**Question 1**

**Describe the two general roles of an operating system, and elaborate why these roles are important.**

* Provides a set of services to system users
* Manages secondary memory and I/O devices on behalf of its users
* Control execution of application programs

Operating systems provide:   
- a fairly standardised abstraction between hardware and userspace software, that allows userspace software to be written with little concern for the hardware it is running on, and having access to a bunch of useful things like memory, files, screens, i/o etc.   
- resource management and allocation to processes & threads. resources include files, memory, cpu runtime, etc.

**Question 2**

**Describe the three-state process model, describe what transitions are valid between the three states, and describe an event that might cause such a transition.**

*States:****ready****-****running****-****blocked***

*Transitions:* ***spawn****(interactive user login,new batch job),****block****(waiting for i/o event),****unblock****(i/o event complete,prioritiy high),****finish****(end of instructions),****suspend****(i/o event taking too long,lack of memory, time out)*

- Fetch Stage

· Processor fetches instruction from memory

· Program Counter holds address of the instruction to be fetched next

· PC incremented after each fetch in order to fetch next instruction

- Execute Stage

· Fetched instructions are loaded into Instruction Register

· Processor interprets instruction and performs the required action

- Interrupt Stage

· Interrupts normal sequencing of processor

· Improves processor utilization

- Fetch -> Execute -> Interrupt -> Fetch

Processes can be Running, Ready, or Blocked. A running process may block on an IO request. A running process may also be placed in a ready queue if it cooperatively or preemtively gets context switched, to allow another program to run. Blocked processes can be moved to a ready queue when the thing it was blocking on becomes available. Ready processes can start running in a context switch.

**Question 3**

**What is a process? What are the attributes of a process?**

* - Process
* · Program in execution
* · Instance of a running program
* · The entity that can be assigned to and executed on a processor
* Program code, set of data, state, priority, context data, program counter, i/o status info,memory pointer etc...

**Question 4**

**What is the function of the ready queue?**

* To keep a queue of the processes that are ready to be executed which the dispatcher selects from.

The ready queue stores threads that aren't currently running, that are capable of resuming execution. There may be several ready queues for each priority level, depending on the scheduling algorithm. The scheduler consults the ready queue to determine which process/thread to run next. As the name suggests, the ready queue is a *queue,*in order to schedule fairly.

**Question 5**

**What is the relationship (or differences) between threads and processes?**

A process has many threads. All threads of a processes have the same shared address. Threads are able to execute different functions of a program. Although a process and a thread maintain different state information,the state of the process can determines the state of the thread. For instance, when a process is suspended all the threads in the process are compulsorily suspended as well

A processes is a container for threads, which has it's own memory space. A process may contain one or more threads, which share that memory space, all of the file descriptors and other attributes. The threads are the units of execution  within the process, they posess a register set, stack, program counter, and scheduling attributes - *per thread.*

**Question 6**

**Name the advantages and disadvantages of user-level and kernel-level threads.**

* ULT advantages:
  + Does not require mode switch to kernel mode
  + Can run on any OS(not OS dependent)
  + Scheduling is specific to application
* ULT disadvantages:
  + Inefficient scheduling
  + Multi Thread cannot work on uniprocessor
  + One thread blocking, blocks other threads from execution as kernel is not aware of what is going on and cannot switch control
* KLT advantages:
  + Even if a thread has been blocked, the kernel can schedule another thread of the same process
  + Kernel control processor hence schedule multiple
* KLT disadvantages:thread
  + Requires plenty of mode switch to kernel mode

**Question 7**

**Describe the process control block and the details of information it maintains?**

The PCB defines the state of the OS and provides all the information needed by the OS to control accessibility etc..It is useful for identifying a process as it contains the following:

i) PID-contains userid and is useful for processes communication and cross referencing (for other resources )as well as identifying parent/child processes

ii)Processor State Information:-PSW,user visible register,stack pointer,control and status registers

iii)Process control information:-data structures,priority,event,memory management... etc

**Question 8**

**Describe the sequence of steps that occur when a timer interrupt occurs that eventually results in a context switch to another application.**

Process is interrupted

Its state saved

Interrupt handler is triggered

Mode switching to kernel

Interrupt handler run

Process switching

**Question 9**

**What is a race condition? Give an example.**

Occurs when processes/threads share certain resources and access/share data results in changes in the data.Eg read/write to files. Can be resolved by mutual exclusion eg read/write actions ,controlling order of execution

A race condition occurs when there is an uncoordinated concurrent access to shared resources (e.g. memory or device registers), which leads to incorrect behaviour of the program, deadlocks or lost  work.  
  
