AID-nummer: 1606 Datum: 2019-03-18 Blad nummer: 1

Kurskod: TDDB68 Provkod: TEN1

1)

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

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

0

/9p

2)
a) We need two global semaphores that control the access to the bus.

```
semaphore busSema = 1;
semaphore commandSema = 1;
```

Both are initialized to 1 because access to the bus is mutual exclusive. "busSema" controls any kind of access to the bus. "commandSema" makes sure that no thread or process tries to send a command while another wait for a status response.

```
bool sendCommandTS(c):
    wait(commandSema)
    wait(busSema)
    sendCommand(c)
    signal(busSema)
    sleep(500)
    wait(busSema)
    bool status = checkStatus()
    signal(busSema)
    signal(commandSema)
    return status
```

2)

a) First we acquire the ability to send commands. Then we acquire access to the bus. We send the command and then we release access to the bus so that other processes or threads can use the bus. We do not release the command semaphore because we don't want others to send a command before we got the status back. The function then puts the thread to sleep for 500 ms in wait for the status to be ready. We request access to the bus, read the status and then signal both the bus and command semaphores. Lastly we return the status.

```
int readValueIS(v):
    wait(busSema)
    int value = readValue(v)
    signal(busSema)
    return value
```

4

The readValueIS function acquires the "busSema" semaphore, reads the desired value and then signals the semaphore. In the end it returns the read value. Because it only uses the "busSema" semaphore, it is possible to call this function even when another thread is waiting for a status response.

2)

a) With mutual exclusion and priority scheduling we could get into a situation called Inversed priority.

Imagine that we have three processes, P_1, P_2 and P_3. P_1 has lowest priority, P_2 has medium priority and P_3 has the highest priority. Also we have 3 resources R_1, R_2, R_3. These resources have mutual exclusive access to them.

2

Considering their priorities the processes would normally run in the order: P_3, P_2, P_1. Imagine that P_1 holds resources R_1 and R_2. Also that P_2 holds R_3 and requests R_2. P_3 requests R_3.

Because of its priority P_3 would run first, but it can't because it needs R_3 which is held by P_2. P_2 can't run either because it needs R_2 which is held by P_1. Thus the order of execution in this case is the opposite of order based on the priorities: P_1, P_2, P_3.

Thus mutual exclusion makes it so that the execution of the processes scheduled by priorities is not as intended.

3)

a) A deadlock detection algorithm can be used that searches for cyklus in a wait-for graph over the running processes or threads. (1)

b) In a deadlock, process do not progress in their execution because their are waiting indefinitely for a resource or a process. For example 2 processes waiting on each other. (1)

Starvation is when a process is not allowed to progress because other processes take higher precedence to the CPU or some resources for an indefinite ammount of time.

3)
c)

$$Max = \begin{pmatrix} 0 & 2 & 2 \\ 3 & 1 & 3 \\ 1 & 5 & 1 \\ 1 & 4 & 1 \end{pmatrix}$$

Max - is a matrix showing the maximum number of each resource that a process may request

$$Allocation = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 0 \\ 0 & 4 & 1 \end{pmatrix}$$

Shows how much of each resource each process holds.

Available = (3, 0, 3) - currently available resources

Request (P_4) = (1, 0, 0) - resources requested by P_4.

$$Need = \begin{pmatrix} 0 & 2 & 2 \\ 3 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$ - shows how much more the processes may request

There are enough resources to grant the request and the process does not exceed its maximum limit.

3) c) After granting the request:

$$Need' = \begin{pmatrix} 0 & 2 & 2 \\ 3 & 0 & 1 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad Allocation' = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 0 \\ 1 & 4 & 1 \end{pmatrix}$$

$Available = (2, 0, 3)$

Let $Finish = [F, F, F, F]$ be a boolean vector showing which processes have finished

P_4 - doesn't need any more resources, so it can finish

$Finish = [F, F, F, T]$
$Available = (3, 4, 4)$

There are enough resources for P_1 to finish:
$Finish = [T, F, F, T]$
$Available = (3, 4, 4)$

P_2 can finish:
$Finish = [T, T, F, T]$
$Available = (3, 5, 6)$

3) c)

Lastly P_3 can finish:

Finish = [T, T, T, T]

Availably = (3, 8, 6)

P_4 request can be granted and the processes can finish for example in the order: P_4, P_1, P_2, P_3.
The system i thus in a safe state.

4

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1606	Datum: *Date:* 2017-03-18	Blad nummer: *Sheet number:* 9
Kurskod: *Course code:* TDDB68	Provkod: *Exam code:* TEN1	

4. a) The time quantum is a time-slice during which a process is allowed to run before being preempted and rescheduled. OK

For every preemption it is required a context switch during which no actual work is being done by the processor. Lowering the time quantum will result in more context switches and thus more wasted CPU-time. OK

Lowering the time quantum results in more processes getting access to the CPU faster. This results in a more responsive system which may be desired for a interactive system for example one with a graphical user interface. OK OK

For systems that are primarily used for heavy computation it may be desired that as little time as possible is wasted, for example through context switches. Thus it may be good to make the time quantum longer in such a case.

