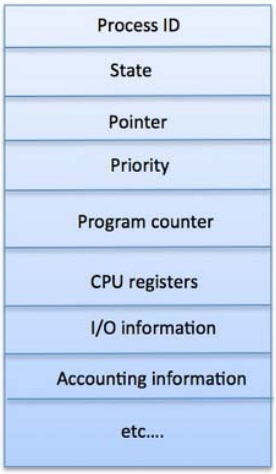
Operating System
Unit-II

PROCESS MANAGEMENT	PROCESS MANAGEMENT (प्रक्रिया प्रबंधन)
Difference between a Process and Program A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. A computer program is a collection of instructions that performs a specific task when executed by a computer. A process is basically a program in execution. We write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.	**एक प्रक्रिया और कार्यक्रम के बीच अंतर** एक कार्यक्रम कोड का एक टुकड़ा है जो एक पंक्ति या लाखों पंक्तियु से बानी हो सकती है। एक कंप्यूटर प्रोग्राम आमतौर पर एक प्रोग्रामिंग भाषा में कंप्यूटर प्रोग्रामर द्वारा लिखा जाता है। एक कंप्यूटर प्रोग्राम उन निर्देशों का संग्रह होता है जो कंप्यूटर द्वारा निष्पादित करते समय एक विशिष्ट कार्य करता है। एक प्रक्रिया मूल रूप से एक कार्यक्रम का निष्पादन होता है। हम अपने कंप्यूटर प्रोग्राम को एक टेक्स्ट फ़ाइल में लिखते हैं और जब हम इस प्रोग्राम को निष्पादित करते हैं, तो यह एक प्रक्रिया बन जाती है जो प्रोग्राम में उल्लिखित सभी कार्यों को निष्पादित करती है।

Basis for comparison तुलना के लिए आधार	Program कार्यक्रम	Process प्रक्रिया
Basic बुनियादी	Program is a set of instruction. कार्यक्रम निर्देश का एक सेट है।	When a program is executed, it is known as process. जब कोई प्रोग्राम निष्पादित होता है, तो इसे प्रक्रिया के रूप में जाना जाता है।
Nature प्रकृति	Passive निष्क्रिय	Active सक्रिय
Lifespan जीवनकाल	Longer लंबे समय तक	Limited सीमित
Required resources स्रोतों की आवश्यकता	Program is stored on disk in some file and does not require any other resources. प्रोग्राम फाइल के रूप में डिस्क पर संग्रहीत होती है और इन्हे किसी अन्य संसाधन की आवश्यकता नहीं होती है।	Process requires resources such as CPU, memory address, disk, I/O etc. प्रक्रिया में संसाधन जैसे सीपीयू, मेमोरी एड्रेस, डिस्क, I/O इत्यादि की जरूरत होती है.

PROCESS CONTROL BLOCK (PCB)	PROCESS CONTROL BLOCK (PCB)
A Process Control Block is a data structure maintained by the Operating System for every process. A PCB keeps all the information needed to keep track of a process. The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems.	एक प्रोसेस कंट्रोल ब्लॉक प्रत्येक प्रक्रिया के लिए ऑपरेटिंग सिस्टम द्वारा बनाए रखा डेटा संरचना है। एक PCB एक प्रक्रिया का ट्रैक रखने के लिए आवश्यक सभी जानकारी रखता है। PCB का आर्किटेक्चर ऑपरेटिंग सिस्टम पर पूरी तरह से निर्भर है और इसमें विभिन्न ऑपरेटिंग सिस्टम में अलग-अलग जानकारी हो सकती है।



The contents of PCB are as follows:

1. **Process ID:** Unique identification for each of the process in the operating system.

2. **Process State:** The current state of the process i.e., whether it is ready, running, waiting, or suspended.

3. Pointer: A pointer to parent process.

4. Process privileges: This is required to allow/disallow access to system resources.

5. Program Counter: Program Counter is a pointer to the address of the next instruction to be executed for this process.

6. CPU registers: Various CPU registers where process need to be stored for execution for running state.

7. IO status information: This includes a list of I/O devices allocated to the process.

8. Accounting information: This includes the amount of CPU used for process execution, time limits, execution ID etc.

9. CPU Scheduling Information: Process priority and other scheduling information which is required to schedule the process.

PCB की सामग्री इस प्रकार है:

1. **Process ID:** ऑपरेटिंग सिस्टम में प्रत्येक प्रक्रिया के लिए अद्वितीय पहचान।

2. **Process State:** प्रक्रिया की वर्तमान स्थिति यानी, चाहे वह तैयार है, चल रही है, प्रतीक्षा कर रही है या निलंबित है।

3. Pointer: मूल प्रक्रिया के लिए एक सूचक।

4. Process privileges: सिस्टम संसाधनों तक पहुंच को अनुमति / न अनुमति देने की आवश्यकता है।

5. Program Counter: कार्यक्रम काउंटर इस प्रक्रिया के लिए अगले निर्देश के पते पर एक सूचक है।

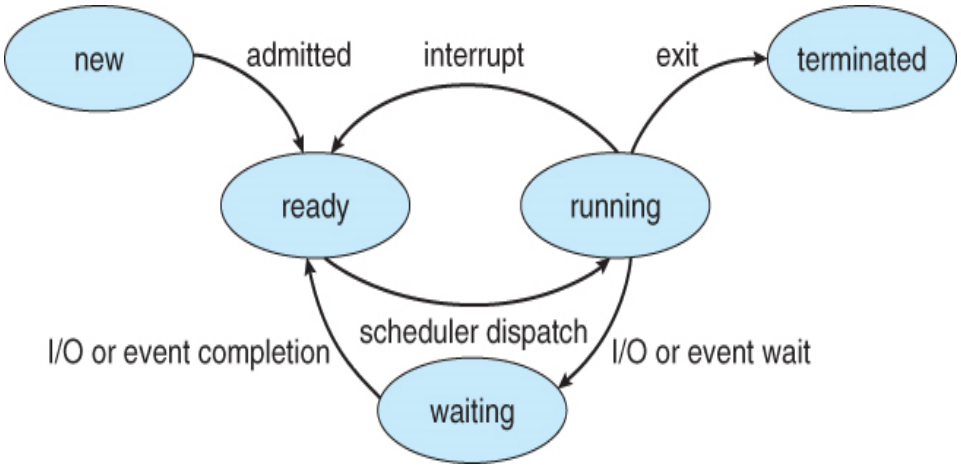
6. CPU registers: विभिन्न सीपीयू रजिस्ट्रार जहां चलने वाले प्रोग्राम के लिए निष्पादन के लिए प्रक्रिया को संग्रहीत करने की आवश्यकता होती है।

7. IO status information: इसमें प्रक्रिया में आवंटित I / O उपकरणों की एक सूची शामिल है।

8. Accounting information: इसमें प्रक्रिया निष्पादन, समय सीमा, निष्पादन आईडी आदि के लिए उपयोग की जाने वाली सीपीयू की मात्रा शामिल है।

9. CPU Scheduling Information: प्रक्रिया प्राथमिकता और अन्य शेड्यूलिंग जानकारी जो प्रक्रिया को निर्धारित करने के लिए आवश्यक है।

PROCESS STATES DIAGRAM	**PROCESS STATES (प्रक्रिया की स्थिति)**
The process, from its creation to completion, passes through various states. The minimum number of states is five. The process state can indicate the status and condition of a process.	प्रक्रिया, इसकी सृष्टि से पूरा होने तक, विभिन्न चरणों के माध्यम से गुजरती है। चरणों की न्यूनतम संख्या पांच है। प्रक्रिया स्थिति प्रक्रिया की स्थिति को इंगित कर सकती है।