An example would be two process updating a counter simultaneously. Say the counter has a value of 1. Process A reads the variable but before updating Process B reads and updates the counter to 2. Process A, having no knowledge of what Process B does, updates the counter to 2. The correct behaviour of the program would have the counter being incremented to 3, not 2.

**Question 10**

**What is a critical region? How do they relate to controlling access to shared resources?**

Path that must be executed without interruption and accessible to only one process at a time. During critical region, only that process/thread has access to the shared resource until the critical region is complete

A critical region is an area of code where the code expects shared resources not to be changed or viewed by other threads while executing inside the critical region.

**Question 11**

**What are the three requirements of any solution to the critical sections problem? Why are the requirements needed?**

Mutual Exclusion: No two processes simultaneously in critical region

Progress: no process outide of a critical region may cause another process to block

Bounded: No process must wait forever to enter its critical region

**Question 12**

**What is the producer consumer problem? Give an example of its occurrence in operating systems.**

The producer consumer problem is a problem that involves filling and depleting a bounded buffer. The producer inserts entries into the buffer, and can sleep if it is full. The consumer must retrieve data from the buffer, possibly sleeping if it is empty. Transactions on the buffer itself may require a mutex. This is seen in operating systems in buffers for everything, particularly i/o device input queues.

Given a single, fix-sized queue, the producer process(es) geenerate(s) an item to put into the queue while the consumer(s) take(s) the item out of the queue and consume(s) it.

Give an example of its occurence in operating systems.

Take file system buffer cache as example, the file system writes are produced by the application requests. OS has to find a free entry in the buffer cache to store the write or blocks the request until a free slot is available on the bounded buffer cache.

The producer consumer problem is a problem that involves filling and depleting a bounded buffer. The producer inserts entries into the buffer, and can sleep if it is full. The consumer must retrieve data from the buffer, possibly sleeping if it is empty. Transactions on the buffer itself may require a mutex. This is seen in operating systems in buffers for everything, particularly i/o device input queues.

**Question 13**

**What is deadlock? What is starvation? How do they differ from each other?**

Starvation: when a thread does not get allocated its required resources for a long time

Deadlock: when two or more threads are waiting on each other to release a resource in order to complete its execution

Starvation is worse

In deadlock, processes halt because they cannot proceed and the resources are never released. In starvation, the system overall makes progress using (and reusing) the resources, but particular processes consistently miss out on being granted their resource request.

Deadlock is the state of a set of threads where all threads in the set are blocked waiting for another thread in the set to release a resource, and a required resource can only be released by a thread in the blocked set. The set remains blocked forever.

Starvation is the phenomenon where the allocation of resources is 'unfair', and something cannot proceed because it never gets the resources it requires even though the resources may become available as the resources are always granted to something else.

In deadlock, processes halt because they cannot proceed and the resources are never released. In starvation, the system overall makes progress using (and reusing) the resources, but particular processes consistently miss out on being granted their resource request.

**Question 14**

**What are the four conditions required for deadlock to occur?**

Mutual exclusion - indefinite wait time for the process accessing the shared resource such that other processes are blocked from accessing it

No preemption - when the resource can’t be removed by force from the process holding it

Circular wait - process holds the resources needed by the next process. There are no copies of that resource so, the process might hold it for an extended period of time leading to a deadlock

Hold and wait –

Mutual exclusion:   
only one thread can use a resource at a time  
Hold and wait:   
a resource can be held, then blocking whilst waiting for more resources   
No preemption:   
resources cannot be forcibly taken away from process holding it   
Circular wait:   
A circular chain of 2 or more processes, each or which are waiting for a resouce held by the next member in the chain.

