A+ Summary

Round 1:
A processor core executes instructions
An instruction set architecture (ISA) defines how data can be interpreted

Which of the following can be considered primary memory?
USB stick (secondary memory)
Processor cache (answer)
SSD (secondary memory)
Network drive (secondary memory)

Thinking of processor, what is cache memory?
Typically a fast memory close to processor core

Unless instructed otherwise processor always increments the program counter after each instruction fetch so that it will fetch the next instruction in sequence.

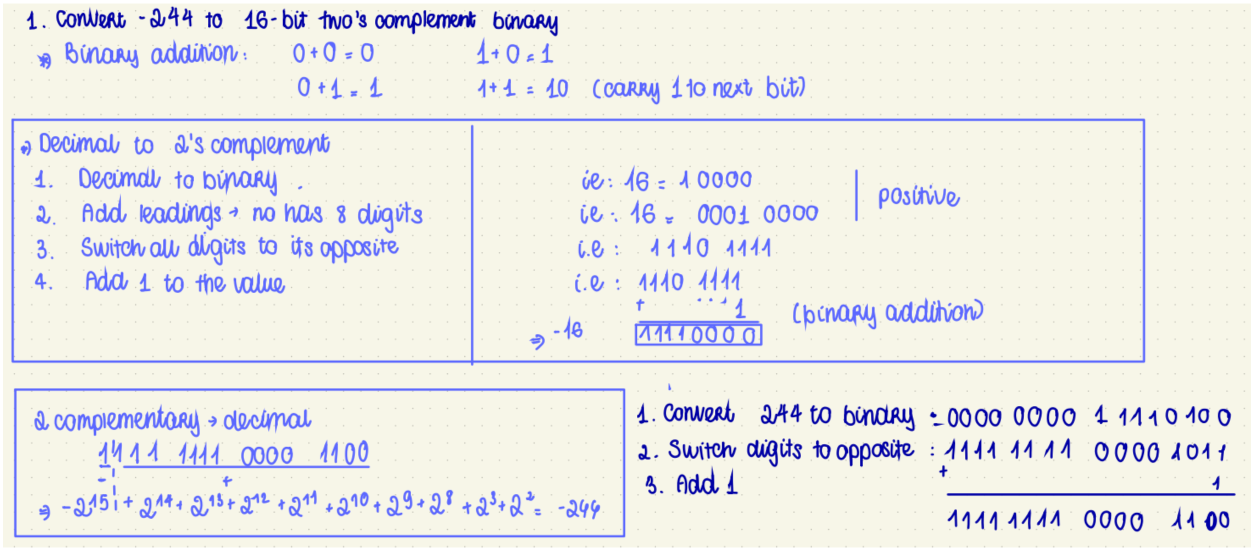
A system program that sets up an executable program in main memory ready for execution is the loader
Compiler: A compiler is a software tool that translates source code written in a high-level programming language into machine code that can be executed by a computer. The compiler typically performs several operations, including lexical analysis, parsing, and code optimization before generating the final executable file.

Assembler: An assembler is a software tool that converts assembly language code into machine code. Assembly language is a low-level programming language that is often used for system-level programming, such as device drivers or operating systems.

Linker: A linker is a software tool that takes object files generated by the compiler and assembles them into a single executable file. The linker performs several important tasks, including resolving dependencies between object files, linking in external libraries, and generating relocation information.

Loader: A loader is a program that loads an executable program into memory and prepares it for execution. The loader typically performs several important tasks, including allocating memory for the program, loading data and code into memory, and resolving external references.

The Program Counter (PC) register contains the address of the next instruction to execute
Principle of locality means that the memory references of a program tend to cluster



1. Convert -244 to 16-bit two's complement binary

⇒ Binary addition: 0 + 0 = 0 1 + 0 = 1

0 + 1 = 1 1 + 1 = 10 (carry 1 to next bit)

⇒ Decimal to 2's complement
1. Decimal to binary.
2. Add leadings → no has 8 digits
3. Switch all digits to its opposite
4. Add 1 to the value

ie: 16 = 1 0000 | positive

ie: 16 = 0001 0000

i.e: 1110 1111

i.e: 1110 1111
+ 1 (binary addition)

⇒ -16 $\boxed{11110000}$

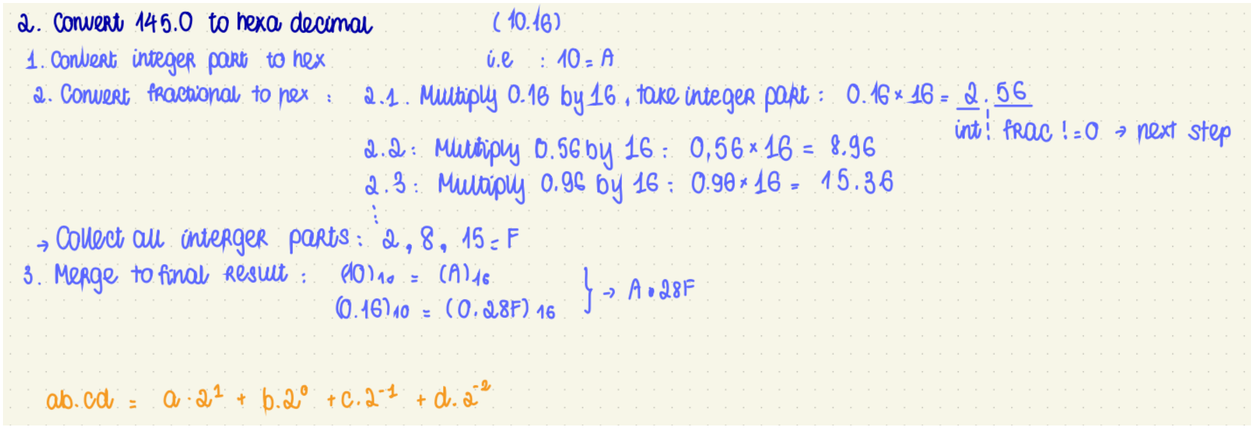
2 complementary → decimal

1111 1111 0000 1100

⇒ $-2^{15} + 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^{9} + 2^{8} + 2^{3} + 2^{2} = -244$

