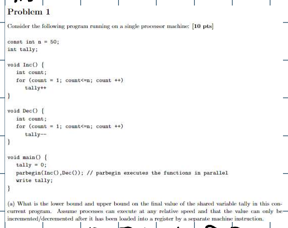
Work

Thursday, April 11, 2024 2:22 PM

1A)



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.

n = 50

Inc → ① reg1 = tally
② reg1 = reg1 + 1
③ tally = reg1

Dec → ④ reg1 = tally
⑤ reg1 = reg1 - 1
⑥ tally = reg1

①	②	③	④	⑤	⑥	0
0		1	1		0	0

Case1)

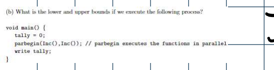
①	④	⑤	⑥	②	③	1
0	0		-1		1	
①	④	⑤	⑥	②	③	2
1	1		0		2	
⋮						
①	④	⑤	②	③		50
49	49	48	50	50		

Case 2)

④	①	②	③	⑤	⑥	-1
0	0	1	1	-1	-1	
④	①	②	③	⑤	⑥	-2
-1	-1	0	0	-2	-2	
⋮						
④	①	②	③	⑤	⑥	-50
-49	-49	-48	-48	-50	-50	

Lower bound for tally : -50
Upper bound for tally : 50

1B)



(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;
}
```

$tally_{count0} = 0$

Inc1) → ① reg1 = tally
② reg1 = reg1 + 1
③ tally = reg1

④ reg1 = tally
⑤ reg1 = reg1 + 1
⑥ tally = reg1

Case 1)

	①	④	⑤	⑥	②	③	Ⓣ
reg1	0	0	1	1	1	1	
tally	0	0	0	1	0	1	1
reg1	1	1	2	2	2	2	
tally	1	1	1	2	1	2	2
reg1	2	2	3	3	3	3	
tally	2	2	2	3	2	3	3
reg1	...						⋮
tally							
							50

Case 2)

	④	①	②	③	⑤	⑥	T
reg1	0	0	1	1	1	1	
tally	0	0	0	1	0	1	1
reg1	1	1	2	2	2	2	
tally	1	1	1	2	1	2	2
reg1							⋮
tally							⋮
reg1							⋮
tally							⋮
							50

Inc1) → ① reg1 = tally
② reg1 = reg1 + 1
③ tally = reg1

④ reg1 = tally
⑤ reg1 = reg1 + 1
⑥ tally = reg1

Case 3)

	①	②	③	④	⑤	⑥	Ⓣ
reg1	0	1	1	1	2	2	
tally	0	0	1	1	1	2	2
reg1	2	3	3	3	4	4	
tally	2	2	3	3	3	4	4
reg1	4	5	5	5	6	6	
tally	4	4	5	5	5	6	6
reg1							⋮
tally							
							100

Case 4)

P_1 ①
P_2 ④⑤⑥) — 49 times tally = 49
P_1 ②③) — talley 1
P_2 ④⑤ — talley 1
P_1 ①②③ - 49 times tally = 50
P_2 ⑥ talley = 2

Lower bound for tally : 50

100

Lower bound for tally : 50
Upper bound for tally : 2

2)

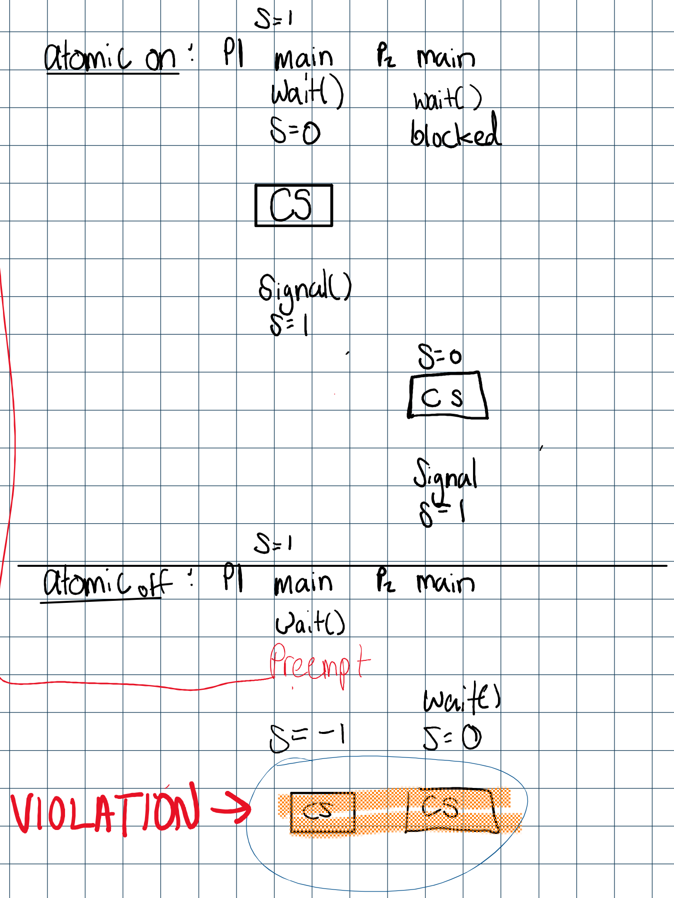
Problem 2

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

atomic ops: finish Entirely

Mutual Exclusion : no two processes can Exist in the critical section at any given time

