ASSIGNMENT – I (SOLUTIONS)

Q1. I have purchased a laptop which consists of three types of memory: cache memory, RAM and Hard disk. I am excited to know why laptop comes with the cache memory. If cache memory is so useful and important, why we cannot replace other memories with it? Kindly help me in finding answers of these two questions.

Answer:

Cache memory is a type of high-speed volatile computer memory that provides high-speed data access to a processor and stores frequently used computer programs, applications, and data. It's an integral part of modern computer architecture and serves a specific purpose that complements the roles of RAM (Random Access Memory) and a Hard Disk Drive (HDD) or Solid-State Drive (SSD).

Cache memory is expensive to manufacture compared to RAM and storage. It is usually smaller in size and faster, which drives up the cost. To maintain affordability, laptops use a limited amount of cache memory, typically in the form of L1 (Level 1) and L2 (Level 2) caches. L3 (Level 3) cache, which is larger and slower than L1 and L2, is sometimes shared among CPU cores.

Q2. Examine the given lines of code. How many numbers of Processes created?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
int i;
for (i = 0; i < 4; i++)
fork();
return 0;
}
```

Answer:

fork () system call is used to create a new process. fork () essentially duplicates the currently executing process, creating a child process.

Total Number of Processes = 2^n, where n is the number of fork system calls.

The code creates a total of 16 processes. Each process is a separate instance of the program, and they all run concurrently.

Explanation:

In the first iteration, one child process is created, resulting in two processes (the original process and the newly created child).

In the second iteration, each of the two existing processes (the original and the first child) creates a new child process. This results in a total of four processes (the original process and three child processes).

In the third iteration, each of the four existing processes creates a new child process. This leads to a total of eight processes (the original process and seven child processes).

In the fourth and final iteration, each of the eight existing processes creates another child process. This results in a total of 16 processes (the original process and 15 child processes).

Q3. Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithms for each queue, the criteria used to move processes between queues, and so on.

These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets?

a. Priority and SJF

b. Multilevel feedback queues and FCFS

c. Priority and FCFS

d. RR and SJF

Answer:

Let's examine the relations between the mentioned pairs of algorithm sets:

a. Priority and SJF (Shortest Job First):
Relation: Priority and SJF are different scheduling algorithms with distinct criteria.
Explanation: Priority scheduling assigns priorities to processes based on some criteria (e.g., process importance), and then it schedules processes with higher priorities first. In contrast, SJF schedules the shortest job (process with the smallest burst time) first, regardless of its priority. These two algorithms have different objectives and selection criteria, so they are not directly related.

b. Multilevel Feedback Queues and FCFS (First-Come, First-Served):
Relation: There is a hierarchical relation between these two.
Explanation: Multilevel Feedback Queues (MLFQ) is a scheduling algorithm that can incorporate various scheduling policies within its multiple queues, including FCFS. MLFQ uses multiple queues with different priorities and scheduling algorithms, including FCFS for some of the queues. Therefore, FCFS can be a part of the set of algorithms used within an MLFQ setup, making MLFQ more versatile and encompassing than FCFS alone.

c. Priority and FCFS (First-Come, First-Served):
Relation: Priority and FCFS are distinct scheduling algorithms.
Explanation: Priority scheduling assigns priorities to processes based on some criteria and schedules processes with higher priorities first. FCFS, on the other hand, simply schedules processes in the order they arrive. These two algorithms have different scheduling policies and objectives, so they are not directly related.

d. RR (Round Robin) and SJF (Shortest Job First):
Relation: RR and SJF are different scheduling algorithms.
Explanation: RR is a preemptive scheduling algorithm that allocates a fixed time slice (quantum) to each process in a cyclic manner. SJF, as mentioned earlier, schedules the shortest job first based on burst time. While both are scheduling algorithms, they have different criteria and scheduling strategies, so they are not directly related.
In summary, the relations between these pairs of scheduling algorithm sets vary. Some are distinct and unrelated, while others involve one algorithm being a part of a more comprehensive algorithm set (e.g., FCFS within MLFQ).

Q4. Consider three CPU-intensive processes, which require 10, 20 and 30 time units and arrive at times 0, 2 and 6, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end. [Gate 2006]
Answer:

2 context switches are required.

Process	Arrival Time	Burst Time
P1	0	10
P2	2	20
P3	6	30

Gantt Chart

P1	P2	P3

0 10 30 60

Since context switches at time zero and at the end (60) is not include, therefore two (2) context switches are required. One at time 10 for P1 to P2 and another at time 30 for P2 to P3.

Q5. You are an IT consultant working with a small business owner who is setting up a computer network for their office. The business owner has specific requirements for the operating system, and your task is to recommend the most suitable classification of operating system based on these requirements.
a) The business owner needs an operating system for their office desktop computers. They want an OS that is user-friendly, supports common office software, and is cost-effective. What classification of operating system would you recommend, and why?
b) The business owner also has a server in the office, and they require an OS that can handle multiple user connections, provide file sharing, and ensure data security. What classification of operating system would you recommend for the server, and why?
c) The business owner plans to integrate a real-time system for monitoring and controlling

Manufacturing equipment. What classification of operating system would you recommend for this specialized task, and why?

Answer:

a) For Office Desktop Computers:

I would recommend a Desktop Operating System for the office desktop computers. A suitable choice could be Microsoft Windows or macOS.

Why:

User-Friendly: Desktop operating systems like Windows and macOS are known for their user-friendly interfaces, making them easy for employees to navigate and use.

Office Software Support: Both Windows and macOS have a wide range of office software applications available, including Microsoft Office, which is commonly used in business environments.

Cost-Effective: These operating systems offer various editions, including cost-effective options, making them suitable for small businesses looking to manage expenses.

b) For the Server:

I would recommend a Server Operating System. Options include Windows Server or Linux Server distributions like Ubuntu Server or CentOS.

Why:

Multiple User Connections: Server operating systems are designed to handle multiple user connections simultaneously, making them ideal for serving data and resources to a network of users.

File Sharing: They include features and services such as file sharing, network printing, and centralized user management, which are essential for an office server.

Data Security: Server operating systems are optimized for security, with features like firewalls, user authentication, and encryption, ensuring the protection of sensitive business data.

c) For Real-Time System for Manufacturing Equipment:

I would recommend a Real-Time Operating System (RTOS) or a specialized industrial control system OS.

Why:

Predictable Timing: RTOSs are designed to provide predictable and deterministic timing for critical real-time tasks. This is essential for monitoring and controlling manufacturing equipment where timing precision is crucial.

Reliability: RTOSs are engineered to be highly reliable and fault-tolerant, minimizing the risk of system failures in critical industrial environments.

Specialized Hardware Support: RTOSs often have support for specialized hardware and communication protocols commonly used in manufacturing and industrial automation.

Minimal Latency: These systems are designed to minimize latency and response times, ensuring that control actions are executed with minimal delay.

It's important to select the right operating systems for each specific use case to optimize functionality, performance, and security in the business environment.

Q6. Find the average waiting time (A.W.T) and (A.T.A.T) for executing the following process using Preemptive short-job first

Process	**P1**	**P2**	**P3**	**P4**	**P5**
Burst time	**5**	**13**	**8**	**4**	**10**
Arrival time	**2**	**3**	**0**	**5**	**1**

Answer:

Process	AT	BT	CT	TAT	WT
P1	2	5	7	5	0
P2	3	13	40	37	24
P3	0	8	17	17	9
P4	5	4	11	6	2
P5	1	10	27	26	36

<u>Gannt Chart</u> :-

P3	P1	P4	P3	P5	P2

0 2 7 11 17 26 40

Average Turn Around Time = 18.2 unit.Average
Waiting Time = 14.2 unit.

Q7. Three processes A, B and C each execute a loop of 100 iterations. In each iteration of the loop, a process performs a single computation that requires tc CPU milliseconds and then initiates a single I/O operation that lasts for tio milliseconds. It is assumed that the computer where the processes execute has sufficient number of I/O devices and the OS of the computer assigns different I/O devicesto each process. Also, the scheduling overhead of the OS is negligible. The processes have the following characteristics:

Process id	**tc**	**tio**
A	**100 ms**	**500 ms**
B	**350 ms**	**500 ms**
C	**200 ms**	**500 ms**

The processes A, B, and C are started at times 0, 5 and 10 milliseconds respectively, in a pure time sharing system (round robin scheduling) that uses a time slice of 50 milliseconds. Calculate the timein milliseconds at which process C would complete its first I/O operation is

Answer:

This is round robin method where Q=50 ms(Given)

Process	AT	BT	I/O Time
A	0	100	500 Ms
B	5	350	500 Ms
C	10	200	500 Ms

Process C (AT=10)

It spends 50 Ms (First) in CPU Completion Which brings us to 60 Ms.At this point,

Process C initiate its I/O Operation.

Process C Completes its first I/O Operation = 60 Ms(Starting) + 500 Ms(I/O)
= 560 Ms

Q8. You are the administrator of a multi-user server. One of the users complains that their commands arenot executing as expected. Upon investigation, you find that their processes are stuck in the "Blocked"state due to waiting for a network resource. Explain how processes transition between states and howyou would address this issue.

Answer:

In the scenario, the user Process are stuck in the "Blocked" state due to waiting for anetwork resources (I/O Event).

- **Process can transition between these states.**

1. **Running to Blocked :-** When it initiates an I/O Operation.
2. **Blocked to Ready :-** Once the Required I/O Operation Completes.
3. **Ready to Running** :- When the CPU scheduler selects the process to run.

Running → Blocked → Ready → Running

Solution For Issues :-

1. Check New Resources.
2. Optimize Code.
3. Load Balancing.
4. Error Handling.

Q9. Imagine a simple round-robin scheduling algorithm with a time quantum of 10 milliseconds. Two processes, X and Y, are competing for CPU time. Process X consistently uses less than 10 millisecondsper quantum, while process Y requires more than 10 milliseconds to complete. How would the CPU scheduler handle this situation, and what might be the user's experience with these processes?

Answer:

Q=10 Ms

Process X Execution :-X

→ Given To CPU
If Process X completely its work within this time frame, It voluntarily fields the CPU andmove to the Terminate State.