1. Convert 244 to binary : 0000 0000 1 1110100
2. Switch digits to opposite : 1111 1111 0000 1011
3. Add 1 + 1

1111 1111 0000 1100



2. Convert 145.0 to hexa decimal (10.16)

1. Convert integer part to hex i.e : 10 = A
2. Convert fractional to hex : 2.1. Multiply 0.16 by 16, take integer part : $0.16 \times 16 = \underset{\text{int}}{\underline{2}}.\underset{\text{frac}}{\underline{56}}$!= 0 → next step

2.2: Multiply 0.56 by 16 : $0,56 \times 16 = 8.96$

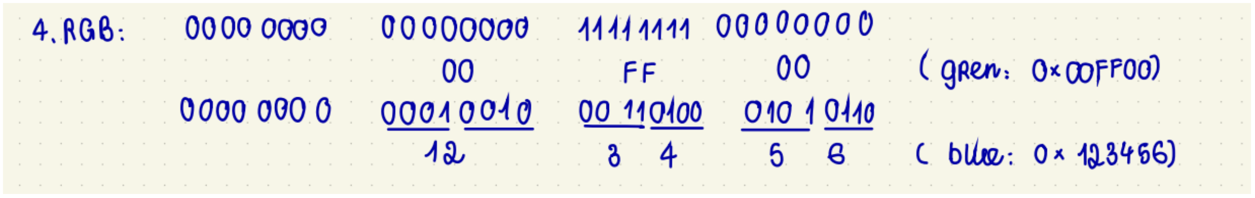
2.3: Multiply 0.96 by 16 : $0.96 \times 16 = 15.36$

⋮

→ Collect all interger parts: 2, 8, 15 = F

3. Merge to final result : $(10)_{10} = (A)_{16}$, $(0.16)_{10} = (0.28F)_{16}$ } → A.28F

$ab.cd = a \cdot 2^1 + b.2^0 + c.2^{-1} + d.2^{-2}$



4. RGB: 0000 0000 00000000 11111111 00000000

00 FF 00 (green: 0x00FF00)

0000 000 0 $\underbrace{0001\ 0010}_{12}$ $\underline{0011}\ \underline{0100}$ $\underline{0101}\ \underline{0110}$

3 4 5 6 (blue: 0x 123456)

Round 2:

What are THREE objectives of an OS design?:

Convenience: An operating system makes a computer more convenient to use

Ability to evolve: An operating system should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service

Efficiency: An operating system allows the computer system resources to be used in an efficient manner

SaaS, PaaS, IaaS (Software as a service/Platform/Infrastructure)

SaaS allows the user to use the software provided by the service.

PaaS allows the user to deploy their own software on a scalable platform provided by the service. This software typically has to be supported by the underlying software stack.
IaaS allows the user to deploy their own software stack on the virtual machines provided by the service.

One of the main advantages of SaaS (from user perspective) is the ease of use.
A typical PaaS allows the available computational resources, along with the usage costs, to be (manually or automatically) scaled based on the current needs.
Cloud service servers typically have hardware that supports virtualization. For example, CPU virtualization allows multiple users to simultaneously use the same CPU in isolation without having any knowledge about the other users.

x86-64 can natively run x86 programs
In their CPU, typical smartphones use ARM
x86-64, AMD64 and Intel 64 are essentially the same architecture with a different marketing name and minor differences
Multiprogramming is a mode of operation that provides for the interleaved execution of two or more computer programs by a single processor
A process is program in execution, which is controlled and scheduled by the operating system

Uniprogramming scheduling and Multiprogramming scheduling:
Uni-programming and multiprogramming are two different scheduling techniques used in operating systems.

In uniprocessing scheduling, only one process executes on the CPU at a time. The CPU executes the process until it finishes, and then starts the next process. It is a simple and easy scheduling technique but it is inefficient because the CPU may remain idle while waiting for I/O or other events. It is generally used in single-user systems and small embedded systems.

In multiprogramming scheduling, multiple processes are loaded into the main memory simultaneously, and they share the CPU. The CPU switches between the processes and executes them in a time-sharing manner. When a process requires I/O or waits for an event, the CPU switches to another process and continues execution. It is more efficient as compared to uniprocessing because it keeps the CPU busy and allows concurrent execution of multiple processes. Multiprogramming scheduling is used in most modern operating systems, including Windows, Unix, and Linux.

Turnaround time = actual time to complete a job = end time - start time for a job
Throughput = average number of jobs completed per time period T = jobs/total running time of all jobs
Processor utilization = percentage of time that the processor is active (not waiting) = CPU time/total running time of all jobs

Suppose we have three jobs in a multiprogramming computer system: JOB1, JOB2, JOB3.

1. JOB1 requires 6s of CPU time
2. JOB2 requires 4s of CPU time
3. JOB3 requires 2s of CPU time

We define the following quantities for system utilization:

1. Turnaround time = actual time to complete a job
2. Throughput = average number of jobs completed per time period T
3. Processor utilization = percentage of time that the processor is active (not waiting)

The multiprogramming system follows a simple round-robin scheduling (processes are scheduled in the order of their number, e.g. JOB1 is first). Each process gets 2s of CPU time turn-wise in a circular manner.

a) **2 points**

What is the turnaround time for JOB1?

Answer is an integer (in seconds)

b) **3 points**

What is the throughput for the system?

Answer is in float (per second)

c) **2 points**

What is the processor utilization percentage?

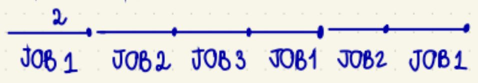
Answer is an integer (percentage)

Multiprogram Scheduling

. JOB 1 = 6s , JOB 2 = 4s , JOB 3 = 2s

. JOB1 > JOB2 > JOB 3 : each process gets 2s of CPU runtime circular manner

a. Turnaround time JOB 1



→ turnaround = 2 * 6 = 12s

b. Throughput = average number of jobs completed in period T

3 jobs / 12 sec ⇒ throughput = $\frac{3}{12}$ = 0.25

c. Process utilization = % time processor is active (not waiting)

There's no wait → utilization = 100%

2. Uniprogramming scheduling

Suppose we have two jobs in a uniprogramming computer system: JOB1, JOB2.

