Test 2

● Graded

Student

Adam Fenjiro

Total Points

70 / 100 pts

Question 1

Semaphore Basic Operation

18 / 18 pts

- ✔ **+ 3 pts** Line 1: Sem A -> 0; no other changes
- ✔ **+ 3 pts** Line 2: SemA -> -1; Waiting Set semA -> {Pc}; no other changes
- ✔ **+ 3 pts** Line 3: SemA -> 0; Waiting Set SemA -> {}; Resumed Thread -> {Pc}; no other changes
- ✔ **+ 3 pts** Line 4: SemB-> -1; Waiting Set SemB -> {Pc}; no other changes
- ✔ **+ 3 pts** Line 5: SemB-> -2; Waiting Set SemB -> {Pc,Pa}; no other changes
- ✔ **+ 3 pts** Line 6: SemB ->-1; Waiting Set SemB -> {Pc}||{Pa}; Resumed thread = {Pa,Pc}

+ 0 pts Entirely incorrect or no answer

Question 2

Semaphore Atomic

12 / 12 pts

- ✔ **+ 4 pts** Gives a completely specified, valid execution of two signal operations one each by Thread A and Thread B
- ✔ **+ 8 pts** Execution shows that W is never released from the waiting set

+ 0 pts Anything else

Question 3

Unprotected Update

Resolved 0 / 12 pts

+ 4 pts Gives a complete and valid execution of the system that runs until all processes have either completed or are blocked

+ 4 pts Recognizes that atomic instructions must be interleaved to produce the identified execution

+ 4 pts Gives a valid execution that produces the output AA

✔ + 0 pts Invalid execution or no answer

1 Thread does not loop

Regrade Request Submitted on: Mar 20

50% of the execution is correct. Starting by the wait(gob), then moving to the other process to print('A'). I understand that the other half is wrong, but 0/12 is like someone that did not answer or had it completely off. Could you please check this one out? Thanks!

The point of this question is to recognize that sometimes you need to go down to the instruction level to encounter a race condition. The execution is not only incorrect (even though it is technically correct for the first part), in its incorrectness you miss the point of the question, and instead answer a much more trivial question. You're answering the wrong question, if that makes sense, and I can't really give you points for that.

Reviewed on: Mar 21

Question 4

Fast Runner

3 / 12 pts

+ 12 pts Entirely correct

✔ + 3 pts Indicates the solution is incorrect

+ 3 pts Gives a valid execution

+ 6 pts Execution shows a process completing **phase i+1** before all processes have complete **phase i**

+ 0 pts Wrong answer and no valid execution or no answer

What is the problem? This does not show a violation of the ordering. mutex has to be signaled by P1 before P2 or P3 can go.

Question 5

Bad Wait in a Mutex

9 / 12 pts

✔ + 3 pts Indicates solution is not correct

+ 3 pts Identifies deadlock from wait in the mutex

✔ + 6 pts Gives a valid solution, completely specified

+ 0 pts Indicates solution is correct

2 Missing a counter example

Question 6

Identify Possible Executions from Output

12 / 12 pts

✔ **+ 12 pts** All output correct

+ 3 pts AABB not possible

+ 3 pts BBAA not possible

+ 3 pts BAAB possible

+ 3 pts Deadlock not possible

+ 0 pts All answers incorrect or no answers

Question 7

Create Simple Synchronization

14 / 14 pts

✔ **+ 3 pts** All shared variable updates are executed mutually exclusively

✔ **+ 3 pts** All variables (including semaphores) declared and initialized

✔ **+ 3 pts** Solution allows the output AABB

✔ **+ 5 pts** Solution **only** allows AABB

+ 0 pts No solution or solution updates shared variables non-exclusively

Question 8

Modification of Existing Solution

2 / 8 pts

+ 8 pts Entirely correct

✔ **+ 2 pts** Gives a well specified solution

+ 6 pts Gives a valid efficient solution

+ 1 pt Disallows inserter and deleter to run at the same time, but not currently with searchers

+ 0 pts No solution, poorly specified solution, incorrect solution.

