Test 3

● Graded

Student

Adam Fenjiro

Total Points

72 / 100 pts

Question 1

General Knowledge — **15** / 18 pts

1.1 (a) — **3** / 3 pts

✔ **+ 3 pts** False

+ 0 pts True

1.2 (b) — **3** / 3 pts

✔ **+ 3 pts** False

+ 0 pts True

1.3 (c) — **3** / 3 pts

✔ **+ 3 pts** False

+ 0 pts True

1.4 (d) — **0** / 3 pts

+ 3 pts False

✔ **+ 0 pts** True

1.5 (e) — **3** / 3 pts

✔ **+ 3 pts** True

+ 0 pts False

1.6 (f) — **3** / 3 pts

✔ **+ 3 pts** False

+ 0 pts True

Question 2

Hoare Semantics

8 / 8 pts

✔ **+ 8 pts** Entirely correct

+ 2 pts 11) T1: barrier.signal()

+ 2 pts 12) T2: barrier.signal()

+ 2 pts 13) T2: x++

+ 2 pts 14) T1: # x++ T3: # x++ (OK if also T2 cnt++)

+ 0 pts Entirely wrong or no answer

Question 3

Mesa Semantics

8 / 8 pts

✔ **+ 8 pts** Entirely correct

+ 2 pts 11) T3: x++

+ 2 pts 12) T1: barrier.signal

+ 2 pts 13) T1: x++

+ 2 pts 14) T2: barrier.signal()

+ 0 pts Entirely incorrect or no answer

Question 4

Lost Signals

6 / 6 pts

✔ **+ 6 pts** Entirely correct

+ 1 pt Indicates the output is not "pingpongpingpong ..."

+ 3 pts Gives a reasonable argument with some ambiguity

+ 1 pt Gives a clear and precise argument, e.g. with an execution table that clearly depicts a deadlock

+ 0 pts Incorrect or no answer

Question 5

Condition Wait in Mesa

3 / 9 pts

+ 9 pts Entirely correct.

✔ **+ 3 pts** Gives a valid execution

+ 3 pts Execution shows producer adding to full buffer or consumer reading from empty buffer

+ 1 pt Concise: continues execution with consumer 2 coming in from the top and consumer 1 reading from full buffer

+ 2 pts Explains the error that is shown.

+ 0 pts Incorrect or no answer

1 Thread will signal here.

Question 6

Write a simple monitor

0 / 12 pts

+ 12 pts Entirely correct

+ 3 pts Completely specified monitor

+ 3 pts Correctly manages the sum of processes currently accessing the file

+ 3 pts Never allows the sum of IDs to exceed MAXIMUM

+ 3 pts Ensures at least one process can access the file when one or more processes are requesting access

✔ **+ 0 pts** No answer or incompletely specified monitor

Show initial value of variables

Question 7

Bankers Algorithm **27** / 27 pts

7.1 **(a)** **3** / 3 pts

✔ **+ 3 pts** 1,2,1

+ 0 pts Anything else

7.2 **(b)** **6** / 6 pts

+ 6 pts Entirely correct

✔ **+ 2 pts** P0: 1,2,0

✔ **+ 2 pts** P1: 2,0,4

✔ **+ 2 pts** P2: 1,1,2

+ 0 pts Entirely incorrect or no answer

7.3 **(c)** **8** / 8 pts

+ 8 pts Entirely correct

✔ **+ 2 pts** 1) Work = [1,2,1] Finish=[F,F,F]

✔ **+ 2 pts** 2) P0 Need=[1,2,0] Work=[2,4,2] Finish=[T,F,F]

✔ **+ 2 pts** 3) P2 Need=[1,1,2] Work=[2,4,4] Finish=[T,F,T]

✔ **+ 2 pts** 4) P1 Need=[2,0,4] Work=[5,6,5] Finish=[T,T,T]

+ 0 pts Entirely incorrect or no answer

7.4 **(d)** **2** / 2 pts

✔ **+ 2 pts** States that all process is in a safe state because there I a safe sequence (acceptable to say all processes can finish)

+ 1 pt Process is safe, but poor explaination

+ 0 pts Incorrect or no answer

7.5 **(e)** **8** / 8 pts

✔ **+ 8 pts** Entirely correct

+ 1 pt Checks REQ<=Available

+ 1 pt Checks REQ <= Need

+ 4 pts Argues by safety algrithm no process can finish from the available

+ 2 pts States that the request cannot be granted.

+ 0 pts Incorrect or no answer

Question 8

Deadlock **5** / 12 pts

8.1 **Necessary Conditions** **0** / 6 pts

+ 6 pts Entirely correct

+ 3 pts States the system cannot deadlock and gives a general argument.

+ 3 pts Indicates circular wait is avoided. Hold-and-wait OK since not specified how fork and knife are returned.

✔ **+ 0 pts** States it will deadlock or no answer

Cannot make an that it will not deadlock using an example execution.

