Sample Exam Question Answers

1. **Explain the structure of the OS and the role of each layer.**
   - The user communicates with the OS through a series of user programs which consist of a library of prewritten functions. These libraries and user programs are connected to the OS kernel. The kernel can be viewed as a collection of functions that user programs may call. The repertoire of commands supported by the kernel defines the "virtual machine" which is platform independent.
   - At the bottom of the OS we have the hardware layer; it accepts primitive commands. Software that interacts directly with the hardware is non-portable and contains low level details: control and state registers, interrupt priorities, DMA starting address etc. The hardware layer includes hardware such as the CPU, memory, buses, keyboard, mouse etc.

2. **Analyse the structure and functions provided by the kernel of an OS.**
   Parts of the OS kernel include:
   - Device Drivers are the hardware unit managers; they hide the low level details.
   - Process Management handles processes, allocating them resources and scheduling their execution.
   - Memory Management is responsible for physical and virtual memory management.
   - File Sys Management involves the code that organises the data storage into files and directories.
   - Network Services provide host-to-host and process-to-process communication across networks.

3. **How can a user program enter the kernel? Explain the trap mechanism.**
   - To enter the kernel, the user program makes a system call by executing a trap instruction of some sort.
   - This instruction switches the processor into a privileged operating mode (kernel mode) and jumps into the kernel through a well-defined trap address.
   - Parameters passed with the trap instruction tell the kernel what service is requested.
   - When the function is completed, the processor flips back to its normal operating mode (user mode) and returns control to the user program.

4. **Give a classification of OS.**
   - General purpose OS are powerful operating systems such as Windows and Linux which can perform multitasking and wide ranging functionality.

5. **Give a definition of a process and analyse it.**
   - A process is an instance of a running program, consisting of executable instructions, data and management information
   - A process is given control of the CPU to execute its instructions. Generally, multiple processes can be executing the same piece of code.

- The context of a process distinguishes it from other processes. The context is information on the instructions the process is currently or will next execute, the data it is working on, and administrative information such as a unique process ID.

6. What are the main components of the process context?

- Process ID
- Parent Process ID
- Real user ID – The ID of the user who started this process.
- Priority
- Current Directory – The start directory for looking up relative pathnames.

7. Analyse the set of operations used for process management, considering their outcome.

- Create – the internal representation of the process is created; initial resources are allocated and the program to run the process is started.
- Terminate – release all resources; possibly inform other processes of the end of this one.
- Change Program – the process replaces the program it is executing (by calling exec).
- Set Process Parameters – e.g. priority.
- Get Process Parameters – e.g. CPU time so far.
- Awaken Process – after waiting the process can run again.

8. Use an example to discuss what a child process is and how it is created.

- A process can create a child process, identical to it, by calling fork() – Unix function. As the kernel creates a copy of the caller, two processes will return from this call.
- The parent and the child will continue to execute asynchronously, competing for CPU time shares.
- Generally users want the child to compute something different than the parent. The fork() command returns the child ID to the parent, while it returns 0 to the child itself. For this reason, fork() is placed inside an if test.
- Example:

```
int i;
if( fork() ) { //must be the parent
        for( i = 0; i < 1000; i++ )
                print( Parent, i );
} else { //must be the child
        for( i = 0; i < 1000; i++ )
                print( Child, i )
}
```

9. Explain the concept of a thread and its benefits. How is a thread managed?

A thread is known as a lightweight process. A process can have one or more threads.

- Threads provide concurrency in a program. This can be exploited in multicore computers.
- Concurrency corresponds to many programs internal structure.

- If a thread calls exit(), the whole process, including all its threads will terminate.
- If a thread is more time consuming than others, all other threads will starve of CPU time.

10. Explain the purpose of process scheduling.

- The purpose of process scheduling is to order the processes that are ready to execute in such a manner that allows them to be completed with the highest level of efficiency. The processes are organized in a queue from where the scheduler selects the next one to take control of the CPU.

11. Use a numeric example to analyse the shortest process first scheduling strategy.

If processes get control in the decreasing order of their execution time, the average turnaround is better than in the random order. The turnaround time is the time consumed from the moment the process is ready for execution until its completion.

3 processes, a(35), b(40), c(15).