```
wait(S)
{
   while(S≤0);
    S--;
}
Signal(S)
{
   S++;
}
```



above ↑ Shows if wait isn't atomic P1 gets preempted before it decrements then P2 is able to decrement then P1 comes back + decrements, now S=-1 + P_1 + P_2 are in the critical section @ the same time.

P3)

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]**

need
Mech 1 : A B C

program/pseudocode to synchronize between these 3 mechanics. Explain whether a deadlock can occur or not in your program.[**20 pts**]

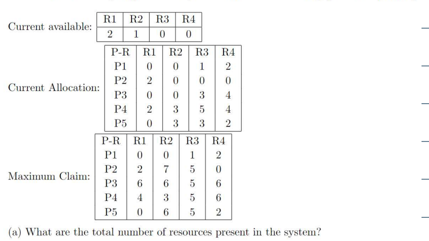
	need		
Mech 1 :	A	B	C
mech 2 :	A		C
mech 3 :		B	C

```
Sem toolA, toolB, toolC

Void Mech1Work()
{
    While(working)
    {
        wait(toola)
        Wait(toolb)
        wait(tool c)

            fix()

        Signal(toola)
        Signal(toolb)
        signal(tool c)

            break()
    }
}

Void Mech2work()
{
    While(working)
    {
        wait(toola)
        wait(tool c)

            fix()

        Si nal(toola)
        signal(tool c)

            break()
    }
}

Void Mech3work()
{
    While(working)
    {
        Wait(toolb)
        Wait(tool c)

            fix()

        Signal(toolb)
        signal(tool c)

            break()
    }
}

int main()
{
    mech mech1, mech2, mech3
    toola, toolb, toolc = 1
    mech1.mech1work()
    mech2.mech2work()
    mech3.mech3work()

        While(!workDAYOVER);

        return 0;
}
```

A deadlock may occur Because there exist a Circular wait which happens when