Process Y **Execution :-**

- Process Y is given the CPU after Process`s time Quantum expires.
- When the Quantum time (10 ms) expire for Process Y, it is forcibly preempted byscheduler, moved back to “Ready State” .
- This scheduler safety the next process in line.

Q10. Imagine you're working with a group of non-technical colleagues who are curious about computers. While discussing operating systems, they ask, "Why do we even need an operating system in our computers?" How would you explain the main purpose and significance of an operating system in a relatable scenario or example?
Answer:

1. **Resource Management :-** The process to manage all the resources efficiently likeCPU, memory, input/output devices, and other hardware resources
2. **User Interface :-** The part of an operating system, program, or device that allows auser to enter and receive information
3. **File Management :-** Defined as manipulating files in a computer system, whichincludes creating, modifying, and deleting files.
4. **Security And Protection :-** Protection deals with who has access to the systemresources. Security gives the system access only to authorized users.
5. **Error Handling :-** The process of handling the possibility of failure.

Q11. If Mr. Abhishek aims to run his five programs, each of which typically takes 15 minutes to complete,but he's constrained to a time window of only 15-16 minutes, which operating system would you recommend for optimizing his task? Please explain your choice.
Answer:

Q 12. Under what circumstances, it is suitable to use time-sharing system rather than a PC or a single user workstation? Explain.
Answer:
Time-sharing systems are useful in a variety of situations where they provide advantages over a single-user PC or workstation. Time-sharing systems are intended to allow numerous users to share the resources of a single computer at the same time, and they are particularly beneficial in the following scenarios:

a. Sharing resources
b. Effectiveness
c. centralized management
d. Collaboration
e. Scalability
f. load balancing and
g. energy efficiency

In brief, time-sharing systems are appropriate for organizations or institutions that value the above factors. They provide an efficient and secure way for several users to share a single computer's resources, making them a viable alternative in a variety of settings.

Q13. . Examine the given lines of code. How many numbers of Processes created?

```
#include <stdio.h>
#include <unistd.h>
int main()
{
int i;
for (i = 0; i < 5; i++)
fork();
return 0;
}
```

Answer:

The included C code includes a loop that uses the fork() system function to create child processes. It is repeated for 5 times, and a new process is created each time by forking the old process. What occurs in each cycle is as follows:

1 st iteration : one process produces two (for a total of two processes).

2nd Iteration: Each of the two processes from the first iteration generates a new process (for a total of four).

3rd Iteration : Each of the four processes from the previous iteration generates a new process (for a total of eight processes).

4th Iteration: Each of the eight processes from the previous iteration generates a new process (for a total of 16 processes).

5th Iteration: Each of the sixteen processes from the previous iteration generates a new process (for a total of 32 processes).

So, in total after the execution of the above code, there will be 32 process in the end.

Q14. Consider a set of 5 processes whose arrival time, CPU time needed and the priority are given below:

Process number	**Arrival time(ms)**	**CPU time**	**Priority**
p1	**0**	**10**	**5**
p2	**0**	**5**	**2**
p3	**2**	**3**	**1**
p4	**5**	**20**	**4**
p5	**10**	**2**	**3**

If the CPU scheduling policy is SJF, the average waiting time (without preemption) will be how much ?

Answer :

Process number	Arrival time(ms)	CPU time	Priority	Completion time	TAT	WT
p1	0	10	5	40	40	30
p2	0	5	2	5	5	0
p3	2	3	1	8	6	3
p4	5	20	4	28	23	3
p5	10	2	3	30	20	18

so average waiting time will be = 54/5=10.8ms

Q15 : Consider the 3 processes, P1, P2, and P3 shown in the table.

Process	**Arrival Time**	**Time Units Required**
P1	**0**	**5**
P2	**1**	**7**
P3	**3**	**4**

The completion order of the 3 processes under the policy round robin scheduling with CPU quantum of 2 time units is

Answer : The order will be p1 p3 p2

Gantt Chart :

P1	P2	P1	P3	P2	P1	P3	P2			

0 2 4 6 8 10 11 13 16

Q16. Consider the following set of processes, with length of CPU bursts given in Millisecond as follows:

Process	**Burst Time**	**Arrival Time**	**Priority**
P1	**8**	**0**	**3**
P2	**1**	**1**	**1**
P3	**3**	**2**	**2**
P4	**2**	**3**	**3**
P5	**6**	**4**	**4**

a. **Draw the Gantt Charts for FCFS, SJF, Preemptive priority and RR(Quantum=2)**
b. **What is the turnaround time of each process for above algorithm?**
c. **What is the waiting time of each process for each of the above algorithm?**
d. **Which algorithm results in minimum average waiting time?**

Answer:

FCFS:

P1	P2	P3	P4	P5

0 8 9 12 14 20

SJF:

P1	P2	P4	P3	P5

0 8 9 11 14 20

RR:

P1	P2	P3	P1	P4	P5	P3	P1	P5

0 2 3 5 7 9 11 12 16 20

From above table, can calculate TAT, WT and AWT.

Process	FCFS	SJF	RR
P1	8	8	16
P2	9	9	3
P3	12	14	12
P4	14	9	9
P5	20	20	20

Q17. You are part of a team working on a robotics project for an industrial automation task. Your team needs to choose an appropriate operating system for the robot's control unit. The choice of operating system will impact the robot's performance and reliability.

a) Explain the key characteristics and requirements of the robotics project that should influence your choice of operating system classification.

b) Considering the project's requirements, justify whether a real-time operating system, a general- purpose operating system, or a specialized embedded operating system would be most suitable for the robot's control unit. Provide reasons for your choice.

c) Outline the potential advantages and drawbacks of your selected operating system classification in the context of the robotics project.

Answer: a) The choice of operating system for the robot's control unit in an industrial automation task should be influenced by several key characteristics and requirements of the robotics project. These include:

1. Real-time Requirements: Consider whether the project requires real-time capabilities, where tasks must be executed within strict time constraints. For example, in manufacturing, precise timing is crucial for tasks like robotic arm movements or conveyor belt control.
2. Processing Power: Assess the computational requirements of the robot's control unit. High-performance tasks may require an operating system that can efficiently manage CPU and memory resources.
3. Hardware Compatibility: Determine the compatibility of the operating system with the robot's hardware components, such as sensors, actuators, and communication interfaces.
4. Safety and Reliability: Evaluate the safety-critical nature of the project. Some applications, like medical robotics or autonomous vehicles, demand a high level of reliability and fault tolerance.
5. Ease of Development: Consider the ease of development and availability of development tools for the chosen operating system. This can impact the efficiency and speed of the project's development phase.
6. Security: Assess the security requirements, especially if the robot will interact with sensitive data or critical systems. Security features and vulnerabilities of the operating system must be taken into account.

b) Based on the project's requirements:

- Real-time Operating System (RTOS): An RTOS would be most suitable if the project demands strict real-time capabilities, such as precise control of robotic movements or synchronization with external devices. RTOSs can provide deterministic response times, ensuring that critical tasks are executed without delay.
- General-Purpose Operating System (GPOS): A GPOS like Linux or Windows may be suitable if real-time requirements are not extremely stringent and the project involves complex computation or human-machine interaction. GPOSs are more versatile but may not guarantee precise timing.

- Specialized Embedded Operating System: If the project requires a balance between real-time capabilities and specific optimizations for embedded hardware, a specialized embedded operating system could be a good choice. These are designed for resource-constrained devices and can provide real-time features when needed.

c) Potential advantages and drawbacks of the selected operating system classification:

Real-time Operating System (RTOS):

- Advantages:
  - Deterministic and predictable response times.
  - High reliability for critical tasks.
  - Precise control in time-sensitive applications.
- Drawbacks:
  - Limited flexibility for non-real-time tasks.
  - Potentially higher development complexity.
  - Smaller ecosystem of software and libraries compared to GPOS.

General-Purpose Operating System (GPOS):

- Advantages:
  - Versatility for a wide range of tasks.
  - Rich ecosystem of software and development tools.
  - Easier development for non-real-time components.

- Drawbacks:
  - Less predictable response times.
  - Unsuitable for applications with strict real-time requirements.
  - May require additional efforts for real-time task management.

Specialized Embedded Operating System:

- Advantages:
  - Balances real-time capabilities with resource efficiency.
  - Tailored for embedded hardware.
  - Suitable for a wide range of applications, including real-time tasks.
- Drawbacks:
  - May have a smaller software ecosystem compared to GPOS.
  - Development tools and support might be less extensive than for mainstream GPOS.
  - Customization may be necessary to meet specific project requirements.

18. A software development team is working on a complex project with multiple components. Describe how you would use process states like "New," "Ready," "Running," and "Terminated" to manage the software development lifecycle, from project initiation to final delivery.

Answer: - The states from project initiation to final delivery are-

1. Project Initiation Phase:

   New: In the initial phase, when a project is proposed, all tasks and requirements are considered as "New." This state represents ideas, project proposals, and high-level requirements.
2. Planning Phase:

   Ready: Once the project is approved and planning begins, requirements are refined, and tasks are broken down into more detailed components. These refined requirements and tasks are marked as "Ready" for development.
3. Development Phase:

   Running: As development starts, tasks move into the "Running" state. Developers work on coding, testing, and implementation based on the detailed requirements. This state indicates that active development is in progress.
4. Testing and Quality Assurance Phase:

   Running: Testing and quality assurance tasks also fall under the "Running" state. Testers perform various tests, identify issues, and work with developers to resolve them. This state reflects ongoing efforts to ensure software quality.

5. Deployment and Release Phase:

   Running: During deployment and release preparations, tasks related to final checks, documentation, and deployment scripts are in the "Running" state. This phase ensures that the software is ready for production release.

6. Project Closure Phase:

   Terminated: Once all project objectives are met, and the software is successfully deployed, tasks are marked as "Terminated." This state signifies that the project has reached its conclusion, and all associated tasks are closed.

Q19. You are managing a real-time operating system for an autonomous drone. The drone has to perform various tasks with different deadlines, such as navigation and obstacle avoidance. How would you choose a scheduling algorithm to ensure that critical tasks meet their deadlines while maximizing CPU utilization?

