CS4328 & CS5305: Homework #3

Due on April 12, 2024

Mina Guirguis

PLEASE READ: You may discuss this problem set with other students. However, you must *write up your answers on your own.* You must also write the names of other students you discussed any problem with. Each problem has a different weight. Please state any assumptions you are making in solving a given problem. *Show your work.* Late assignments will not be accepted with prior arrangements. By submitting this assignment, you acknowledge that you have read the course syllabus.

Problem 1

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

```
const int n = 50;
int tally;

void Inc() {
   int count;
   for (count = 1; count<=n; count ++)
      tally++
}

void Dec() {
   int count;
   for (count = 1; count<=n; count ++)
      tally--
}

void main() {
   tally = 0;
   parbegin(Inc(),Dec()); // 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/decremented after it has been loaded into a register by a separate machine instruction.

The minimum value tally is reached when all increments happen before decrements. For above, the final value would be n because Inc() increases tally n times and Dec() doesn't have chance to decrease it. Lower bound = -n = -50

The maximum value tally can be reached when all decrements happen before any increment. For above, final value would be -n because Dec() decreases tally n times, and Inc() does not have a chance to increase it. Upper bound = n = 50

(b) What is the lower and upper bounds if we execute the following process?

```
void main() {
   tally = 0;
   parbegin(Inc(),Inc()); // parbegin executes the functions in parallel
   write tally;
}
```

If 3 instances of the Inc() function in parallel then:
minimum value tally can be reached when all increments happen before decrement. In this case, final value would be 3n because Inc() increases tally n times and Dec() doesn't have chance to decrease it. Lower bound = n = 50
maximum value tally can be reached when all decrements happen before an increment. For this case, final value would be -2n because Dec() decreases tally n times and one Inc() increases it n times.
Upper bound = 2

Problem 2

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

A wait operation will automatically decrement values associated with a semaphore. So, if two wait operations execute on a semaphore when value is 1 and the two operations are not performed automatically, then it is possible that both operations might proceed to decrement semaphore value. This then violates mutual exclusion.

ex: Suppose we have P1 and P2 with a shared resource. Resource is protected with a semaphore and semaphore set to 1. This indicates that the resource is available. The wait() operation decrements semaphore and Signal() increments. A potential problem when these operations aren't atomic,

```
//P1
wait(semaphore); //not atomic
// critical section
// Access shared resource
signal(semaphore);

//P2
wait(semaphore); //not atomic
// critical section
// Access resource
signal semaphore;
```

- If operations aren't executed atomically, it can lead to a race condition where multiple processes seemingly acquire semaphore simultaneously, violating mutual exclusion

Problem 3

The Under-equipped Mechanic Shop problem is a synchronization problem in which 3 mechanics work in an under-equipped shop and are forced to share 3 available tools (A, B and C). The 3 mechanics continuously repair parts and take breaks after fixing a part. Mechanic 1 needs the 3 tools to repair a part, Mechanic 2 only needs A and C to repair a part, and Mechanic 3 only needs B and C to repair a part. Write a program/pseudocode to synchronize between these 3 mechanics. Explain whether a deadlock can occur or not in your program.**[20 pts]** To avoid deadlock, I'm going to use semaphores

```
SemTool A = 1 // Semaphore for tool A
SemTool B = 1 // Semaphore for tool B
SemTool C = 1 // Semaphore for tool C

Mechanic 1:

while true:
    wait(A)
    wait(B)
    wait(C)
    // Repair part using tools A, B, and C
    signal(C)
    signal(B)
    signal(A)
    // Take a break after done

Mechanic 2:
while true:
    wait(A)
    wait(C)
    // Repair part using tools A and C
    signal(C)
    signal(A)
    // Take a break after done

