TENTAMEN (EXAMINATION)

6

Tentamensdatum/*Examination date:* 19-06-11
(åå-mm-dd/*yy-mm-dd*)

AID-nummer
AID number

Ifylles av student

1	2	9	1		

Completed by student

Ifylles av vakt

1	2	8	1		

Completed by supervisor

Utbildningskod/*Education code:* TDDB68 **Modul/*Module:*** TEN1

Kursnamn/*Course title:* Processprogrammering och operativsystem

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 10 **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*	7	5.5	5	5	5	3									
X här/*here*	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Erhållna poäng *Points obtained*															

Anvisningar/*Instructions*

1. Skriv AID-nummer, datum, kurskod och provkod på varje blad som lämnas in/
   Write AID number, date, course code and exam code 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

_____

Σ Poäng/*Points:* 30,5+4 Betyg/*Grade:* 5

Examinator/*Examiner:* _____

LINKÖPINGS UNIVERSITET

AID: 1281 Date: 19-06-11 Sheet num: 1

Ed. 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			0
1 b)		X		X	0
1 c)	X	X		X	0
1 d)	X	X			1
1 e)			X	X	1
1 f)	X		X		1
1 g)	X		X		1
1 h)			X	X	1
1 i)	X	X			1
1 j)	X	X		X	1

/7

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)	
1 b)	
1 c)	
1 d)	
1 e)	
1 f)	
1 g)	
1 h)	
1 i)	
1 j)	

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1281	Datum: *Date:* 19-06-11	Blad nummer: *Sheet number:* 2
Utbildningskod: *Education code:* TDDB68	Modul: *Module:* TEN1	

2.

(a)

```
struct stack {
    int array[256];
    int top;
};

void push(stack* s, int n){
    s->array[s->top] = n;
    s->top++;
}

int pop(stack* s){
    if (s->top > 0){
        s->top--;
        return s->array[s->top];
    } else {
        return -1;
    }
}

void stackInit(stack* s){
    s->top = 0;
}
```

2.

(b) If for example two different processes try to pop at the same time, we don't know which process is going modify the stack or return first. The statement (top>0) could be true for both, and both processes might decrement top before any of them return. This would lead to both instances of the function call returning array[top-2], which is one of the possible unexpected outcomes.

The critical sections are anywhere we modify or read information from the stack. The variables involved are array and top (in the stack). The statements involved are both lines in push, the comparison in pop and both lines in the if statement, and the line in stackInit. This is because these are all either reading or modifying the stack.

2.
(c)

One solution is to use locks or semaphores (with it's value set to 1 since we only want one process to modify the stack at a time).

The critical sections in each function could be protected as follows:

push:

```
wait(semma);
S->array[S->top] = n;
S->top++;
signal(sema);
```

1.5

pop:

```
wait(sema);
if(S->top>0){
    S->top--;
    int ret = S->array[S->top]
    signal(sema);
    return ret;
} else {
    signal(sema);
    return -1;
}
```

StackInit:

```
wait(sema);
S->top = 0;
signal(sema);
```

3.
(a)

Mutual exclusion: There is a limited amount of chopsticks and and only one philosopher can use a specific chopstick at a time. ✓

Hold and wait: A philosopher can hold a chopstick, and wait for a chopstick used by another philosopher. ✓

Voluntary release: Only the philosopher holding the chopstick can choose when to stop using it. ✓

4th condition: A chain of resource allocation can occur, each philosopher trying to claim a chopstick to the left, all the way around the table. ✓

circular wait ↙

1

3. (b)

$$Allocated = \begin{pmatrix} 0 & 1 & 1 \\ 3 & 0 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad Max = \begin{pmatrix} 9 & 3 & 1 \\ 3 & 2 & 2 \\ 1 & 4 & 1 \\ 6 & 5 & 3 \end{pmatrix} \Rightarrow$$

$$\Rightarrow Need = \begin{pmatrix} 9 & 2 & 0 \\ 0 & 2 & 0 \\ 0 & 3 & 1 \\ 6 & 4 & 3 \end{pmatrix}$$

Available = (5, 3, 0) Request = (0, 1, 0).

Request ≤ Available: true ✓ Request ≤ Need[2]: true ✓

Assume allocation of request ⇒ $Allocated = \begin{pmatrix} 0 & 1 & 1 \\ 3 & 1 & 2 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

$Need = \begin{pmatrix} 9 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \\ 6 & 4 & 3 \end{pmatrix}$ Available = (5, 2, 0)

Let Work = (5, 2, 0), Finish = (F, F, F, F) ✓

Need[1] ≤ Work: false

Need[2] ≤ Work: true, Work := (5,2,0) + (3,1,2) = (8,3,2)
Finish := (F, T, F, F) ✓

Need[3] ≤ Work: true, Work := (8,3,2) + (1,1,0) = (9,4,2)
Finish := (F, T, T, F) ✓

Need[4] ≤ Work: false

Need[1] ≤ Work: true, Work := (9,4,2) + (0,1,1) = (9,5,3)
Finish := (T, T, T, F) ✓

Need[4] ≤ Work: true, Work := (9,5,3) + (0,1,0) = (9,6,3)
Finish := (T, T, T, T) ✓

Finish = (T, T, T, T) ⇒ safe state, request from P2 should be granted.

