**Operating system 2 Project – Cover sheet**

Project Title :

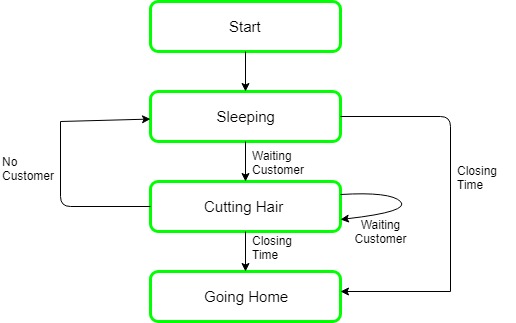
Multiple Sleeping Barber Problem

Group# ……………………………………………………..

Discussion time:- …………………………………………………….. Instructor ……………………………………………………

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Name(Arabic) | Bounce | Minus | Total Grade | Comment |
| 201900353 | سمير احمد محمد |  |  |  |  |
| 20180196 | حسام محمد يوسف |  |  |  |  |
| 201900693 | محمد صبري عبدالسلام |  |  |  |  |
| 201900743 | محمد هاني محمد الشريف |  |  |  |  |
| 201900901 | نسرين خالد فايز |  |  |  |  |
| 201900615 | محمد ابراهيم محمد سليمان |  |  |  |  |
| 20190037 | شوقى سيد شوقى |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Critrial |  | | | | | | | | Grade | | | Team Grade | Comment |
| Documentation | Solution pseudocode | | | | | | | |  | 1 | |  |  |
| Examples of Deadlock | | | | | | | |  | 1 | |  |  |
| How did solve deadlock | | | | | | | |  | 1 | |  |  |
| Examples of starvation | | | | | | | |  | 1 | |  |  |
| How did solve starvation | | | | | | | |  | 1 | |  |  |
|  | Explanation for real world application and how did apply the problem | | | | | | | |  | 1 | |  |  |
| GitHub | Upload project files | | | | | | | |  | 2 | |  |  |
| Submitted before discussion time (shared GitHub project link with TA and Dr) | | | | | | | |  | 1 | |  |  |
| Only one contribution | | | | | | | | -1 | | |  |  |
| Implementation | Run correctly (correct output) | | | | | | | |  | 5 | |  |  |
| Run but with incorrect output | | | | | | | | -3 | | |  |  |
| Not run at all (error and exceptions) | | | | | | | | -8 | | |  |  |
| Free from Deadlock | | | | | | | |  | 3 | |  |  |
| Free from deadlock in some cases and not free in other cases | | | | | | | | -2 | | |  |  |
| Free from Starvation | | | | | | | |  | 2 | |  |  |
| Free from Starvation in some cases and not free in other cases | | | | | | | | -1 | | |  |  |
| Apply problem to real world application | | | | | | | |  | 6 | |  |  |
| Total |  | Total grade for Team | | | | |  | |  | 25 |  |  |  |
|  | Total Team Grade(after adjustment) | | | | | |  |  | 25 |  |  |  |
| Bounce | Multithreading GUI Based Java Swing | | | | | | | | +5 | | |  |  |
| Multithreading GUI Based Java | | | | | | | |  |  |
| Swing( | | adjustment | | ) | | | |
| Multithreading GUI Based JavaFX | | | | | | | | +10 | | |  |  |
| Multithreading GUI Based | | | | | | | |  |  |
| JavaFX( | | | adjustment | | ) | | |
| Bounce Graphic and animation | | | | | | | | +5 | | |  |  |
| Total with  Bounce |  | Total Team Grade | | |  | | | |  | | |  |  |
|  | Total Team Grade(after adjustment) | | | | | |  |  | | |  |  |



 Solution : The solution to this problem includes three semaphores.First is for the customer which counts

the number of customers present in the waiting room (customer in the barber chair is not included because he is not waiting).

Second, the barber 0 or 1 is used to tell whether the barber is idle or is working,

And the third mutex is used to provide the mutual exclusion which is required for the process to execute.

In the solution,

the customer has the record of the number of customers waiting in the waiting room if the number of customers is equal to

the number of chairs in the waiting room then the upcoming customer leaves the barbershop.

When the barber shows up in the morning, he executes the procedure barber,

causing him to block on the semaphore customers because it is initially 0.

Then the barber goes to sleep until the first customer comes up.

When a customer arrives, he executes customer procedure the customer acquires the mutex for entering the critical region,

if another customer enters thereafter,

the second one will not be able to anything until the first one has released the mutex.

The customer then checks the chairs in the waiting room if waiting customers are less then

the number of chairs then he sits otherwise he leaves and releases the mutex.

If the chair is available then customer sits in the waiting room and increments the variable waiting value

and also increases the customer’s semaphore this wakes up the barber if he is sleeping.