1. JOB1 requires 2s of disk time, followed by 4s of CPU time, followed by 2s of I/O time
2. JOB2 requires 0s of disk time, followed by 8s of CPU time, followed by 8s of I/O time

We define the following quantities for system utilization:

1. Turnaround time = actual time to complete a job
2. Throughput = average number of jobs completed per time period T
3. Processor utilization = percentage of time that the processor is active (not waiting)

We assume the processor is NOT utilized for any disk or I/O tasks.

a) **2 points**

What is the turnaround time for JOB2?

Answer is an integer (in seconds)

b) **3 points**

What is the throughput for the system?

Answer is a floating point number (3 decimal places)

c) **2 points**

What is the processor utilization percentage?

Answer is an integer (percentage)

Uniprogram Scheduling

JOB 1 : 2s disk time + 4s CPU time + 2s I/O time

JOB 2 : 0s ______ + 8s ______ + 8s ______

(processor not utilize for any disk or I/O task)

a. Turn around time JOB 2 :

JOB 1 > JOB 2 $\rightarrow$ turnaround = $2+4+2+0+8+8 = 24$ (s)

b. Throughput = $\frac{2 \text{ jobs}}{24} = 0.083$

c. Utilization : $\frac{4+8}{2+4+2+0+8+8} = \frac{12}{24} = 0.5$ (50%)

Round 3:

1. What is an instruction trace? a listing of the sequence of instructions that execute for a process
2. What does it mean to preempt a process?
Preemption is defined to be the reclaiming of a resource from a process before the process has finished using it.
The OS interrupts the running process and dispatches another process.

3. Which are characteristics of a suspended process?
The process is not immediately available for execution.
The process is always waiting on an event.
The process is not in main memory.

4. For what types of entities does the OS maintain tables of information for management purposes? memory tables and I/O tables

5. What are the elements of a process image?
the user stack
the private address space of the process

6. What are the steps performed by an OS to create a new process?
The OS builds the data structures used to manage the process, and allocates address space in main memory to the process.

7. Select the most complete definition of an interrupt:
A signal from a hardware device and is independent of the currently running process, such as the completion of an I/O operation.

8. Which are valid examples of interrupts?
software interrupts, hardware interrupts
I/O interrupts, timer interrupts

1. Priority scheduling

A system adopts a priority-based preemptive scheduling where a process arrives with an initial priority number. The system will always schedule the process with the highest priority (LOWER priority number = HIGHER priority). If multiple processes have the same priority, the one that arrived sooner will be scheduled (e.g. P1 will be scheduled over P2). Every time a process is running uninterrupted for 5ms, the system will increment its priority number by 1 - making the scheduling "fairer" by trying to unblock processes. The dispatcher takes 2 ms for a process switch, where no process is being executed during the switch. We also assume processing can always resume from where it stopped when a process is re-scheduled.

In a recorded time span, the system has 3 processes: P1, P2, P3. As shown in the following table:

Process ID	Initial Priority	Arrival time(ms)	CPU time required to finish(ms)
P1	1	0	15
P2	2	5	8
P3	1	10	4

Hint: This exercise is much easier if you draw a timing diagram - to visualize what is happening at every 1 ms.

a) **2 points**

What is the turnaround time (in ms) for process P1?

Answer is an integer

b) **2 points**

What is the turnaround time (in ms) for process P2?

Answer is an integer

c) **1 point**

What is the turnaround time (in ms) for process P3?

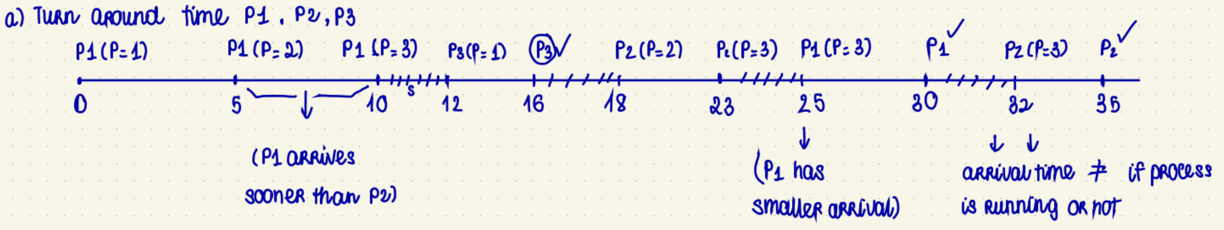
Answer is an integer

Priority Scheduling

- if processes have same priority : FCFS
- every time process run 5s w/o interrupt → priority +1
- process switch = 2s (no process allowed to run during switch)
- process resumes from where it stops

PID	Initial Priority	Arrived Time	Completed time
P1	1	0	15
P2	2	5	8
P3	1	10	4

a) Turn around time P1, P2, P3



turn around time = time complete - time init (start switching from prev process to current process)

turn around time P1 = 30 - 0 = 30ms

turn around (P2) = 35 - 16 = 29

turn around (P3) = 16 - 10 = 6 (ms)

(4 : running time + 2 sw time)

3. CPU and DMA

With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller when the operation is done. DMA module is transferring characters to main memory from an external device transmitting at 10800 bits per second (bps). Each character is 8 bits wide. The processor can fetch instructions at the rate of 1 million instructions per second. Assume that: a) the CPU is only fetching instructions, b) each fetch takes one cycle, c) interrupt handling does not consume CPU cycles.

a) 4 points

By how much will the processor be slowed down due to the DMA related activity (initiate transfer and handle interrupt)? In other words, what is the percentage of CPU cycles "stolen" by the DMA operations?

Answer is a int obtained after multiplying the two decimals precision (in percentage) float by 100. For example, for 0.99% enter 99 and for 12.67% enter 1267.

CPU and DMA

- DMA : CPU init transfer ⇒ does other operation while transfer in process ⇒ receive interrupt when transfer done
- Transfer Rate DMA : 10800 bps (bits / second) . Each char = 8 bits wide
- Processor fetch rate : 10^6 instructions/s , Each fetch takes 1 cycle

How much does CPU slow down due to DMA activity

- No of character transfered = 10800 / 8 = 1350 (chars)
- No of instructions / cycle = 10^6 (instruction/ cycle)

$\Big\rangle$ Slowdown = $\dfrac{\text{no of chars}}{\text{no of instruction}} = \dfrac{1860}{10^6} = 0.135$

2. Process trace

Suppose that three interleaved processes are running in a system having start addresses 4050, 3200, 6700. The system's dispatcher has a starting address of 200 and the dispatcher cycle has 2 instructions. The system uses a Round-Robin scheduler where the dispatcher is invoked after 4 instructions OR for an interrupt OR when a process finishes.