- $t_a = 35$, $t_b = 75$, $t_c = 90$ $t_{aa} = 200/3$
- In decreasing order of execution time: $t_c = 15$, $t_a = 50$, $t_b = 90$, $t_{aa} = 155/3$

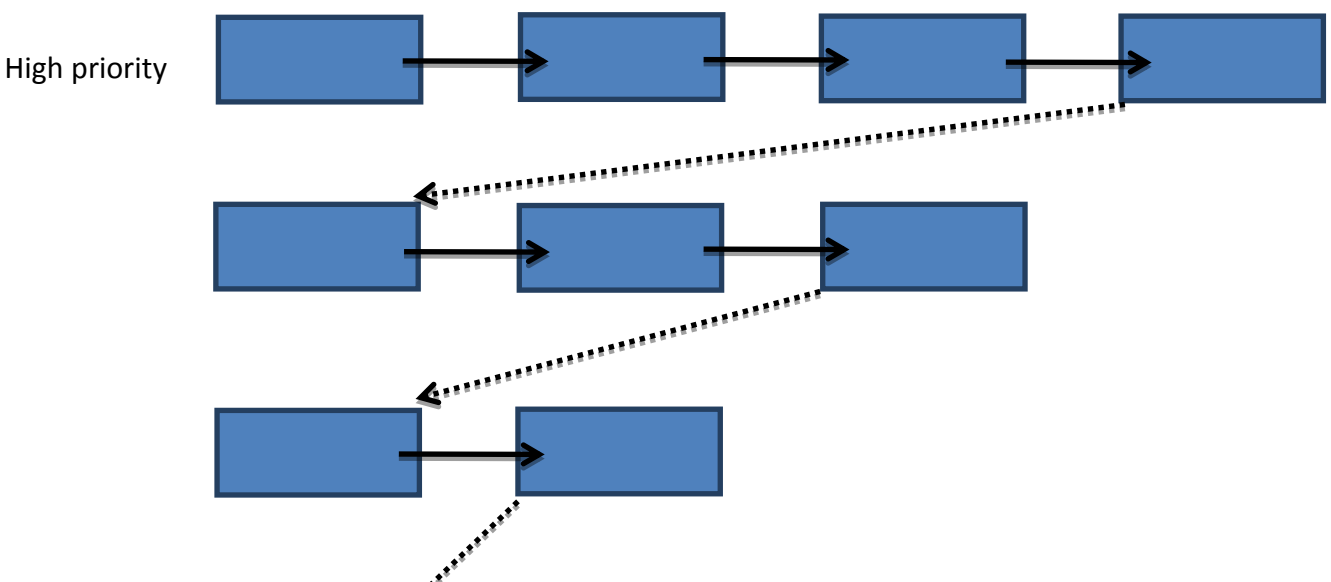
12. What is priority scheduling? In this context, explain the concept of dynamic priorities.

Priority Scheduling: multilevel feedback queues

- This is one implementation of dynamic priorities.
- Initially, a process gets a priority that puts it on a certain level.
  After each time slice, the priority is lowered to the next level, until it reaches the lowest acceptable priority. At that level, the strategy is round robin.
- However, after being blocked, the process gets a higher priority (priority boost).

- Dynamic priorities allow to avoid process starvation when, for example, a medium-level priority process is computation strong and never blocks. Lower priority processes will starve waiting for their time slice. In this case, their priorities can be raised at the medium or even higher level.

13. Use a diagram to analyse the multilevel feedbacks queue scheduling strategy.



Low priority 

14. **Explain the rationale behind two-level scheduling.**
    - Not all active processes can reside in memory at the same time, some of them must be kept on disk temporarily.
    - Moving processes from disk to memory is difficult and time consuming.
    - To prevent the CPU scheduler from wasting time moving processes, there can be a higher level, slower scheduler which selects the subset of active processes to reside in memory over a longer period of time.