1. Start: This is the initial state when a process is first started / created. **2. Ready:** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after Start state or while running it by but interrupted by the scheduler to assign CPU to some other process. **3. Running:** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions. **4. Waiting:** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available. **5. Terminated or Exit:** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.	**1. Start:** यह प्रारंभिक स्थिति है जब एक प्रक्रिया पहली बार शुरू / बनाई गई है। **2. Ready:** प्रक्रिया प्रोसेसर को सौंपने की प्रतीक्षा कर रही है। तैयार प्रक्रियाएं ऑपरेटिंग सिस्टम द्वारा आवंटित प्रोसेसर रखने की प्रतीक्षा कर रही हैं ताकि वे चल सकें। प्रारंभ स्थिति के बाद या इसे चलाने के दौरान प्रक्रिया इस स्थिति में आ सकती है लेकिन सीपीयू को किसी अन्य प्रक्रिया में असाइन करने के लिए शेड्यूलर द्वारा बाधित होती है। **3. Running:** एक बार O.S शेड्यूलर द्वारा प्रोसेसर को प्रक्रिया सौंपी जाने के बाद, प्रक्रिया स्थिति चलने के लिए सेट हो जाती है और प्रोसेसर इसके निर्देश निष्पादित करता है। **4. Waiting:** प्रक्रिया प्रतीक्षा स्थिति में जाती है अगर उसे संसाधन की प्रतीक्षा करने की आवश्यकता होती है, जैसे कि उपयोगकर्ता इनपुट की प्रतीक्षा करना, या फ़ाइल उपलब्ध होने का इंतजार करना। **5. Terminated or Exit:** प्रक्रिया समाप्त होने के बाद, या इसे ऑपरेटिंग सिस्टम द्वारा समाप्त कर दिया जाता है, इसे समाप्त राज्य में स्थानांतरित कर दिया जाता है जहां यह मुख्य स्मृति से हटाया जाता है।

PROCESS SCHEDULING

The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

SCHEDULING CRITERIA

1. CPU utilization: Keep the CPU as busy as possible. CPU utilization may range from 0 to 100 percent.

2. Throughput: The number of process executed by the system in a specific period of time is called throughput.

3. Burst Time: Time required by a process for CPU execution.

4. Turnaround time: The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O. In an interactive system, turnaround time may not be the best criterion.

•Turn Around Time = Completion Time - Arrival Time

5. Waiting time: The amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

Waiting time = turnaround time – Burst time

6. Response time: It is the time from the submission of a request until the first response is produced.

Response time = Arrival time – schedule time

PROCESS SCHEDULING (प्रक्रिया अनुसूची)

यह निर्धारित करने का कार्य की कौन सी प्रक्रिया ready state में है और कौन सी प्रक्रिया running state में स्थानांतरित किया जाना चाहिए, को प्रक्रिया अनुसूची (process scheduling) के नाम से जाना जाता है।
प्रक्रिया शेड्यूलिंग का मुख्य उद्देश्य CPU को हर समय व्यस्त रखना है और सभी कार्यक्रमों के लिए न्यूनतम प्रतिक्रिया समय देना है। इसे प्राप्त करने के लिए, शेड्यूलर को CPU के अंदर और बाहर प्रक्रियाओं को स्वैप करने के लिए उपयुक्त नियम लागू करना पड़ता है।

SCHEDULING CRITERIA (अनुसूची मानदंड)

1. सीपीयू उपयोग: सीपीयू को यथासंभव व्यस्त रखें। सीपीयू उपयोग 0 से 100 प्रतिशत तक हो सकता है।

2. थ्रूपुट: एक समय में एक सिस्टम कितनी प्रक्रियाओं को चला सकता है ,उसे सिस्टम का थ्रूपुट कहा जाता है।

3. बर्स्ट टाइम: CPU निष्पादन के लिए प्रक्रिया द्वारा आवश्यक समय।

4. टर्नअराउंड समय: प्रक्रिया जमा करने के समय से पूरा करने के समय तक के अंतराल को टर्नअराउंड टाइम कहा जाता है। टर्नअराउंड टाइम मेमोरी में आने के लिए इंतजार कर रहे समय की अवधि, तैयार कतार में प्रतीक्षा, सीपीयू पर निष्पादन, और किसी I / O प्रक्रिया हो सकती है। एक इंटरेक्टिव सिस्टम में, टर्नअराउंड टाइम सबसे अच्छा मानदंड नहीं हो सकता है।
•Turn Around Time = Completion Time - Arrival Time

5. प्रतीक्षा समय: यह एक प्रक्रिया के तैयार कतार में प्रतीक्षा कर रहे समय को दर्शाती है। प्रतीक्षा समय तैयार कतार में प्रतीक्षा की गई अवधि का योग है।

Waiting time = turnaround time – Burst time

6. प्रतिक्रिया समय: अनुरोध जमा करने के बाद आने वाली पहली प्रतिक्रिया में लगा समय होता है।
Response time = Arrival time – schedule time

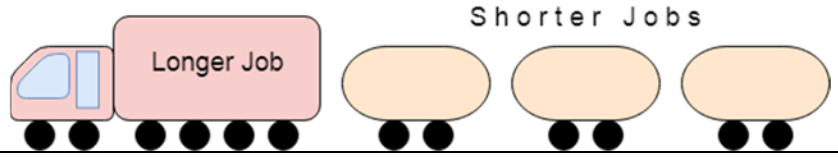
SCHEDULING ALGORITHM A scheduling algorithm is the algorithm which dictates how much CPU time is allocated to Processes and Threads. The objectives of scheduling algorithm are: 1. Maximize CPU Untilization. 2. Fair allocation of CPU. 3. Maximize throughput. 4. Minimize turnaround time. 5. Minimize waiting time. 6. Minimize response time. **Category of Scheduling Algorithm** **1. Non-preemptive:** In this once a process enters the running state; it cannot be preempted until it completes its allotted time. Ex: - FCFS, SJF, Priority, etc. **2. Preemptive:** It is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state. Ex: - SRTF, Priority, Round Robin, etc.	**शेड्यूलिंग कलन विधि** एक शेड्यूलिंग एल्गोरिदम, एक एल्गोरिदम है जो यह निर्धारित करता है कि प्रोसेस और थ्रेड के लिए कितना CPU समय आवंटित किया जाना है। शेड्यूलिंग एल्गोरिदम के उद्देश्य हैं: 1. सीपीयू आवंटन को अधिकतम करें। 2. सीपीयू का उचित आवंटन। 3. थ्रूपुट अधिकतम करें। 4. टर्नअराउंड समय को कम करें। 5. प्रतीक्षा समय कम करें। 6. प्रतिक्रिया समय को कम करें। **शेड्यूलिंग एल्गोरिदम की श्रेणी** **1. Non-preemptive:** इसमें जब एक प्रक्रिया चलती स्थिति में प्रवेश करती है; इसे तब तक छूट नहीं दी जा सकती जब तक कि यह अपना आवंटित समय पूरा न करे। Ex: - FCFS, SJF, Priority, etc. **2.Preemptive:** यह प्राथमिकता पर आधारित है जहां एक शेड्यूलर किसी भी समय एक कम प्राथमिकता वाले प्रोसेस को एक उच्च प्राथमिकता वाले प्रोसेस को अपना control दे सकता है। Ex: - SRTF, Priority, Round Robin, etc.

Preemptive Scheduling	Non-Preemptive Scheduling
Processor can be preempted to execute a different process in the middle of execution of any current process. प्रोसेसर को किसी भी मौजूदा प्रक्रिया के निष्पादन के बीच में एक अलग प्रक्रिया निष्पादित करने की अनुमति दी जा सकती है।	Once Processor starts to execute a process it must finish it before executing the other. It cannot be paused in middle. एक बार प्रोसेसर किसी एक प्रक्रिया निष्पादित करना शुरू कर देता है, तो किसे दूसरे को निष्पादित करने से पहले इसे खत्म करना होगा। इसे बीच में रोका नहीं जा सकता है।
CPU utilization is more. सीपीयू उपयोग अधिक है।	CPU utilization is less. इसमें CPU का उपयोग कम होता है।
Waiting time and Response time is less. प्रतीक्षा समय और प्रतिक्रिया समय कम है।	Waiting time and Response time is more. प्रतीक्षा समय और प्रतिक्रिया समय अधिक है।
The preemptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized. इसमें प्राथमिकता दी जाती है। सर्वोच्च प्राथमिकता प्रक्रिया हमेशा वह होनी चाहिए जिसका उपयोग वर्तमान में किया जाता है।	When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time. जब कोई प्रक्रिया चलने की स्थिति में प्रवेश करती है, तो उस प्रक्रिया की स्थिति शेड्यूलर से तब तक नहीं हटाई जाती जब तक कि यह अपने सेवा समय को पूरा न करे।
If a high priority process frequently arrives in the ready queue, low priority process may starve. यदि तैयार कतार में अक्सर उच्च प्राथमिकता प्रक्रिया आती है, तो कम प्राथमिकता प्रक्रिया भूखा हो सकती है।	If a process with long burst time is running CPU, then another process with less CPU burst time may starve. यदि लंबे समय के साथ एक प्रक्रिया CPU पर चल रही है, तो कम CPU वाले समय के साथ एक प्रक्रिया भूखा हो सकता है।
Preemptive scheduling is flexible. यह शेड्यूलिंग लचीला है।	Non-preemptive scheduling is rigid. यह शेड्यूलिंग कठोर है।

