3

TENTAMEN (EXAMINATION)

Tentamensdatum/*Examination date:* 17-06-07
(åå-mm-dd/*yy-mm-dd*)

AID-nummer
AID number

Ifylles av student

		1	8	2	4

Completed by student

Ifylles av vakt

		1	8	2	4

Completed by supervisor

Kurskod/*Course code:* TDDB68 **Provkod/*Exam code:*** TEN1

Kursnamn/*Course title:* Processorprogrammering & operativsystem

Institution/*Department:* IDA

Inlämnat: antal blad 12 **provformulär** ☐
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	7.5	5.5	2,5	1	2,5									
	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, 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

_____

∑ Poäng/*Points:* 25+4=29 Betyg/*Grade:* 4

Examinator/*Examiner:* _____

LINKÖPINGS UNIVERSITET

AID	1824	DATUM	17-06-07
KURS	TPDB68	TENTA	TEN1

1

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

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

/ 6p

LINKÖ

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1824	Datum: *Date:* 17-06-07	Blad nu *Sheet n* 2
Kurskod: *Course code:* TPDB68	Provkod: *Exam code:* TEN1	

1.

kopia

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

2. a) P1: book(3,6) P2: book(5,6)

If P1 & P2 are concurrent there is a chance that P1 finds seat k available but before it is able to book it P2 also finds it available. This results in seat k being double booked for segment 5.

1

b) The critical sections are the areas of the code where k is being looked at (read) or operated on (written to). i.e. every time k changes.

```
book(...)
  int k,t;
  for(...) {
  critical section
      :
      }
return -1;
}
```

```
cancel(...)
int t;
critical section
      :
}
```

1.5

2.

c)

```
mutex lock;
book(...){
  int k, t;
  for(...){
    wait(lock);
    :
    signal(lock);
    return t; }
  signal(lock);
}
return -1;
```

```
cancel(...){
  int t;
  wait(lock);
  :
  signal(lock)
}
```

1

d)

```
Seat[k][i];
lock[k];

int book( int i , int j ) {
  int k,t;
  for(k=0; k< S; k++){
     int found_k = -1;
     wait(lock[k]);
     for(t=i; t<=j; t++) {
        if (seat[k][t]) {
           found_k=0;
           break;
        }
     }
     if (found_k) {
        for(t=i; t≤j; t++){
           seat[k][t] = -1;
        } signal (lock[k]);
        return k;
     } signal (lock[k])
  } signal(lock[k])
  return -1
}

void cancel(int k, int i , int j ) {
   int t; wait(lock[k]);
   for (t=i; t<=j; t++)
      seat[k][t] = 0;
   signal(lock[k]);
}
```

2.5

explanation →

... d) The critical section only becomes slightly shorter (1 line in back(...)). The reasoning for this is because the seat k should only be read AND written with mutual exclusion.

It is not tolerable for a process to read a seat k for a segment if it is being written to or if another process is also reading seat k.

This means that at ~~least~~ most row k needs to be mutually exclusive in its entirety. (My solution). Or at least seat k during segment i → j to be mutually exclusive in its entirety.

e) A reader-writer lock allows multiple processes to read at a time but only one process allowed to read-write. i.e. You can read if there is no process writing, you can write if ~~there~~ are no processes reading.

1.5

A reader writer lock is more suitable when the system does not use the data it reads from the resource as a way to decide what to later write to the resource.

3. a)

```
struct mutex {
    int value = 1;
    int holding_pid = -1
}
void wait(*mutex, size_t pid) {
    if( pid == mutex->holding_pid ) {
        mutex->value = 0;        // raise exception
        return;
    }
    while (mutex->value == 0) { }   // block
    mutex->value = 0;
    mutex -> holding_pid = pid;     // 1.5
}
void signal (*mutex) {              // check for same pid
    mutex->value = 1;
    mutex->pid = -1;
}
```

This solution assumes the process has the same pid. For it to work on multiple of the same process one could instead use a process identifier which is constant across all same processes.

This solution prevents self lock since it allows a process to skip the while loop if it has the same pid as the current lock holder.