4.
(a) A process is a program in execution represented in user memory and has an associated process control block. ✓

Kernel level threads are units of processing power that can run in paralell to eachother, they are mapped to user level threads (not necessarily one to one).

User level threads are a subset of a process and executes instructions individually. Each process can have one are more user level threads, and they can run (logically) in paralell to eachother. 2

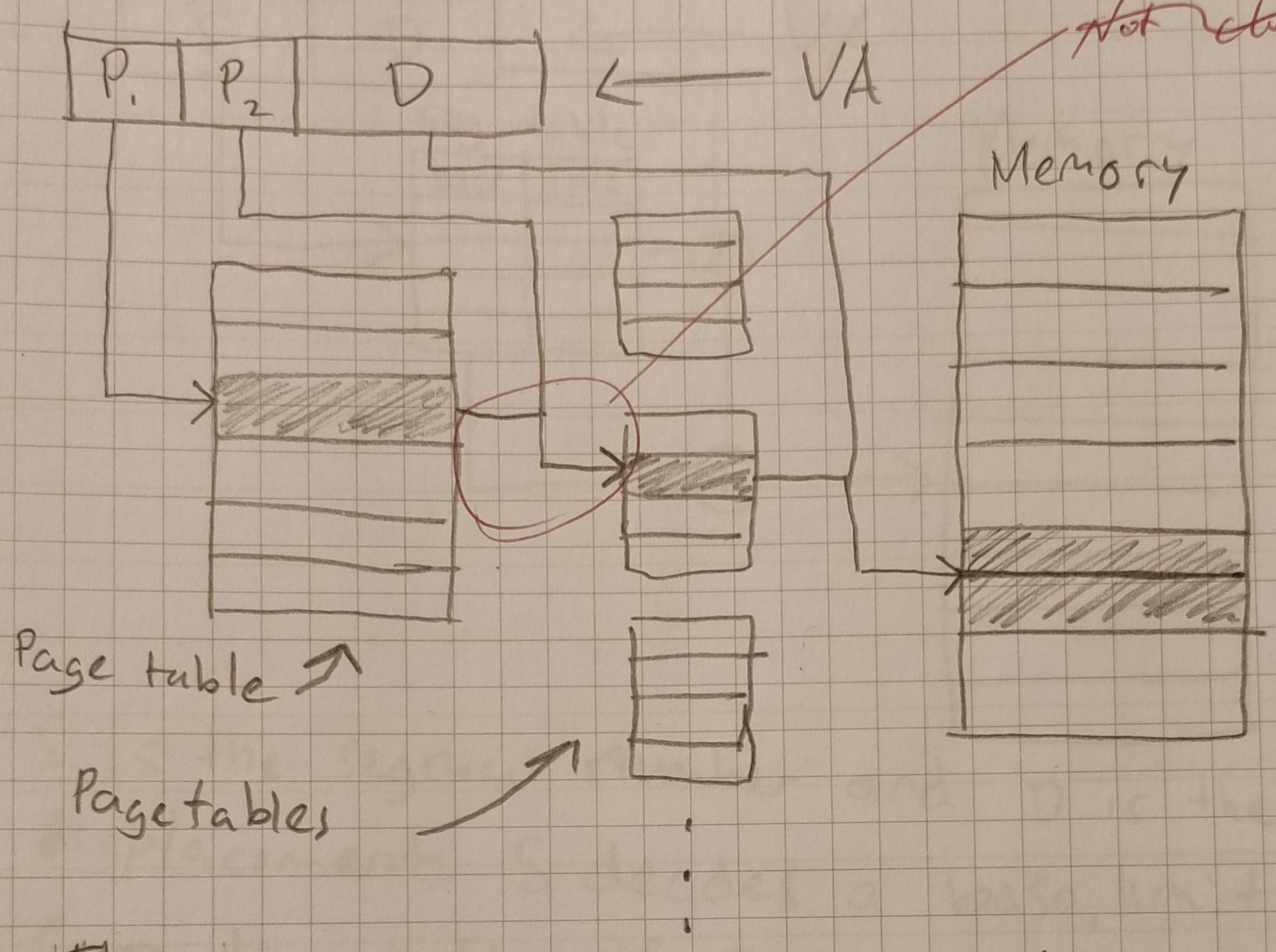
(b) Processes can share part of their memory, and have shared variables. By reading and modifying these they can communicate with eachother.

Message passing works by a process sending information to a "mailbox" in kernel memory, which can then be retreived by the process for which the information was meant for.

Shared memory is faster because both processes modify and read from their own memory, as opposed to going via kernel memory, where they have to find the relevant information. 3

5.

(a) One technique is multi level paging.