The traces of the processes and the dispatcher are as follows:

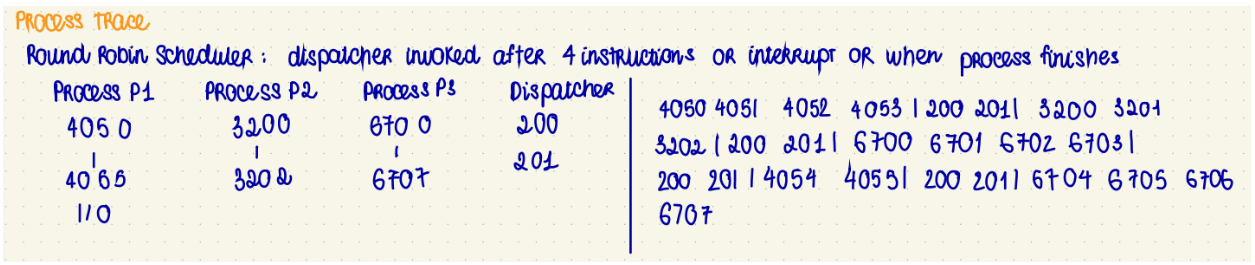
Process P1	Process P2	Process P3	Dispatcher
4050	3200	6700	200
4051	3201	6701	201
4052	3202	6702	
4053		6703	
4054		6704	
4055		6705	
I/O		6706	
		6707	

a) 5 points

What is the trace of instructions being executed until all 3 processes complete?

Write out the answer in the field below as a list of space-separated addresses. E.g:

```
4050 4051 3020 3021 3033 200 203
```



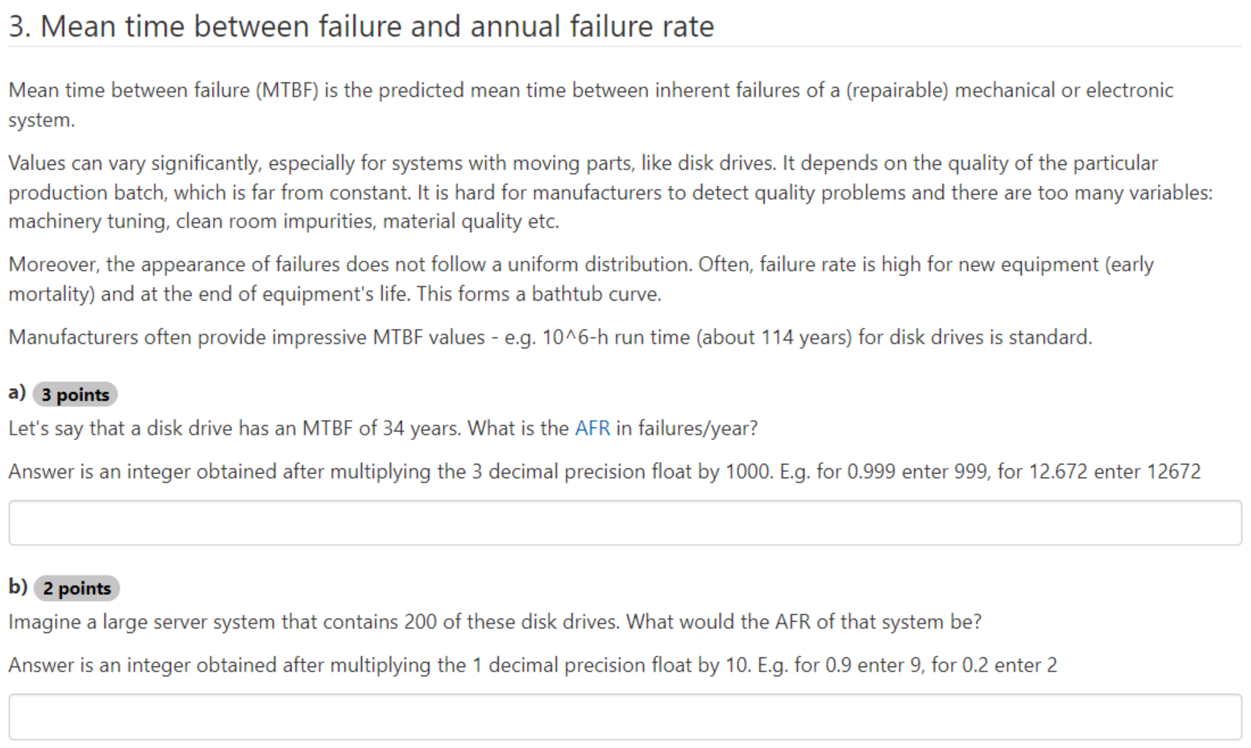
PROCESS TRACE

Round Robin Scheduler: dispatcher invoked after 4 instructions OR interrupt OR when process finishes

PROCESS P1	PROCESS P2	PROCESS P3	Dispatcher
4050	3200	6700	200
|	|	|	201
4055	3202	6707	
I/O			

4050 4051 4052 4053 | 200 2011 3200 3201
3202 | 200 2011 6700 6701 6702 6703 |
200 201 | 4054 4055 | 200 201 | 6704 6705 6706
6707

A trace of a process typically refers to a log or recording of the sequence of events and actions that occur during the execution of a program. This can include information such as the instructions executed, system calls made, and data accessed or modified by the program.



3. Mean time between failure and annual failure rate

Mean time between failure (MTBF) is the predicted mean time between inherent failures of a (repairable) mechanical or electronic system.

Values can vary significantly, especially for systems with moving parts, like disk drives. It depends on the quality of the particular production batch, which is far from constant. It is hard for manufacturers to detect quality problems and there are too many variables: machinery tuning, clean room impurities, material quality etc.

Moreover, the appearance of failures does not follow a uniform distribution. Often, failure rate is high for new equipment (early mortality) and at the end of equipment's life. This forms a bathtub curve.

Manufacturers often provide impressive MTBF values - e.g. 10^6-h run time (about 114 years) for disk drives is standard.

a) **3 points**

Let's say that a disk drive has an MTBF of 34 years. What is the AFR in failures/year?

