B 1.

TENTAMEN (EXAMINATION)

Tentamensdatum/*Examination date:* 2019-08-26
(åå-mm-dd/*yy-mm-dd*)

	Ifylles av student	Ifylles av vakt
AID-nummer ***AID number***	1 3 7 1	1 3 7 1
	Completed by student	*Completed by supervisor*

Utbildningskod/*Education code:* TPDB68 **Modul/*Module:*** TEN1

Kursnamn/*Course title:* Concurrent programming and operating systems

Institution/*Department:* IDA

Jag intygar att varken mobil eller något annat otillåtet hjälpmedel finns tillgängligt under tentamen.
I confirm that no mobile or other non-permitted aids are available during the examination. ☒

Inlämnat: antal lösblad 9 **tentamensformulär** ☐
Enclosed: number of sheets ***exam booklet***

Markera behandlade uppgifter med X/*Mark tasks attempted with an X*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
X här/*here*	X	X	X	X	X	X									
Erhållna poäng *Points obtained*	6	6	3	4	3	1.5									
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
X här/*here*															
Erhållna poäng *Points obtained*															

Anvisningar/*Instructions*

1. Skriv AID-nummer, datum, utb.kod, modul på varje blad som lämnas in/*Write AID number, date, edu.code and module on every sheet that is handed in*
2. På varje papper får högst en uppgift lösas om inget annat anges/
Maximum one task per sheet unless otherwise instructed
3. Skriv endast på papprets ena sida om inget annat anges/
Use only one side of each sheet unless otherwise instructed
4. Numrera de papper som lämnas in/*Number every sheet that is handed in*
5. Använd inte röd penna/*Do not use a red pen/pencil*

Sen inlämning ☐
Late hand in

Klockslag _____
Time

Orsak _____
Reason

_____

∑ Poäng/*Points:* 23,5 **Betyg/*Grade:*** 3

Examinator/*Examiner:* _____

LINKÖPINGS UNIVERSITET

AID-nr: 1371 **Date:** 2019-08-26 **Page number:** 1

Course code: TDDB68 **Module:** TEN1

Multiple choice form for answering question 1. Please put X:s in the appropriate cells:

	A	B	C	D	
1 a)			X		1
1 b)				X	1
1 c)			X		0
1 d)		X	X		0
1 e)			X	X	1
1 f)	X		X		1
1 g)		X			0
1 h)		X	X	X	1
1 i)		X			0
1 j)	X		X		1

/8

Optional: if you feel you need to clarify your interpretation of the question you can do so here. This is not needed if your answer is correct.

1 a)	4 times
1 b)	A safe state means no deadlock can occur while in that state.
1 c)	
1 d)	
1 e)	
1 f)	
1 g)	
1 h)	
1 i)	
1 j)	

2. a) If available connections is 1 then two threads running concurrently could pass the while-loop and then decrement N_available_connects by two, making it -1. This would lead to additional threads passing the while-loop even though there are no available connections. 1

b)

```
bool var_lock=false;

connection *void handle_connection_request(void) {
    bool tmp=false;
    while (N_available_connects==0) {
        While(tmp==false) {
            atomic_swap(tmp, var_lock); }
        tmp=false; }
    N_available_connects=N_available_connects-1;
    return make_new_connection;
}

Void close_connection(connection *c) {
    release_connection(c);
    N_available_conneds=N_available_connects+1;
    Var_lock=true;
}
```

2. b) The system is now abscent of race conditions.
If there are 0 available connects and several threads are requesting to connect they will wait in the while-loop. Thanks to the atomic_swap function it is ensured that only one thread will leave this wait-loop when a connect becomes available. 3

---

c) It's not fair. A thread could potentially skip ahead the waiting threads if it requests connection in the same moment that a connection becomes avilable.
Furthermore, the thread that gets to leave the wait-loop when a connection becomes available is the first thread to execute atomic_swap(). Age and priority doesn't matter in this situation. 1

---

d)
```
handle_connection_request()
{ bool tmp=false;
  while(no connections available) {
      while(tmp==false) {
          block thread;                <----------
          atomic_swap(tmp, var_lock); }
      tmp=false; }
  available connects +1;
  return make new connection;
}
```

Examinator/*Examiner:* ——— /

2. d)

```
close_connection()
{   release_connection;
    available connections +1;
    var_lock = true;
    unblock thread;
```

By blocking the process in the waiting loop until a connection becomes available.

lp

/6

3.