15. **In the context of real-time scheduling, explain the earliest deadline first scheduling technique.**
    - One popular technique is earliest deadline first (EDF). For each process, there is an attribute value that indicates the time by which it should be completed.
    - The scheduler always selects the process with the earliest deadline. After the process used its time slice, the deadline is updated and the process is added to the ready list.
    - Example; time slice = 100ms and three processes, a with const period of 300ms, b with 500ms and c with 1000ms.
    - All are ready at t = 0 and have deadlines equal to their periods.
    - a is selected first, its next deadline is 400ms which makes it the next candidate. Then, the deadline is 200ms + 300ms = 500ms, after b. b is scheduled and its new deadline is 300ms + 500ms = 800ms…

16. **What is the main challenge of the scheduler in a multi-core system?**
    - The main challenge before the scheduler is to identify and predict the resource needs of each process and schedule them in a fashion that will minimize shared resource contention, maximize shared resource utilization, and exploit the advantage of shared resources between cores.

17. **Analyse the idea of process group scheduling.**
    - Process group scheduling is grouping processes together to be scheduled in order to minimise shared resource contention.
    - Threads and processes sharing memory segments can be co-scheduled on processors sharing the same cache. Since they share data and other context, this minimises resource contention and resource duplication.

18. **Explain how the scheduling domain works. Present examples of policies.**

    How The Domain Works:
    - Scheduling domains are sets of cores which share properties and scheduling policies that can be balanced against each other. Processes from a core in one scheduling domain can be moved to another to spread to load.
    - Each scheduling domain contains one or more core groups, which are subsets of cores that are treated as one entity. The actual balancing of process load happens between core groups.

- As process attributes change and resources become used up, processes are moved to scheduling domains with certain policies.

Policy Examples:

- If a processor becomes idle, and its domain has the SD_BALANCED_NEWIDLE flag set, the scheduler will go looking for processes to move over from a busy processor within the domain.
- If one processor in a shared pair is running a high-priority process, and a low-priority process is trying to run on the other processor, the scheduler will actually idle the second processor for a while. In this way, the high-priority process is given better access to the shared package.

19. What is active balancing?

- Active balancing is especially necessary when CPU-hungry processes are competing for access to a hyper threaded processor. The scheduler will not normally move running processes, so a process which just cranks away and never sleeps can be hard to dislodge. The balancing code, can push the CPU clog out of the processor for long enough to allow it to be moved and the load spread more widely.

20. Explain a couple of UNIX process system calls.

- By calling fork(), a new process is created, which is a copy of the calling process.
- A call to exec() replaces the code of the existing program with a new one.
- A process can voluntarily terminate by calling exit().
- Another possibility to terminate a process is to invoke kill() from another process (with appropriate privileges).

21. Explain how UNIX uses the process table.

The process table is divided in two:

- The first part is an array of structures (proc). These structures hold admin info, state info, id, scheduling info.
- Other data is not needed when the process is swapped out. They are stored in the user structure. User structures are swapped along with the rest of the process's memory space.

22. What elements define UNIX scheduling?

- The scheduler uses process priorities. The actual scheduling code is in the context switching function swtch(). It searches the process table for the highest priority process in memory.
- Processes migrate between memory and disk under the control of the function sched().
- Swtch() and sched() represent a two-level scheduler.
- Periodically, the priority of each process is updated