Answer is an integer obtained after multiplying the 3 decimal precision float by 1000. E.g. for 0.999 enter 999, for 12.672 enter 12672

b) **2 points**

Imagine a large server system that contains 200 of these disk drives. What would the AFR of that system be?

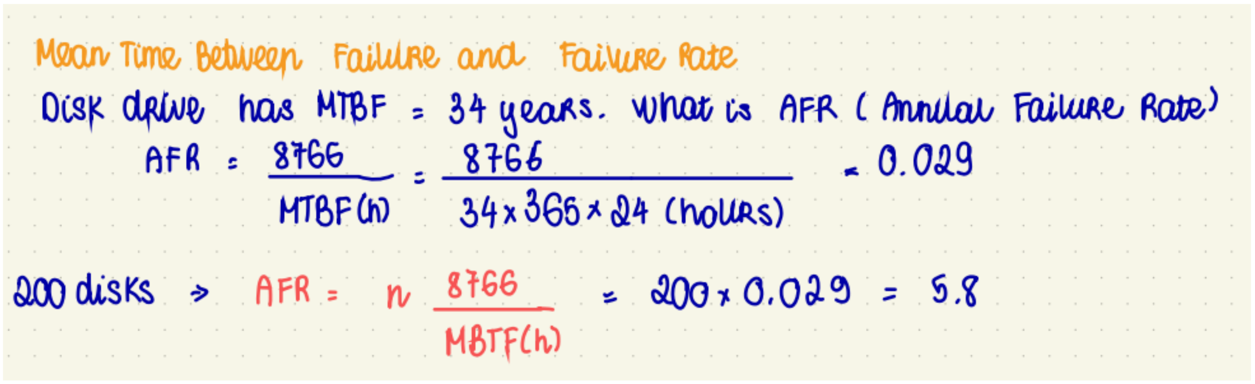
Answer is an integer obtained after multiplying the 1 decimal precision float by 10. E.g. for 0.9 enter 9, for 0.2 enter 2

Mean Time Between Failure (MTBF) is a measure of the reliability of a system. It represents the average amount of time that elapses between two failures of a system. MTBF is often used in the context of complex systems, such as computer hardware or industrial machinery, and is a key metric in determining the system's overall reliability.

MTBF is calculated by dividing the total time the system was operational by the number of failures that occurred during that time. For example, if a system operated for 1,000 hours and experienced 10 failures during that time, the MTBF would be 100 hours.

Annual failure rate (AFR) is a metric used in the computer industry to estimate the likelihood of a hardware component failing during a one-year period. It is usually expressed as a percentage or in hours of operation. AFR is often used as a measure of the reliability of a device or system and is an important factor in determining the warranty and maintenance requirements for the

hardware. A higher AFR indicates a greater likelihood of failure and may require more frequent maintenance or replacement of the hardware.
a) AFR single = 1/MTBF = 1/34 = 0.029
b) AFR large system = AFR single * number of systems = 1/34 * 200 = 5.8

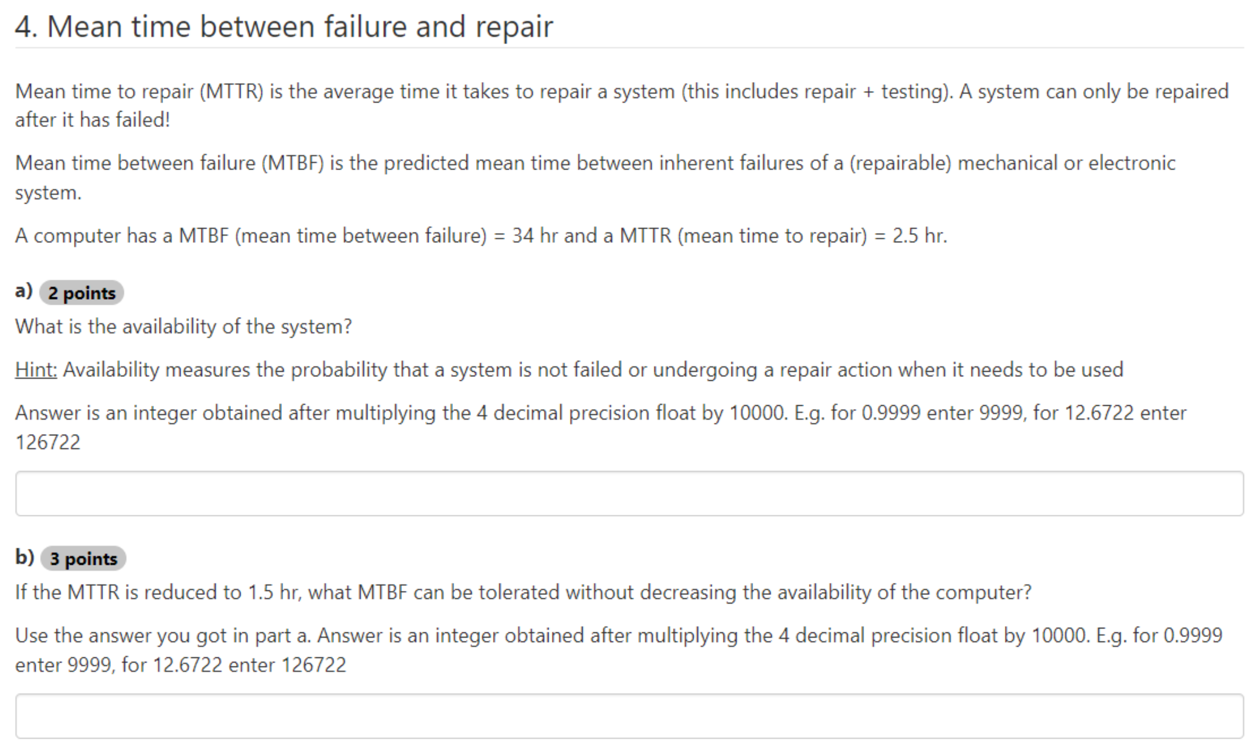


Mean Time Between Failure and Failure Rate

Disk drive has MTBF = 34 years. What is AFR (Annual Failure Rate)

$$AFR = \frac{8766}{MTBF(h)} = \frac{8766}{34 \times 365 \times 24 \text{ (hours)}} = 0.029$$

200 disks $\Rightarrow$ $AFR = n \dfrac{8766}{MBTF(h)} = 200 \times 0.029 = 5.8$



4. Mean time between failure and repair

Mean time to repair (MTTR) is the average time it takes to repair a system (this includes repair + testing). A system can only be repaired after it has failed!