3.

b)

Process (R1,R2,R3)	Need	Max	Available
P1	(0,0,0)	(4,1,1)	(1,0,2)
P2	(3,3,4)	(3,3,6)	
P3	(0,0,1)	(3,0,2)	
P4	(0,0,0)	(0,0,1)	

P2 → P1 → P3 → P4

Process (R1,R2,R3)	Need	Max	Available
P1	(1,0,1)	(4,1,1)	(0,0,1)
P2	(3,3,4)	(3,3,6)	
P3	(0,0,1)	(3,0,2)	
P4	(0,0,0)	(0,0,1)	

P4 → X

4

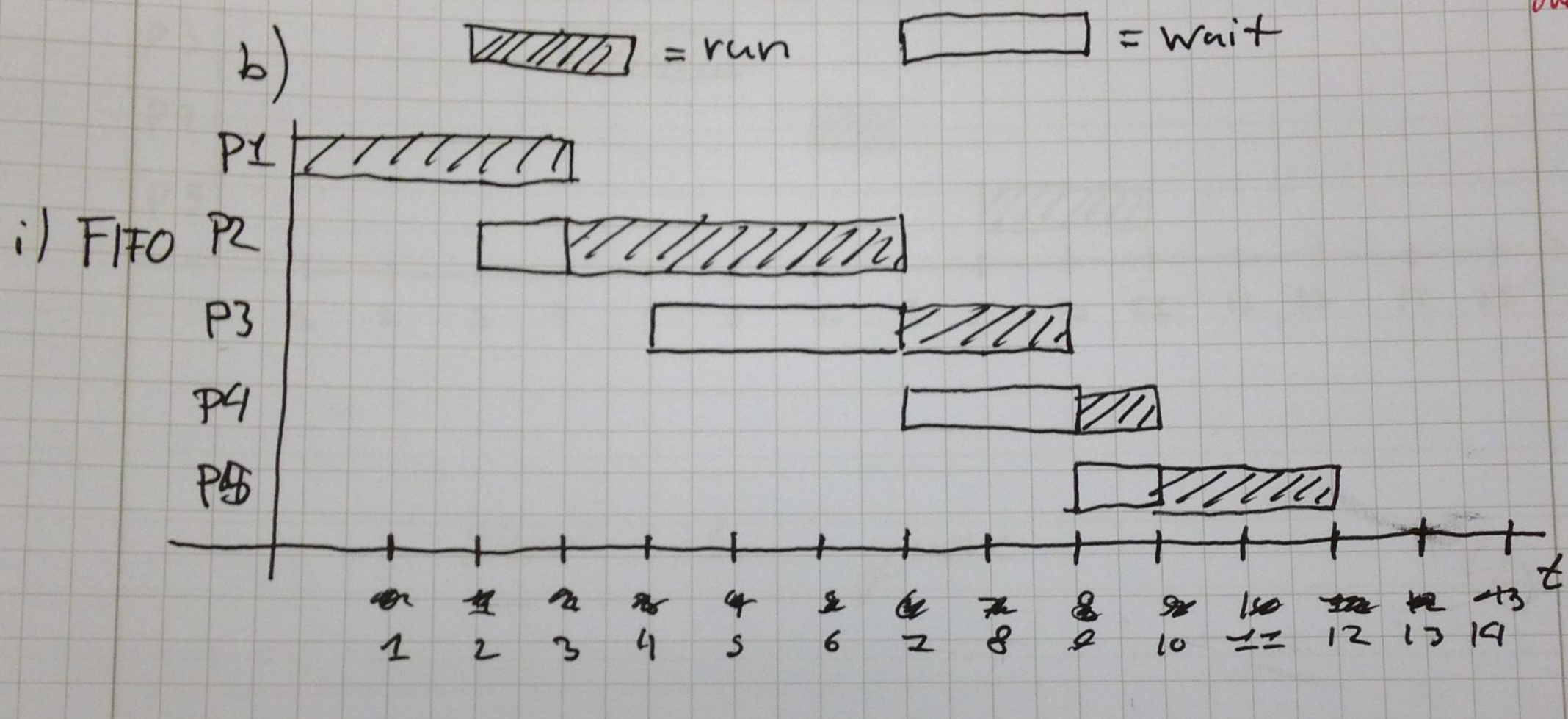
P1 should not be granted the request since there is no way for all processes to be able to execute with (0,0,1) available resources.