Types of Proces Scheduling Algorithms

1. First Come First Serve (FCFS): FCFS scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation (Convoy Effect) if the burst time of the first process is the longest among all the jobs. Turn Around Time = Completion Time - Arrival Time Waiting Time = Turnaround time - Burst Time	**1. First Come First Serve (FCFS):** FCFS शेड्यूलिंग एल्गोरिदम अपने आने वाले समय के अनुसार नौकरियों को शेड्यूल करता है। तैयार कतार में जो पहले होता है वह पहले सीपीयू प्राप्त करता है। नौकरी में आने वाले समय में जितना कम होगा, उतना ही जल्द उसे सीपीयू मिलेगा। FCFS शेड्यूलिंग भुखमरी (कन्वॉय प्रभाव) की समस्या का कारण बन सकती है यदि पहली प्रक्रिया का burst समय सभी नौकरियों में सबसे लंबा हो। Turn Around Time =Completion Time - Arrival Time Waiting Time = Turnaround time - Burst Time

The Convoy Effect, Visualized Starvation



2. **Shortest Job Next/First (SJF) (Non-Preemptive):** In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next. However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.	2. **Shortest Job Next/First (SJF) (Non-Preemptive):** SJF शेड्यूलिंग में, तैयार कतार में उपलब्ध प्रक्रियाओं की सूची के बीच सबसे कम burst समय वाली प्रक्रिया को निर्धारित की जाती है। हालांकि, प्रक्रिया के लिए आवश्यक burst समय की भविष्यवाणी करना बहुत मुश्किल है इसलिए इस एल्गोरिदम को सिस्टम में लागू करना बहुत मुश्किल है।

3. Shortest Remaining Time First (SRTF) or Preemptive SJF: This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process. Once all the processes are available in the **ready queue**, no preemption will be done and the algorithm will work as **SJF scheduling**.	**3. Shortest Remaining Time First (SRTF) or Preemptive SJF:** यह एल्गोरिदम SJF शेड्यूलिंग का प्रीपेप्टिव संस्करण है। SRTF में, प्रक्रिया के निष्पादन को कुछ समय के बाद रोक दिया जाता है। प्रत्येक प्रक्रिया के आगमन पर, शॉर्ट टर्म शेड्यूलर उपलब्ध प्रक्रियाओं और चल रही प्रक्रिया की सूची में कम से कम शेष burst के साथ प्रक्रिया को शेड्यूल करता है। एक बार सभी प्रक्रियाएं तैयार कतार में उपलब्ध हो जाने के बाद, कोई छूट नहीं की जाएगी और एल्गोरिदम SJF शेड्यूलिंग के रूप में काम करेगा।

4. Round Robin Scheduling (RR): Round Robin is the preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called a **quantum**. Once a process is executed for a given time period, it is preempted and other process executes for a given time period. Context switching is used to save states of preempted processes.	**4. Round Robin Scheduling (RR):** राउंड रॉबिन प्रीपेप्टिव प्रक्रिया शेड्यूलिंग एल्गोरिदम है। प्रत्येक प्रक्रिया को निष्पादित करने के लिए एक निश्चित समय प्रदान किया जाता है, इसे क्वांटम(quantum) कहा जाता है। एक बार एक दी गई अवधि के लिए प्रक्रिया को निष्पादित करने के बाद, इसे किसी भी समय अवधि के लिए पूर्ववत किया जाता है और अन्य प्रक्रिया निष्पादित की जाती है। संदर्भ स्विचिंग (Context switching) का इस्तेमाल preempted प्रक्रियाओं के states को बचाने (save) के लिए किया जाता है।

5. Priority Based Scheduling: Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.	**5. Priority Based Scheduling:** प्राथमिकता शेड्यूलिंग एक गैर-प्रीपेप्टिव एल्गोरिदम है और बैच सिस्टम में सबसे आम शेड्यूलिंग एल्गोरिदम में से एक है। प्रत्येक प्रक्रिया को प्राथमिकता दी जाती है। सर्वोच्च प्राथमिकता के साथ प्रक्रिया पहले और इतनी पर निष्पादित की जानी चाहिए। समान प्राथमिकता वाले प्रक्रियाओं में पहले आये प्रक्रियाओं को निष्पादित किया जाता है। प्राथमिकता तय की जा सकती है स्मृति आवश्यकताओं, समय आवश्यकताओं या किसी अन्य संसाधन आवश्यकता के आधार पर।

6. Multilevel Queue Scheduling: In multilevel queue scheduling, the ready queue is divided into different queues. In each queue, processes are allocated according to their priority. In the multi-level queue scheduling, which will have high priority processes, they will place them in the top-level queue and keep those processes which have less priority in the lower-level queue. In multilevel queue scheduling, if a process is allocated to a queue once, then it can not allocate the process to another queue. That is, the processes will always remain in the same-queue. In multilevel queue scheduling until all the processes present in the top level queue are executed, the processes present in the level queue below can not be executed. In multilevel queue scheduling, each queue can have its own scheduling algorithm. It is not necessary that all the queues have the same scheduling algorithm applied.	**6. Multilevel queue scheduling**: में, ready queue को विभिन्न queue में विभाजित किया जाता है. प्रत्येक queue में processes को उनकी priority के अनुसार store (allocate) किया जाता है. Multi-level queue scheduling में, जो high priority वाली processes होती है उन्हें सबसे ऊपर (top-level) के queue में रखेंगे और जिन processes की priority कम होगी उन्हें नीचे वाले (low-level) के queue में रखेंगे. Multilevel queue scheduling में, अगर किसी process को एक बार किसी queue को allocate कर दिया जाए तो उस process को दूसरे queue को allocate नही कर सकते है. अर्थात् processes हमेशा के लिए उसी queue में रहेंगी. Multilevel queue scheduling में जब तक top level queue में उपस्थित सभी processes execute नहीं हो जाती है तब तक उसके नीचे level के queue में उपस्थित processes execute

Consider for Example:

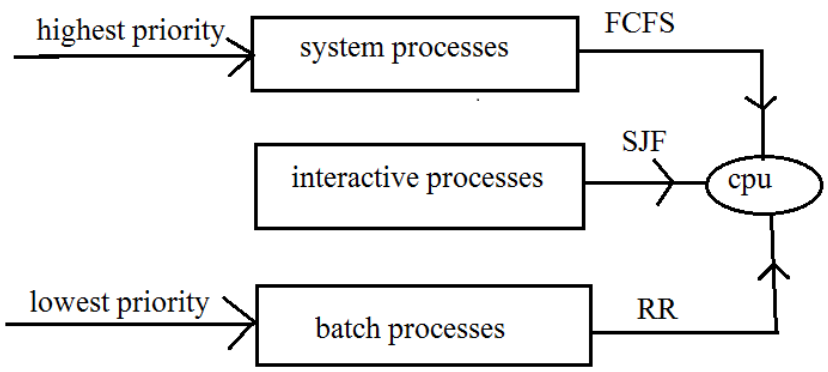
Suppose we have 3 queues: -

1. System processes: There are many system processes in the operating system such as: - interrupt.
If the system is interrupt, then all the work in the system will stop. So the first operating system will only execute the interrupt.
2. Interaction processes: The interaction processes are those in which the user directs interact with an application. Such as: - Watching movies, programming, game play etc.

3. Batch processes: The priority of the batch processes is the lowest; they are also called background processes. In this we give lots of processes to the system and the system keeps executing these processes in the background.