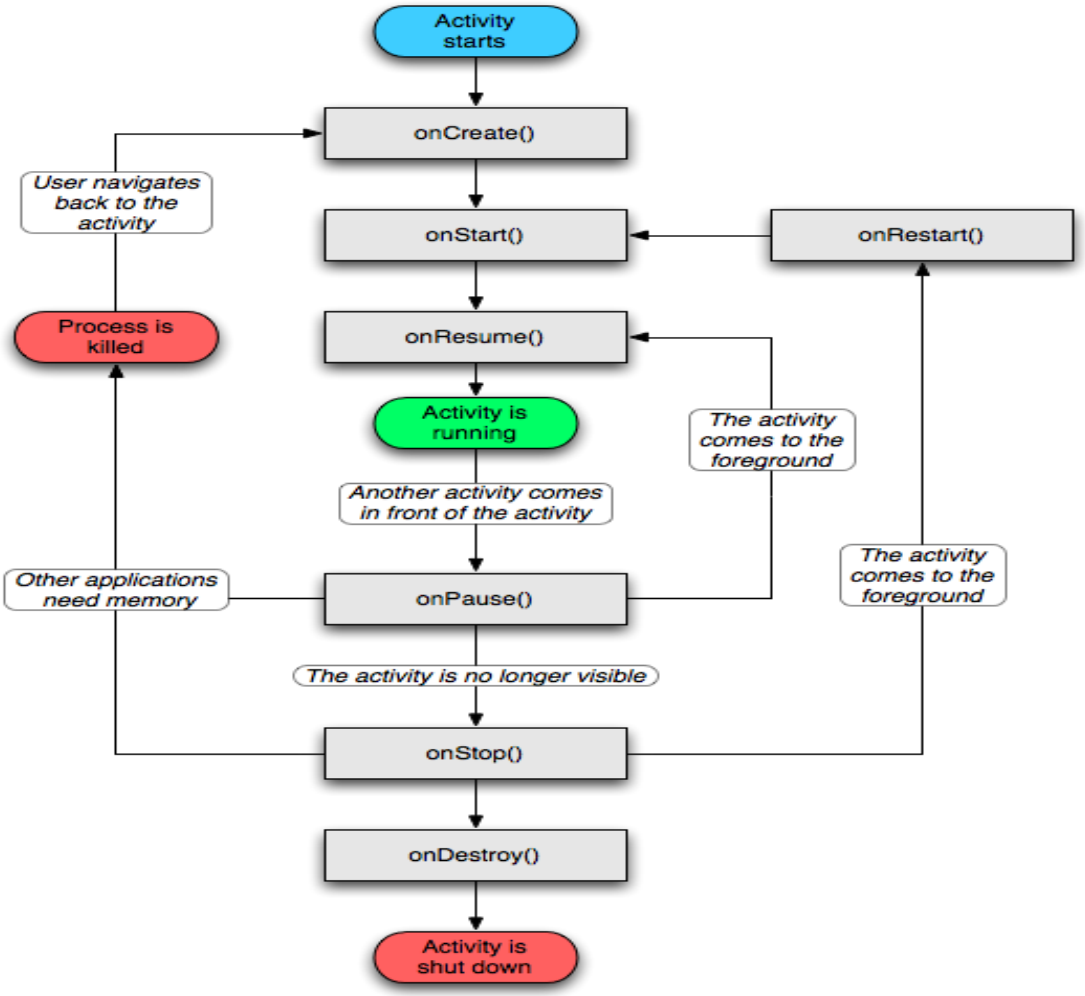
23. What scheduling strategies are used by Win NT?

- It uses a multilevel feedback queue with 32 priority levels.
- The lowest, 0, is reserved for a special kernel thread that clears free memory pages.
- Levels 1-15 are dynamic levels: are allocated to application threads.
- Levels 16-31 are real-time levels. They are not real-time in the sense of guaranteed response time or in terms of scheduling by deadline. They provide a higher priority level than normal applications and a more predictable response because they are not dynamically adjusted by the system.

24. Explain the component structure of TinyOS.]

- This is an OS for tiny sensors.
- Only the necessary parts of the OS are compiled with the application -> each application is built into the OS.
- It provides a set of system SW components.
- An application wires OS components together with application-specific components – a complete system consists in a scheduler and a graph of components.
- A component has four interrelated parts: a set of command handlers, a set of event handlers, a fixed-size frame and a bundle of tasks.
- Tasks, events and commands execute in the context of the frame and operate on its state.