Not clear, explain

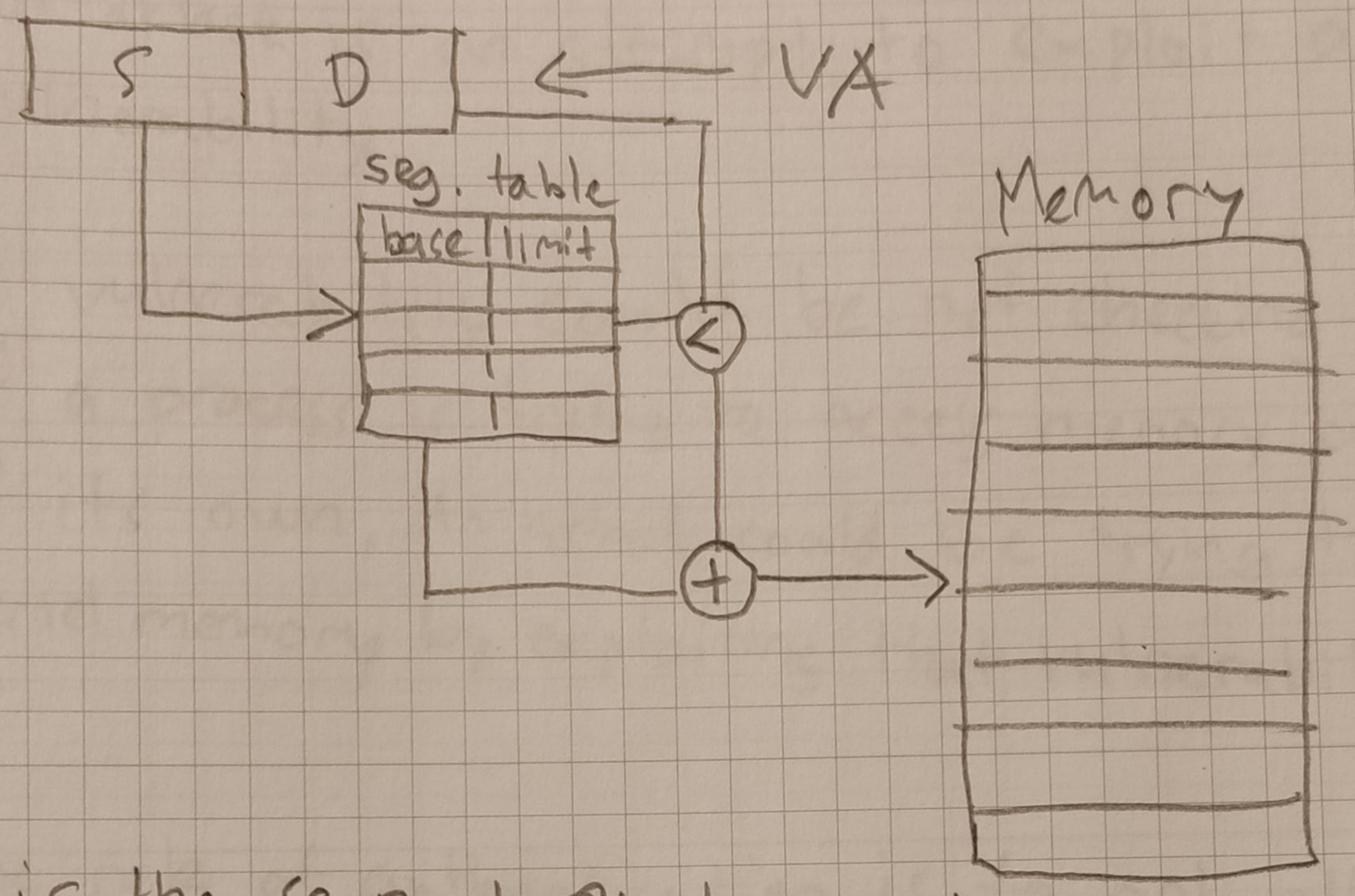
The page number is split in two so that the first part, P_1, decides a page number from the first table. The output from the first table decides a page table from the second layer. P_2 will then decide a page number from that table. The output from the table in the second layer decides the page frame in memory, and the displacement, D, is added to finally get the physical memory adress.

3p

5.

(b)

Simple segmented memory management system:



S is the segment number and D is the displacement. S decides a base, limit pair from the table. If D<limit the base and displacement are added, and the result is the physical memory address.

The displacement<limit comperison and base + displacement addition should be provided by the MMU in hardware.

2p

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1281	Datum: *Date:* 19-06-11	Blad nummer: *Sheet number:* 10
Utbildningskod: *Education code:* TDDB68	Modul: *Module:* TEN1	

6.

(a) A vulnerability is a security flaw, and an attack is an attempt to exploit a vulnerability.

A vulnerability could be not checking if a process is trying to access memory outside of its own. An attack could be trying to modify kernel memory by exploiting that vulnerability.

2

(b) The role of authentication is to only allow certain users to do a certain thing. In a distributed file system a user could be trying to modify data which it is not allowed to. To prevent this one can implement permissions and groups to the system.

↩ AUTHORIZATION

1