नहीं हो सकती है. Multilevel queue scheduling में प्रत्येक queue की अपनी अलग scheduling algorithm हो सकती है.जरुरी नहीं है कि सभी queues में एक ही scheduling algorithm अप्लाई हो.

आइये इसे अब हम एक उदाहरण के द्वारा समझते है.
माना कि हमारे पास 3 queues है:-
1. System processes: ऑपरेटिंग सिस्टम में बहुत प्रकार की system processes होती है जैसे:- interrupt. यदि सिस्टम में interrupt आ गया तो सिस्टम की सारीं working बंद हो जाएगी. तो सबसे पहले ऑपरेटिंग सिस्टम interrupt को ही execute करेगा.
2. Interaction processes: interaction processes वह होती है जिसमें user किसी एप्लीकेशन से direct interact करता है. जैसे:- मूवी देखना, प्रोग्रामिंग करना, गेम खेलना आदि.
3. Batch processes: batch processes की priority सबसे कम होती है इन्हें background processes भी कहते है. इसमें हम बहुत सारीं processes सिस्टम को दे देते है और सिस्टम इन processes को background में execute करते रहता है.



In this example, system processes have the highest priority, so it is placed in the top of the queue and likewise the priority of the batch processes is the lowest so it is placed in the bottom queue.

In the above picture, you can see FCFS (first come first serve) algorithm used for system processes. RJ (round robin) algorithms have been used for SJF (shortest job first) and batch processes for interaction processes.

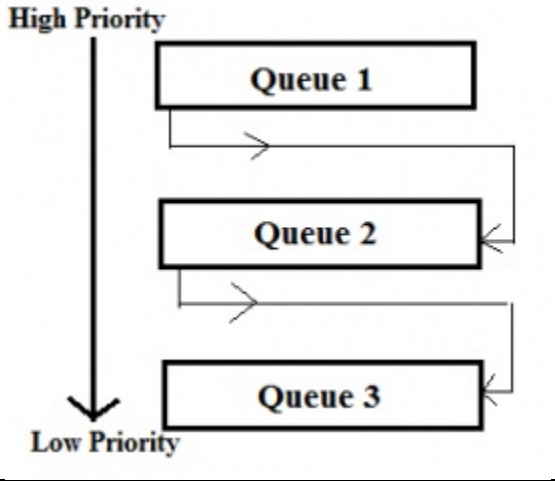
इस उदाहरण में system processes की priority सबसे ज्यादा है इसलिए इसे सबसे top के queue में रखा गया है और इसी तरह batch processes की priority सबसे कम है इसलिए इसे सबसे नीचे के queue में रखा गया है.

उपर दिए गये चित्र में आप देख सकते है system processes के लिए FCFS (first come first serve) अल्गोरिथम का प्रयोग किया है. Interaction processes के लिए SJF (shortest job first) तथा batch processes के लिए RR (round robin) अल्गोरिथम का प्रयोग किया गया है.

7. Multilevel Feedback Queue Scheduling:

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has

7. Multilevel Feedback Queue Scheduling:
एक बहुस्तरीय कतार-शेड्यूलिंग एल्गोरिदम में, प्रक्रियाओं को सिस्टम में प्रवेश पर एक कतार में स्थायी रूप से असाइन किया जाता है। प्रक्रियाएं कतारों के बीच नहीं बढ़ती (move) हैं। इस

the advantage of low scheduling overhead, but the disadvantage of being inflexible. Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.	सेटअप में कम शेड्यूलिंग ओवरहेड का लाभ है, लेकिन असंगत (inflexible) होने का नुकसान है। मल्टीलेवल फीडबैक कतार शेड्यूलिंग, हालांकि, कतारों के बीच एक प्रक्रिया को स्थानांतरित करने की अनुमति देता है। इसका मुख्य उद्देश्य विभिन्न सीपीयू-विस्फोट(burst) विशेषताओं के साथ प्रक्रियाओं को अलग-अलग करना है। यदि कोई प्रक्रिया बहुत अधिक CPU समय का उपयोग करती है, तो इसे कम प्राथमिकता वाली कतार में ले जाया जाएगा। इसी तरह, एक प्रक्रिया जो कम प्राथमिकता वाली कतार में बहुत अधिक प्रतीक्षा करती है उसे उच्च प्राथमिकता वाली कतार में स्थानांतरित किया जा सकता है। उम्र बढ़ने इस रूप से भूख(starvation) से बचाता है।
	

SCHEDULING ALGORITHM A scheduling algorithm is the algorithm which dictates how much CPU time is allocated to Processes and Threads. The objectives of scheduling algorithm are: 1. Maximize CPU Untilization. 2. Fair allocation of CPU. 3. Maximize throughput. 4. Minimize turnaround time. 5. Minimize waiting time. 6. Minimize response time. **Category of Scheduling Algorithm** **1. Non-preemptive:** In this once a process enters the running state; it cannot be preempted until it completes its allotted time. Ex: - FCFS, SJF, Priority, etc. **2. Preemptive:** It is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state. Ex: - SRTF, Priority, Round Robin, etc.	**शेड्यूलिंग कलन विधि** एक शेड्यूलिंग एल्गोरिदम, एक एल्गोरिदम है जो यह निर्धारित करता है कि प्रोसेस और थ्रेड के लिए कितना CPU समय आवंटित किया जाना है। शेड्यूलिंग एल्गोरिदम के उद्देश्य हैं: 1. सीपीयू आवंटन को अधिकतम करें। 2. सीपीयू का उचित आवंटन। 3. थ्रूपुट अधिकतम करें। 4. टर्नअराउंड समय को कम करें। 5. प्रतीक्षा समय कम करें। 6. प्रतिक्रिया समय को कम करें। **शेड्यूलिंग एल्गोरिदम की श्रेणी** **1. Non-preemptive:** इसमें जब एक प्रक्रिया चलती स्थिति में प्रवेश करती है; इसे तब तक छूट नहीं दी जा सकती जब तक कि यह अपना आवंटित समय पूरा न करे। Ex: - FCFS, SJF, Priority, etc. **2.Preemptive:** यह प्राथमिकता पर आधारित है जहां एक शेड्यूलर किसी भी समय एक कम प्राथमिकता वाले प्रोसेस को एक उच्च प्राथमिकता वाले प्रोसेस को अपना control दे सकता है। Ex: - SRTF, Priority, Round Robin, etc.

Preemptive Scheduling	Non-Preemptive Scheduling
Processor can be preempted to execute a different process in the middle of execution of any current process. प्रोसेसर को किसी भी मौजूदा प्रक्रिया के निष्पादन के बीच में एक अलग प्रक्रिया निष्पादित करने की अनुमति दी जा सकती है।	Once Processor starts to execute a process it must finish it before executing the other. It cannot be paused in middle. एक बार प्रोसेसर किसी एक प्रक्रिया निष्पादित करना शुरू कर देता है, तो किसे दूसरे को निष्पादित करने से पहले इसे खत्म करना होगा। इसे बीच में रोका नहीं जा सकता है।
CPU utilization is more. सीपीयू उपयोग अधिक है।	CPU utilization is less. इसमें CPU का उपयोग कम होता है।
Waiting time and Response time is less. प्रतीक्षा समय और प्रतिक्रिया समय कम है।	Waiting time and Response time is more. प्रतीक्षा समय और प्रतिक्रिया समय अधिक है।
The preemptive scheduling is prioritized. The highest priority process should always be the process that is currently utilized. इसमें प्राथमिकता दी जाती है। सर्वोच्च प्राथमिकता प्रक्रिया हमेशा वह होनी चाहिए जिसका उपयोग वर्तमान में किया जाता है।	When a process enters the state of running, the state of that process is not deleted from the scheduler until it finishes its service time. जब कोई प्रक्रिया चलने की स्थिति में प्रवेश करती है, तो उस प्रक्रिया की स्थिति शेड्यूलर से तब तक नहीं हटाई जाती जब तक कि यह अपने सेवा समय को पूरा न करे।
If a high priority process frequently arrives in the ready queue, low priority process may starve. यदि तैयार कतार में अक्सर उच्च प्राथमिकता प्रक्रिया आती है, तो कम प्राथमिकता प्रक्रिया भूखा हो सकती है।	If a process with long burst time is running CPU, then another process with less CPU burst time may starve. यदि लंबे समय के साथ एक प्रक्रिया CPU पर चल रही है, तो कम CPU वाले समय के साथ एक प्रक्रिया भूखा हो सकता है।
Preemptive scheduling is flexible. यह शेड्यूलिंग लचीला है।	Non-preemptive scheduling is rigid. यह शेड्यूलिंग कठोर है।

FCFS (First Come, First Served) Scheduling Algorithm (Non-Preemptive Type)

FCFS is a scheduling algorithm used in operating systems and computer systems to manage the execution of processes or tasks in a queue. In FCFS scheduling, processes are executed in the order they arrive in the ready queue. The process that arrives first gets executed first, and so on. It is a non-preemptive scheduling algorithm, which means once a process starts executing, it continues until it finishes or goes into a waiting state.