Mechanic 3:
while true:
    wait(B)
    wait(C)
    // Repair part using tools B and C
    signal(C)
    signal(B)
    // Take a break after done
```

-start by initializing semaphore to each tool of 1 value
-mechanics will try to grab tools and release after finishing
-simple approach where mechanics acquire and release tools in a fixed order.

Deadlock: Yes possibly because each mechanic might hold one or more tools while waiting for others. Like if Mechanic 1 holds A and B and is waiting for C, mechanic 3 could have C and be waiting for B. This leads to deadlock where neither mechanic will proceed.

To ensure no deadlock will occur, I could of enforced strict ordering in tool acquisition and release. Also could do Resource heirarchy solution and a acquire all or none at an instance.

Problem 4

Consider the following snapshot of a system. Answer the following questions: **[24 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

(a) What are the total number of resources present in the system?

total = available + allocation

R1	R2	R3	R4
6	7	12	12

(b) Compute what each process might still need. Need = max claim - allocation

Allocation

	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

Max

	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

Need

	R1	R2	R3	R4
P1	0	0	0	0
P2	0	7	5	0
P3	6	6	2	2
P4	2	0	0	2
P5	0	3	2	0

all finish

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

work = 2 1 0 0
need1 = 0 0 0 0
work = 2 1 0 0
+ 0 1 2 2
= 2 2 2 2
*will finish

need4 = 2 0 0 2
work = 2 2 2 2
+ 2 3 5 4
4 5 7 6
*will finish

need5 = 0 3 2 0
work = 4 5 7 6
+ 0 6 5 2
4 11 12 8
*will finish

need2 = 0 7 5 0
work = 4 11 12 8
2 0 0 0
6 11 12 8
*will finish

need 3 = 6 6 2 2
work = 6 11 12 8
+ 0 0 3 4
= 6 11 15 12
*will finish

All will finish

System is overall safe state because Bankers safety test passed. $P_1 \to P_4 \to P_5 \to P_2 \to P_3$

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

No it isn't deadlocked because the system runs through and will remain safe if in order P_1, P_4, P_5, P_2, P_3.

(e) Which processes, if any, are or may become deadlocked?

None for above case

(f) 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?

current available: 2100 - 0100 = 2000

need 3 = 6622 - 0100 = 6522

P_3 = 0034 + 0100 = 0134

available: 2 0 0 0

Allocation

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

Max

	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

Need

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

work = 2000
+ 0012
= 2012

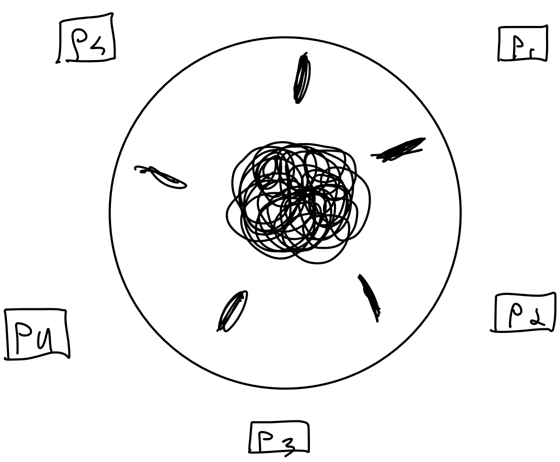
Need4
work = 2012
+ 2354
= 4366
* safe

Need5
work = 4366
+ 0332
= 4698
* not safe

— Request can't be given safely. If so, then deadlock will occur. $P_2 \to P_3$ are deadlocked at $P_1 \to P_4 \to P_3$ since not enough resources

Problem 5

Write a program/pseudo-code to show how solving the dining philosophers problem can be done by allowing each philosopher to grab both chopsticks at once. Discuss any drawbacks of your solution. **[10 pts]**



```
int mutex = 1;
int philospher[N] = {0,1,2,3,4} // # of philosphers
int main()
{
  philospher[N]=0

  do {
     wait(mutex);
     grabChopstikks()
     Signal(mutex)
     eat()
     think()
  } while(mutex)
  philospher[N]+1;
```

Drawbacks

1. Can lead to starvation
2. can lead to deadlock
3. very basic hard code
4. one philospher can eat at a time only

Problem 6

(a) Three processes share M resources units that can be reserved and released only one at a time. Each process needs a maximum of 3 units. What is the minimum value of M so that a deadlock cannot occur? **[5pts]**

	Allocation	max	need
P_1	x	3	3 - X
P_2	X	3	3 - X
P_3	X	3	3 - X

total = 6

Reserve and release = 1 → R = 1

total = 6

$R >= p(N-1)+1$ N=max need, p= # process

$M >= 3(3-1)+1 = 7$

A process can request max, complete operation and release resources. So this shows no deadlock

(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**] X = max resources req , at max X-1

$$\sum_{i=1}^{n} (X-1) < M \rightarrow \sum_{i=1}^{n} X - \sum_{i=1}^{n} 1 < M$$

$$\rightarrow \sum_{i=1}^{n} X - n < M \rightarrow \sum_{i=1}^{n} X < m+n$$

$$\text{Allocation} = \sum_{i=1}^{n} \text{Need} < m+n \rightarrow \sum_{i=1}^{n} \text{need}_i - N \rightarrow \sum_{i=1}^{n} \text{need}_i = m$$

Deadlock can't occur since there is always sufficient resources to meet max needs. Each process can be allocated its max resources, finish job, and release. Then another process can do same after. Sum of max needs is less than m+n and each at most has $\text{need}_i - 1$ $(X-1)$. Always satifies.