**Question 15**

**Describe general strategies for dealing with deadlocks.**

Banker’s algorithm, having the process to request all processes at once and block the process until all the requests can be granted simultaneously. Preemption, resources held by a process can be forcefully released

Prevention-design in a way that adopts policy that eliminates one of the conditions.More conservative Can be direct(prevent circular wait) or indirect(prevent on of the conditions)

Avoidance(bankers algorithm)-make appropriate dynamic choices based on the current situation eg process intialisation denial and resource allocation denial

Detection-detect deadlock possibilities by checks and control them eg rollback,abort all or abort some(until no deadlock)

1) ignore it   
- easy   
- especially if the other strategies are very costly, and deadlock is a rare occurence  
  
2) detect and recover   
  
detection:  
- do pages of maths   
- and some more maths   
- then some more   
- or perhaps explore a graph   
- with maths  
  
recover :  
- then either steal preemptible resources, or try and save valid states of processes and roll back to them   
- looks complicated...   
  
3) deadlock avoidance   
- disallow deadlock by trying to determine if allowing a particular resource allocation will cause deadlock   
- only allow progression of programs into 'safe states', which guarantee process completion.   
- difficult, because we need to know process requirements in advance, or do checks before allocating resources. this can often be impossible practically   
  
4) deadlock prevention   
- negate one of the 4 requirements of deadlock.   
- solve hold and wait, by making threads drop resources they hold if they cannot get all of the resources they require for progress. this can cause starvation.   
- usually done by using ordered resources, to break the circular dependancy condition. breaking other conditions is rarely a feasable option.

**Question 16**

**For single unit resources, we can model resource allocation and requests as a directed graph connecting processes and resources. Given such a graph, what is involved in deadlock detection?**

Deadlock detection can be done by finding closed loops in the graph, which involve two or more processes requesting resources which are hold by other processes.

**Question 17**

**Assuming the operating system detects the system is deadlocked, what can the operating system do to recover from deadlock?**

Roll back /restart,abort all processes,abort till no deadlock,preempt resources till no deadlock

An operating system can take resources if they are preemptible (meaning the process doesn't die if it's taken away). It can kill a process to free up resources. Otherwise, it can try and take snapshots of process states, and roll back to state where the system was not deadlocked.

**Question 18**

**What must the banker’s algorithm know prior in order to prevent deadlock?**

The bankers algorithm must now the maxmimal resource requirements of all processes.  
  
We assume that the banker can keep track of resource availability and usage, and the process that receive the resources they require eventual finish and release them.

**Question 19**

**Describe the general strategy behind deadlock prevention, and give an example of a practical deadlock prevention method.**

Deadlock prevention is removing the capability of deadlock to occur, by negating one of the 4 conditions of deadlock.

A practical example of how to achieve this is to use ordered resources, such that circular dependancies cannot form. see above

**Question 20**

**Explain short term and medium term scheduling policy.**

Short term scheduling involves decision on which processes to run from ready queue during execution. Conversion for ready state to running state. Medium term scheduling involves decision on suspended/swapped processes..it manages the degree of multiprogramming.

**Question 21**

**What are four general characteristics of processor scheduling policies?**

* Turnaround time
* Response time
* Throughput
* Predictability

**Question 22**

**Define turnaround time and normalized turnaround time. Why are these useful for measuring the performance of a scheduling algorithm?**

Turnaround time is the time that it takes for the process to complete from its submission.