Here's how FCFS scheduling works:

1. When a process arrives in the ready queue, it is placed at the end of the queue.
2. The CPU scheduler selects the process at the front of the queue (the one that arrived first) for execution.
3. The selected process runs until it completes or enters a waiting state (e.g., I/O operation). During this time, no other process can run.
4. Once the currently running process finishes or goes into a waiting state, the CPU scheduler selects the next process in the queue for execution.
5. This process continues until all processes in the queue have been executed.

FCFS scheduling is simple to implement and ensures fairness in the sense that the first process to arrive will be the first to be executed. However, it has some drawbacks:

1. It may lead to poor average waiting times: If a long CPU-bound process arrives first, it can cause shorter processes to wait for a long time, leading the problem of starvation **(Convoy Effect)** if the burst time of the first process is the longest among all the jobs.
2. It is not suitable for time-sensitive tasks: FCFS doesn't consider the priority or execution time of processes, so important or urgent tasks may be delayed if they arrive later.
3. It doesn't adapt well to changing workload: FCFS doesn't dynamically prioritize processes based on their characteristics or system load.

Because of its limitations, FCFS is not commonly used in modern operating systems for general-purpose scheduling. Instead, more sophisticated algorithms like Round Robin, Priority Scheduling, and Multilevel Queue

FCFS (पहले आओ, पहले पाओ) शेड्यूलिंग एल्गोरिदम ((गैर-निवारक प्रकार)

FCFS शेड्यूलिंग एल्गोरिदम एक शेड्यूलिंग एल्गोरिदम है जिसका उपयोग ऑपरेटिंग सिस्टम और कंप्यूटर सिस्टम में एक कतार में प्रक्रियाओं या कार्यों के निष्पादन को प्रबंधित करने के लिए किया जाता है। एफसीएफएस शेड्यूलिंग में, प्रक्रियाओं को तैयार कतार में आने के क्रम में निष्पादित किया जाता है। जो प्रक्रिया पहले आती है वह पहले क्रियान्वित होती है, इत्यादि। यह एक गैर-प्रीमेप्टिव शेड्यूलिंग एल्गोरिदम है, जिसका अर्थ है कि एक बार प्रक्रिया निष्पादित होने के बाद, यह तब तक जारी रहती है जब तक यह समाप्त नहीं हो जाती या प्रतीक्षा स्थिति में नहीं चली जाती।

यहां बताया गया है कि एफसीएफएस शेड्यूलिंग कैसे काम करती है:

1. जब कोई प्रक्रिया तैयार कतार में आती है, तो उसे कतार के अंत में रखा जाता है।

2. सीपीयू शेड्यूलर निष्पादन के लिए कतार के सामने की प्रक्रिया (वह जो पहले पहुंची) का चयन करता है।

3. चयनित प्रक्रिया तब तक चलती है जब तक वह पूरी नहीं हो जाती या प्रतीक्षा स्थिति में प्रवेश नहीं कर जाती (उदाहरण के लिए, I/O ऑपरेशन) । इस दौरान कोई अन्य प्रक्रिया नहीं चल सकती.

4. एक बार जब वर्तमान में चल रही प्रक्रिया समाप्त हो जाती है या प्रतीक्षा स्थिति में चली जाती है, तो सीपीयू शेड्यूलर निष्पादन के लिए कतार में अगली प्रक्रिया का चयन करता है।

5. यह प्रक्रिया तब तक जारी रहती है जब तक कतार में सभी प्रक्रियाएं निष्पादित नहीं हो जातीं।

एफसीएफएस शेड्यूलिंग को लागू करना सरल है और यह इस अर्थ में निष्पक्षता सुनिश्चित करता है कि आने वाली पहली प्रक्रिया निष्पादित होने वाली पहली प्रक्रिया होगी। हालाँकि, इसमें कुछ कमियाँ हैं:

1. इससे औसत प्रतीक्षा समय खराब हो सकता है: यदि एक लंबी सीपीयू-बाउंड प्रक्रिया पहले आती है, तो इससे छोटी प्रक्रियाओं को लंबे समय तक इंतजार करना पड़ सकता है, जिससे इष्टतम प्रदर्शन कम हो सकता है।

2. यह समय-संवेदनशील कार्यों के लिए उपयुक्त नहीं है: एफसीएफएस प्रक्रियाओं की प्राथमिकता या निष्पादन समय पर विचार नहीं करता है, इसलिए महत्वपूर्ण या जरूरी कार्य देर से आने पर विलंबित हो सकते हैं।

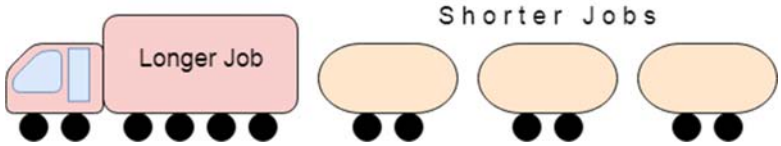
3. यह बदलते कार्यभार के अनुकूल नहीं है: एफसीएफएस प्रक्रियाओं को उनकी विशेषताओं या सिस्टम लोड के आधार पर गतिशील रूप से प्राथमिकता नहीं देता है।

अपनी सीमाओं के कारण, एफसीएफएस का उपयोग आमतौर पर सामान्य प्रयोजन शेड्यूलिंग के लिए आधुनिक ऑपरेटिंग सिस्टम में नहीं किया जाता है। इसके बजाय, संसाधन उपयोग और

Scheduling are often employed to optimize resource utilization and responsiveness.	प्रतिक्रिया को अनुकूलित करने के लिए राउंड रॉबिन, प्रायोरिटी शेड्यूलिंग और मल्टीलेवल क्यू शेड्यूलिंग जैसे अधिक परिष्कृत एल्गोरिदम को अक्सर नियोजित किया जाता है।

Turn Around Time = Completion Time - Arrival Time
Waiting Time = Turnaround time - Burst Time

The Convoy Effect, Visualized Starvation



Example 1 of FCFS Consider the following table of arrival time and burst time for five processes P1, P2, P3, P4 and P5.	एफसीएफएस का उदाहरण पाँच प्रक्रियाओं P1, P2, P3, P4 और P5 के आगमन समय और बर्स्ट समय की निम्नलिखित तालिका पर विचार करें।

Processes	Arrival Time	Burst Time
P1	0	4
P2	1	3
P3	2	1
P4	3	2
P5	4	5

The FCFS CPU Scheduling Algorithm will work on the basis of steps as mentioned below:

Step 0: At time = 0,
- The process begins with P1
- As it has an arrival time 0

Step 1: At time = 1,
- The process P2 arrives
- But process P1 still executing,
- Thus, P2 is kept on a waiting table and waits for its execution.

Step 3: At time = 2,
- The process P3 arrives and kept in a waiting queue
- While process P1 is still executing as its burst time is 4.