At this point, customer and barber are both awake and the barber is ready to give that person a haircut.

When the haircut is over, the customer exits the procedure and if there are no customers in waiting room barber sleeps.

the following pseudocode guarantees synchronization between barber and customer and is deadlock free,

but may lead to starvation of a customer. The problem of starvation can be solved with a first-in first-out (FIFO) queue.

The semaphore would provide two functions: wait() and signal(),

which in terms of C code would correspond to P() and V(), respectively.

# The first two are mutexes (only 0 or 1 possible)

Semaphore barberReady = 0

Semaphore accessWRSeats = 1 # if 1, the number of seats in the waiting room can be incremented or decremented

Semaphore custReady = 0 # the number of customers currently in the waiting room, ready to be served

int numberOfFreeWRSeats = N # total number of seats in the waiting room

def Barber():

while true: # Run in an infinite loop.

wait(custReady) # Try to acquire a customer - if none is available, go to sleep.

wait(accessWRSeats) # Awake - try to get access to modify # of available seats, otherwise sleep.

numberOfFreeWRSeats += 1 # One waiting room chair becomes free.

signal(barberReady) # I am ready to cut.

signal(accessWRSeats) # Don't need the lock on the chairs anymore.

# (Cut hair here.)

def Customer():

while true: # Run in an infinite loop to simulate multiple customers.

wait(accessWRSeats) # Try to get access to the waiting room chairs.

if numberOfFreeWRSeats > 0: # If there are any free seats:

numberOfFreeWRSeats -= 1 # sit down in a chair

signal(custReady) # notify the barber, who's waiting until there is a customer

signal(accessWRSeats) # don't need to lock the chairs anymore

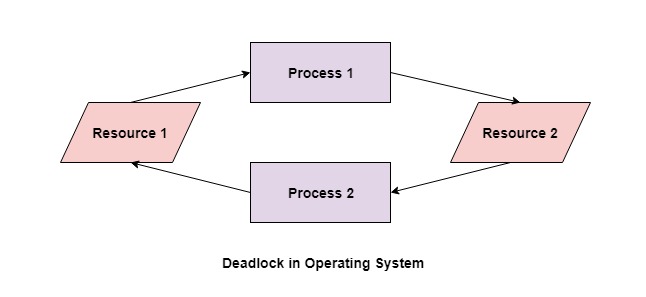
wait(barberReady) # wait until the barber is ready

# (Have hair cut here.)

else: # otherwise, there are no free seats; tough luck --

signal(accessWRSeats) # but don't forget to release the lock on the seats!

# (Leave without a haircut.)



**#when the deadlock occurs in our project**

the deadlock occurs when the barbeer waits for his customer and the customer also waits for the barber to call out his name

**#the solution**

* 1. Ensure that the system will neverenter a deadlock state:Deadlock prevention
  2. Deadlock avoidance
  3. Allow the system to enter a deadlock state and then recover
  4. Ignore the problem and pretend that deadlocks never occur in the system.

Deadlock Prevention

1. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.
2. Invalidate one of the four necessary conditions for deadlock:
3. Mutual Exclusion–not required for sharable resources (e.g., read-only files); must hold for non-sharable resources

**There are two ways to eliminate hold and wait:-**

1. **By eliminating wait**:

The process specifies the resources it requires in advance so that it does not have to wait for allocation after execution starts.

**For Example:** Process1 declares in advance that it requires both Resource1 and Resource2

1. **By eliminating hold**:

The process has to release all resources it is currently holding before making a new request.

**For Example:** Process1 has to release Resource2 and Resource3 before making request for Resource1

1. For example, the lock ordering in the deadlock example could be
2. F(first mutex) = 1
3. F(second mutex) = 5
4. We can now consider the following protocol to prevent deadlocks:
5. Each thread can request resources only in increasing order of enumeration. That is, a thread can initially request an instance of a resource—say, Ri. After that, the thread can request an instance of resource Rjif and only if F(Rj) >F(Ri).
6. For example, using the function defined above, a thread that wants to use both first mutex and second mutex at the same time must first request first mutex and then second mutex.

The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

Resource-allocation states defined by the number of available and allocated resources and the maximum demands of the processes

\*In our project we use the banker algorithm

**#when the starvation occurs in our project**

the starvation happens to the customer that has to wait for along time because he doesnt know that

the previous customer has already done

and there is another suctomer taking his turn

**#the solution**

**Aging is a technique of gradullay increasing the priority of processes that wait for a long time**

**For example if priority range from 127 to 0 we could increase the priority of a waiting process by 1 every 15 minutes eventually ,even a process with an initial priority of 127 would take no more than 32 hours to age a piriority -0 process**