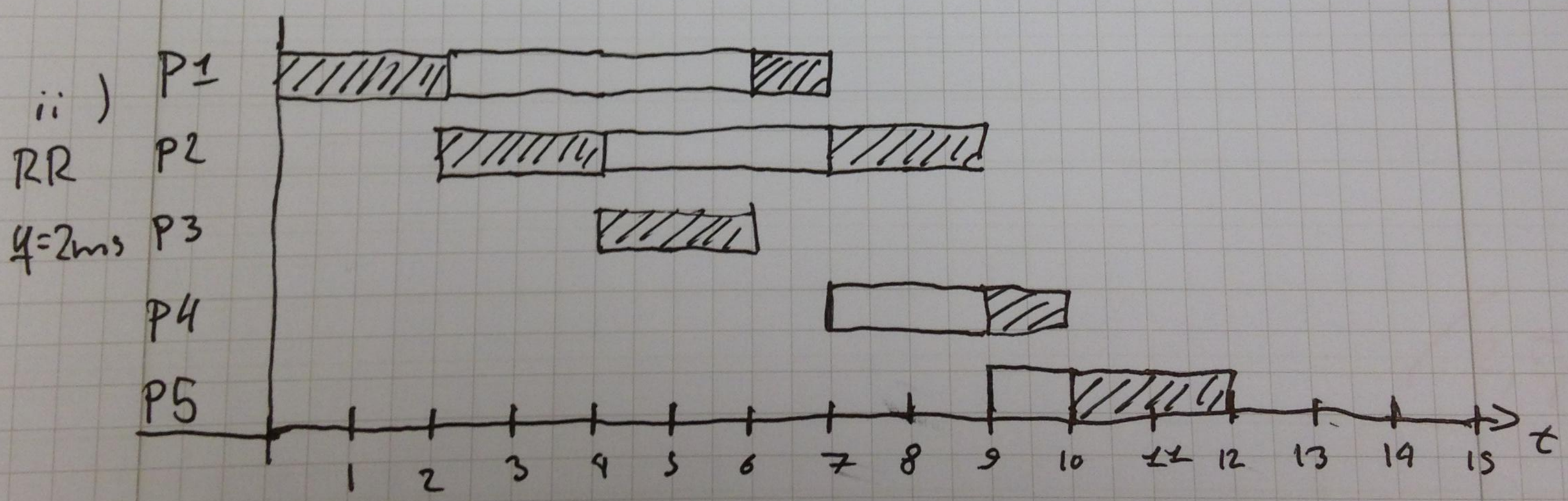
4.

a) The PCB contains all necessary information to start and resume a process. This can include page information, resource allocation, priority ... 0.5p

You miss the important ones

b)