allocated (max)

	R_1	R_2	R_3
P_1	0(4)	2(2)	0(1)
P_2	0(3)	2(4)	0(1)
P_3	3(3)	1(2)	3(4)
P_4	1(3)	0(0)	2(3)

Need = max - allocated

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

available = [1,2,4]

First check if $request_{P_1} \leq available$: ✓

[0,0,1] ≤ [1,2,4] is true so lets pretend we grant process P_1 its request of resources.
This gives us the new tables: is request ≤ need?

allocated (max)

	R_1	R_2	R_3
P_1	0(4)	2(2)	1(1)
P_2	0(3)	2(4)	0(1)
P_3	3(3)	1(2)	3(4)
P_4	1(3)	0(0)	2(3)

need

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

available

R_1	R_2	R_3
1	2	3

Now lets use the safety algorithm to determine if we are in a safe state.
We can grant P_3 its needed resources to complete its task. Then P_3 can release all its resources so that available = [4,3,6].

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1371	Datum: *Date:* 2019-08-26	Blad nummer: *Sheet number:* 6
Utbildningskod: *Education code:* TDDB68	Modul: *Module:* TEN1	

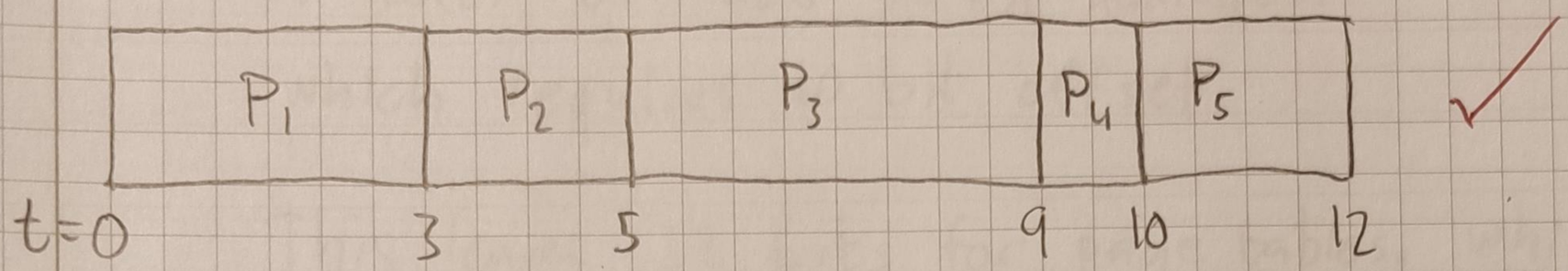
3 Now, P_1, P_2 and P_4 can their needed resources in turn and complete their tasks.

The safety algorithm succeeds and we are in a safe state.
Therefor it is safe to grant P_1 its request.

The sequence for a safe allocation of resources can be $<P_3, P_1, P_2, P_4>$ why P_1 and P_2 before P_4?

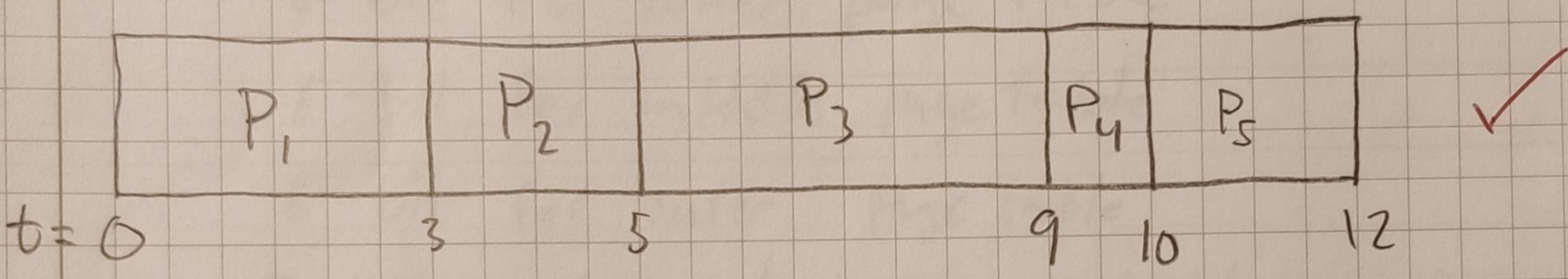
4 b)