Answer:

In this scenario, where tasks have different deadlines and priorities, a priority-based scheduling algorithm, such as Earliest Deadline First (EDF) or Rate Monotonic Scheduling (RMS), is typically suitable. Here's how you can make the choice and implement it effectively-

1.Task Analysis:

- Priority levels: Assign priorities based on the criticality of the task. Navigation and obstacle avoidance are likely to have higher priorities than less critical tasks.
- Execution times: Estimate the worst-case execution time for each task. This information is essential for scheduling decisions.

2.Priority Assignment:

- Use a priority assignment strategy that reflects the importance and deadlines of tasks.

3.Scheduling Algorithm Selection:

- Earliest Deadline First (EDF): EDF is a dynamic priority scheduling algorithm where the task with the earliest deadline is scheduled next.

4.Optimize CPU and memory resource usage to maximize CPU utilization while still ensuring that critical tasks have the necessary resources for timely execution.

5. Implement safety mechanisms to handle exceptional situations or task failures gracefully. For example, if a critical task misses a deadline, ensure that the drone can take corrective actions to mitigate potential risks.

Q20. In a computer lab at a university, students are using shared workstations for various tasks, including coding, word processing, and browsing. Describe how a fair-share scheduling algorithm could be employed to ensure that all students have a fair opportunity to use the CPU resources, regardless of their specific tasks.

Answer:

- First, classify the users or tasks into different groups based on criteria such as their academic programs, project requirements, or the nature of their tasks. For example, you might have groups for coding, word processing, and web browsing.
- Determine the fair share of CPU resources that each group or user should receive during a given time period.
- Implement a scheduling algorithm that dynamically allocates CPU time to each group or user based on their fair share. These algorithms ensure that CPU time is distributed proportionally to each group's fair share.
- Continuously monitor the CPU usage and adjust the allocation as needed.
- Depending on the importance of tasks or users, you can introduce a priority mechanism.

Q21. Consider a GLA university computer lab 330 where various students and researchers have access toa shared computing environment. Some students want to use this lab for doing programmingassignments and data analysis while Professors want to utilize this lab to train their deep learning models. Describe which type of operating system in this lab can be utilized that effectively allocates resources, such as CPU time, memory, and peripheral devices, to ensure fairness and optimize productivity among multiple users.

Answer: A Time-Sharing Operating System can be used in the lab.

A Time-Sharing Operating System is an operating system that allows multiple users to share a single computer or computing resource simultaneously. It achieves this by rapidly switching between different user tasks, giving each user the illusion of having their own dedicated system even though the computer's resources are being shared.

Here are some key features and characteristics of Time-Sharing Operating Systems:

Multitasking: Time-sharing systems support multitasking, where multiple processes or tasks runconcurrently on the same system. Each task gets a slice of CPU time, and the operating system rapidly switches between them to give the appearance of parallel execution.

Resource Sharing: Users share the computer's resources such as CPU time, memory, and peripheral devices. The operating system must manage these resources efficiently to ensure fair and equitable access among users.

User Interaction: Time-sharing systems are typically interactive, allowing users to interact with the computer in real-time through a command-line interface or graphical user interface (GUI). This is in contrast to batch processing systems where jobs are submitted and run without direct user interaction.

Time Slicing: The operating system allocates CPU time to each task in small time slices, often referred to as time quantum or time slice. This time slicing allows multiple tasks to run concurrently and provides the illusion of simultaneous execution.

Task Scheduling: The operating system employs task scheduling algorithms to determine the order in which tasks receive CPU time. Common scheduling algorithms include round-robin, priority-based scheduling, and more advanced algorithms like Completely Fair Scheduler (CFS).

Security and Isolation: Time-sharing systems need to provide isolation between users and processes to prevent unauthorized access to each other's data and resources. User accounts and permissions are typically used to enforce security.

Fairness: Ensuring fairness among users is a critical aspect. The system should aim to distribute resources equitably so that no user monopolizes the system's resources at the expense of others.

Response Time: Time-sharing systems prioritize low response times to maintain an interactive user experience. This requires efficient scheduling and resource management.

Virtualization: Many modern time-sharing systems support virtualization, allowing the creation of virtual machines (VMs) or containers to further isolate and manage user tasks and workloads.

Logging and Accounting: Time-sharing systems often include logging and accounting mechanisms to track user activity, resource usage, and generate billing or usage reports for accounting purposes.

Time-sharing operating systems are commonly used in scenarios where multiple users or clients need to access a shared computing resource, such as in multi-user servers, cloud computing environments, and shared hosting services. They play a crucial role in maximizing the utilization of computing resources and ensuring efficient sharing among users while maintaining system responsiveness.

Q22. Predict the output?

```
  #include<stdio.h>
  #include<sys/types.h>
  #include <unistd.h>
  void forkexample()
  {
          int x = 1;

          if (fork() == 0)
                  printf("Child has x = %d\n", ++x);
else
  }
  printf("Parent has x = %d\n", --x);
   int main()
  {
          forkexample();
          return 0;
 }
```

Answer:

In this C program, you have a function forkexample() that uses the fork() system callto create a child process. The child and parent processes each have their own copy of the x variable.

Here's what happens step by step:

Initially, the value of x in both the parent and child processes is 1.

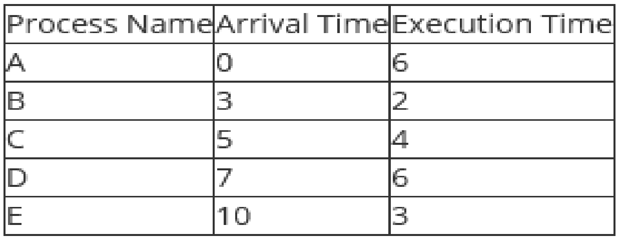
When the fork() call is made, a new child process is created, and the code following the fork()call is executed in both the parent and child processes.

In the child process, fork() returns 0, so it executes the code inside the if block, which incrementsx by 1 and prints "Child has x = 2".

In the parent process, fork() returns the child's process ID, so it executes the code inside the elseblock, which decrements

x by 1 and prints "Parent has x = 0".
So, the expected output of this program will be:
Child has x = 2
Parent has x = 0

Q23. Consider the following set of processes that need to be scheduled on a single CPU. Allthe times are given in milliseconds.



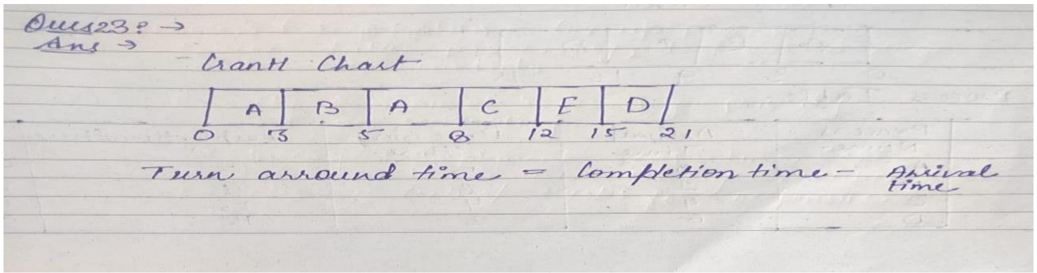
Process Name	Arrival Time	Execution Time
A	0	6
B	3	2
C	5	4
D	7	6
E	10	3

Using the shortest remaining time first scheduling algorithm, the average process turnaroundtime (in msec) is?

Turnaround time (TAT) = Completion time (CT) – Arrival Time (AT)Turnaround time for Process A = 8 ms - 0 ms = 8 ms
Turnaround time for Process B = 5 ms - 3 ms = 2 ms Turnaround time for Process C = 12 ms - 5 ms = 7 ms Turnaround time for Process D = 21 ms - 7 ms = 14 msTurnaround time for Process E = 15 ms - 10 ms = 5 msNow, we can calculate the average turnaround time:
Average Turnaround Time = (8 + 2 + 7 + 14 + 5) / 5 = 36 ms / 5 processes = 7.2 ms
So, the average process turnaround time using the SRTF scheduling algorithm is 7.2milliseconds.

Process	Arrival Time (ms)	Execution Time (ms)	Completion Time (ms)
A	0	6	8
B	3	2	5
C	5	4	12
D	7	6	21
E	10	3	15

Q23 - Q25.



Ques23? →
Ans →
Grantt Chart

A
B
A
C
E
D
0
3
5
8
12
15
21

Turn arround time = Completion time − Arrival time

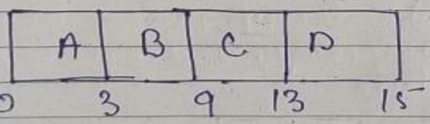
Process Name	Arrival time	Execution Time	Completion Time	Ta Time
A	0	6	8	8
B	3	2	5	2
C	5	4	12	7
D	7	6	21	14
E	10	3	15	5

Average turnaround time = $\frac{(8+2+7+14+5)}{5} = 7.2\text{msec}$

Ques24 :→
Ans

i) Scheduling Algorithm –
First Come First Serve

GANTT Chart



Process Table

Process Name	Arrival Time	Processing Time	Completion time	TA time
A	0	3	3	3
B	1	6	9	8
C	4	4	13	9
D	6	2	15	9

Averag Turn arround time = 29/4 = 7.25

ii) Scheduling Algorithm.-
Non - preemtive Job first

GANTT Chart

A	B	D	C

0 3 9 11 15

Process table

Process Name	Arrival Time	Processing Time	Completion Time	Turn arround Time
A	0	3	3	3
B	1	6	9	8
C	4	4	15	11
D	6	2	11	5

Average turnaround time = 27/4 = 6.75

iii) Scheduling Algorithm
Shortest remaining time first

GANTT CHART

A	B	C	D	B

0 3 4 8 10 15.