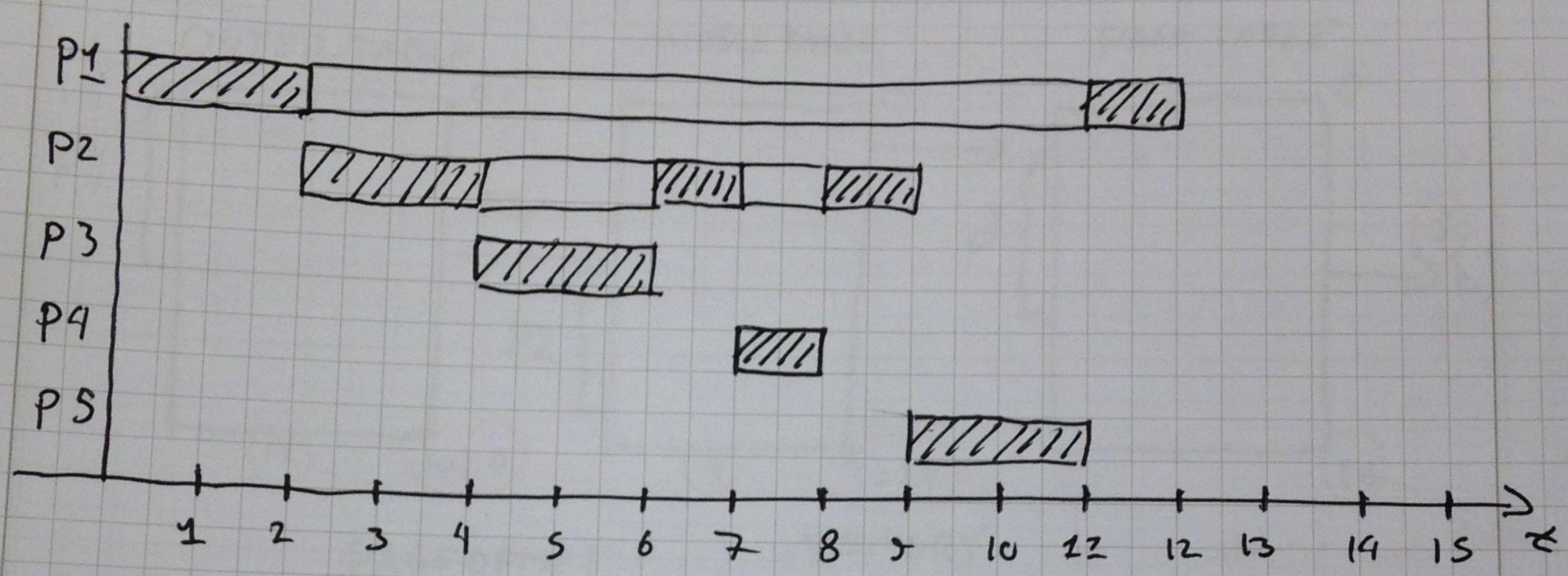


Assumption: RR cycles from 1 -> 5. If a process has not yet arrived it is skipped. ~~the effect restored~~

Not how it usually operates

4. ...b)

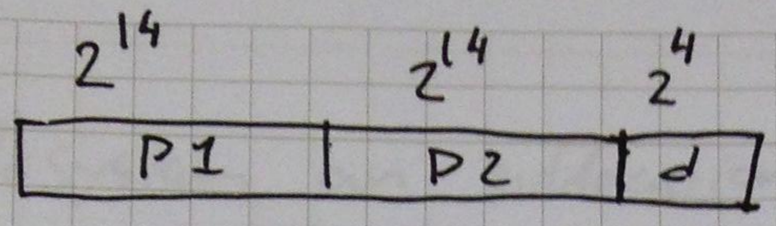
iii)