Mean time between failure (MTBF) is the predicted mean time between inherent failures of a (repairable) mechanical or electronic system.

A computer has a MTBF (mean time between failure) = 34 hr and a MTTR (mean time to repair) = 2.5 hr.

a) **2 points**

What is the availability of the system?

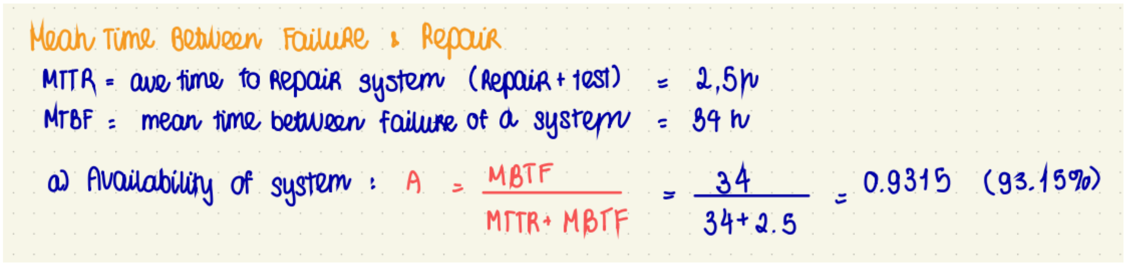
Hint: Availability measures the probability that a system is not failed or undergoing a repair action when it needs to be used

Answer is an integer obtained after multiplying the 4 decimal precision float by 10000. E.g. for 0.9999 enter 9999, for 12.6722 enter 126722

b) **3 points**

If the MTTR is reduced to 1.5 hr, what MTBF can be tolerated without decreasing the availability of the computer?

Use the answer you got in part a. Answer is an integer obtained after multiplying the 4 decimal precision float by 10000. E.g. for 0.9999 enter 9999, for 12.6722 enter 126722



Mean Time Between Failure & Repair

MTTR = ave time to repair system (repair + test) = 2,5h
MTBF : mean time between failure of a system = 34 h

a) Availability of system : $A = \dfrac{MBTF}{MTTR + MBTF} = \dfrac{34}{34 + 2.5} = 0.9315$ (93.15%)

a) availability = MTBF/(MTBF + MTTR) = 34/(34+2.5) = 93.15%
b) keeping the ratio similar, we need to solve the equation
MBTF/(MBTF + 1.5) = 34/(34+2.5) => 20.4 hours

Round 4:
1. What is one reason mode switch between threads may be cheaper than a mode switch between processes. Less processing of information is involved
A mode switch, also known as a context switch, is a process by which a computer's operating system switches the processor from running in one context to running in another. Context switching can be expensive in terms of time and resources, and the cost is typically related to the amount of memory that must be saved and restored for each context.

In a multi-threaded environment, threads share the same memory space and can access the same resources, which makes the context switch between threads relatively cheap compared to a context switch between processes. When a thread is switched out, the operating system only needs to save and restore the processor state and the thread's stack, which is a relatively small amount of memory compared to the entire process's memory space.

On the other hand, a context switch between processes is more expensive because the operating system needs to save and restore the entire memory space of the process, which includes the program code, data, and stack. This can be a large amount of memory and can take more time to save and restore.

2. What are the two separate and potentially independent characteristics embodied in the concept of a process? Resource ownership and scheduling/execution

3. What are the factors affecting relocation of a program?
In a multiprogramming system, the available main memory is generally shared among a number of processes
Once a program is swapped out to disk, it is limiting to specify when it is next swapped back in

Program relocation refers to the process of changing the memory address at which a program is loaded into memory. In a typical computer system, a program is compiled with absolute addresses, which means that the program expects to be loaded into a specific memory location. However, this can create problems if multiple programs are to be executed at the same time or if a program needs to be moved to a different location in memory.

To address these issues, program relocation is used. The relocation process involves modifying the absolute addresses in the program to relative addresses that are independent of the location in memory where the program is loaded. This allows the program to be loaded into any available memory location, making it possible for multiple programs to be executed simultaneously without interfering with each other.

4. Frame and page
a frame is a fixed-size block of physical memory while a page is a fixed-size block of virtual memory. The operating system divides the virtual address space of a process into pages, which are mapped to physical frames in main memory. Each page corresponds to a contiguous block of virtual addresses, and each frame corresponds to a contiguous block of physical addresses.

The mapping between virtual pages and physical frames is maintained by the page table, a data structure that keeps track of the correspondence between virtual pages and physical frames. When a process accesses a virtual address, the operating system consults the page table to determine the physical address corresponding to that virtual address, and then performs the necessary memory operation (e.g., read or write) on the physical memory.

Main memory is divided into blocks of the same size called frames

5. disadvantage of using kernel-level threads.
The transfer of control from one thread to another within the same process requires a mode switch to the kernel

One of the main disadvantages of using kernel-level threads is that they are slower to create and manage than user-level threads. This is because creating and switching between kernel-level threads requires system calls to the operating system kernel, which can be more time-consuming than user-level thread operations, which can be done entirely in user space.

Another disadvantage of kernel-level threads is that they can be less efficient in terms of resource usage. Since kernel-level threads are managed by the operating system, they may require more system resources, such as memory and CPU time, than user-level threads, which can be managed more efficiently by the application itself.

6. Segment of a program
a segment is a portion of a program or process that contains related code or data. It is a logical unit of memory management that allows the operating system to allocate and manage memory in a more flexible way than a simple, continuous block of memory.

It is not required that all segments of all programs be of the same length, although there is a maximum segment length

user vs kernel
In operating systems, the term "user" typically refers to an entity or process that interacts with the system or uses its resources. Specifically, a user is a person or program that runs on top of the operating system and requests its services or accesses its features, such as running applications or accessing files. User-level programs operate within their own address space and have limited access to the system resources, whereas kernel-level programs, which operate within the operating system's kernel, have full access to all resources and system operations.