Process table

Process Name	Arrival Time	Processing Time	Completion Time	Turn around Time
A	0	3	3	3
B	1	6	15	14
C	4	4	8	4
D	6	2	10	4

Average turnaround time = 25/4 = 6.25

→ So shortest Average TAtime is taken in Shortest remaing time first Algo

Ques25

Ans→

Process	AT	BT
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

AT → Arrival Time
BT → Burst Time
CT → Completion time
TAT → Turn around Time

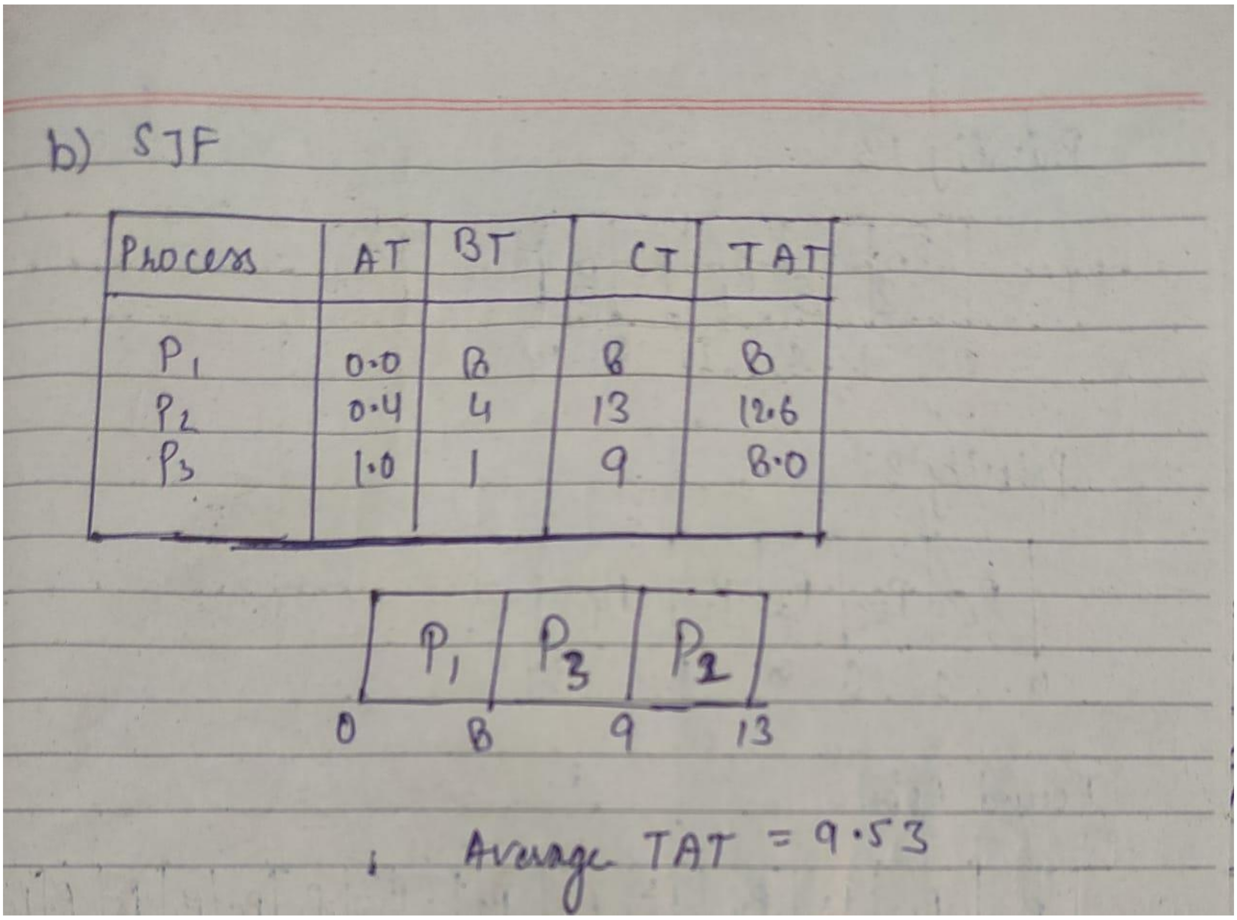
a) FCFS

Process	AT	BT	CT	TAT
P_1	0.0	8	8	8.0
P_2	0.4	4	12	11.6
P_3	1.0	1	13	12.0

P_1	P_2	P_3

0.0 08 12 13

Average TAT = 10.53



b) SJF

Process	AT	BT	CT	TAT
P1	0.0	8	8	8
P2	0.4	4	13	12.6
P3	1.0	1	9	8.0

P1	P3	P2

0 8 9 13

Average TAT = 9.53

Q26. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

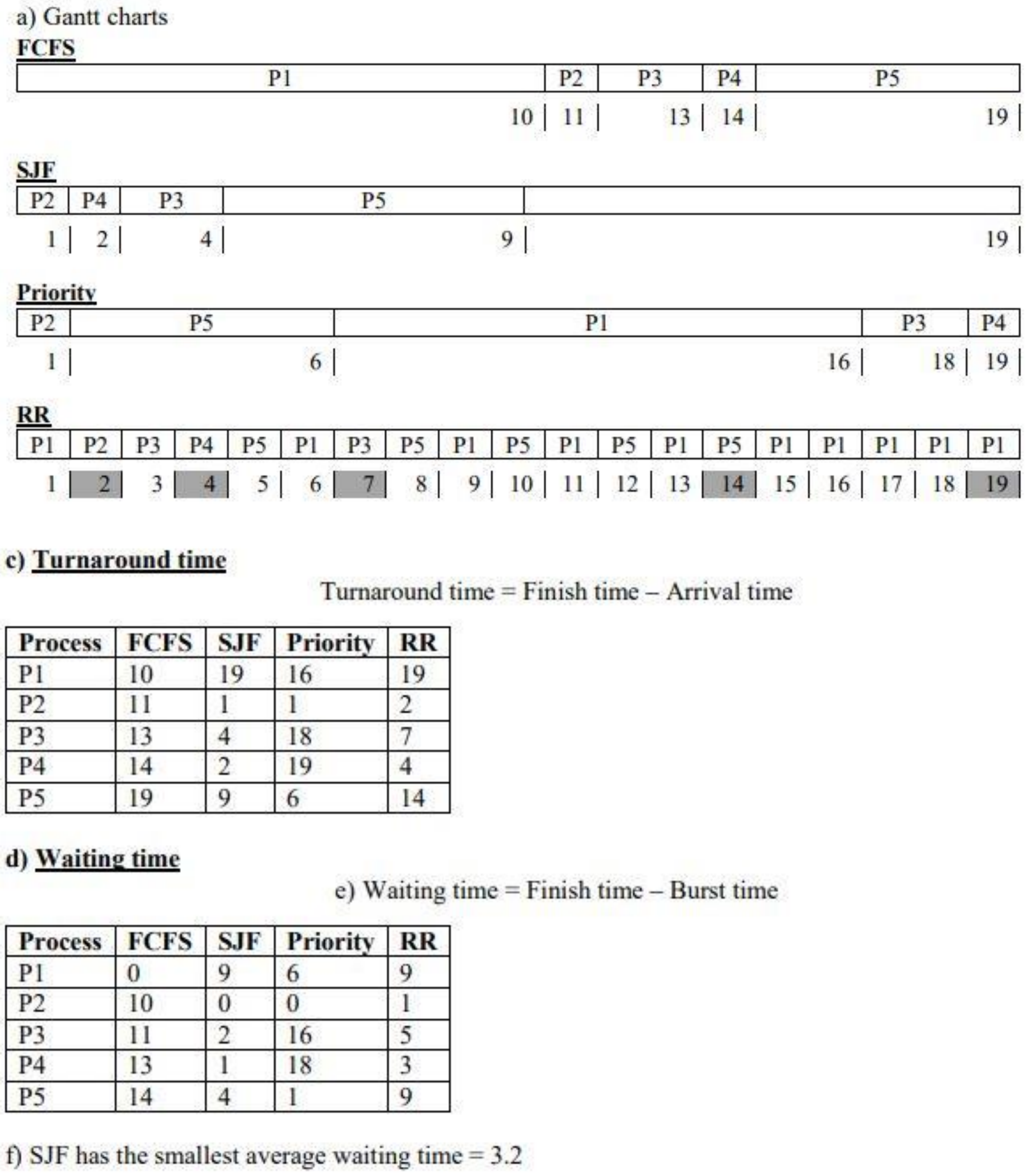
Process	**Burst Time**	**Priority**
P1	**10**	**3**
P2	**1**	**1**
P3	**2**	**3**
P4	**1**	**4**
P5	**5**	**2**

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time.

a. **Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR scheduling.**

b. **What is the turnaround time of each process for each of the scheduling algorithms in part a?**

c. **What is the waiting time of each process for each of the scheduling algorithms in part a?**
Assume an operating system maps user-level threads to the kernel using the many-to- many model where the mapping is done through LWPs. Furthermore, the system allows the developers to create real-time threads. Is it necessary to bound a real-time thread to an LWP? Explain.



a) Gantt charts

FCFS

P1	P2	P3	P4	P5
10	11	13	14	19

SJF

P2	P4	P3	P5	
1	2	4	9	19

Priority

P2	P5	P1	P3	P4
1	6	16	18	19

RR

P1	P2	P3	P4	P5	P1	P3	P5	P1	P5	P1	P5	P1	P5	P1	P1	P1	P1	P1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

c) **Turnaround time**

Turnaround time = Finish time – Arrival time

Process	FCFS	SJF	Priority	RR
P1	10	19	16	19
P2	11	1	1	2
P3	13	4	18	7
P4	14	2	19	4
P5	19	9	6	14

d) **Waiting time**

e) Waiting time = Finish time – Burst time

Process	FCFS	SJF	Priority	RR
P1	0	9	6	9
P2	10	0	0	1
P3	11	2	16	5
P4	13	1	18	3
P5	14	4	1	9

f) SJF has the smallest average waiting time = 3.2

Q27. You are developing an operating system for a personal computer. A user opens a word processing application and starts typing a document. Describe the process creation steps that occur from the user'saction to the actual execution of the word processing program.

Answer:

The process of creating and executing a word processing program within an operating system involves several steps. Below is an overview of these steps, starting from the user's action to the actual execution of the word processing program:

User Interaction:

Input Interpretation:

Process Creation Request:

Loading the Program:

Initialization:

Execution:

User Interaction:

Saving Data:

Resource Management:

Termination:
Cleanup:
Return to the OS:
.