(i) FIFO



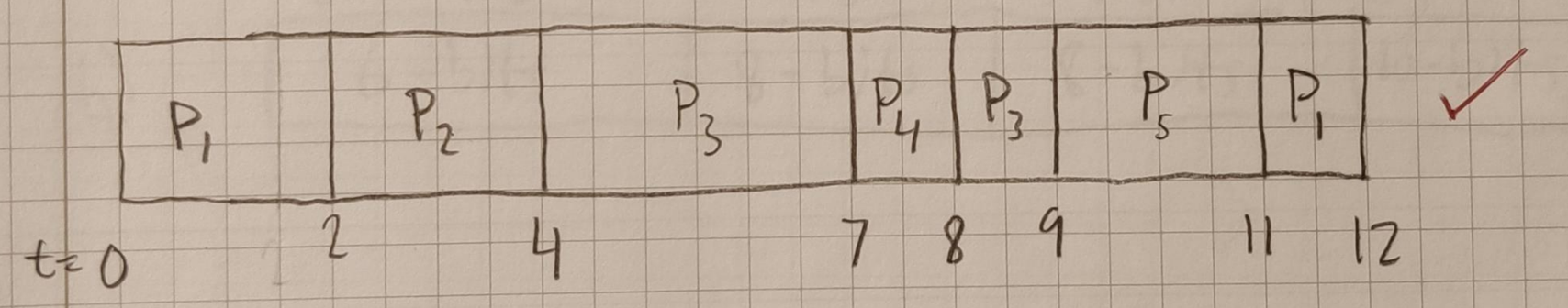
✓

(ii) shortest-job first scheduling without preemption



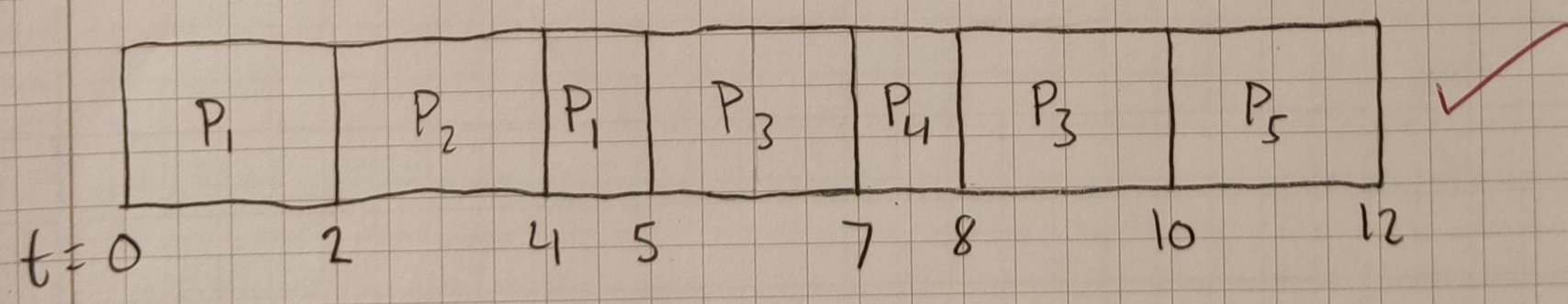
✓

(iii) Priority scheduling with preemption



✓

(iv) Round-robin scheduling



✓

/4

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1371	Datum: *Date:* 2019-08-26	Blad nummer: *Sheet number:* 8
Utbildningskod: *Education code:* TDDB68	Modul: *Module:* TEN1	

5. a) 1024 bytes with 32 bit addresses gives a total of 256 32 bit addresses. which requires 10 bit offset.

This leaves 22 bits for page tables, which can be distributed as such:
8-bit for inner page table
8-bit for middle page table
6-bit for outer page table
Totaling 3-levels of page tables, giving the virtual address:

b)

outer page table	middle page table	inner page table	offset
6-bits	8-bits	8-bits	10-bits

3

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1371	Datum: *Date:* 2019-08-26	Blad nummer: *Sheet number:* 9
Utbildningskod: *Education code:* TDDB68	Modul: *Module:* TEN1	

6. a) A buffer-overflow attack is when you overflow a buffer with the purpose to push important data and instructions on the stack so that the execution fails, or gives the attacker control. An attacker could do this by overflowing an input field with massive amount of data and potentially crash the program or gain unauthorized privileges. The solution is to limit amount of data that can get passed in an input field. 1

b) A virtual machine is completely isolated and separated from the main operating system, making it highly unlikely that a virus in a virtual machine could infect the rest of the system. 0.5