Completion - Arrival. It is the total times a process spends in system

Normalized tt is the turnaround time divided by the service time. It is useful to provide the average waiting time.ie how long process spends waiting in queue. Normalised tt closer to 1 are preferred as they signal little to no waiting time

Nb:tt -turnaround time

**Question 23**

**List and describe the four memory allocation algorithms. Which two of the four are more commonly used in practice?**

Best-fit, Next-fit, First-fit, Worst-fit

First and Next

First fit - allocate into the first available gap found of adequate size, starting from the beginning of memory.   
Next fit - allocate into the first available gap found, resuming the search from the last allocation.   
Best fit - search all of memory to find the smallest valid gap and allocate into it, on the basis that it'll leave the smallest unusable gap.   
Worst fit - search and place into the largest area, on the basis that it is likely to leave a gap big enough for something else to use.   
  
First and next fit are used in practice as they are faster, with comparable performance.

**Question 24**

**Describe the difference between external and internal fragmentation. Indicate which of the two are most likely to be an issue on: (a) a simple memory management machine using static partitioning; and (b) a similar machine using dynamic partitioning.**

 Internal fragmentation: the space wasted internal to the allocated region.Allocated memory might be slightly larger than requested memory.  
- External fragmentation: the space wasted external to the allocated region.memory exists to satisfy a reqest but it's unsuable as it's not contiguous.  
  
Stattic partitioning is more likely to suffer from internal fragmentation.  
Dynamic partitioning is more likely to suffer from external fragmentation.

**Question 25**

**What is thrashing ? How might it be detected? How might one recover from it once detected?**

When a processor spends most of its time switching process pieces instead of doing work.

Thrashing is when you're trying to run java 'hello world', and any other process on a system. The memory requirements of the currently running processes are so large that the working sets of the executing processes cannot be contained in physical memory. So one process tries to run, gets a page fault, retrieves the page from swap, boots out another processes' frame doing that. Then it context switches and the other process tries to run, but it's memory has been booted, so it page faults to swap, etc. The time is spent swapping frames rather than getting anything done.   
  
It could be detected by monitoring swapping levels compared to real cpu utilization. If swapping is huge, and utilization is down, thrashing is occuring. Suspending processes can alleviate thrashing; hopefully some processes will go below the thrashing threshold, run, terminate, then free up memory for other processes. Installing more ram helps, and not running java hello world helps.

**Question 26**

**Enumerate some pros and cons for increasing the page size.**

Increasing the page size decreases the number of page faults

Increasing the page size increases internal fragmentation

pros:   
reduces pagetable size   
increases tlb coverage   
increases swapping I/O throughput, as small disk transaction times are dominated by seek & rotational delays.  
  
cons:   
increases page fault latency, as swap response time is slower due to more data.   
increases internal fragmentation of pages, as there is more 'wasted page' in the working set

**Question 27**

**Describe two virtual memory page fetch policies. Which is less common in practice? Why?**

Demand paging: Only the pages that are requested are brought in

Pre-paging: Pages surround the requested pages are also brought in

Demand paging - relevant pages are loaded as page faults occur   
Pre paging - try to load pages for a process before they're accessed. Wastes I/O bandwidth if pages are loaded unnecessarily, and worse if an unnecessary page kicks out a necessary page.

**Question 28**

**What is the maximum file size supported by a file system with 16 direct blocks, single, double, and triple indirection? The block size is 512 bytes. Disk block numbers can be stored in 4 bytes.**

block size = 512  
  
number of block numbers in an indirection block  
= block size / 4  
= 128  
  
number of blocks for file data in that file object  
  
= 16 + 128 + 128^2 + 128^3   
Maximum file size:   
(direct + single indirect + double indirect + triple indirect) \* (blocksize)   
= (16 + 512/4 + (512/4)^2 + (512/4)^3) \* (512)   
= 68853964800 bytes, ~64 gigs

**Question 29**

**What are temporal locality and spatial locality?**

Temporal locality is a principle of locality where it is likely that a resource that is referenced once will be referenced again

Spatial locality is where a resource near the resource that is referenced is likely to be referenced.

temporal locality (locality of time) is the phenomenon where if a piece of memory is accessed, it is likely that that piece of memory will be accessed again in the near future. spatial locality is where if a piece of memory is accessed, it is likely that future accesses will be to similar addresses. These phenomena are why caching works; memory accesses are rarely random.