Q28. In a real-time control system for an autonomous robot, there are multiple processes responsible for sensor data processing, decision-making, and motor control. Describe how you would manage and prioritize processes in different states to ensure the robot's responsiveness to changing environmentalconditions.

Answer:

Managing and prioritizing processes in a real-time control system for an autonomous robot is critical to ensure the robot's responsiveness to changing environmental conditions. To achieve this, you can implement a multitasking system where different processes run concurrently, and their priorities are adjusted dynamically based on the robot's state and the urgency of the tasks. Here's a high-level overview of how you can manage and prioritize processes in different states of the robot:

Task Partitioning

Identify the main tasks and processes that the robot needs to perform, such as sensor data processing, decision-making, and motor control. Each of these tasks should be implemented as separate processes or threads.

State Machine:

Implement a state machine that represents the robot's operational states. The robot can transition between states based on its perception of the environment and its internal goals. For example, states might include "Idle," "Exploring," "Obstacle Avoidance," and "Task Execution."

Process Priority Assignment:

Define initial priorities for each process based on their importance and time sensitivity. For example:
Sensor data processing processes may have high priority since timely and accurate sensor data is crucial.
Decision-making processes may have medium priority.
Motor control processes may have lower priority.

Dynamic Priority Adjustment:

Implement a dynamic priority adjustment mechanism that allows the system to adapt to changing conditions. For instance:
In a critical situation, such as detecting an obstacle, the obstacle avoidance process may be given the highest priority to ensure quick response.
If the robot is in a relatively safe state, lower-priority processes can be given more CPU time or resources to perform tasks like exploration or battery management.

Interrupt Handling:

Utilize hardware and software interrupts to handle high-priority events promptly. For example, a collision detection sensor triggering an interrupt should cause an immediate switch to obstacle avoidance behavior.

Resource Management:

Implement resource management techniques to ensure fair allocation of CPU time, memory, and other resources among processes. This prevents one process from monopolizing resources and affecting others.

Feedback Control Loops:

Integrate feedback control loops to continuously monitor and adjust robot actions. For instance, use feedback from motor control to ensure that the robot moves accurately and smoothly.

Fault Tolerance:

Implement fault tolerance mechanisms to handle unexpected failures or errors gracefully. This might include redundancy in critical components and failover mechanisms.

Testing and Simulation:

Thoroughly test the system in various scenarios, including both expected and unexpected conditions. Simulation environments can be used to validate the responsiveness and robustness of the control system.

Logging and Diagnostics:

Implement logging and diagnostic capabilities to record system behavior and facilitate debugging in case of issues.
By managing and prioritizing processes in this way, the real-time control system can adapt to different situations and ensure that the robot remains responsive to changing environmental conditions while maintaining overall system stability and reliability.

Q29. When a user on a personal computer opens multiple applications, the operating system manages these processes and their states to provide a smooth user experience, especially when switching between applications. The operating system utilizes the concepts of "Running," "Ready," and "Blocked" states to efficiently allocate CPU time and system resources. Here's how it handles these states:

Answer:

Running State:

When an application is in the "Running" state, it means that it is actively executing instructions on the CPU. The operating system uses a scheduler to allocate CPU time to running processes.
The scheduler assigns a time slice or quantum to each running process, allowing it to execute for a short period.

The running process may voluntarily yield the CPU (e.g., when it waits for user input) or involuntarily relinquish it due to time-slicing, allowing other processes to run.

Ready State:

Processes in the "Ready" state are those that are eligible to run but are waiting for their turn on the CPU. These processes are typically stored in a queue or data structure known as the "ready queue."

When a process in the "Running" state yields the CPU, completes its task, or is preempted by a higher-priority process, the operating system selects a process from the "Ready" state to run next.

The transition from "Ready" to "Running" state is managed by the scheduler based on priority, time-sharing algorithms, and other factors.

Blocked State:

Processes in the "Blocked" state are not ready to execute because they are waiting for some event or resource, such as user input, data from a slow I/O device, or completion of a file operation.

When a process is in the "Blocked" state, it is typically removed from the ready queue to make room for other processes.

Once the blocking condition is satisfied (e.g., user input is received), the process transitions back to the "Ready" state, becoming eligible to run again.

Now, let's see how these process states are managed to provide a smooth user experience, especially when switching between applications:

Q30. Assume that, you work for a space exploration company, and your team is responsible for programming the software that controls a rover on a distant planet. Your colleague, who is new to thisproject, asks, "What exactly is a real-time operating system, and why are we using it for the rover's software?" How would you explain the concept of a real-time operating system and its importance inthe context of controlling the rover's actions and data processing on a distant planet?

Answer:

"A real-time operating system, or RTOS, is a crucial component of the software that controls our rover on the distant planet. Unlike typical operating systems found on personal computers, an RTOS is specifically designed to handle tasks and processes with strict timing requirements. It's like the conductor of an orchestra, ensuring that each instrument plays its notes at the right time to create a harmonious piece of music. Similarly, an RTOS ensures that the rover's actions and data processing occur precisely when they are supposed to, which is vital for the success of our mission.

Timing Precision: In space exploration, timing is everything. The rover must perform tasks like collecting samples, avoiding obstacles, and transmitting data back to Earth with extreme precision. An RTOS ensures that these tasks occur at the exact moment they are scheduled, which is critical for mission success.

Predictability: RTOSes provide a high level of predictability in task execution. We can precisely determine when a particular task will start and finish, allowing us to plan and coordinate rover activities effectively.

Concurrency: Our rover needs to perform multiple tasks simultaneously, such as moving its wheels, capturing images, and analyzing data. An RTOS manages these concurrent tasks, ensuring that they do not interfere with each other and that critical tasks get priority when needed.

Fault Tolerance: In the harsh and unpredictable environment of space, things can go wrong. An RTOS can incorporate fault tolerance mechanisms, allowing the rover to recover from unexpected issues and continue its mission.

Resource Management: With limited resources on the rover, like memory and processing power, an RTOS efficiently manages these resources to maximize performance and minimize energy consumption.

Real-time Feedback: The RTOS can provide real-time feedback on the rover's state, allowing us to monitor its health, diagnose problems, and make rapid adjustments if necessary.

Q31. The following C program is executed on a Unix/Linux system:

```
#include < unistd.h > int main
  ()
  {
      int i ;
      for (i=0; i<10; i++)
          if (i%2 == 0) fork ( ) ; return 0 ;
  }
```

The total number of child processes created is _______and justify your output.

Answer:

The total number of child processes created is 5 and justify your output.

Justification:

Child processes are created when i is even (i.e., i % 2 == 0).

The loop runs from i=0 to i<10, and for each even i, a child process is created. The odd values of i do not result in child process creation.

Therefore, the total number of child processes created is 5.

Q32. In a multiprogramming and time-sharing environment, several users share the system simultaneously. This situation can result in various security problems. What are two such problems?

Answer:

There are two security problems arise multiprogramming and time-sharing environment:

Unauthorized Access: Multiple users are accessing the same system concurrently, and each user may have different levels of privilege and access rights. Unauthorized users may attempt to gain access to files, resources, or data that they should not have access to. Users can potentially exploit vulnerabilities or weaknesses in the system to bypass access controls and gain unauthorized access to sensitive information or perform unauthorized actions.

Data Leakage and Privacy Concerns: When multiple users shared same resource, memory spaces, or insufficient isolation between user processes. Data leakage can result in privacy breaches, where one user gains access to another user's data, potentially compromising confidentiality and privacy. To mitigate these security problems, system administrators and developers need to implement robust authentication and authorization mechanisms, properly isolate user processes, enforce access controls, regularly update and patch the system to address vulnerabilities, and monitor system activity for any suspicious behavior.

Q33. Assume arrival order is: P1, P2, P3, P4, P5 at time 0, 1, 2, 3, 4 respectively and a smaller priority number implies a higher priority. Draw the Gantt charts for pre-emptive and non-pre-emptive priority scheduling. Calculate Average Turnaround Time and Average Waiting Time.

Answer:

Process	Burst Time (Assume)	Arrival Time	Priority (Assume)
P1	1	0	1
P2	2	1	2
P3	3	2	3
P4	4	3	4
P5	5	4	5

Non-Preemptive Priority Scheduling:

P1	P2	P3	P4	P5

0 2 3 6 10 15

Process	Burst Time (Assume)	Arrival Time	CT	TAT	WT
P1	1	0	2	2	1
P2	2	1	3	2	0
P3	3	2	6	4	1
P4	4	3	10	7	3
P5	5	4	15	11	6

Average Turnaround Time = (2 + 2 + 4 + 7 + 11) /5 = (26/5) = 5.2 units

Average Waiting Time = (1 + 0 + 1 + 3 + 6) /5 = (11/ 5) = 2.2 units

Preemptive Priority Scheduling:

P1	P2	P3	P4	P5

0 2 3 6 10 15

Process	Burst Time (Assume)	Arrival Time	CT	TAT	WT

P1	1	0	2	2	1
P2	2	1	3	2	0
P3	3	2	6	4	1
P4	4	3	10	7	3
P5	5	4	15	11	6

Average Turnaround Time = (2 + 2 + 4 + 7 + 11) /5 = (26/5) = 5.2 units
Average Waiting Time = (1 + 0 + 1 + 3 + 6) /5 = (11/ 5) = 2.2 units

Q34. Consider a set of 5 processes whose arrival time, CPU time needed and the priority are given below:

Process number	Arrival time(ms)	CPU time	Priority
p1	0	10	5
p2	0	5	2
p3	2	3	1
p4	5	20	4
p5	10	2	3

If the CPU scheduling policy is SJF, the average waiting time (with preemption) will be?

Answer:

Average Waiting Time = 6

P1	P2	P3	P3	P3	P2	P1	P5	P1	P4

0 1 2 3 4 5 9 10 12 20 40

Process	Complete Time	TAT	Waiting Time
P1	20	20	10
P2	9	9	4
P3	5	2	1
P4	40	35	15
P5	12	2	0
		68	30

Average Waiting Time = 30/5 = 6

Q35. Find the waiting time of P4 process using priority scheduling algorithm (Assume lowest number has highest priority).