```
mech1 wait(a)
mech1 wait(b)
mech wait(c)
mech1 signal(a)
mech1 signal(b)
mech2 wait(a)
mech3 wait(b)
mech3 wait(c)   Lock
```

P4)


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

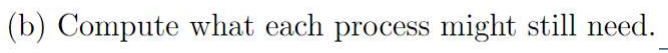
Maximum Claim:

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?

a) tot = allocation + available

R_1	R_2	R_3	R_4
6	7	12	12

b) 
(b) Compute what each process might still need.

Need = Max - allocation

ALLOCATION

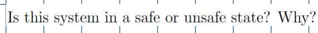
		R_1	R_2	R_3	R_4
✓	P_1	0	0	1	2
	P_2	2	0	0	0
	P_3	0	6	3	4
✓	P_4	2	3	5	4
✓	P_5	0	3	3	2

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	Finish
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	✓

C 
Is this system in a safe or unsafe state? Why?

① work = available

② Find i st

- $finish[i] \,!= true$
- $need_i \leq work$

if no i → ④

③ work = work + allocation

① work = 2 1 0 0

② $need_{i=1}$ = 0 0 0 0

③ work = 2 1 0 0
 + 0 1 2 2
 = 2 2 2 2

$finish_{i=1}$ = TRUE

② $need_{i=4}$ = 2 0 0 2

③ work = 2 2 2 2
 + 2 3 5 4

② $need_{i=5}$ = 0 3 2 0

③ work = 4 5 7 6
 + 0 6 5 2
 4 11 12 8

$finish_{i=5}$ = true

② $need_{i=2}$ = 0 7 5 0

③ work = 4 11 12 8
 + 2 0 0 0
 = 6 11 12 8

② $need_{i=3}$ = 6 6

③ work = 6 11
 + 0 0
 6 11

$finish_i$ = tr

④ all finish(i) =

2 2

12 8
3 4

15 12

ue

TRUE

if no i → ④

③ work = work + allocation
Finish[i] = TRUE
→ ②

④ all finish[i] = TRUE?
SAFE

③ work =

$$\begin{array}{rcccc} & 2 & 2 & 2 & 2 \\ + & 2 & 3 & 5 & 4 \\ \hline & 4 & 5 & 7 & 6 \end{array}$$

$finish_{i=4}$ = TRUE

$$\begin{array}{rcccc} + & 2 & 0 & 0 & 0 \\ \hline = & 6 & 11 & 12 & 8 \end{array}$$

$finish_{i} = 2$ = TRUE

The system is safe b/c it passed BANKERS SAFETY TEST $P_1 \Rightarrow P_4 \Rightarrow P_5 \Rightarrow P_2 \Rightarrow P_3$

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

No b/c if the system runs in the sequence
$P_1 \to P_4 \Rightarrow P_5 \Rightarrow P_2 \Rightarrow P_3$
it will Remain safe

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

NONE when system follows sequence

(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?

2100

Request

① if $Request_i > Need_i$ → ERROR
② if $Request_i \leq$ availible goto ③ else wait
③ $ALLOCATION_i = ALLOCATION_i + REQUEST_i$
$Need_i = Need_i - Request_i$
$availible = available - Request_i$

④ Run SAFE, if safe: grant ? deny

① ✓
② ✓
③

$$\begin{array}{rcccc} & 0 & 0 & 3 & 4 \\ + & 0 & 1 & 0 & 0 \\ \hline & 0 & 1 & 3 & 4 \end{array}$$

$$\begin{array}{rcccc} & 2 & 1 & 0 & 0 \\ - & 0 & 1 & 0 & 0 \\ \hline & 2 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{rcccc} & 6 & 6 & 2 & 2 \\ - & 0 & 1 & 0 & 0 \\ & 6 & 5 & 2 & 2 \end{array}$$

ALLOCATION

	R1	R2	R3	R4
P1	0	0	1	2
P2	2	0	0	0
P3	0	1	3	4

Max

	R1	R2	R3	R4
P1	0	0	1	2
P2	2	7	5	0
P3	6	6	5	6

Need

	R1	R2	R3	R4	
P1	0	0	0	0	✓
P2	0	7	5	0	
P3	6	5	2	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

P_2	2	7	5	0
P_3	6	6	5	6
P_4	4	3	5	6
P_5	0	6	5	2

P_2	0	7	5	0	
P_3	6	5	2	2	
P_4	2	0	0	2	✓
P_5	0	3	2	0	✓

AVAILABLE= 2000

① work = available

① Find i st
- finish[i] != true
- need$_i$ ≤ work

if no i → ④

③ work = work + allocation
finish[i] = TRUE
→ ②

④ all finish[i] = TRUE?
SAFE

① work = 2000

② need$_i$ = P_1

③ work =
$$\begin{array}{r} 2\,0\,0\,0 \\ +\ 0\,0\,1\,2 \\ \hline 2\,0\,1\,2 \end{array}$$

② Need$_i$ = 4

work =
$$\begin{array}{r} 2\,0\,1\,2 \\ 2\,3\,5\,4 \\ \hline 4\,3\,6\,6 \end{array}$$

Need$_i$ = 5

Work =
$$\begin{array}{r} 4\,3\,6\,6 \\ +\ 0\,3\,3\,2 \\ 4\,6\,9\,8 \\ \hline \end{array}$$

NOT SAFE

!

Request cannot be safely given

If request given, DEADLOCKED

P_2 + P_3 would be deadlocked at sequence $P_1 \rightarrow P_4 \rightarrow P_5 \rightarrow$ b/c 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 ;

int main( )
{
  Mutex = 1 ;
  do {
      wait(mutex)
       grab-both( )
      signal(mutex)
```

DRAWBACKS:

① ONLY ONE Philo can eat at a time

② Can lead to starvation

```
eat()
think()
} while(1);
```

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]**

m = tot
allocation = 3X
avail = M − 3X

	allocation	max	need
P_1	X	3	3-X
P_2	X	3	3-X
P_3	X	3	3-X

$M - 3X \geq 3$
$M - 3 \geq 3$
$M \geq 6$

X=1, Max num proc hold while still allowing 1 proc to rec all units to finish

with M=6, a process can request max & complete op & release resources so 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]**

need < M+N $\quad \sum_{i=1}^{N} Need_i < M+N$

ASSUME ALL Procs holding some resources but fewer than max

$$ALLOCATION = \sum_{i=1}^{N} Need_i - N$$

$$\sum_{i=1}^{N} need_i < N+N-N$$

$$= \sum_{i=1}^{N} need_i < M$$

$$AVAILABLE = M - \left(\sum_{i=1}^{N} Need_i - N\right) \geq 0$$

Because sum of max needs of all procs is Less than M+N & each process has at most $\text{Need}_i - 1$ Resources the remaining available resources will always be enough to satisfy the max needs of one process

Since there's always sufficient Resources to meet max needs of at least one process, a sequence of allocations will always avoid a deadlock,

EACH Proc can eventually be allocated its max resources, complete work, release & allow another proc to proceed

THUS deadlock cannot occur