Step 4: At time = 3,
- The process P4 arrives and kept in the waiting queue
- While process P1 is still executing as its burst time is 4

Step 5: At time = 4,
- The process P1 completes its execution
- Process P5 arrives in waiting queue while process P2 starts executing

Step 6: At time = 5,
- The process P2 completes its execution

Step 7: At time = 7,

FCFS CPU शेड्यूलिंग एल्गोरिदम नीचे बताए गए चरणों के आधार पर काम करेगा:

चरण 0: समय = 0 पर,
- प्रक्रिया P1 से शुरू होती है
- चूंकि इसका आगमन समय 0 है

चरण 1: समय = 1 पर,
- प्रक्रिया P2 आती है
- लेकिन प्रक्रिया P1 अभी भी क्रियान्वित हो रही है,
- इस प्रकार, P2 को प्रतीक्षा तालिका पर रखा जाता है और इसके निष्पादन की प्रतीक्षा की जाती है।

चरण 3: समय = 2 पर,
- प्रक्रिया P3 आती है और प्रतीक्षा कतार में रखी जाती है
- जबकि प्रक्रिया P1 अभी भी क्रियान्वित हो रही है क्योंकि इसका विस्फोट समय 4 है।

चरण 4: समय = 3 पर,
- प्रक्रिया P4 आती है और प्रतीक्षा कतार में रखी जाती है
- जबकि प्रक्रिया P1 अभी भी क्रियान्वित हो रही है क्योंकि इसका विस्फोट समय 4 है

चरण 5: समय = 4 पर,
- प्रक्रिया P1 अपना निष्पादन पूरा करती है
- प्रक्रिया P5 प्रतीक्षा कतार में आती है जबकि प्रक्रिया P2 निष्पादित होना शुरू हो जाती है

चरण 6: समय = 5 पर,
- प्रक्रिया P2 अपना निष्पादन पूरा करती है

चरण 7: समय = 7 पर,

- Process P3 starts executing, it has burst time of 1 thus, it completes execution at time interval 8

Step 8: At time 8,
- The process of P3 completes its execution
- Process P4 starts executing, it has burst time of 2 thus, it completes execution at time interval 10.

Step 9: At time 10,
- The process P4 completes its execution
- Process P5 starts executing, it has burst time of 5 thus, it completes execution at time interval 15.

Step 10: At time 15,
- Process P5 will finish its execution.

The overall execution of the processes will be as shown below:

- प्रक्रिया P3 निष्पादित करना शुरू करती है, इसका बर्स्ट समय 1 है, इस प्रकार यह समय अंतराल 8 पर निष्पादन पूरा करती है

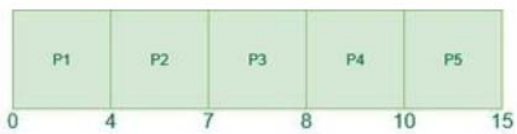
चरण 8: समय 8 पर,
- P3 की प्रक्रिया इसके निष्पादन को पूरा करती है
- प्रक्रिया P4 निष्पादित करना शुरू करती है, इसका बर्स्ट समय 2 है, इस प्रकार, यह समय अंतराल 10 पर निष्पादन पूरा करती है।

चरण 9: समय 10 पर,
- प्रक्रिया P4 अपना निष्पादन पूरा करती है
- प्रक्रिया P5 निष्पादित करना शुरू करती है, इसका बर्स्ट समय 5 है, इस प्रकार, यह समय अंतराल 15 पर निष्पादन पूरा करती है।

चरण 10: समय 15 पर,
- प्रक्रिया P5 अपना निष्पादन समाप्त कर देगी।

प्रक्रियाओं का समग्र निष्पादन नीचे दिखाए अनुसार होगा:

Gantt chart for above execution:



Gantt chart for First come First serve Scheduling

Calculate Turnaround Time (TT)

$T_{P1}=4-0=4$
$T_{P2}=7-1=6$
$T_{P3}=8-2=6$
$T_{P4}=10-3=7$
$T_{P5}=15-4=11$

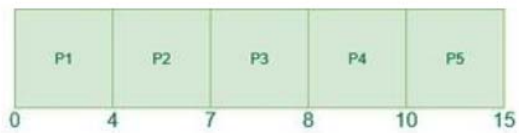
Calculate Waiting Time(WT)

$W_{P1}=4-4=0$
$W_{P2}=6-3=3$
$W_{P3}=6-1=5$
$W_{P4}=7-2=5$
$W_{P5}=11-5=6$

Average Waiting Time:

$W_{avg} = (0+3+5+5+6) / 5$
= 3.8

उपरोक्त निष्पादन के लिए गैंट चार्ट:



Gantt chart for First come First serve Scheduling

Calculate Turnaround Time (TT)

$T_{P1}=4-0=4$
$T_{P2}=7-1=6$
$T_{P3}=8-2=6$
$T_{P4}=10-3=7$
$T_{P5}=15-4=11$

Calculate Waiting Time(WT)

$W_{P1}=4-4=0$
$W_{P2}=6-3=3$
$W_{P3}=6-1=5$
$W_{P4}=7-2=5$
$W_{P5}=11-5=6$

Average Waiting Time:

$W_{avg} = (0+3+5+5+6) / 5$
= 3.8

Example 2 of FCFS

Consider the set of 5 processes whose arrival time and burst time are given below:

Process Id	Arrival time	Burst time
P1	3	4
P2	5	3
P3	0	2
P4	5	1
P5	4	3

If the CPU scheduling policy is FCFS, calculate the average waiting time and average turn around time.

Gantt Chart:

Here, black box represents the idle time of CPU.

Process Id	Exit time	Turn Around time	Waiting time
P1	7	7 – 3 = 4	4 – 4 = 0
P2	13	13 – 5 = 8	8 – 3 = 5
P3	2	2 – 0 = 2	2 – 2 = 0
P4	14	14 – 5 = 9	9 – 1 = 8
P5	10	10 – 4 = 6	6 – 3 = 3

Now,

Average Turn Around time = (4 + 8 + 2 + 9 + 6) / 5 = 29 / 5 = **5.8 unit**
Average Waiting Time = (0 + 5 + 0 + 8 + 3) / 5 = 16 / 5 = **3.2 unit**

Operating System

Unit II

Numerical Practice Questions based on FCFS Scheduling Algorithm

Q1. Consider the set of 3 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	2
P2	3	1
P3	5	6

If the CPU scheduling policy is FCFS, calculate the average waiting time and average turnaround time.

Check Your answer:

- *Average Turn Around time = 3 unit*
- *Average waiting time = 0 unit*

Q2. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	2
P2	0	1
P3	2	3
P4	3	5
P5	4	5

Check Your answer:

- *Average Turn-around time = 6*
- *Average Waiting time = 2.8*

Q3. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P0	0	10
P1	1	6
P2	3	2
P3	5	4

Check Your answer:

- *Average Turn-around time = 14.25*
- *Average Waiting time = 8.75*

Q4. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	2
P2	5	6
P3	0	4
P4	0	7
P5	7	4

Check Your answer:

- *Average Turn-around time = 11.2*
- *Average Waiting time = 6.6*

Q5. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	6
P2	1	8
P3	0	3
P4	4	4

Check Your answer:

- *Average Turn-around time = 9.3*
- *Average Waiting time = 5.6*

Q6. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0.0	8
P2	0.4	4
P3	1.0	1

Check Your answer:

- *Average Turn-around time = 10.53*
- *Average Waiting time = ??*

Q7. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	2
P2	1	3
P3	2	5
P4	3	4
P5	4	6

Check Your answer:

- *Average Turn-around time = 8.2*
- *Average Waiting time = 4.2*

Q8. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	24
P2	0	3
P3	0	3

Check Your answer:

- *Average Turn-around time = 27*
- *Average Waiting time = 17*

Q9. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Check Your answer:

- *Average Turn-around time = 15.25*
- *Average Waiting time = 8.75*

Q10. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	6
P2	1	2
P3	2	5
P4	3	6
P5	7	1

Check Your answer:

- *Average Turn-around time = 10.6*
- *Average Waiting time = 6.6*

Operating System

Unit II

Numerical Practice Questions based on SJF Scheduling Algorithm

Q1. Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	3	1
P2	1	4
P3	4	2
P4	0	6
P5	2	3

If the CPU scheduling policy is SJF non-preemptive, calculate the average waiting time and average turnaround time.