**Question 30**

**Explain the following basic algorithms that are used for the selection of a page as replacement algorithms: clock policy, first-in-first-out (FIFO), least recently used (LRU), and optimal policy?**

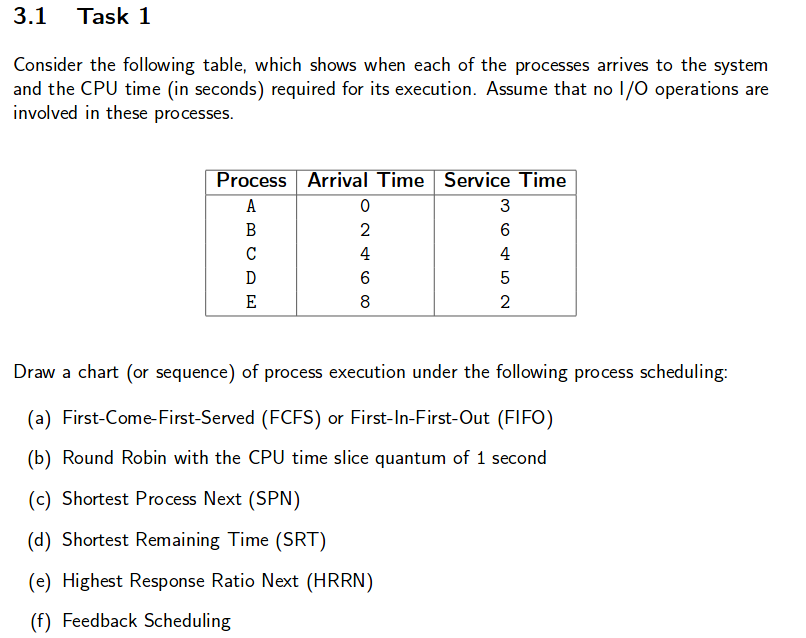
Clock policy: Checks in a circular manner if a page should be replaced or not. If the use bit is 1, it will not be replaced and the use bit will be changed to 0. If the use bit is 0, it will be replaced.

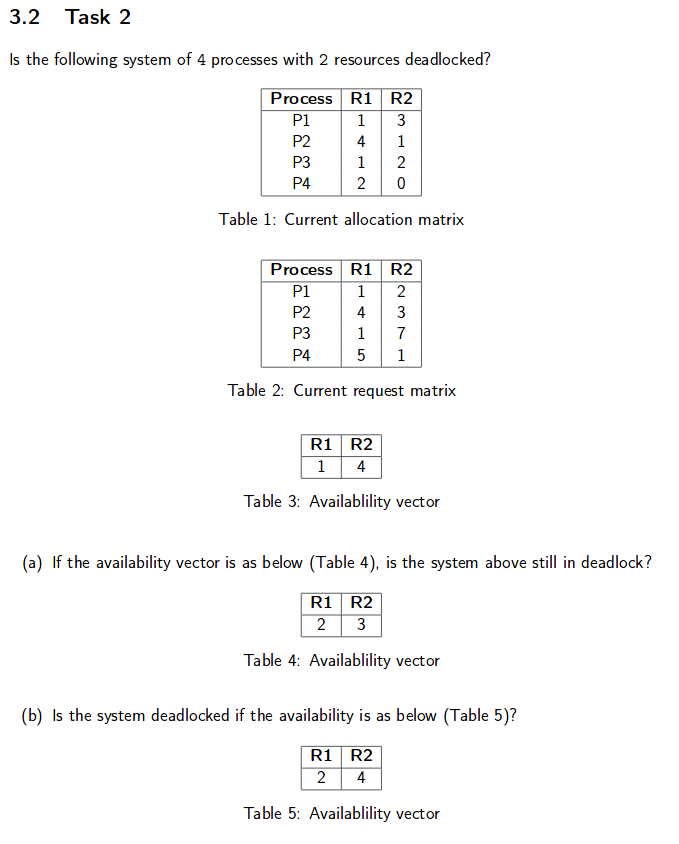
FIFO: The first page that was brought in will be the first page to be replaced

LRU: The page that was used the longest time before the replacement will be replaced

OPT: The page that will not be used for the longest time from the replacement will be replaced.