Process	**P1**	**P2**	**P3**	**P4**	**P5**
Burst time	**5**	**13**	**8**	**6**	**12**
Priority	**1**	**3**	**0**	**4**	**2**

Answer:

Waiting time of P4 process is **38**

P3	P1	P5	P2	P4

0 8 13 25 38 44

Process	Complete Time	TAT	Waiting Time
P1	13	13	8
P2	38	38	25
P3	8	8	0
P4	44	44	**38**
P5	25	25	13

Q36. The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function: Priority = (recent CPU usage / 2) + base where base = 60 and recent CPU usage refers to a value indicating how often a process has used the CPU since priorities were last recalculated. Assume that recent CPU usage for process P1 is 40, for process P2 is 18, and for process P3 is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

Answer:

To calculate the new priorities for processes P1, P2, and P3 using the traditional UNIX scheduler formula, you can use the given formula:

Priority = (recent CPU usage / 2) + base

Where:

- base = 60 (as given)
- recent CPU usage is different for each process.

Let's calculate the new priorities for each process:

1. Process P1 with recent CPU usage = 40 Priority = (40 / 2) + 60 Priority = 20 + 60 Priority = 80
2. Process P2 with recent CPU usage = 18 Priority = (18 / 2) + 60 Priority = 9 + 60 Priority = 69
3. Process P3 with recent CPU usage = 10 Priority = (10 / 2) + 60 Priority = 5 + 60 Priority = 65

So, after priorities are recalculated, the new priorities for the three processes are as follows:

- Process P1: Priority 80
- Process P2: Priority 69
- Process P3: Priority 65

The traditional UNIX scheduler raises the priority of a CPU-bound process when its recent CPU usage is higher, as seen with Process P1 having the highest recent CPU usage and the highest priority among the three processes. Conversely, a process with lower recent CPU usage will have a lower priority, as seen with Process P3 having the lowest recent CPU usage and the lowest priority. This behavior is in line with the concept that CPU-bound processes are given higher priority to ensure they get CPU time.

Q37. You are designing a task scheduler for an operating system that supports both foreground and background tasks. Explain how the "Ready" and "Running" states are managed in the context of a multi-core CPU, and how the scheduler decides which process to move from "Ready" to "Running."

Answer:

In an operating system that supports both foreground and background tasks on a multi-core CPU, the management of the "Ready" and "Running" states is a crucial aspect of the task scheduler's functionality. Here's an explanation of how these states are managed and how the scheduler decides which process to move from "Ready" to "Running":

1. **Ready Queue**:
   - Processes that are ready to execute but are waiting for CPU time are placed in a data structure called the "Ready Queue." The Ready Queue holds a list of all processes that are in the "Ready" state.
2. **Running State**:
   - The "Running" state represents the state of the process that is currently executing on a CPU core. In a multi-core CPU environment, there are multiple cores, and each core can execute a separate process concurrently.
3. **Scheduler's Role**:
   - The task scheduler's primary responsibility is to decide which process from the "Ready Queue" should be moved to the "Running" state on each CPU core. The scheduler makes this decision based on various scheduling algorithms and policies, including:

   a. **Priority Scheduling**: Assigning priorities to processes and selecting the one with the highest priority to run next.

b. **Round Robin Scheduling**: Allowing each process in the "Ready Queue" to run for a fixed time quantum before moving to the back of the queue.
c. **Shortest Job First (SJF)**: Selecting the process with the shortest expected execution time.
d. **Multi-Level Feedback Queue**: Using multiple priority queues and adjusting priorities based on process behavior over time.
e. **CPU Burst Prediction**: Predicting the next CPU burst for each process and scheduling the one with the shortest predicted burst.

4. **Scheduling Decision**:
   - When it's time to make a scheduling decision (for example, when the current running process completes its time slice or voluntarily yields the CPU), the scheduler evaluates the Ready Queue and selects the next process to run based on the chosen scheduling algorithm.
5. **Assigning to Cores**:
   - In a multi-core CPU environment, the scheduler must also decide which core to assign the selected process to. This can be done using load balancing techniques to evenly distribute the workload across all available cores.
6. **Moving to Running State**:
   - Once a process is selected, it is moved from the "Ready Queue" to the "Running" state on the assigned core, and its execution begins.
7. **Context Switching**:
   - When a process is moved from the "Running" state to the "Ready" state (e.g., due to completion or preemption), a context switch is performed to save the current process's state and load the state of the selected process.

The specific scheduling algorithm and policies used will determine how processes are prioritized and selected for execution. The goal is to efficiently utilize the CPU resources, provide fair access to CPU time for all processes, and potentially optimize for specific criteria such as responsiveness, throughput, or fairness, depending on the system's requirements.

Q38. In a multi-user operating system, several users are running CPU-bound tasks. One user has a high-priority task, while the others have lower-priority tasks. Explain how a priority-based scheduling algorithm would handle this situation and its potential advantages and drawbacks

Answer:

In a multi-user operating system with multiple CPU-bound tasks running concurrently, a priority-based scheduling algorithm can be used to determine which task gets access to the CPU and in what order. Here's how such an algorithm would handle a situation where one user has a high-priority task while others have lower-priority tasks, along with its potential advantages and drawbacks:

Handling the Situation:

1. **Priority Assignment:** In a priority-based scheduling algorithm, each process is assigned a priority value. Higher priority values indicate higher priority tasks, and lower values indicate lower priority tasks. In this situation, the user with the high-priority task would have their task assigned a higher priority compared to the others.
2. **Scheduling Decision:** When it's time to make a scheduling decision (e.g., when the currently executing process completes its time slice or voluntarily yields the CPU), the scheduler will select the process with the highest priority from the pool of ready processes.
3. **Advantages:**
   - **Priority Differentiation:** The primary advantage of a priority-based scheduling algorithm is its ability to differentiate between tasks based on their importance or urgency. This ensures that high-priority tasks are executed promptly, which can be critical for certain applications and users.
   - **Responsiveness:** High-priority tasks, such as real-time or interactive processes, can respond quickly to events and user input without significant delays.
   - **Resource Allocation:** It allows system administrators to allocate CPU time more effectively to meet specific requirements, such as guaranteeing that certain users or applications get the necessary resources.
4. **Drawbacks:**
   - **Starvation:** A major drawback of priority-based scheduling is the potential for lower-priority tasks to suffer from starvation. If high-priority tasks constantly arrive, lower-priority tasks may rarely get CPU time, leading to unfairness and potential resource contention issues.
   - **Priority Inversion:** In some cases, lower-priority tasks might indirectly affect higher-priority tasks due to priority inversion. This occurs when a high-priority task is waiting for a resource held by a lower-priority task, causing delays in execution.
   - **Potential for Misuse:** If not managed carefully, users may exploit priority assignment to gain an unfair advantage or disrupt the system by assigning artificially high priorities to their tasks.
   - **Complexity:** Priority-based scheduling can be more complex to configure and manage than simpler scheduling algorithms like round-robin or first-come-first-served. Determining appropriate priority values for tasks can be challenging.

To mitigate some of the drawbacks associated with priority-based scheduling, operating systems often use techniques like priority aging (to prevent starvation), priority inheritance (to address priority inversion), and limits on how much a process can increase its priority over time.

Q39. In a computer lab, there are multiple student workstations running various applications. A student is complaining that their computer is running slowly because of a CPU-intensive application running on another workstation. How could a CPU scheduler address this issue and ensure fair CPU allocation among all users?
Answer:
To address the issue of a CPU-intensive application on one workstation affecting the performance of other workstations in a computer lab and ensure fair CPU allocation among all users, the CPU scheduler can employ various scheduling techniques and strategies. Here are some ways a CPU scheduler can help mitigate this problem:

1. **Priority-Based Scheduling**:
   - Assign priorities to different user processes or workstations. Lower-priority processes will yield the CPU to higher-priority processes when needed. This allows more critical or interactive tasks to get preferential treatment.
2. **Fair Share Scheduling**:
   - Implement a fair share scheduler that allocates CPU time based on each user's fair share. Each user or workstation is allocated a portion of the CPU time, ensuring equitable access to computing resources.
3. **CPU Quotas**:
   - Set CPU quotas for each workstation or user. This approach guarantees that no single workstation or user can consume all available CPU resources. If a workstation exceeds its quota, its processes are throttled.
4. **Round-Robin Scheduling**:
   - Employ a round-robin scheduling algorithm that assigns CPU time slices to different processes or workstations in a cyclic manner. This helps prevent any single process from monopolizing the CPU for extended periods.
5. **CPU Affinity**:
   - Use CPU affinity to restrict certain CPU-intensive tasks to specific processor cores. This can help isolate the impact of one workstation's CPU-intensive application on others.
6. **Load Balancing**:
   - Implement load balancing across multiple workstations or CPUs. Load balancing redistributes tasks to ensure even distribution of CPU load, preventing any single workstation from becoming overloaded.
7. **Process Prioritization**:
   - Allow users or administrators to set the priority of their processes. Users can prioritize their important tasks and reduce the priority of background or non-essential processes.
8. **Resource Monitoring**:
   - Continuously monitor system resource utilization and performance. If a workstation's CPU usage exceeds a certain threshold, the scheduler can take corrective action, such as limiting that workstation's CPU allocation.
9. **Dynamic Adjustment**:
   - Use dynamic adjustment of priorities or quotas based on system load. When the system is under heavy load, the scheduler can allocate more CPU resources to interactive tasks to maintain system responsiveness.
10. **Feedback Control**:
    - Employ feedback control mechanisms that learn from past behavior. If a workstation consistently consumes excessive CPU resources, the scheduler can adapt and allocate fewer resources to it.

It's important to note that the choice of scheduling strategy will depend on the specific requirements of the computer lab, the types of applications being run, and the expected usage patterns. Fairness and performance goals should be balanced to ensure that users have a satisfactory experience while preventing resource abuse by any single workstation or user.

Q40. Consider you're in a tech support role, and a colleague from another department asks you to clarify what the term "kernel" means in the context of an operating system. They also want to know about the different types of kernels that exist. How would you explain the concept of a kernel, and what examples?