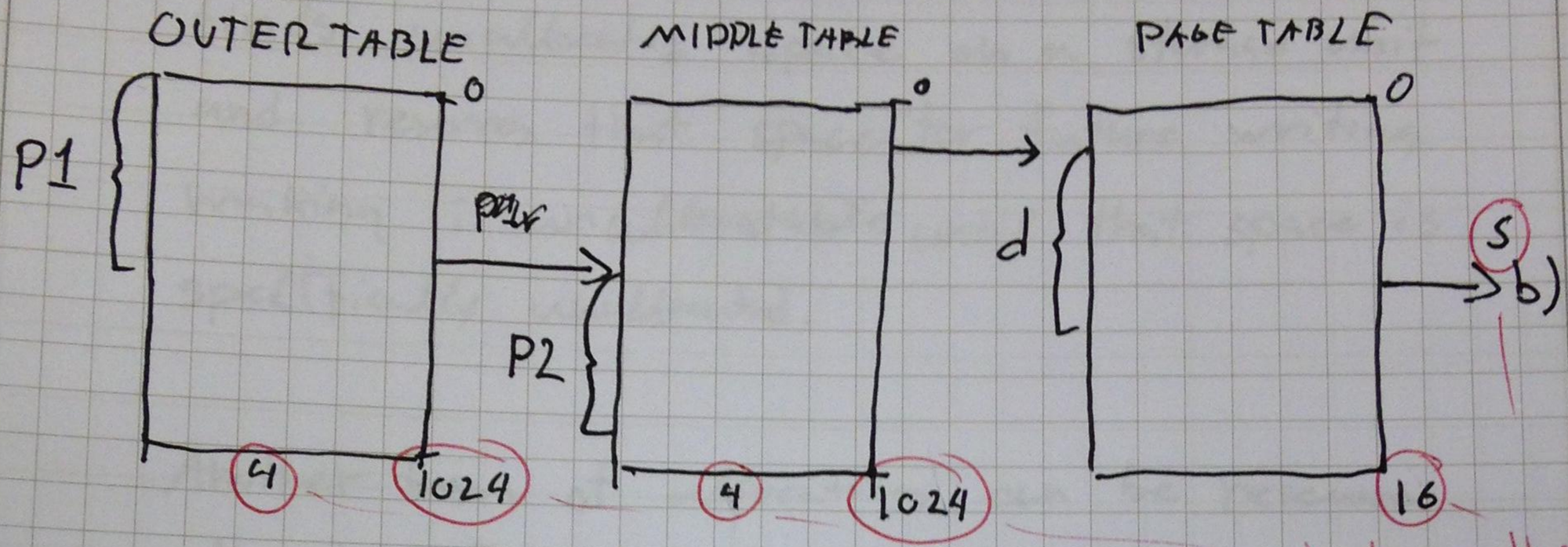


Show the queue 2p

/2,5p

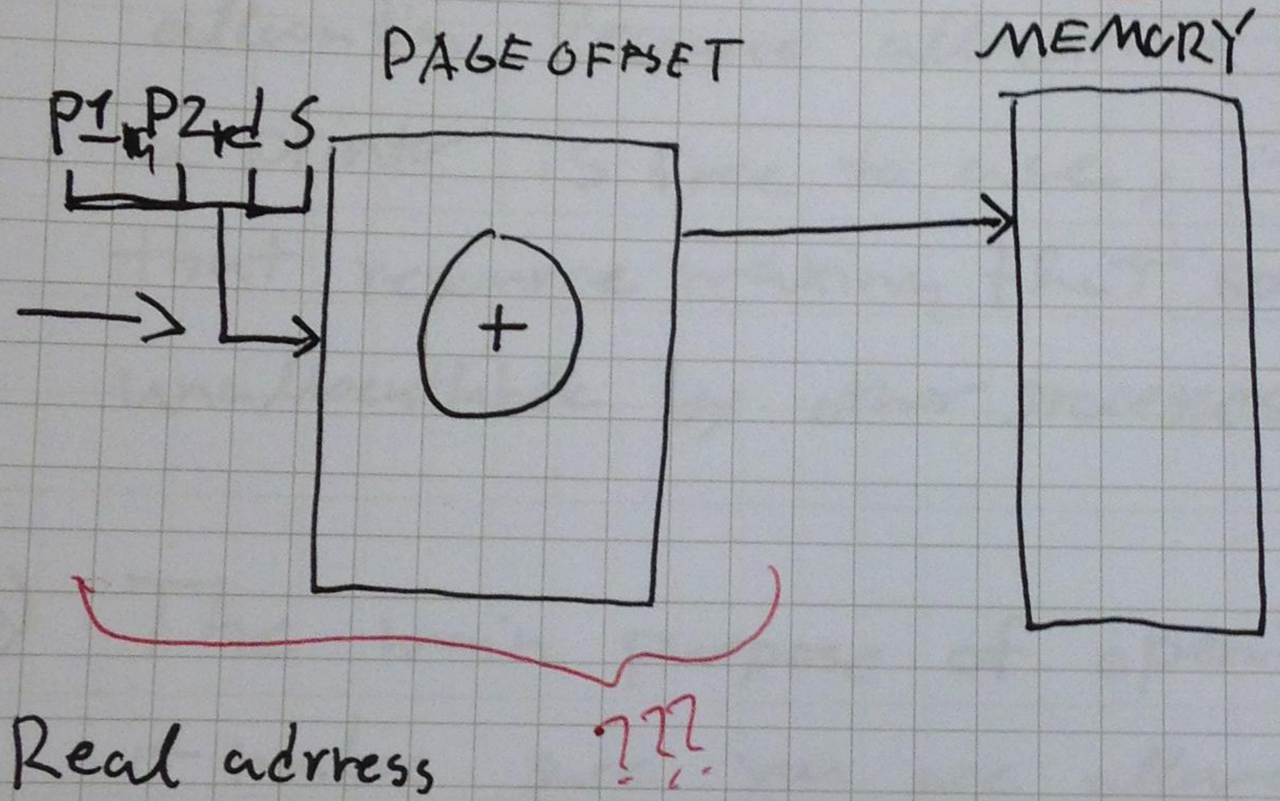
5 a)





what are these and how did you get them?