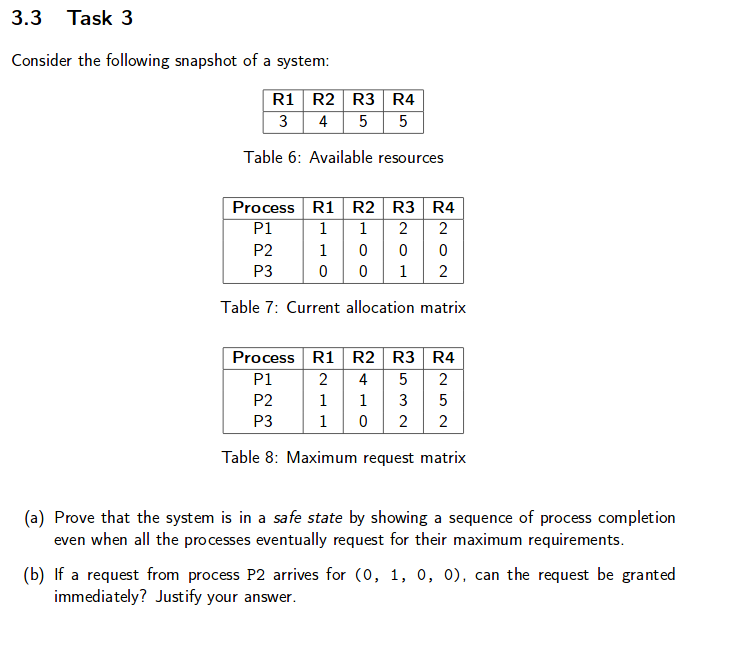
From best to worst (lecture notes):   
  
1) Optimal - use time travel to find the page that won't be used for the most time and boot it. Impossible to implement - used only as a theoretical reference point for comparing over algorithms.  
  
2) Least Recently Used - calculate whatever page hasn't been used the longest, and boot it. impossible to implement efficiently in practice - requires a timestamp on every memory reference.   
  
3) Clock page replacement - set 'referenced' bit to 1 when something is used. when looking for an entry to boot, set these bits to 0 if they're 1, and kick the first 0 candidate found. resume searches from the last one booted. Efficient (implementable) approximation of LRU - used in practice.   
  
4) FIFO - remove the page that has been there longest - does not consider actual memory usage in its decision - it will evict the most frequently used page in the system.

****

****

1. **Yes**
2. **No**

**^Unconfirmed (But I think it’s right)**

****

**a)**

**b) No, request cannot be granted as it is in an unsafe state**

• A smaller page size leads to smaller page tables – False – need more entries because we have more pages   
• A smaller page size leads to more TLB misses – True – less likely that page will encompass address we are after.   
• A smaller page size leads to fewer page faults – True   
  
• A smaller page size reduces paging I/O throughput – True   
• Threads are cheaper to create than processes – True   
• Kernel-scheduled threads are cheaper to create than user-level threads – False   
• A blocking kernel-scheduled thread blocks all threads in the process – False. This is true for user level threads   
• Threads are cheaper to context switch than processes – True – don’t have to save the address space   
• A blocking user-level thread blocks the process - True   
• Different user-level threads of the same process can have different scheduling priorities – False. User level threads aren’t scheduled, they yield() the CPU   
• All kernel-scheduled threads of a process share the same virtual address space – True   
• The optimal page replacement algorithm is the best choice in practice – False - it’s impossible to implement   
• The operating system is not responsible for resource allocation between competing processes – False – it is responsible for this   
• System calls do not change to privilege mode of the processor – False – we trap into the kernel so we do change the privilege mode   
• The Challenge-Response authentication protocol is susceptible to replay attacks by intruders snooping on the network – False   
• A scheduler favouring I/O-bound processes usually does not significantly delay the completion of CPU-bound processes – True – the I/O processes get the I/O and then yield the CPU again.