Answer:
I would explain the concept of a kernel in the context of an operating system and provide information about the different types of kernels that exist.
Explanation of a Kernel:
In the context of an operating system, the "kernel" is a crucial component that serves as the core or central part of the operating system itself. It acts as an intermediary between software applications and the hardware of a computer. The kernel is responsible for managing and controlling various hardware resources, such as the CPU (Central Processing Unit), memory, input/output devices (like disks and network interfaces), and more. It plays a pivotal role in maintaining system stability, security, and overall functionality.
Types of Kernels:
There are primarily three types of kernels, each with its own characteristics:

1. **Monolithic Kernel:**
    - In a monolithic kernel architecture, all operating system services and functionalities, including device drivers and system call implementations, are part of a single, large, and tightly integrated kernel space.
    - This type of kernel provides fast communication between kernel components since they are all part of a single address space.
    - Examples of operating systems that use monolithic kernels include Linux and earlier versions of Unix.
2. **Microkernel:**
    - A microkernel architecture takes a minimalist approach by keeping the core kernel small and moving most of the operating system services, such as device drivers and file systems, into user-space processes.
    - The microkernel's primary function is to provide essential services like inter-process communication (IPC) and process management.
    - This design promotes modularity and flexibility but can have some performance overhead due to the need for inter-process communication.
    - Examples of microkernel-based operating systems include QNX and Minix.
3. **Hybrid Kernel:**
    - Hybrid kernels aim to combine the best of both monolithic and microkernel designs. They keep some critical operating system components in kernel space for performance reasons while moving less critical components to user space for better modularity and stability.
    - This approach tries to strike a balance between performance and modularity.
    - Examples of operating systems using hybrid kernels include Microsoft Windows (starting with Windows NT) and macOS.

Summary:
In summary, a kernel in the context of an operating system is the core component responsible for managing hardware resources and providing a foundation for running software applications. Different types of kernels, such as monolithic, microkernel, and hybrid kernels, offer varying trade-offs between performance, modularity, and complexity. The choice of kernel type often depends on the specific design goals and requirements of the operating system.

Q41. There are two different ways that commands can be processed by a command interpreter. One way is to allow the command interpreter to contain the code needed to execute the command. The other way is to implement the commands through system programs. Compare and contrast the two approaches.

Answer:
First way is fast but changing the code is difficult.
Speed and Simplicity in first case.
Second way is slower but you can add command but maintain a relatively simple interface.

Q42. The issue of resource utilization shows up in different forms in different types of operating systems. List what resources must be managed carefully in the following settings:

a. Mainframe or minicomputer systems
b. Workstations connected to servers

Answer:
Mainframes: memory and CPU resources, storage, network bandwidth. b. Workstations: memory and CPU resources

Q43. Predict the output?

```
#include <stdio.h>
 #include <unistd.h>
 int main()
{
fork();
fork() && fork() || fork(); fork();

printf("forked\n"); return 0;
}
```

Solution:
fork() system call spawn processes as leaves of growing binary tree. If we call fork() twice, it will spawn 22 = 4 processes. All these 4 processes forms the leaf children of binary tree. In general if we are level **l**, and fork() called unconditionally, we will have **2l** processes at level (**l+1**). It is equivalent to number of maximum child nodes in a binary tree at level (**l+1**).

As another example, assume that we have invoked fork() call 3 times unconditionally. We can represent the spawned process using a full binary tree with 3 levels. At level 3, we will have 23 = 8 child nodes, which corresponds to number of processes running.

Q44. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of does the CPU remain idle?

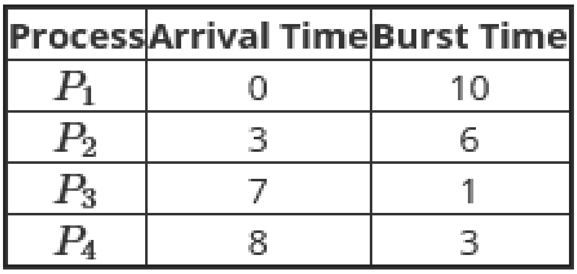
Solution:

Let three processes be p0, p1 and p2.
Their execution time is 10, 20 and 30 respectively.
p0 spends first 2 time units in I/O, 7 units of CPU time and finally 1 unit in I/O.
p1 spends first 4 units in I/O, 14 units of CPU time and finally 2 units in I/O. p2 spends first 6 units in I/O, 21 units of CPU time and finally 3 units in I/O.

Idle	**P0**	**P1**	**P2**	**Idle**
0 2	9	23	44	47

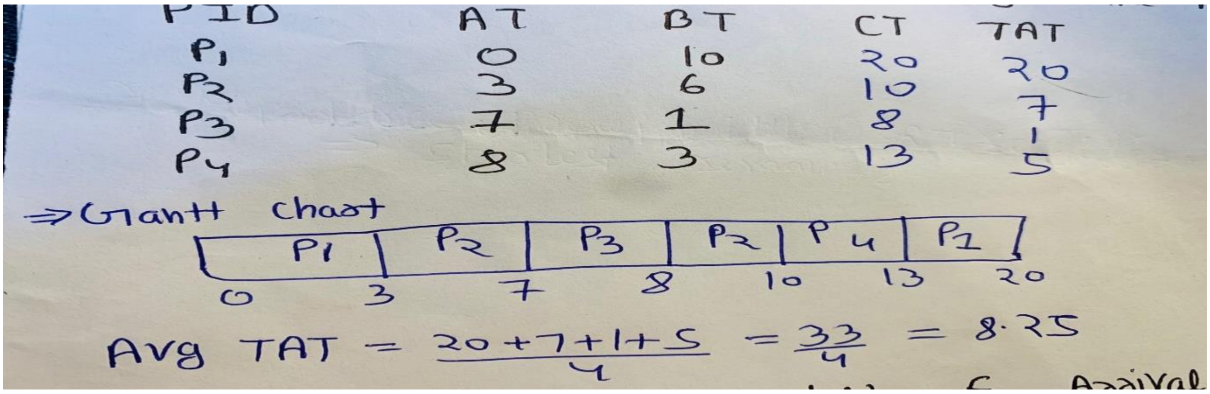
Total time spent = 47
Idle time = 2 + 3 = 5
Percentage of idle time = (5/47)*100 = 10.6

Q45.Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is preemptive shortest remaining-time first.



Process	Arrival Time	Burst Time
P_1	0	10
P_2	3	6
P_3	7	1
P_4	8	3

The average turnaround time of these processes is milliseconds ____



PID	AT	BT	CT	TAT
P1	0	10	20	20
P2	3	6	10	7
P3	7	1	8	1
P4	8	3	13	5

⇒ Gantt Chart

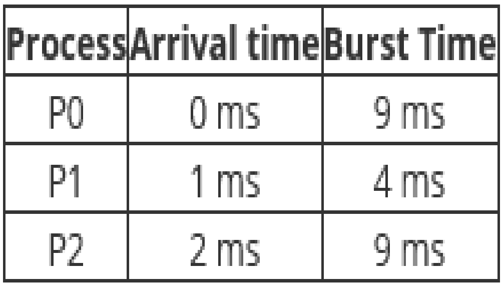
P1	P2	P3	P2	P4	P1

0 3 7 8 10 13 20

$$\text{Avg TAT} = \frac{20+7+1+5}{4} = \frac{33}{4} = 8{\cdot}25$$

Arrival

Q46. Consider the following table of arrival time and burst time for three processes P0, P1 and P2.



Process	Arrival time	Burst Time
P0	0 ms	9 ms
P1	1 ms	4 ms
P2	2 ms	9 ms

The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?

Answer:

Process P0:

- Starts at time 0 and has a burst time of 9.
- Executes for 3 time units (t=0 to t=3).
- There is a context switch.

Process P1:

- Starts at time 3 and has a burst time of 4.
- Executes for 4 time units (t=3 to t=7).
- There is a context switch.

Process P0:

- Resumes execution at time 7.
- Executes for 2 time units (t=7 to t=9).

Process P1:

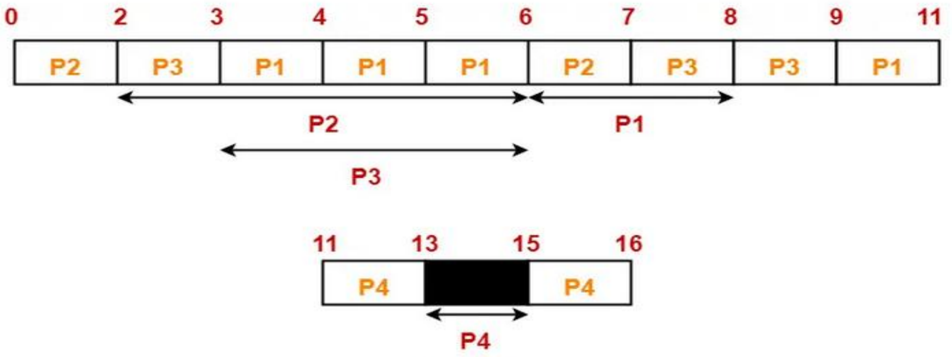
- Resumes execution at time 9.
- Executes for 1 time unit (t=9 to t=10).

Process P2:

- Starts at time 10 and has a burst time of 9.
- Executes for 9 time units (t=10 to t=19).

