# CS4328: Homework #3

Due on April 11, 2019 at 11:59PM

Jamal Rasool

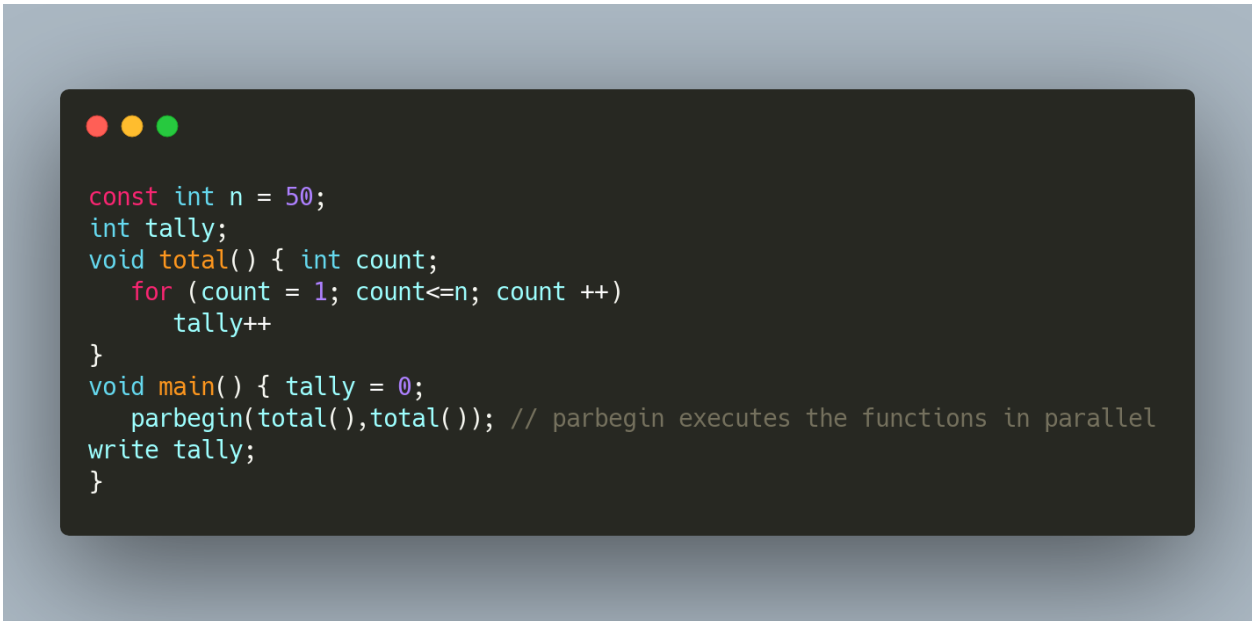
In Collaboration with:

Kristof York, John Daloni, Fernando Valdez

Jamal Rasool
Tuesday, April 9, 2019

# Problem 1

**Consider the following program running on a single processor machine: [10 pts]**



```
const int n = 50;
int tally;
void total() { int count;
   for (count = 1; count<=n; count ++)
      tally++
}
void main() { tally = 0;
   parbegin(total(),total()); // parbegin executes the functions in parallel
write tally;
}
```

**(a) What is the lower bound and upper bound on the final value of the shared variable tally in this concurrent program? Assume processes can execute at any relative speed and that the value can only be incremented after it has been loaded into a register by a separate machine instruction.**

Lower bound = 2
Higher bound = 100

The following can result in account of 50, p1 reads the tally, p2 then also reads the tally, p1 will increment and then p2 increments and stores the data. When this is repeated 50 times, the tally will become 50 as a result since p2's store will overwrite P1's. This is not the lower bound however it is possible for the count to be 2 as stated above. The way this happens is in this following order:

1. p1 loads tally to a register
2. p2 loads tally and increments 49 times stores 49.
3. p1 increments its register and stores 1 in tally
4. p2 ready's tally and stores it in a register
5. p1 loads tally and increments it 49 times, stores 50.
6. p2 increments tally, stores 2.
7. The upper bound of 100 will occur if for example p1 always does the entire load increment and store before p2.