b)



OP

c) ~~[illegible]~~1: When a process doesn't use all its pages fully internal fragmentation occurs.

2: When a process request contigious memory larger than a page.

(1P)

6.

a) In a filesystem an allocation method finds unallocated space on a storage unit and reserves that space for future writing making i unallocatable until that space is specifically unallocated.

Another form of allocation can be resource allocation. Resource allocation checks if say a printer is free to use, if then allocates that resource making that specific resource unallocatable by other processes.

0.5

b) The main purpose of opening a file is it makes sure you are allowed to open the fail in the way you want. This allows files to be synchronized between processes.

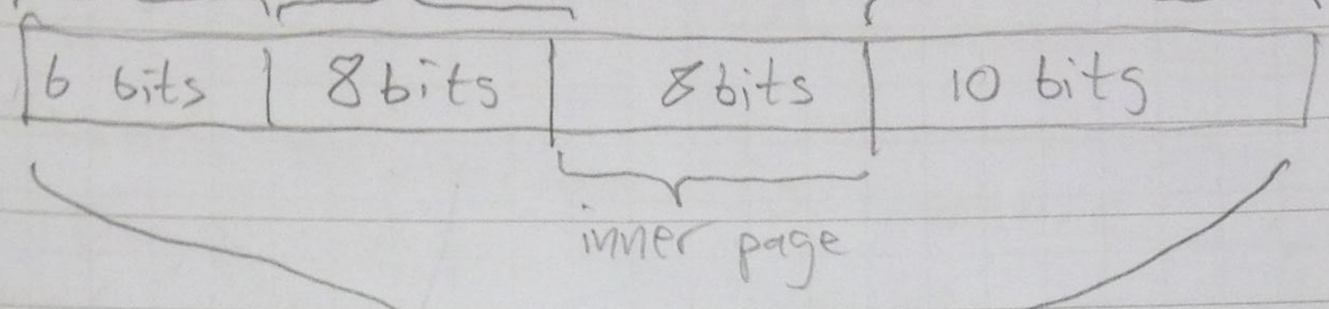
It also allows the OS to add the file to an open_files table for easier access and synchronization.

The open() system calls manipulates the global open_file table as well as the process specific open_file table. Open returns a file descripter which is usually a number which the process can use to finde the file in its open_file table.