Check Your answer:

- *Average Turn Around time = 8 unit*
- *Average waiting time = 4.8 unit*

Q2. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	6
P2	5	2
P3	1	8
P4	0	3
P5	4	4

Check Your answer:

- *Average Turn-around time = ??*
- *Average Waiting time = 5.2*

Q3. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	10
P2	3	8
P3	6	9
P4	10	4

Check Your answer:

- *Average Turn-around time = ??*
- *Average Waiting time = 6.75*

Q4. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	10
P2	1	6
P3	3	2
P4	5	4

Check Your answer:

- *Average Turn-around time = 12.75*
- *Average Waiting time = 7.25*

Q5. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	1	7
P2	3	3
P3	6	2
P4	7	10
P5	9	8

Check Your answer:

- *Average Turn-around time = ??*
- *Average Waiting time = 27/5*

Q6. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	1	6
P2	2	3
P3	3	2
P4	4	3
P5	5	4

Check Your answer:

- *Average Turn-around time = 9.4*
- *Average Waiting time = 5.8*

Q7. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	20
P2	1	1
P3	2	1

Check Your answer:

- *Average Turn-around time = 20.3*
- *Average Waiting time = 13*

Q8. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	20
P2	0	1
P3	1	1

Check Your answer:

- *Average Turn-around time = 7.3*
- *Average Waiting time = 0*

Q9. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	21
P2	0	3
P3	0	6
P4	0	2

Check Your answer:

- *Average Turn-around time = ??*
- *Average Waiting time = 4.5*

Q10. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	1	6
P2	3	5
P3	4	3
P4	7	9
P5	9	2

Check Your answer:

- *Average Turn-around time = ??*
- *Average Waiting time = ??*

Operating System

Unit II

Numerical Practice Questions based on Preemptive SJF Scheduling Algorithm (SRTN Algorithm)

Q1. Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	2	6
P2	5	2
P3	1	8
P4	0	3
P5	4	4

If the CPU scheduling policy is SJF preemptive, calculate the average waiting time and average turnaround time.

Check Your answer:

- *Average waiting time = 4.6 unit*

Q2. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	6
P2	1	4
P3	2	2
P4	3	3

Check Your answer:

- *Average Waiting time = 3.75*

Q3. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	2	6
P2	5	2
P3	1	8
P4	0	3
P5	4	4

Check Your answer:

- *Average Turn-around time = 9.2*
- *Average Waiting time = 4.6*

Q4. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	3
P2	0	6
P3	2	3
P4	3	2

Check Your answer:

- *Average Turn-around time = ??*
- *Average Waiting time = ??*

Q5. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	8
P2	1	4
P3	2	9

Check Your answer:

- *Average Turn-around time = 11.66*
- *Average Waiting time = 4.66*

Q6. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	18
P2	1	4
P3	2	7
P4	3	2

Check Your answer:

- *Average Turn-around time = 12.75*
- *Average Waiting time = 5*

Q7. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Check Your answer:

- *Average Turn-around time = ??, Average Waiting time = 3*

Q8. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	10
P2	1	6
P3	3	2
P4	5	4

Check Your answer:

- *Average Turn-around time = 10*
- *Average Waiting time = 4.5*

Q9. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

Check Your answer:

- *Average Turn-around time = 12.75*
- *Average Waiting time = 5.5*

Q10. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

Process ID	Arrival time	Burst time
P1	0	6
P2	3	2
P3	5	4
P4	7	6
P5	10	3

Check Your answer:

- *Average Turn-around time = 7.2*
- *Average Waiting time = ??*

Procedure to solve RR questions.

Example-1: Consider the following table of arrival time and burst time for four processes P1, P2, P3, and P4 and given Time **Quantum = 2.**
Draw Gantt Chart and calculate average turnaround time according to RR scheduling.

Process	Arrival Time	Burst Time
P1	0	5
P2	1	4
P3	2	2
P4	4	1

Solution:

At time = 0,

- At time =0, we find from the above table which processes have entered the system.
- We see that P1 has arrived in the system. So select it and bring in the Gantt Chart and execute P1.
- P1 will execute for time quantum time of 2 sec as given in the question. After executing for 2 sec its remaining time will be now 3.

Process	Arrival Time	Burst Time
P1	0	~~5~~ 3
P2	1	4
P3	2	2
P4	4	1

Note: *We have to maintain two queues: Ready Queue and Running Queue. Ready queue will contain those processes which are ready for execution and Running queue will contain those process which are currently running in the system.*

Ready Queue: Initially at time 0 only P1 is ready and we bring it in the ready queue.

~~P1~~

From ready queue select the first process P1 and bring it in running queue and execute for 2 sec as follows:

0	2								
P1									

At time = 2,

- At time =2 we find from the above table which processes have entered the system.

- We see that P2 and P3 have arrived in the system. Bring P1 and P2 in the ready queue and also bring P1 at last position in the queue.
- So our ready queue looks like as follows:

~~P1~~	P2	P3	P1

- Now Select P2 from ready queue and bring it in running queue and execute for 2 sec.

~~P1~~	~~P2~~	P3	P1

- The following is the gantt chart:

0	2	4							
P1	P2								

Process	Arrival Time	Burst Time	
P1	0	~~5~~	3
P2	1	~~4~~	2
P3	2	2	
P4	4	1	

At time = 4,

- At time =4 we find from the above table which processes have entered the system.
- We see that P4 has arrived in the system. Bring P4 in the ready queue and also bring P2 at last position in the queue.
- So our ready queue looks like as follows:

~~P1~~	~~P2~~	P3	P1	P4	P2

- Now Select P3 from ready queue and bring it in running queue and execute for 2 sec.

~~P1~~	~~P2~~	~~P3~~	P1	P4	P2

- The following is the gantt chart:

0	2	4	6						
P1	P2	P3							

Process	Arrival Time	Burst Time	
P1	0	~~5~~	3
P2	1	~~4~~	2
P3	2	~~2~~	0
P4	4	1	

At time = 6,

- At time =6 we find from the above table which processes have entered the system.
- We see that all processes have arrived in the system. That means we have covered all the programs. Now select the next process from the ready queue that is P1 and bring it in the running queue.
- So our ready queue looks like as follows:

~~**P1**~~	~~**P2**~~	~~**P3**~~	~~**P1**~~	**P4**	**P2**

- The following is the gantt chart:

0 2 4 6 8

P1	**P2**	**P3**	**P1**						

Process	Arrival Time	Burst Time		
P1	0	~~5~~	~~3~~	1
P2	1	~~4~~	2	
P3	2	~~2~~	0	
P4	4	1		

- Now P1 still requires 1 sec to complete but its chance has gone now. So we pre-empt the process P1 and put in the last of ready queue as follows:

~~**P1**~~	~~**P2**~~	~~**P3**~~	~~**P1**~~	**P4**	**P2**	**P1**

- Now select the next process in the ready queue and bring it in the running queue and execute it for 2 sec. (here it is now P4). Since P4 require only 1 sec to complete so it will run only for 1 sec.

~~P1~~	~~P2~~	~~P3~~	~~P1~~	~~P4~~	P2	P1

❖ The following is the gantt chart:

0 2 4 6 8 9

P1	P2	P3	P1	P4					

Process	Arrival Time	Burst Time		
P1	0	~~5~~	~~3~~	1
P2	1	~~4~~	2	
P3	2	~~2~~	0	
P4	4	~~1~~	0	

❖ Now select the next process that is P2 from the Ready queue and bring it in the gantt chart and execute it for 2 sec as follows:

~~P1~~	~~P2~~	~~P3~~	~~P1~~	~~P4~~	~~P2~~	P1

❖ The following is the gantt chart:

0 2 4 6 8 9 11

P1	P2	P3	P1	P4	P2				

Process	Arrival Time	Burst Time		
P1	0	~~5~~	~~3~~	1
P2	1	~~4~~	~~2~~	0
P3	2	~~2~~	0	
P4	4	~~1~~	0	

❖ Now only one process that is P1 is remaining and it requires only 1 sec to complete. So the final gantt chart is:

0 2 4 6 8 9 11 12

P1	P2	P3	P1	P4	P2	P1			

Calculating Turnaround Time:

T.T=Completion Time – Arrival Time

T_{P1}=12-0=12

T_{P2}=11-1=10

T_{P3}=6-2=4

T_{P4}=9-4=5

Calculating Waiting Time:

W.T= T.T – Burst Time

W_{P1}=12-5=7

W_{P2}=10-4=6

W_{P3}=4-2=2

W_{P4}=5-1=4

Average Waiting Time = (7+6+2+4)/4 =4.7

Operating System

Unit II

Numerical Practice Questions based on Round Robin Scheduling Algorithm (RR Algorithm)

Q1. Consider the set of 5 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

If the CPU scheduling policy is Round Robin with time **quantum = 2 unit**, calculate the average waiting time.