Jamal Rasool
Tuesday, April 9, 2019

**(b) If we have N processes executing the function total() in parbegin as opposed to 2, what effect will this modification have on the range of final values of tally?**

Yes, the effect of this is that the final value of the arbitrary numbers of process are executed at parallel. In the case that there are N concurrent processes, then the lower bound is unaffected and the upper bound is N*50.

## Problem 2

**Show that, if the wait() and signal() semaphore operations are not executed atomically, then mutual exclusion may be violated. [10 pts]**

Within the case a wait operation atomically decrements the value associated with a semaphore. If you have two wait operations are executed on a semaphore, when it has a value of 1. In the case that the two operations are not performed atomically, we see that it is possible that both operations might end up decrementing the semaphore values, thereby violating mutual exclusion.

## Problem 3

**The Cookie Eater Problem. [20 pts]**

```
Mutex lock = lock
Bool IngredientReady;
Bool ingredient Consumed
Int sugar, flour, milk;

void Agent()
{
   lock.acquire();
   while(1){
     while (!((Milk == 0) &&  (Flour == 0) && (Sugar == 0))){
        IngredientConsumed.wait(&lock);
     }
     chooseIngredients(&Milk, &Flour, &Sugar);
     IngredientReady.broadcast(&lock);
   }
   lock.release();
}

void cookieEater()
{
   lock.acquire();
   while(1){
     while( !( (Milk ==1) && (Sugar ==1) )){
       IngredientReady.wait(&lock);
     }
    //Initialization values
    Milk = 0;
    Sugar = 0;
    IngredientConsumed.signal(&lock);
   }
   lock.release();
}
```

Jamal Rasool
Tuesday, April 9, 2019

# Problem 4

**Consider the following snapshot of a system. Answer the following questions: [20 pts]**

Current available:

| R1 | R2 | R3 | R4 |
|---|---|---|---|
| 2 | 1 | 0 | 0 |

Current Allocation:

| P-R | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| P1 | 0 | 0 | 1 | 2 |
| P2 | 2 | 0 | 0 | 0 |
| P3 | 0 | 0 | 3 | 4 |
| P4 | 2 | 3 | 5 | 4 |
| P5 | 0 | 3 | 3 | 2 |

Maximum Claim:

| P-R | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| P1 | 0 | 0 | 1 | 2 |
| P2 | 2 | 7 | 5 | 0 |
| P3 | 6 | 6 | 5 | 6 |
| P4 | 4 | 3 | 5 | 6 |
| P5 | 0 | 6 | 5 | 2 |

Current Allocation

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| $P_1$ | 0 | 0 | 1 | 2 |
| $P_2$ | 2 | 0 | 0 | 0 |
| $P_3$ | 0 | 0 | 3 | 4 |
| $P_4$ | 2 | 3 | 5 | 4 |
| $P_5$ | 0 | 3 | 3 | 2 |

Currently Avdible

| $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|
| 2 | 1 | 0 | 0 |

Need

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| $P_1$ | 0 | 0 | 0 | 0 |
| $P_2$ | 0 | 7 | 5 | 0 |
| $P_3$ | 6 | 6 | 2 | 2 |
| $P_4$ | 2 | 0 | 0 | 2 |
| $P_5$ | 0 | 3 | 2 | 0 |

Max claim

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| $P_1$ | 0 | 0 | 1 | 2 |
| $P_2$ | 2 | 7 | 5 | 0 |
| $P_3$ | 6 | 6 | 5 | 6 |
| $P_4$ | 4 | 3 | 5 | 6 |
| $P_5$ | 0 | 6 | 5 | 2 |