25. Analyse Android application's lifecycle using a diagram.



26. Compare two different OS in terms of process management.

UNIX

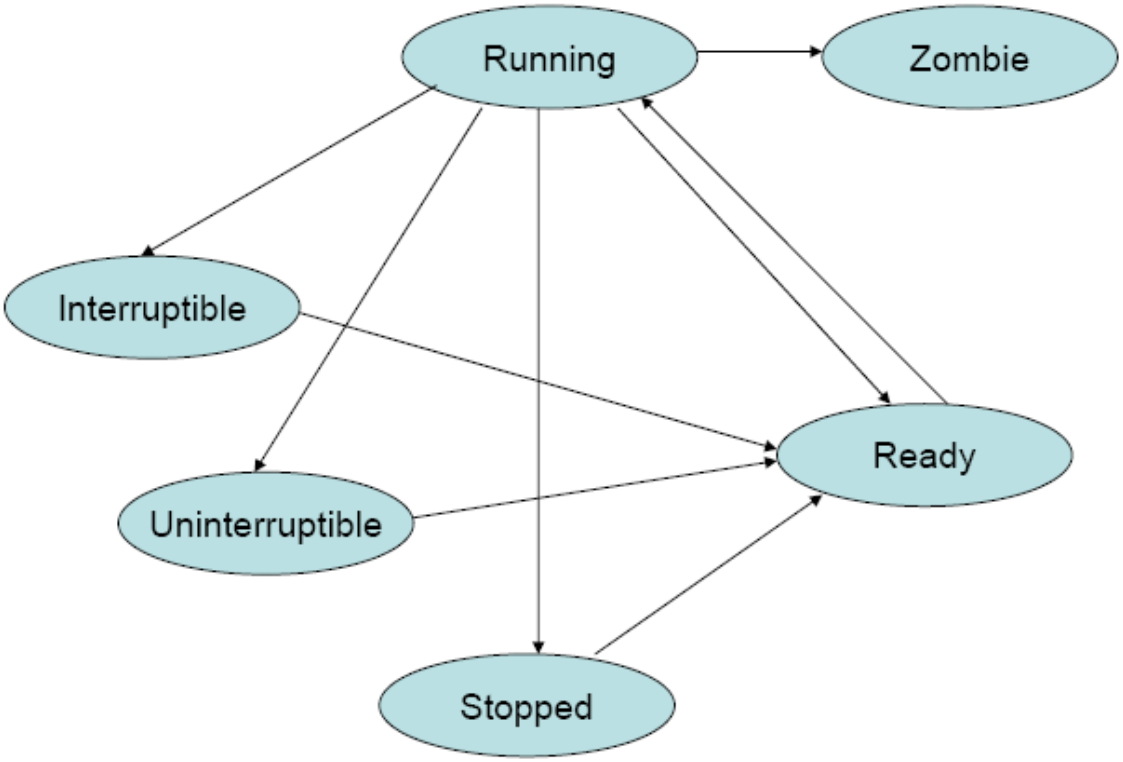
- For each program commanded to execute, a process is created. All processes are created by other ones, except the very first one started after the system is booted.
- By calling fork(), a new process is created, a copy of the calling process.
- A call to exec() replaces the code of the existing program with a new one.
- A process can voluntarily terminate by calling exit().

- Another possibility to terminate a process is to invoke kill() from another process (with appropriate privileges).

Win NT

- Processes in Win NT are sets of one or more threads that all share a common memory space.
- Threads are the units of execution that are scheduled and managed.
- The CreateProcess() Win 32 call creates a new process and implicitly a new thread. The call takes as argument the name of the program to run in the newly created process.
- createThred() starts a thread from the mentioned function of the process.
- ExitProcess() and ExitThread() terminate processes and threads respectively.

27. Show and comment Linux process state diagram.



- Running refers to both running and ready
- Interruptible: process blocked, can be awakened by signals
- Uninterruptible: process blocked cannot be awoken by signals
- Stopped: process halted
- Zombie: child process terminated early, still in memory

28. How is the process ID allocated in Linux?

- The function alloc_pid maintains a pointer to the last process id allocated.
- It will try to allocate last_pid + 1, but since process ids can be reused, it must check for availability.
- A bitmap of process id values being used is checked, starting at last_pid + 1. If the desired id is not available, the next available is in the bitmap will be located using linear search.

- If there are no available ids, the error code EAGAIN will be issued. alloc_pid() can be attempted again when a process terminates and its process id becomes free.
- When a free id is found, it is returned and used as the process id for the newly created process.

29. Characterise Linux scheduling by its key features.

- There are two multilevel feedback queues for active expired designated processes.
- It uses a bitmap to quickly determine which queue is the highest priority nonempty one.
- Interactive processes that completely use their quantum are placed back into the active queue and are scheduled in a round-robin fashion.
- Compute-bound processes that completely use their quantum are moved to the expired queue.
- When the active queue becomes empty, it is swapped with the expired queue.
- Priority and time slice length are computed each time a process is placed into a queue.