3p

4.

b) A multilevel-feedback-queue scheduler is a scheduler that uses several queues to schedule processes, each queue with a different priority. With this scheduling algorithm processes can move between queues as needed. Each queue may also have different scheduling algorithms. For exampl one queue may use a round-robin algorithm while another may use a FCFS algorithm. The processes are moved between the queues by a cross-queue scheduler. The main idea is that a process in a queue will only be able to run if all the queues with higher priorities are empty.

To prevent starvation, aging is implemented so that a process that gets older is moved down to queues with lower priorities. To improve responsiveness a process is moved to higher priority when it gets a I/O response.

This is used because it is the most flexible scheduling algorithm as it can combine several different scheduling algorithms with priority scheduling. It can be adapted to most needs for scheduling.

Not clear which criteria for moving processes

1,5p

LINKÖPINGS UNIVERSITET

AID-nummer: *AID-number:* 1606	Datum: *Date:* 2017-03-18	Blad nummer: *Sheet number:* 11
Kurskod: *Course code:* TDDB68	Provkod: *Exam code:* TEN1	

4)

c) The turnaround time is the time that passes from when a process arrives into a system to when a process has finish its execution. 1p

/5,5

5)

a) External fragmentation refers to the unused memory that is outside the memory parts that are allocated to processes. In particular it refers to the small memory 'holes' that occur between memory partitions because of the memory allocation scheme. These holes are sometimes too small to allocate to any process and are essentially wasted memory.

Paging completely avoids external fragmentation as it divides all the memory into fixed-size pages that are allocated to a process as needed. Internal fragmentation occurs instead though.

2p

b) Let the following variables be:

Main memory access time: $t = 90$ ns

TLB lookup time: $l = 10$ ns

Hit rate: $\alpha = 90\% = 0,9$

$$EAT = (t + l) \cdot \alpha + (2t + l)(1 - \alpha) =$$

$$= t\alpha + l\alpha + 2t + l - 2t\alpha - l\alpha =$$

$$= 2t + l - \alpha t =$$

$$= 2 \cdot 90 + 10 - 0,9 \cdot 90 =$$

$$= 190 - 81 = 109 \text{ ns}$$

5)

a) If we have a hit in the TLB we access the TLB once and then the main memory once. Thus the time is $(t+l)$.

If we have a miss we have accessed the TLB once and we need to access the main memory twice. Thus the time for a miss is $(2t+l)$.

Given the hit rate α, the miss rate is $(1-\alpha)$ and the EAT can be calculated as:

$$EAT = (t+l)\cdot\alpha + (2t+l)(1-\alpha)$$

2p

6) a) Indexed allocation uses a index table that points to all the blocks that a file occupies on a disk. The advantage with indexed allocation is that it allows direct access to any block of the file. The disadvantage is that the table may become very big for big files and may not even fit inside a block itself. For efficiency this table must also be kept in main memory during use, which is also a disadvantage as it can take quite a bit of space.

2p

LINKÖPINGS UNIVERSITET

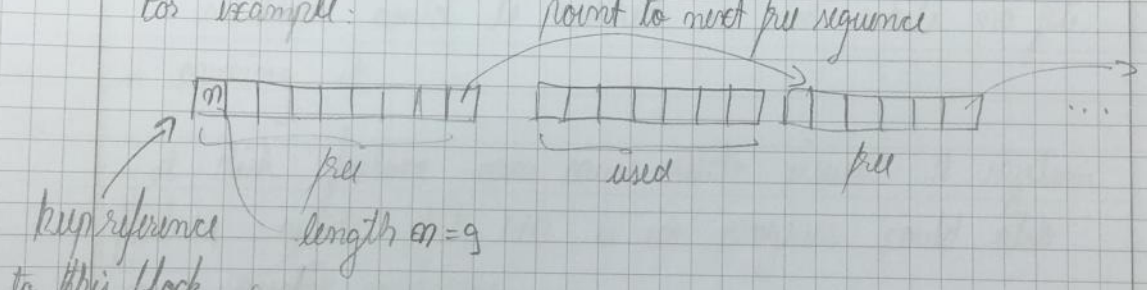
AID-nummer: 1606	Datum: 2017-03-18	Blad nummer: 15
Kurskod: TDDB68	Provkod: TEN1	

6) b)

The file system has to keep track of all the blocks that have not been allocated.

One possible technique is to keep references to all the contiguous sequences of free blocks by keeping a pointer to the first block of the sequence and how long the sequence is. This could be implemented in a linked manner if each free sequence points to the next.

For example:



2p

6)

c) Inconsistency could mean that we have conflicting information about what state disk blocks are in.

For example:

- a block is marked as being used by a file and free at the same time
- a block could be marked as being used by multiple files at the same time
- a block could be marked as used while no file is owning it.

A disk failure can cause such issues. A system failure while disk I/O is in progress could also cause such problems.

2p