Jamal Rasool
Tuesday, April 9, 2019

**(a) Compute what each process might still need.**

Need

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| $P_1$ | 0 | 0 | 0 | 0 |
| $P_2$ | 0 | 7 | 5 | 0 |
| $P_3$ | 6 | 6 | 2 | 2 |
| $P_4$ | 2 | 0 | 0 | 2 |
| $P_5$ | 0 | 3 | 2 | 0 |

**(b) Is this system safe or unsafe state? Why?**

B)

Currently available

| | $R_1$ | $R_2$ | $R_3$ | $R_4$ |
|---|---|---|---|---|
| $P_1$ | 2 | 1 | 0 | 0 |
| $P_4$ | 2 | 1 | 1 | 2 |
| $P_5$ | 4 | 4 | 6 | 6 |
| $P_2$ | 4 | 7 | 9 | 8 |
| $P_3$ | | | | |

$P_1 \rightarrow P_4 \rightarrow P_5 \rightarrow P_2 \rightarrow P_3$

← Everything can complete.

The system is safe for the following combination P1 -> P4 -> P5 -> P2 -> P3, this is due to none of the processes becoming deadlock if you run it in that order.

**(c) Is this system currently deadlocked? Why or why not?**

No if the system follows the combination above the system can finish the sequence without any issues.

**(d) Which process, if any, are or may become deadlocked**

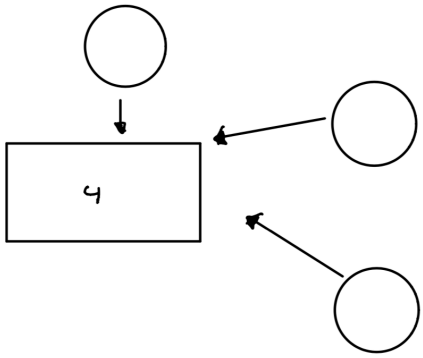
No processes are deadlocked with the above sequence, with the values in step e, we see that process 2 and process 3 are the potential deadlocks in that case.

**(e) If a request from P3 arrives (0,1,0,0), can that request be safely granted? If granted, what would be the resulting state (safe, unsafe, deadlocked)? Which processes, if any, are or may become deadlocked if this request was immediately granted?**

P1 -> p4 -> p5 sequence which would lead to deadlock at p2 and p3. The resulting state would be if the request is granted would be a deadlock due to there being not enough resources in available to handle the request.

## Problem 5

**(a) Three processes share four resources units that can be reserved and released only one at a time. Each process needs a maximum of two units. Show that a deadlock cannot occur. [5pts]**




In the case of the figure above the maximum graph illustrates the initial state of the system. Due to all of the requests being for a single unit, we can observe is that this system is proceeding to be deadlocked. With needing to have four units allocated and with all the processes holding resources; we only have one unit since otherwise what we would see is that the processes would request no further resources before releasing. If all the resources are allocated, then at least one of the three process has to be allocated two units. Hence that process is not blocked, and therefore the system would be unable to deadlock.

Jamal Rasool
Tuesday, April 9, 2019

**(b) N processes share M resource units that can be reserved and released only one at a time. The maximum need of each process does not exceed M, and the sum of all maximum needs in less than M+N. Show that a deadlock cannot occur. [5 pts]**

Suppose we set the following variables to these values below:
N = Sum of all Need(i)
S = Sum of all Allocation(i)
M = Sum of all Max(i)

By using a proof by contradiction one can assume that the system is not deadlock free. In this case if there exists a deadlock state, then S = m due to there is only one type of resource and with that resources can be request and released one at a time. Therefore, from the condition N + S = (M < m + n), we must break it into separate parts. We get N = m < m + n. So, we get N < n. It shows that at least one process at a specific point = 0. From part a above, a process can release at least one resource. So there are n-1 processes sharing m resources now, therefore we find that both conditions still hold in which no process will wait permanently, so hence no deadlock.