Round 5:

1. What are valid examples of the use of threads in a single-user multiprocessing system.
Foreground/background work
Asynchronous processing
Modular program structure

2. What are the advantages of using multithreading over multiple processes?
Multithreading saves the overhead of two mode switches (user to kernel; kernel back to user)
In multithreading the scheduling algorithm can be tailored to the application without disturbing the underlying OS scheduler

3. Which of the statements are valid for threads/threading in the context of Clouds operating system?
The approach provides an effective way of insulating both users and programmers from the details of the distributed environment
Threads may move from one address space to another, and actually span computer boundaries
A thread in clouds is a unit of activity from the users perspective

4. What requirements is memory management intended to satisfy?
- Relocation - A process that has been swapped out to a disk can be moved to a different memory location than the one it was in previously.
when a process is swapped out to disk, it means that the process's memory contents are moved from the computer's main memory (RAM) to the hard disk. This is done to free up memory for other processes that are actively running, as RAM is a finite resource.

When a process is swapped out, its memory contents are written to a special area on the hard disk called the swap space. The swap space is typically allocated when the operating system is installed, and it is used to store the memory contents of processes that are not currently being used.

When the operating system needs to run the swapped-out process again, it will read the process's memory contents from the swap space back into the computer's main memory. This is known as swapping in. Swapping in and out is managed by the operating system's memory management system and is transparent to the user and the running processes.

- Protection - Each process should be protected from unwanted interference by other processes, so programs in other processes should not be able to reference memory locations in a process for reading or writing purposes without permission; satisfied by the processor (hardware)
- Sharing - Allowing several processes to access the same portion of main memory. Memory management system must allow controlled access to shared areas of memory without compromising essential protection

- Logical organization - Enabling the OS and computer hardware to deal with user programs and data in the form of modules of some sort
- Physical organization - The organization of the flow of information between main and secondary memory

5. thrashing is a phenomenon in which the processor spends most of its time swapping pages between physical memory and disk, instead of executing actual user processes. Thrashing typically occurs when a system is under heavy memory pressure and does not have enough physical memory to hold all of the required pages in memory.

6. module
a module refers to a self-contained block of code that can be loaded and unloaded dynamically into the kernel at runtime. Modules provide a way to add functionality to a running system without the need to recompile the entire kernel or reboot the system. Modules can be device drivers, filesystems, or other kernel features that are not required for the basic operation of the system. The use of modules can help to reduce the size of the kernel and improve its performance by allowing only the necessary code to be loaded into memory.

What are the advantages of organizing programs and data into modules?
Modules can be written and compiled independently
With modest additional overhead, different degrees of protection can be given to different modules
It is possible to introduce mechanisms by which modules can be shared among processes

7. What are some reasons to allow two or more processes to all have access to a particular region of memory?
Processes that are cooperating on some task may need to share access to the same data structure
If a number of processes are executing the same program, it is advantageous to allow each process to access the same copy of the program rather than have its own separate copy

8. System Calls with Linux

The UNIX system provides several system calls to create and end program, to send and receive software interrupts, to allocate memory, and to do other useful jobs for a process.

Round 6: Driver in Linux
alloc_chrdev_region allocates and registers a range of char device numbers. The third parameter may be equal to MAX_DEVICE_NUM, to cover all possible minor numbers

The kernel will dynamically allocate a major number for our character device if this function is called

What is true about device initialization
Device initialization does not make the device available to the system immediately

Device initialization has the file operations for the device as an argument

What is true about setting the owner of the device in this case?
Owner field of the structure should be initialized in order to protect against ill-advised module unloads while the device is active.
The owner is initialized using a macro defined in <linux/module.h>
Setting the owner of the device is mandatory

What is true about adding a device to the system in this case?
The function used for adding the device has the number of consecutive minor numbers corresponding to the device as an argument
Together with initialization, the function used for adding a character device registers it to the VFS (Virtual File System)
The device is live immediately to the system after the function used for adding is called and the kernel can invoke its operations

What is true regarding device_create?
After calling device_create, a struct device will be created in sysfs, registered to the specified class
If it succeeds, device_create will create a /dev/chdev-x device (where x is one of the i values) in the /dev directory
The struct class passed to this function must have previously been created with a call to class_create

TODO 8. What is true regarding device_destroy?
Its call unregisters and cleans up a device that was created with a call to device_create
One cannot call device_destroy on a device that is still opened by a process
The function takes a pointer to the struct class the device was registered with as an argument

Round 7:

1. Which of the following design issues is the concept of concurrency relevant to?
communication among processes
sharing of and competing for resources
synchronization of the activities of multiple processes
allocation of processor time to processes

2. Which of the statements are true about competing processes and cooperating processes?
competing processes need access to the same resource at the same time, such as a disk, file, or printer.
cooperating processes either share access to a common object, such as a memory buffer or are able to communicate with each other

3. Which are the control problems associated with competing processes?
deadlock
starvation

4. What is true about binary and general semaphores?
a general semaphore can have any integer values
a binary semaphore may only take on the values 0 and 1

5. What is true about a mutex and a binary semaphore?
the process that locks a mutex must be the one to unlock it
for a semaphore, if the operation wait(s) is executed by one process, the operation signal(s) can be executed by any process
multiple number of threads can acquire a binary semaphore at the same time concurrently

6. Which characteristics of monitors mark them as high-level synchronization tools?
condition variables
locks

7. What is true about direct and indirect addressing with respect to message passing?
in direct addressing, the process must know ahead of time from which process a message is expected
In direct addressing, the sender of the message explicitly specifies the process that should receive the message. This is similar to sending a letter through the postal service: the sender knows the address of the recipient and writes it on the envelope. Direct addressing is simple and efficient, but it requires the sender to know the identity of the recipient.

In indirect addressing, the sender of the message specifies a "mailbox" or "port" that is associated with the recipient process, rather than the process itself. The recipient process can then read the message from the mailbox at its convenience. This is similar to leaving a message in a voicemail box: the sender doesn't need to know where the recipient is, but can leave a message that will be picked up later. Indirect addressing is more flexible than direct addressing, but can be less efficient due to the additional indirection involved.

8. What conditions are generally associated with the readers/writers problem?
The readers/writers problem is a common synchronization problem in concurrent programming. It is associated with a shared resource that can be accessed by multiple processes or threads, where some processes may only read the resource while others may write to it.

The main conditions associated with the readers/writers problem are:

Multiple processes or threads have access to a shared resource.
Some processes or threads may only read from the shared resource, while others may write to it.