2p

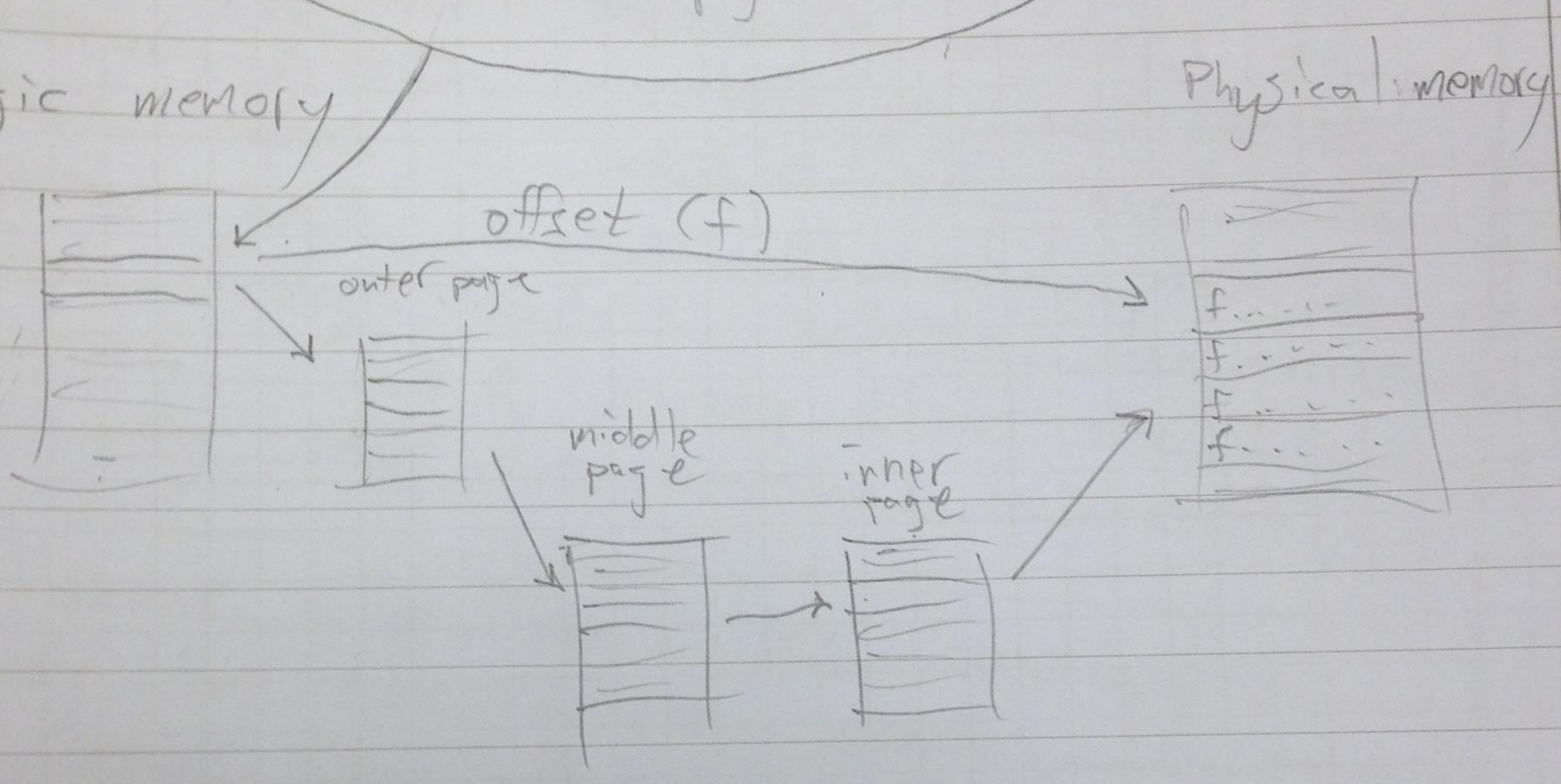
5

a) We will need three levels of paging



2p

b)



2p

c) If the pages are large internal fragmentation can occur. Example: a process that requires 20 bytes and the page size comes in chunks of 16 bytes, the process will end up allocating 32 bytes.

If the page size is small it can leave small "holes" in physical memory that is hard to fill with anything, thus leading to external fragmentation.

No

1F

4

a) The PCB:s purpose is to hold information of a running process. It contains, starting time, execution time, information of different threads that the process contains and priority of the process. It also contains CPU register states. A lot missing 1p

b) time quantum = 2ms

FIFO

time	0	1	2	3	4	5	6	7	8	9	10	11
Running process	1	1	1	2	2	2	2	3	3	1 (4)	5	5
Arrival	1		2	3			4				5	

✓

Round-robin

time	0	1	2	3	4	5	6	7	8	9	10	11
Running	1	1	2	2	1	3	3	2	2	4	5	5
Arrival	1		2	3			4				5	
Queue (first, ..., last)	-	-	1	1	3,2	2	2	4	4	5	-	-

0

4 b) Priority scheduling with preemption

time	0	1	2	3	4	5	6	7	8	9	10	11
Running	1	1	2	2	3	3	2	4	2	5	5	1
Arrival	1		2		3			4		5		
Queue	-	-	1	1	2,1	2,1	1	2,1	1	1	1	-

ok

3 - p

4