Student Name: Hemalatha Veeranki

Student ID: 11702443

Email Address: chinnuneeluv@gmail.com

GitHub Link: https://github.com/Hemalatha-Veeranki/Os-readers-and-writers.git

Q1

**Code:**

**void \*writer(void \*arg)**

**{**

**int f;**

**f = ((int) arg);**

**sem\_wait(&writeblock);**

**data++;**

**printf("Data writen by the writer%d is %d\n",f,data);**

**//sleep(1);**

**sem\_post(&writeblock);**

**}**

**int main()**

**{**

**int i,b;**

**pthread\_t rtid[5],wtid[5];**

**sem\_init(&mutex,0,1);**

**sem\_init(&writeblock,0,1);**

**for(i=0;i<=5;i++)**

**{**

**pthread\_create(&wtid[i],NULL,writer,(void \*)i);**

**pthread\_create(&rtid[i],NULL,reader,(void \*)i);**

**pthread\_join(wtid[i],NULL);**

**pthread\_join(rtid[i],NULL);**

**}**

**return 0;**

**}**

**Description:**

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

Resources--

* How much of each resource each process could possibly request[MAX]
* How much of each resource each process is currently holding[ALLOCATED]
* How much of each resource the system currently has available[AVAILABLE]

Resources may be allocated to a process only if the amount of resources requested is less than or equal to the amount available; otherwise, the process waits until resources are available.

Some of the resources that are tracked in real systems are [memory](https://en.wikipedia.org/wiki/Memory_(computers)), [semaphores](https://en.wikipedia.org/wiki/Semaphore_(programming)) and

[interface](https://en.wikipedia.org/wiki/Interface_(computer_science)) access.

The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.

Basic data structures to be maintained to implement the Banker's Algorithm:

Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

* Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type Rj available.
* Max: An *n*×*m* matrix defines the maximum demand of each process. If Max[i,j] = k, then Pi may request at most k instances of resource type Rj.
* Allocation: An *n*×*m* matrix defines the number of resources of each type currently allocated to each process. If Allocation[i,j] = k, then process Pi is currently allocated k instances of resource type Rj.
* Need: An *n*×*m* matrix indicates the remaining resource need of each process. If Need[i,j] = k, then Pimay need k more instances of resource type Rj to complete the task.

Note: Need[i,j] = Max[i,j] - Allocation[i,j]. n=m-a.

Safe and Unsafe States--

 The algorithm determines if a state is **safe** by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an **unsafe** state.

Requests--

When the system receives a request for resources, it runs the Banker's algorithm to determine if it is safe to grant the request. The algorithm is fairly straightforward once the distinction between safe and unsafe states is understood.

1. Can the request be granted?
   * If not, the request is impossible and must either be denied or put on a waiting list
2. Assume that the request is granted
3. Is the new state safe?
   * If so grant the request
   * If not, either deny the request or put it on a waiting list

**Algorithm:**

1. How much of each resource each person could maximum request [MAX]
2. How much of each resource each person currently holds [Allocated]
3. How much of each resource is available in the system for each person [Available]

So we need MAX and REQUEST. If REQUEST is given MAX = ALLOCATED + REQUEST

NEED = MAX – ALLOCATED

A resource can be allocated only for a condition.

REQUEST<= AVAILABLE or else it waits until resources are available.

Let **‘n’**be the number of processes in the system and ‘m**’**be the number of resource types.

* Available **–**It is a 1D array of size ’m’. Available [j] = k means there are k occurrences of resource type Rj.
* Maximum **–**It is a 2D array of size ‘m\*n’ which represents maximum demand of a section. Max[i,j] = k means that a process i can maximum demand ‘k’ amount of resources.
* Allocated **–** It is a 2D array of size ‘m\*n’ which represents the number of resources allocated to each process. Allocation [i,j] =k means that a process is allocated ‘k’ amount of resources.
* Need **–**2D array of size ‘m\*n’. Need [i,j] = k means a maximum resource that could be allocated.
  + Need [i,j] = Max [i,j] – Allocation[i,j]

**Boundary conditions:**

Process rarely know in advance how many resources they will need.

The number of processes change as time progresses.

Resources once available can disappear.

The available assumes processes will return their resources within a reasonable time.

Processes may only get their resources after an arbitrarily long delay.

**Test Cases:**

**GitHub Link:**