8.2 **Deadlock Sequence** **5** / 6 pts

✔ **+ 6 pts** Entirely correct

+ 3 pts States the system can deadlock and gives a general argument

+ 3 pts Gives a clear sequence of operations that lead to deadlock in a tabular format.

+ 0 pts Indicates cannot deadlock or no answer.

– 1 pt Show the waits to complete the deadlock and explain where philosophers are seated.

Name: Adam FENJIRO

CS 3331 Exam 3 Spring 2025

SOLUTIONS

MAKE SURE THAT YOUR COPY OF THE SOLUTION SHEET HAS 10 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.

1. (a) [3 points] One thread in a process can never write to the stack of another thread within the same process.
   - ○ True
   - ● False

   (b) [3 points] A routine that is thread-safe is also reentrant.
   - ○ True
   - ● False

   (c) [3 points] Under MESA semantics, when process P_i signals a condition, the awoken process gets the monitor.
   - ○ True
   - ● False

   (d) [3 points] A system that is an unsafe state will eventually deadlock.
   - ● True
   - ○ False

   (e) [3 points] If a system contains a deadlock, then its Resource Allocation Graph will contain a cycle.
   - ● True
   - ○ False

   (f) [3 points] Suppose a monitor has two procedures *ProcA* and *ProcB*. Then only one thread can execute the code of *ProcA* at a time and only one thread can execute the code of *ProcB* at a time, but there can be one thread executing the code of *ProcA* at the same time that a different thread is executing the code of *ProcB*.
   - ○ True
   - ● False

2. (a) [8 points]

Time	T_1	T_2	T_3	cnt
1				0
2	cnt++			1
3	if (cnt<n)			
4	barrier.wait()			
5		cnt++		2
6		if (cnt<n)		
7		barrier.wait()		
8			cnt++	3
9			if (cnt<n) else	
10			barrier.signal()	
11	barrier.signal()			
12		barrier.signal()		
13		X++		
14	#X++		#X++	

(b) [8 points]

Time	T_1	T_2	T_3	cnt
1				0
2	cnt++			1
3	if (cnt<n)			
4	barrier.wait()			
5		cnt++		2
6		if (cnt<n)		
7		barrier.wait()		
8			cnt++	3
9			if (cnt<n) else	
10			barrier.signal()	
11			X++	
12	barrier.signal()			
13	X++			
14		barrier.signal()		

3. [6 points]

Ping 1	Ping 2	Pong 1	Pong 2	start	output
				1	

if(start==0)wait(gping)
start = 0
while("ping")
signal(gpong)
				0	"ping"

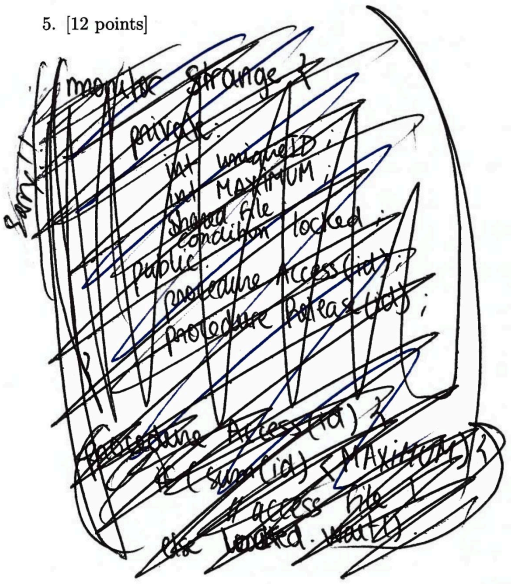
	if(start==0)wait(gping)				
		wait(gpong)			

The output will not always be "pingpongpingpong...".
In the execution I provided above, the process Ping1 takes over the monitor, the the process Ping2 takes over the monitor to then wait on gping, then the process Pong1 takes the monitor to then wait on gpong. Since the process Ping waits on a signal Pong and no other proces can continue execution, then the system will deadlock.

4. [9 points]

Time	C_1	C_2	P_1
1			
2	if (!full)		
3	wait(notEmpty)		
4			if (full)
5			buffer[0]=item
6			full=true
7			signal(notEmpty)
8	item = buffer[0]		
9	full = false		
10		wait(full)	
11		wait(not Empty)	
12			if(full)
13			buff[0] = item
14			full = true
15			signal (not Empty)
16		item = buffer[0]	
17		full = false	
18			if (full)

5. [12 points]



```
monitor Strange {
    private:
        int uniqueID
        int MAXIMUM
        shared file
        condition locked
    public:
        procedure Access(id) {
            if(sum(id) < MAXIMUM)
                //access file
                signal(locked)
            else wait(locked)
        }
        procedure Release(id) {
            signal(locked)
        }
}
```

example:
provided in instructions → Currently available (A) = 5 − (1 + 3) = 1 (sum allocation (A))

Still Needs (A) = Max(A) − Allocation (A)
example: still Needs (A) of P0 = 2 − 1 = 1

6. (a) [3 points] Use the table below.

(b) [6 points] Use the table below.