Now, calculate the waiting time for each process:

Waiting Time P0 = (Start Time P0 - Arrival Time P0) + (Start Time P0 - Finish Time P1) = (0 - 0) + (3 - 7) = 0 + (-4) = -4 Waiting Time P1 = (Start Time P1 - Arrival Time P1) = (3 - 1) = 2 Waiting Time P2 = (Start Time P2 - Arrival Time P2) + (Start Time P2 - Finish Time P0) = (10 - 2) + (10 - 9) = 8 + 1 = 9



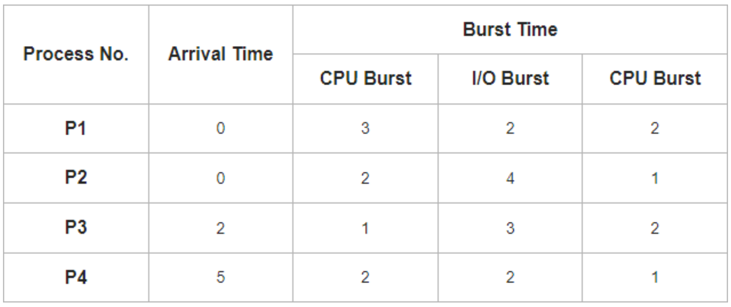
Gantt Chart

Now, calculate the average waiting time:

Average Waiting Time = (Waiting Time P0 + Waiting Time P1 + Waiting Time P2) / 3 Average Waiting Time = (-4 + 2 + 9) / 3 Average Waiting Time = 7 / 3 = 2.33 (rounded to two decimal places)

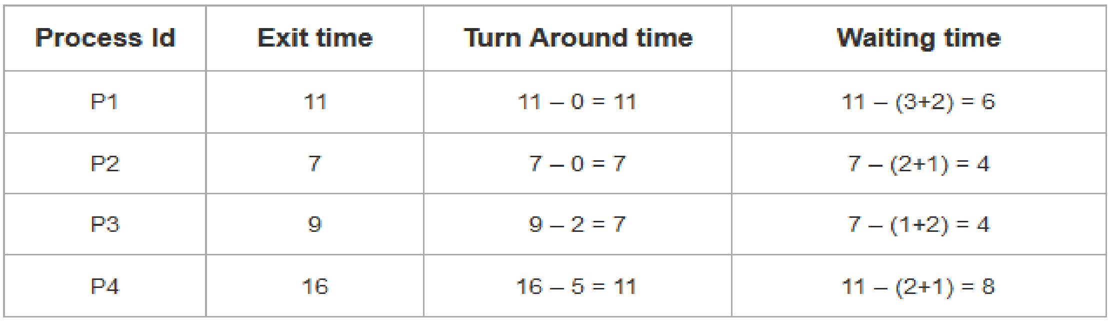
So, the average waiting time for the three processes is approximately 2.33 time units.

Q47. Consider the set of 4 processes whose arrival time and burst time are given below-

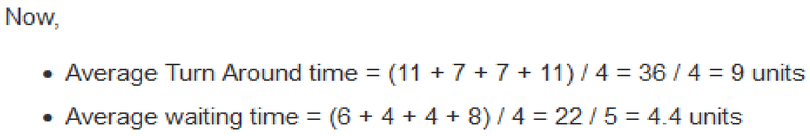


Process No.	Arrival Time	Burst Time		
		CPU Burst	I/O Burst	CPU Burst
P1	0	3	2	2
P2	0	2	4	1
P3	2	1	3	2
P4	5	2	2	1

If the CPU scheduling policy is Shortest Remaining Time First, calculate the average waiting time and average turnaround time.



Process Id	Exit time	Turn Around time	Waiting time
P1	11	11 – 0 = 11	11 – (3+2) = 6
P2	7	7 – 0 = 7	7 – (2+1) = 4
P3	9	9 – 2 = 7	7 – (1+2) = 4
P4	16	16 – 5 = 11	11 – (2+1) = 8



Now,

- Average Turn Around time = (11 + 7 + 7 + 11) / 4 = 36 / 4 = 9 units
- Average waiting time = (6 + 4 + 4 + 8) / 4 = 22 / 5 = 4.4 units

Q48. While using your computer, you notice that one application has become unresponsive. Describe how you would use the operating system's task manager or equivalent feature to identify and terminate the unresponsive process.

Answer:

If you encounter an unresponsive application on your computer, you can use the operating system's task manager or an equivalent feature to identify and terminate the unresponsive process. Here are the general steps for Windows, macOS, and Linux:

For Windows:

1. **Ctrl+Shift+Esc**: Press `Ctrl + Shift + Esc` on your keyboard. This keyboard shortcut opens the Windows Task Manager directly.

   Alternatively, you can also press `Ctrl + Alt + Delete` and then select "Task Manager" from the options menu.

2. **Identify the unresponsive application**: In the Task Manager, you'll see a list of running processes and applications. Look for the application that is marked as "Not Responding" in the "Status" column.
3. **End the unresponsive process**: Select the unresponsive application by clicking on it once to highlight it. Then, click the "End Task" button at the bottom right of the Task Manager window.
4. **Confirmation**: You might get a warning that ending the task could cause data loss. If you're sure, click "End Process."

Q49. How does the FCFS scheduling algorithm work in the context of allocating computers to students in the lab? What are the potential advantages and drawbacks of using FCFS for this purpose?

The First-Come-First-Serve (FCFS) scheduling algorithm is a simple and intuitive method used to allocate resources, including computers in a lab, based on the order in which requests are received. In the context of allocating computers to students in a lab, FCFS works as follows:

1. **Order of Allocation**: The FCFS algorithm allocates computers to students in the order in which they arrive at the lab or submit their requests for computer use. The first student to arrive or make a request is served first, followed by the second student, and so on.
2. **Sequential Allocation**: Computers are allocated sequentially. The first available computer is assigned to the first student, the second available computer is assigned to the second student, and this continues until all computers are allocated or all students have been served.

Advantages of Using FCFS for Allocating Computers in a Lab:

1. **Simplicity**: FCFS is straightforward and easy to implement. It requires minimal computational overhead and does not involve complex decision-making processes.
2. **Fairness**: FCFS is inherently fair as it follows the principle of "first come, first served." It ensures that all students have an equal chance to access lab computers, and there is no bias in allocation.
3. **Predictability**: Users can predict when they will get access to a computer based on their position in the queue. This predictability can be helpful for planning and time management.

Drawbacks of Using FCFS for Allocating Computers in a Lab:

1. **Inefficiency**: FCFS may not be the most efficient way to allocate resources. It doesn't consider the varying needs or usage patterns of different students. For example, a student who only needs a computer for a short task may occupy a computer for an extended period.
2. **Resource Wastage**: Under FCFS, a computer allocated to a student might remain in use even when the student is idle. This can lead to resource wastage, especially in a lab with limited computers.
3. **Lack of Prioritization**: FCFS does not take into account any priority levels or urgency of tasks. This can be problematic when some tasks are more time-sensitive or critical than others.
4. **No Adaptability**: FCFS does not adapt to changing circumstances. It continues to allocate resources based solely on the order of requests, even if some students have more urgent needs.
5. **Potential for Bottlenecks**: If there are long-running tasks or students occupying computers for extended periods, it can lead to bottlenecks, causing delays for other students.

In practice, FCFS can be suitable for scenarios where all users have relatively similar needs, and it's important to maintain fairness in resource allocation. However, for situations with varying priorities, time-sensitive tasks, and a desire to maximize resource utilization, more sophisticated scheduling algorithms, such as priority scheduling or round-robin, may be more appropriate. These algorithms allow for better resource management and prioritization while maintaining fairness to some extent.

Q50. Imagine you're leading a team meeting at a software development company. You want to discuss thevarious types of operating systems and their pros and cons to help your team make informed decisionsabout which OS to use for their upcoming project. How would you guide the discussion, explaining the different types of operating systems while highlighting their advantages and disadvantages?

Answer:

Leading a team meeting to discuss the selection of an operating system for an upcoming software development project is an important decision. To guide this discussion effectively, you can structure the conversation by explaining the different types of operating systems and highlighting their respective advantages and disadvantages. Here's a suggested approach:

Introduction:

Set the Agenda: Start the meeting by outlining the agenda and the importance of selecting the right operating system for the project.

1. **Objectives**: Clearly state the project's requirements, constraints, and goals to provide context for the discussion.

Types of Operating Systems:

3. **Desktop Operating Systems**:
    - Explain that desktop OSes are designed for individual users and workstations.
    - Mention common examples such as Windows, macOS, and Linux distributions designed for desktop use.
    - Highlight their advantages and disadvantages.
4. **Server Operating Systems**:
    - Describe server OSes used for running server applications, websites, databases, and services.
    - Examples include Windows Server, various Linux distributions (e.g., Ubuntu Server), and others.
    - Discuss the pros and cons, especially emphasizing stability, security, and performance.
5. **Mobile Operating Systems**:
    - Introduce mobile OSes like Android and iOS, used for mobile devices.
    - Explain the relevance if mobile development is part of the project.
6. **Embedded Operating Systems**:
    - Mention embedded OSes like Android Things, Free RTOS, and others for devices like IoT appliances.
    - Discuss their suitability for specialized applications.

Advantages and Disadvantages:

7. **Desktop Operating Systems**:
    - Advantages: Familiarity, broad application compatibility, user-friendly interfaces.
    - Disadvantages: Overhead, licensing costs, potential security vulnerabilities.
8. **Server Operating Systems**:
    - Advantages: Stability, performance, security features, scalability.
    - Disadvantages: Cost, resource requirements, complexity.
9. **Mobile Operating Systems**:
    - Advantages: User base, app store support, well-defined platforms.
    - Disadvantages: Vendor lock-in, limited hardware choices.
10. **Embedded Operating Systems**:
    - Advantages: Lightweight, tailored for specific tasks, power efficiency.
    - Disadvantages: Limited functionality, specialized knowledge required.

Project Requirements:

11. **Review Project Requirements**: Discuss the specific needs of the upcoming project, such as hardware compatibility, real-time performance, scalability, and security.

Decision-Making:

12. **Engage the Team**: Encourage team members to share their insights and preferences based on the project requirements.
13. **Weighted Decision Matrix**: Consider creating a weighted decision matrix that factors in project requirements and ranks different OS options based on these criteria.
14. **Discussion and Consensus**: Open the floor for discussion, questions, and concerns. Encourage team members to voice their opinions and concerns.
15. **Summary and Decision**: Summarize the key points discussed and aim to arrive at a consensus on the choice of the operating system for the project.
16. **Action Items**: Outline the next steps, assign responsibilities, and set a timeline for the implementation of the chosen OS.
17. **Follow-Up**: Schedule a follow-up meeting to ensure that the chosen OS aligns with the project's evolving needs and to address any issues or challenges that arise during the implementation phase.

By structuring the discussion in this way, you can help your software development team make an informed decision about the operating system for the upcoming project while considering all relevant factors.