30. How does Linux compute priorities?

[Can somebody summarize this damn question please?] – let me know

31. Comment on the difference between virtual addresses and physical addresses.

- The memory addresses are physical (real) addresses. They uniquely identify more locations and create the physical address space.
- On the other hand, the process layout is specified in terms of virtual addresses. Their set represents the virtual address space.

32. Present methods used to translate virtual addresses into physical addresses.

- Methods of address translation:
  - Base registers
    - The virtual address is added to the content of a base register. The result is the physical address.
  - Segmentation
    - In this case, different memory segments store different parts of the program: code, data, and stack. Each segment will have separate base and limit register.
  - Paging
    - Memory is separated into areas of 2^k bytes, called page frames. If virtual memory addresses are n bits in length, there can be 2^{n-k} pages. The first n-k bits of the virtual address point to an entry in a page table. Page table entries hold, along with other information, a page frame number, which added to the last n bits of virtual address, make up a physical address.

33. Explain the mechanism of memory pages and the general content of the page table entry.

- Memory pages are 2^k byte sized areas of memory. They are part of a virtual to physical address mapping technique called paging.
- If virtual memory addresses are n bits in length, there can be 2^{n-k} pages. The upper n-k bits of the virtual address point to an entry in a page table. Page table entries

hold, along with other information, a page frame number, which added to the lower n bits of virtual address, make up a physical address.

- There can also be hierarchical paging. In this case, the upper bits of the virtual address are split in two. The upper bits indexes a page directory, whose entries in turn index various page tables. The lower bits index into this page table the same as above to get the physical address.
- Page table entries hold 5 main pieces of information
  - Page frame number – the start of a physical page frame. An offset is added to it to get physical address.
  - Protection bits – Indicate protection policies such as read-only for shared memory pages.
  - Present bit – A bit which indicates a valid address translation. A page fault occurs if a physical address does not exist, or the process cannot access it.
  - Modified bit – A bit which indicates whether this page has been written to recently.
  - Accessed bit – This bit is set if this page is accessed by a process.

34. What mechanisms can be used and how for the effective management of page tables?

- Let's assume a 32-bit virtual address and that the page size is 2^{12} = 4096 bytes. There are 2^{20} pages and therefore the same number of PTE. If each PTE is 4 bytes, the page table will be 4MB.
- A better solution is to organize the virtual space into a two-level page table. The 20bits are split up into two groups of 10.The most significant 10 bits index a page table, called page directory. The page frame number stored in the selected PTE identifies the page holding the page table, which is indexed by the other 10 bits of the page number.

35. How does the OS manage free memory space? Explain the free bitmap solution.

- The OS needs to locate free memory blocks which can then be allocated to processes.
- If the system is using pages of a fixed-size, one bit/page can show its state, free or not – this solution is called free bitmap.

36. How does the OS manage free memory space? Explain the linked list solution. Discuss how this solution can be made more efficient.

- The OS needs to locate free memory blocks which can then be allocated to processes.
- Free memory blocks can also be represented in a linked list. Again, when dealing with fixed_size pages, allocation is easy – take first off the list, or when it becomes free, append it to the list.
- If memory is allocated in variable-sized blocks, the list needs to be searched to find a suitable block. To speed up the search process, binary trees or hash tables could be used.

37. What is memory fragmentation and how can it be minimized?

- If memory blocks are fixed in size, the allocation process can result in waste. More memory allocated than is necessary = internal fragmentation. Memory left unallocated = external fragmentation.
- The simplest method of allocating memory is based on dividing memory into areas with fixed partitions.

- However, the flexibility of allocating memory in either large or small blocks is needed: e.g. a free block is selected and split into two parts, the first is allocated to the process and the second is returned as free space. The allocation is done in multiples of some minimum allocation unit (OS parameter). This helps to reduce external fragmentation.

38. Compare the first fit, next fit, best fit and worst fit memory allocation strategies by using an example.

If more than one block can satisfy a request, then which one to select?