Reads and writes must be mutually exclusive. That is, only one process or thread can write to the shared resource at a time, and no other process or thread can read or write while a write is in progress.
Multiple reads can occur simultaneously, as long as there are no writes in progress. This allows for higher concurrency and better performance.
The solution must avoid starvation and ensure fairness. That is, all processes or threads must eventually have the opportunity to access the shared resource, and no process or thread should be unfairly prevented from doing so.

9. What are the conditions(some/all) that must be present for deadlock to occur?
mutual exclusion
hold-and-wait
no pre-emption
circular wait

10. How can the hold-and-wait condition be prevented?
by requiring that a process requests all of its required resources at one time, and blocking the process until all requests can be granted simultaneously

11. Why you cannot disallow mutual exclusion in order to prevent deadlocks?
there are some resources (like printers) that are inherently non-sharable, and it is impossible to disallow mutual exclusion. Then if mutual exclusion is disallowed, then all non-sharable resources become sharable

12. How can the circular wait condition be prevented?
by defining a linear ordering of resource types
The circular wait condition can be prevented by imposing a total ordering on all resource types and ensuring that processes can only request resources in a increasing order of the assigned number to each resource type. This ensures that processes can only acquire resources in a certain order, breaking any circular dependency between resources.

13. Which of the methods may be adopted to recover from deadlocks.
Abort all deadlocked processes
Successively abort deadlocked processes until deadlock no longer exists
Successively preempt resources until deadlock no longer exists
Back up each deadlocked process to some previously defined checkpoint, and restart all processes.

Round 8: Using the Linux shell
14. Which ones of the following are examples of a UNIX shell?
Bash
Zsh

In Unix and Unix-like operating systems, a shell is a command-line interpreter that provides a user interface for accessing the operating system's services. It allows users to interact with the system by typing commands and running scripts, which the shell then executes. The shell can also perform basic programming tasks, such as file and process management, and can be customized through the use of shell scripts and environment variables. Popular Unix shells include Bash, Korn shell, and C shell.

Which of the below commands will list the contents of the current directory, including hidden files and the read/write/execute rights of each file?
ls -la
ls -l -a
ls -al

Which of the below commands will print out the value of the HOME variable?
echo $HOME
echo ${HOME}

What commands can we use to look at some contents of a file without fully loading it into memory?
tail
head
less

What will happen when the following command is executed?: touch data_a.txt && mkdir data_a || mkdir data_b && touch data_b.txt
2 files are created(data_a.txt and data_b.txt). 1 directory is created: (data_a)
The command touch data_a.txt && mkdir data_a || mkdir data_b && touch data_b.txt will create a file named data_a.txt and a directory named data_a if it doesn't already exist. If data_a already exists, then it will move to the next command, which is to create a directory named data_b. If data_b is successfully created, the last command will create a file named data_b.txt in the data_b directory.

Which commands will result in a new text file data.txt being created in the current directory?
echo data >> data.txt
touch data.txt

Which commands will print out the contents of the data.txt file?
cat data.txt
cat data.txt >> proxy.txt && cat proxy.txt
cat data.txt | cat

The command cat data.txt will print out the contents of the data.txt file.

The command cat data.txt >> proxy.txt && cat proxy.txt will append the contents of data.txt to the proxy.txt file and then print the contents of the proxy.txt file.

The command cat data.txt | cat will also print out the contents of the data.txt file by using a pipe to send the output of the cat command to another cat command.

What is the function of the /dev/null node in the Linux file system?
It is a node that discards any written data

Which Linux command can be used to terminate a background process via process signalling?
kill

Which one of the following is NOT a standard UNIX data stream?
stderr
stdio (answer)
stdout
stdin

Which pairs of commands can we use to create and delete directories in the Linux file system?
mkdir & rmdir
cp & rmdir

Given a Linux terminal, how can we determine the shell we are currently running?
Using an environment variable
The command “ps -p $$”

Which one of the following is NOT a Linux process state?
Running
Zombie
Stopped
Paused (answer)

"Paused" is not a Linux process state. The valid process states in Linux are:

Running: the process is currently running on a CPU.
Sleeping: the process is waiting for a condition to be met or an event to occur.
Stopped: the process has been stopped, usually by a signal.
Zombie: the process has completed execution but its parent process has not yet read its exit status.
Dead: the process has completed execution and its parent process has read its exit status.

Signals are short, fast, one-way messages sent to processes such as scripts, programs, and daemons. They let the process know about something that has happened.

The Linux kernel sends signals to processes about events they need to react to. Well-behaved scripts handle signals elegantly and robustly and can clean up behind themselves even if you hit Ctrl+C. Linux uses a lot of signals, as we shall see, but from a scripting point of view, there's only a small subset of signals that you're likely to be interested in. In particular, in non-trivial scripts, signals that tell the script to shut down should be trapped (where possible) and a graceful shutdown performed.

For example, scripts that create temporary files or open firewall ports can be given the chance to delete the temporary files or to close the ports before they shut down. If the script just dies the instant it receives the signal, your computer can be left in an unpredictable state.

We can use the trap command with the -l (list) option to show us the entire list of signals that Linux uses.

In a Linux or Unix shell environment, "trap" is a command used to set up an action to be taken when a signal is received. Signals are software interrupts that are used to communicate events or requests to a process or to terminate a process. The "kill" command is used to send a signal to a process or a group of processes to terminate them or to request that they take some other action, depending on the signal sent.

What is true regarding the kill and trap command?
we can use the number of a signal in the kill command instead of its name
we can use multiple signals with the kill command
we need to use signals QUIT and INT to quit and interrupt the process
the signal given to the kill command will be caught by the trap statement written in the script
the kill command sends the signal (given as a parameter) to the process created by the script
we need to use the PID of the process we just created for running kill
Hitting Ctrl+C has the same effect as sending SIGINT to our process

Round 9: Pthreads
Pthreads (short for POSIX threads) is a standard for multithreading in the C programming language. It is a set of library functions and associated headers that implement thread creation, thread synchronization, and thread termination. Pthreads provides a way to create lightweight threads within a process, which can run concurrently and share resources, such as memory and file descriptors. This allows programs to take advantage of multi-core processors and perform multiple tasks at the same time, increasing their efficiency and performance. The Pthreads library is standardized by the IEEE and is available on many Unix-like operating systems, including Linux, macOS, and FreeBSD.