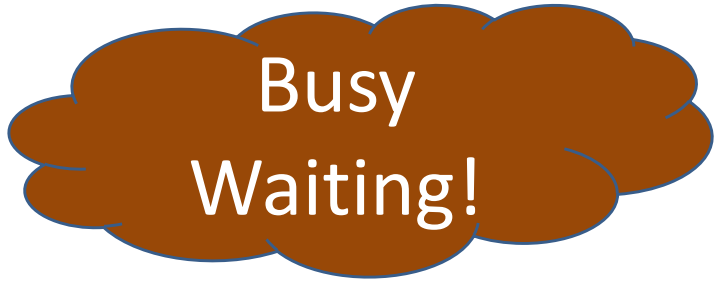
```
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
5.                                  if(current.right == null){
6.                                                  current.right = new node(key,data)
7.                                                  done = true
8.                                  }
9.                                  else{
10.                                                 current = current.right
11.                                 }
12.                         }
13.         else{
14.                                 if(current.left == null){
15.                                                 current.left = new node(key,data)
16.                                                 done = true
17.                                 }
18.                                 else{
19.                                                 current = current.left
20.                                 }
21.         }
22. }
```

Solution

```
bool lock1, lock2 = false;
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
                                                    while(TSL(lock1));
5.                                                  if(current.right == null){
6.                                                          current.right = new node(key,data)
                                                            lock1 = false;
7.                                                          done = true
8.                                                  }
9.                                                  else{
                                                            lock1 = false;
10.                                                         current = current.right
11.                                                 }
12.                         }
13.         else{
                                                    while(TSL(lock2));
14.                                                 if(current.left == null){
15.                                                         current.left = new node(key,data)
                                                            lock2 = false;
16.                                                         done = true
17.                                                 }
18.                                                 else{
                                                            lock2 = false;
19.                                                         current = current.left
20.                                                 }
21.         }
22. }
```

TSL

- What is the main drawback of TSL?



Reichman University

Avoid busy waiting – use Mutex

- fields:
  - Bool locked
  - Queue Sleeping
- Lock and Unlock are atomic.

```
lock(){
        If(locked){
                sleeping.enqueuer(curr_thread);
                sleep();
        }
        locked = true;
}
```

```
unlock(){
        locked = false;
        If(sleeping.count>0){
                tid = sleeping.dequeuer();
                wakeup(tid);
        }
}
```

Question 2.4

Use mutex to protect critical section

```
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
5.                                  if(current.right == null){
6.                                                  current.right = new node(key,data)
7.                                                  done = true
8.                                  }
9.                                  else{
10.                                                 current = current.right
11.                                 }
12.                         }
13.         else{
14.                                 if(current.left == null){
15.                                                 current.left = new node(key,data)
16.                                                 done = true
17.                                 }
18.                                 else{
19.                                                 current = current.left
20.                                 }
21.         }
22. }
```

Solution

```
Mutex m1, m2;
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
                                        m1.lock();
5.                                      if(current.right == null){
6.                                              current.right = new node(key,data)
                                                m1.unlock();
7.                                              done = true
8.                                      }
9.                                      else{
                                                m1.unlock();
10.                                             current = current.right
11.                                     }
12.                         }
13.         else{
                                        m2.lock();
14.                                     if(current.left == null){
15.                                             current.left = new node(key,data)
                                                m2.unlock();
16.                                             done = true
17.                                     }
18.                                     else{
                                                m2.unlock();
19.                                             current = current.left
20.                                     }
21.         }
22. }
```

Semaphore

Sometimes we want to allow a limited multiple entrance to a CS.

- Fields:
  - Counter
- Down and Up are atomic.

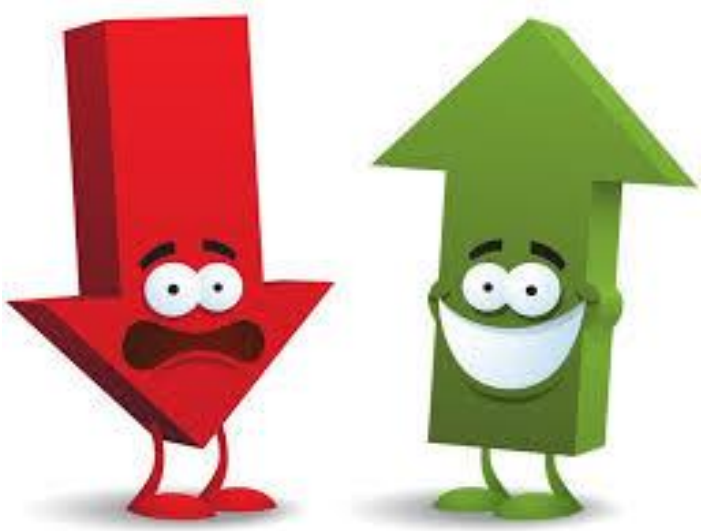
```
down(){
        If(counter==0){
                sleeping_id = curr_thread;
                sleep();
        }
        counter --;
}
```

```
up(){
        counter ++;
        If(sleeping_id!=null){
                wakeup(sleeping_id);
        }
}
```

Reichman University

Semaphore

- 2 methods: up and down
- Up increase the counter
- Down decrease the counter, when a thread tries to decrease the counter when its value is 0 the tread will sleep (until awaked)



Question 2.5

Use semaphore to protect critical section

```
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
5.                                  if(current.right == null){
6.                                                  current.right = new node(key,data)
7.                                                  done = true
8.                                  }
9.                                  else{
10.                                                 current = current.right
11.                                 }
12.                         }
13.         else{
14.                                 if(current.left == null){
15.                                                 current.left = new node(key,data)
16.                                                 done = true
17.                                 }
18.                                 else{
19.                                                 current = current.left
20.                                 }
21.         }
22. }
```

Solution

```
s1 = new Semaphore(1), s2 = new Semaphore(1);
1.  insert(key, data)
2.  current←root  ,done ← false
3.  while(!done){
4.          if(current.key < key){
                                        s1.down();
5.                                      if(current.right == null){
6.                                              current.right = new node(key,data)
                                                s1.up();
7.                                              done = true
8.                                      }
9.                                      else{
                                                s1.up();
10.                                             current = current.right
11.                                     }
12.                         }
13.         else{
                                        s2.down();
14.                                     if(current.left == null){
15.                                             current.left = new node(key,data)
                                                s2.up();
16.                                             done = true
17.                                     }
18.                                     else{
                                                s2.up();
19.                                             current = current.left
20.                                     }
21.         }
22. }
```

Mutex vs. Semaphore

- Lock --- Down
- Unlock --- Up
- We can use a semaphore to implement a mutex.
- An important note:
  - If a thread t1 locked the mutex, t1 is the “owner” of the mutex and it is the only one who can unlock it (and awake other thread waiting for it)
  - When using semaphore there is no “ownership” on the semaphore, t1 can up the semaphore and t2 can down it.