- First fit is when the first memory block of equal or greater size to memory requested in the list is selected for allocation. This generally leads to allocations being clustered to lower addresses and fragmentation in lower addresses while larger free blocks remain unused in upper addresses.
- Next fit is when the search for a free block of required size starts at the next block after the previous allocation. In this case, the linked list of free blocks is treated as a circular list. If the search returns to the starting point, allocation fails. This leads to more evenly distributed blocks of memory.
- Best fit is when a block closest to the requested memory size is chosen for allocation. This can lead to external fragmentation as small blocks are allocated and de-allocated, but keeps large blocks available for larger requests.
- Worst fit is when the largest blocks are used for all requests. If allocation requests are of similar size, this can reduce external fragmentation.

39. Analyse the buddy memory allocation algorithm.

- The buddy memory allocation algorithm's main feature is the splitting of a free block of memory into two equally sized "buddy" blocks, where one is returned for allocation and the other is kept free. These blocks can be combined again at a later stage to form the original sized free block.
- All free memory blocks have a size of some power of 2. If the smallest block size equal or greater than a memory request cannot be found, a block of twice that size is split into two buddies. The first buddy is offered for allocation.

40. Analyse the swapping technique of memory management. What is demand paging?

- Swapping involves transferring one blocked process memory space on disk. Then when the process becomes active, it will be restored.
- When a page fault occurs, the OS must decide if the process tries to access a page not allocated to it or a swapped one. The loading of pages when they are needed is called demand paging.

41. Compare two memory replacement strategies, first in first out and second chance.

- First In First Out starts from the idea that pages are used for a finite amount of times after which they become "old". The page selected here is the one that has been in memory the longest. Implementation is done by a queue – all new pages are added to the tail of the queue.
- Second chance is an extension of FIFO: when a page if pulled off the head of the queue, the accessed (A) bit is examined. If its 0, the page is swapped out, else the bit is cleared and the page is reinserted at the tail of the queue.

42. Compare two memory replacement strategies, second chance and the clock algorithm.

- Second chance [refer to above]
- The clock algorithm is similar to second chance: two clock hands are moving synchronously above pages; the distance between them determines how long the page is given to be accessed. If it has not been accessed within that time, it can be swapped out.

43. Compare two memory replacement strategies, not recently used and least recently used.

- Not recently used is similar to second chance, except the modified bit M of memory blocks is checked as well as the accessed bit A. The blocks with the lowest AM values are chosen for swapping.
- Least recently used is based on timing when a block was last used. The block with the most time since its last use is chosen for swapping.

44. Compare two memory replacement strategies, least recently used and not frequently used.

- Least recently used [refer to above]
- Not frequently used is based on counting memory references. Periodically, all pages in the memory are swept and for each one with A=1, A is cleared and a page counter is incremented. Then the page with the smallest value of the counter is selected for swapping.

45. Analyse the working set strategy for memory replacement. What criteria are used to set the values of the two thresholds? Discuss how Win NT implements this strategy.

- The working set of pages corresponds to the number of pages a specific process is using at a time.
- If a process has allocated more pages than its upper threshold for the working set, it is a good candidate for page swapping. On the other hand, if by taking a page the lower threshold is passed, it makes sense to swap all the pages.

- Win NT uses a working set page management strategy – each process has an admin determined min and max working set size. The default values are based on the amount of physical memory in the system.
- When a process needs to read a page and is using its max working set size, then the choice of pages to be replaced comes from its set. If it needs to read a page and its set size is below the min, one page will be added to it.

46. Explain Linux slab allocator system.

- Slabs are collections of free memory blocks of a particular size. When a request matches that size, the slab can satisfy it. If the slab is empty, one or more pages are divided into blocks of the required size and added to the slab. Also, when a block is released, it is added to the slab.

47. How does a device driver work?

- Device drivers interface with device controllers on behalf of the OS.
- A device driver has two threads of control. The first thread of control deals with user processes' requests for I/O. The second thread of control deals with interrupts.
- The two threads of control share a request queue. If a user process requests a read operation, the driver loads the called function's arguments into the controller registers and starts the read. Until this read is finished, any other requests are put in the request queue. However, the user control can be interrupted.

48. Explain the structure of a device driver. What are water marks?

The driver can be divided in two halves:

- The upper deals with user requests;
- The lower deals with the controller.
- The two threads of control use mutual exclusion techniques in order to prevent corruption of the shared queue.