+ 3 pts Valid solution but concurrency significantly diminished,

Since deleter does wait on mutex, this process can run concurrently with a deleter.

Name: Adam FENJIRO

CS 3331 Exam 2 Spring 2024

SOLUTIONS

MAKE SURE THAT YOUR COPY OF THE SOLUTION SHEET HAS 8 PAGES. You may add additional pages using the supplied paper (only). Be sure to put your name and a page number on each sheet that you add.
Please only use a landscape layout for your answers. Otherwise, it increases the grading time unnecessarily.

1. [18 points]

P_A	P_B	P_C	SemA	Sem B	Waiting Set SemA	Waiting Set Sem B	Resumed Thread
			1	0	{}	{}	
	Wait(SemA)		0		{}	{}	
		Wait(SemA)	-1		{PC}	{}	
Signal(SemA)			0		{}	{}	PC
		Wait(SemB)		-1	{}	{PC}	
Wait(SemB)				-2	{}	{PC, PA}	
	Signal(SemB)			-1	{}	{PC} or {PA}	PC or PA

2. [12 points]

Thread A instructions	Register	Thread B instructions	Register	S.count	S waiting set
				-1	{W}
LOAD register, S.count	-1				
ADD register, #1	0				
STORE register, S.count				0	
		LOAD register, S.count	0		
		ADD register, #1	1		
		STORE register, S.count		1	
if(S.count<=0){}				1	{W}
//resumes					
		if(S.count<=0){}		1	{W}
		//resumes			
					{W}

3. [12 points]

Thread A_1	Thread A_2	Thread B_1	count	output
			0	
		wait(gob)		
count++			1	
if(count==2)				
else				
write("A")				'A'
	count++		2	
	if(count==2)			
	signal(gob)			
	wait(bdone)			
	count=0		0	
count++				
if(cont==2)				
else				
write("A")				'AA'

4. [12 points]

NO, the solution is not correct. the problem here is the mutex that is signaled on line 07.

P_1 instructions	P_2 instructions	P_3 instructions	count
wait(mutex) cont ++ if (count < 3) ; } (			0 1
	Wait(mutex) count ++ if(count < 3) ; }		2
		wait(mutex count ++ if(count < 3) else ; }	3
		//problem line 7	

5. [12 points]

No, the solution is not correct. the mutex is misplaced here.
Here is a proposed solution: 2

Increment:
```
wait(increment);
wait(mutex);
count++;
signal(decrement);
signal(mutex);
```

decrement:
```
wait(decrement);
wait(mutex);
count--;
signal(increment);
signal(mutex);
```

The thing with the original solution and why it was incorrect is that they don't and won't have a matching pace on decrement and increment, where one of them is considerably faster. My solution, thus, is correct.

6. [12 points]

Type A Processes

```
wait(A)
wait(A)
write("A")
signal(A)
signal(B)
```

Type B Processes

```
wait(B)
write("B")
signal(A)
```

Semaphore	Initial Value
A	2
B	1

Output	Possible	Impossible
AABB		X
BBAA		X
BAAB	X	
No output. Deadlock.		X

7. [14 points] 'AABB'

semaphore A=1, B=0, mutex=1
shared count=0;

Thread *A*

```
while(1==1){

    wait(A);
    wait(mutex);
    count++;

    write("A");
    if(count==2){
        count=0;
        signal(mutex);
        signal(B);
    }
    else{
        signal(mutex);
        signal(A);
    }
}
```

Thread *B*

```
while(1==1){

    wait(B)
    wait(mutex)
    count++;

    write("B");
    if(count==2){
        count=0;
        signal(mutex);
        signal(A);
    }
    else{ signal(mutex);
          signal(B);
    }
}
```

8. [8 points]

inserters :

```
while (1) {                    // use mutex to reach the end of the
                                  singly linked-list
    wait (mutex);                 and make sure only
    count --;                        one insertion can
    if (count == 0)                     proceed.
        signal (listProtection)
    signal (mutex);
        // add two nodes
}
```

Inserters add new nodes at the end of the list while mutually exclusive to prelude two inserts from inserting new nodes at the same time, and one insertion can process in parallel with any number of searches. This solution should be able to achieve this.