	Allocation			Max			Still Needs			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	1	2	1	2	4	1	1	2	0	1	2	1
P1	3	2	1	5	2	5	2	0	4			
P2	0	0	2	1	1	4	1	1	2			

(c) [8 points]

Process	Need for Finishing Process	Work	Finish
		[1, 2, 1]	[F, F, F]
P0	[1, 2, 0]	[1,2,1] + [1,2,1] = [2, 4, 2]	[T, F, F]
P2	[1, 1, 2]	[2,4,2] + [0,0,2] = [2, 4, 4]	[T, F, T]
P1	[2, 0, 4]	[2, 4, 4] + [3, 2, 1] = [5, 6, 5]	[T, T, T]

(d) [2 points]

yes, this system is in a safe state.
This is because all the processes finished execution without deadlock.

(e) [8 points] The table from parts a-c is repeated below for convenience.

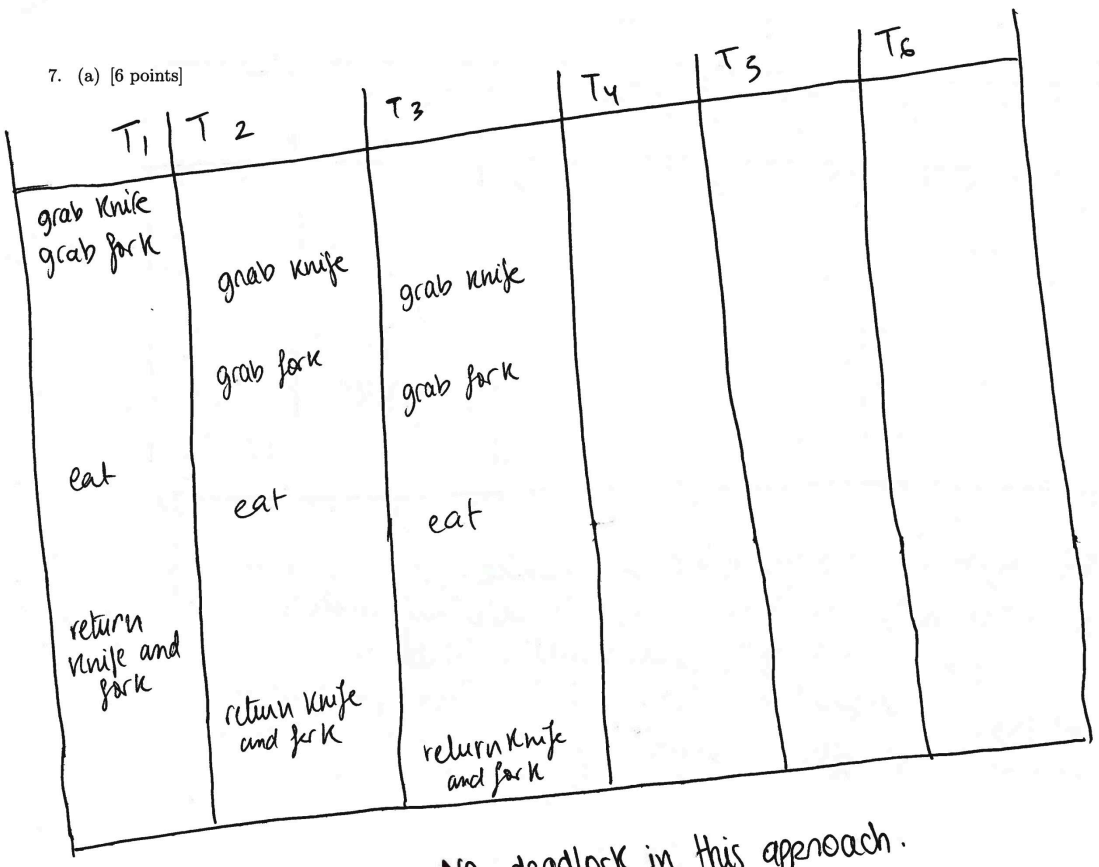
	Allocation			Max			Still Needs			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	1	2	1	2	4	1	1	2	0	0	2	1
P1	3	2	1	5	2	5	2	0	4			
P2	1	0	2	1	1	4	0	1	2			

safety check:
1. P_2 request $[1,0,0]$ < P_2 needs $[1,1,2]$
2. P_2 request $[1,0,0]$ < available $[1,2,1]$
then, yes! we can proceed to next step.
3. Modify the table to the new allocation of P_2 and needed resources of P_2
4. Modify currently available resources.

Process	Need for Finishing Process	Work	Finish
		[0, 2, 1]	[F, F, F]

Since there are no available resources for any of the processes, we cannot grant the new P2 [1, 0, 0] request.

7. (a) [6 points]



No deadlock in this approach.

(b) [6 points]

T_1	T_2	T_3	T_4	T_5	T_6
grab fork	grab fork	grab fork	grab knife	grab knife	grab knife

In this approach, we will have a deadlock. This is because we have circular waiting where every process is waiting for a resource held by the next member in the chain. The execution above is a good example.