Water marks are used with buffers:

- The mark close to the full end signals when the buffer is about to become full.
- The mark close to the empty end signals when the buffer is about to become empty.

49. How are I/O devices represented in UNIX?

- When a device name is referenced, the file system translates it to three data items: a major device number, a minor device number and a flag indicating if the device is a character or block one.

50. Compare two I/O schedulers.

- Noop scheduler merges adjacent requests; when a request addresses sectors adjacent to sectors accessed by a request already in the queue, the two can be combined into a single request.
- Complete fair queuing scheduler maintains a separate queue for each process. Requests are merged and inserted in order of sector number. It schedules among the queues in a round-robin order.

51. How does the OS provide exclusive access to a file?

This is done by means of a special algorithm:

- If Q (queue of processes awaiting access) is nonempty, then add P to the tail of Q and return.
- If no process has currently exclusive access to the file, P gets access and return.
- If P requests read-only access and Q is empty and the processes with current access are readers, P gets access and return.
- Add P to the tail of Q and return.

52. Explain the concept of file metadata. What is included?

- The file metadata is managed by the OS and in some cases by applications as well.
- The file metadata includes data about the file such as name, size, last modification date, owner etc.

53. Explain the concept of file system metadata. What is included?

- File system metadata is information about a file system and its contents.
- The file system metadata includes the total size of the system, the amount of free space, the date of the last mount etc.

54. How does the OS manage free storage space?

- The allocation of space in a file system is done in fixed-sized blocks.
- For keeping track of free blocks, the free bitmap can be used.
- Another solution is to use free lists, embedded in the free blocks.
- Finally, a simple list stored in a free block can be used.

55. Explain the purpose of Linux Virtual File System.

- The Virtual File System implements a number of common services and generic file operations.

56. What is a superblock and what is an i-node?

- When a file system is mounted, a file system function is called to load an internal representation of the file system metadata, this is called the superblock.
- A member of the internal superblock structure points to a structure of type struct super_operations. This structure contains a number of functions needed to fetch and manipulate other metadata structures called i-nodes.

57. Explain how RAID improves reliability.

To improve reliability, RAID implements a few different levels of increasing reliability:

- RAID level 0: Data striping – stripping at the level of blocks but without any redundancy.
- RAID level 1: Data mirroring – a drive has its data duplicated on two different drives.
- RAID level 2: Uses error-correcting code(ECC) – one disk stores the 1^{st} bit of all bytes, another stores the 2^{nd} bit of all bytes, and so on.
- RAID level 3: Bit-interleaved parity organisation – improves on level 2 by using only one disk for parity.
- RAID level 4: Block-interleaved parity organisation – keeps a parity block on a separate disk.
- RAID level 5: Block-interleaved distribution parity – spreads data and parity among all disks.

58. Explain how resources are protected in a computer system, using the access matrix strategy as an example.

- In the case of the access matrix strategy, access is either granted or denied for certain types of users with certain files. For example, a computer administrator could have access to delete any file in a particular shared directory, but a simple user would not have permission to perform such an action.

59. What are the attributes of access rights revocation?

- Immediate vs delayed
- Selective vs general
- Partial vs total
- Temporary/permanent

60. Does Java provide any protection mechanism?

- When a class is loaded, the JVM assigns to it a protection domain that gives permissions.
- The protection domain depends on the URL from which the class was loaded and any digital signature on the class file.

61. Explain methods used by OS for user authentication.

- User names and passwords – the user will have to supply a name that exists in the computers user database, with a matching password in order to gain access to the computer.
- Finger print – some systems require a finger print for user authentication, a unique way of gaining access to a computer system.

62. Discuss the MIDP security model.

Concepts:

- Trusted MIDlet suites: origin and integrity can be trusted on the basis of some objective criterion;
- Protected APIs: APIs to which access is restricted; the level of access is determined by permissions (Allowed or User) allocated to the API.
- [not sure about this answer]

63. Present the public key infrastructure, discussing the hierarchy of certification authorities.

- The public key is distributed by a trusted certificate authority (CA), as part of a certificate.
- Root certificates (or root keys) contain details of the CAs and their public keys and are self-signed
- The PKI allows for a hierarchy of CAs up to the trust anchor.
- [not sure about this answer either]