Check Your answer:

- *Average waiting time = 5.8 unit*

Q2. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time. **(time quantum = 3)**

Process ID	Arrival time	Burst time
P1	5	5
P2	4	6
P3	3	7
P4	1	9
P5	2	2
P6	6	3

Check Your answer:

- *Average Turn-around time = 21.33*
- *Average Waiting time = 16*

Q3. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time. **(time quantum = 2)**

Process ID	Arrival time	Burst time
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

Check Your answer:

- *Average Turn-around time = 10.84*
- *Average Waiting time = 7.33*

Q4. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

(time quantum = 1)

Process ID	Arrival time	Burst time
P1	0	4
P2	0	1
P3	0	8
P4	0	1

Check Your answer:

- *Average Turn-around time = ??*
- *Average Waiting time = ??*

Q5. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

(time quantum = 2)

Process ID	Arrival time	Burst time
P1	0	5
P2	1	7
P3	2	3
P5	3	4

Check Your answer:

- *Average Turn-around time = 14.25*
- *Average Waiting time = 9.5*

Q6. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

(time quantum = 8)

Process ID	Arrival time	Burst time
P1	0	10
P2	1	9
P3	2	12
P4	3	6

Check Your answer:

- *Average Turn-around time = 31.5*
- *Average Waiting time = 22.25*

Q7. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

(time quantum = 1)

Process ID	Arrival time	Burst time
P1	0	10
P2	1	6
P3	3	2
P4	5	4

Check Your answer:

- *Average Turn-around time =19.5*
- *Average Waiting time = 14*

Q8. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

(time quantum = 2)

Process ID	Arrival time	Burst time
P1	0	4
P2	0	3
P3	0	5

Check Your answer:

- *Average Turn-around time =*
- *Average Waiting time = 5.6*

Q9. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

(time quantum = 2)

Process ID	Arrival time	Burst time
P1	0	6
P2	0	5
P3	0	2
P4	0	3
P5	0	7

Check Your answer:

- *Average Turn-around time = 16.6*
- *Average Waiting time = 12*

Q10. Consider the given table below and find Completion time (CT), Turn-around time (TAT), Waiting time (WT), Average Turn-around time and Average Waiting time.

(time quantum = 5)

Process ID	Arrival time	Burst time
P1	0	21
P2	0	3
P3	0	6
P4	0	2

Check Your answer:

- *Average Turn-around time = 19*
- *Average Waiting time = 11*

Inter Process Communication (IPC)

Before understanding IPC let us learn the concept of types of processes running in the operating system.

In multitasking concept several processes are executed concurrently. Based on this they can be categorized as:

1. **Independent Process**: A process is independent if it cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is independent.
2. **Cooperative Process:** A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

Now let us see what are the reasons for providing an environment for cooperating processes:

1. **Information Sharing**: Since several users may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent access to such information.
2. **Computation Speed**: If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others.
3. **Modularity**: We may want to construct the system in a modular fashion, dividing the system functions into separate processes.
4. **Convenience**: Even an individual user may work on many tasks at the same time. For instance, a user may be editing, listening to music, and compiling in parallel.

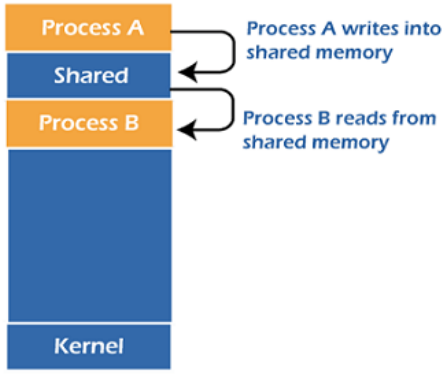
So after understanding the concept of cooperating processes, it is clear that Cooperating processes require an inter-process communication (IPC) mechanism that will allow them to exchange data and information.

How to solve the IPC situation?

There are two fundamental models of inter-process communication:

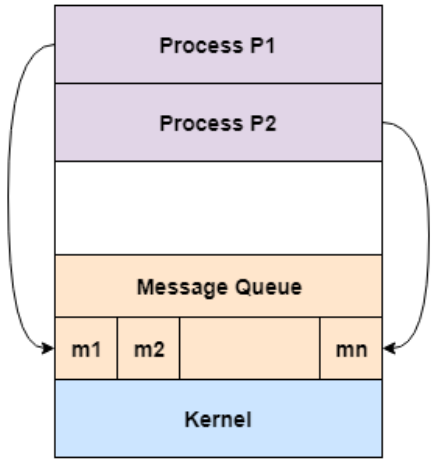
1. **Shared Memory**
2. **Message Passing**

Shared Memory Model: Shared memory is a memory shared between two or more processes. Each process has its own address space; if any process wants to communicate with some information from its own address space to other processes, then it is only possible with IPC (inter-process communication) techniques. The operating system maps a memory segment in the address space of several processes to read and write in that memory segment. Shared memory is the fastest inter-process communication mechanism.



Message Passing:

In this model, the processes communicate with others by exchanging messages. A communication link between the processes is required for this purpose, and it must provide at least two operations: transmit (message) and receive (message). Messages are stored on the queue until their recipient retrieves them.



Message Passing Model

In the above diagram, both the processes P1 and P2 can access the message queue and store and retrieve data.

Process Synchronization

Process synchronization is a critical concept in operating systems and concurrent programming. It refers to the coordination and control of multiple processes to ensure that they execute in a way that doesn't lead to data corruption, race conditions, or other undesirable outcomes. The primary goals of process synchronization are to:

1. **Prevent Data Corruption**: Processes may **share** common data or resources, and without proper synchronization, they can concurrently access and modify that data in an unpredictable manner, leading to data corruption.
2. **Avoid Race Conditions**: Race conditions occur when the behaviour of a program depends on the relative timing of events, and the program's outcome is sensitive to this timing. **Synchronization is used to prevent race conditions.**
3. **Ensure Consistency**: Synchronization mechanisms ensure that the data and resources accessed by multiple processes are in a consistent state. This helps maintain data integrity and application correctness.

Race Condition

A situation, where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a **race condition**.

Example:

Consider the code below:

Shared Variable

```
a=10
P()
{
    R(a)
    a=a+1
    W(a)
}
```

Here there is a shared variable 'a' having initial value 10. Let's see the result when the sequence of execution changes.

Case 1: (P1→P2)

- Suppose two processes P1, P2 accesses the shared variable and are executed in sequence i.e. first P1 and then P2.
- When first P1 is executed the final value of a will become: 11
- After this P2 executes the same code, then the final value of 'a' becomes: **12**

Case 2: (Interleave execution)

- Suppose we execute only two instructions of P1 i.e first we read the value of 'a' via R(a) which gives us 10, then we execute a=a+1, which gives us a=11.
- After this a context switch occurs and P2 is given chance to execute. P2 executes all its instructions.

  R(a) gives a=10,
  a=a+1 gives a=11,
  W(a), makes the final value of 'a' as 11.
- After execution of P2, P1 resumes and executes its final instruction that is W(a). After this the final value of 'a' after execution of P1 is **11.**

Here the (value 'a' = 12) and (value 'a'=11) are racing, if we execute these two processes in our computer system then sometime we will get 12 and sometime we will get 11 as the final value of 'a'. **This phenomenon is called race condition.**

Critical Section Problem

A critical section is a code segment that can be accessed by only one process at a time. The critical section contains shared variables that need to be synchronized to maintain the consistency of data variables. So the critical section problem means designing a way for cooperative processes to access shared resources without creating data inconsistencies.

```
do {
    entry section
        critical section
    exit section
        remainder section
  } while (TRUE);
```

Example of Critical Section Problem

Let us consider a classic bank example

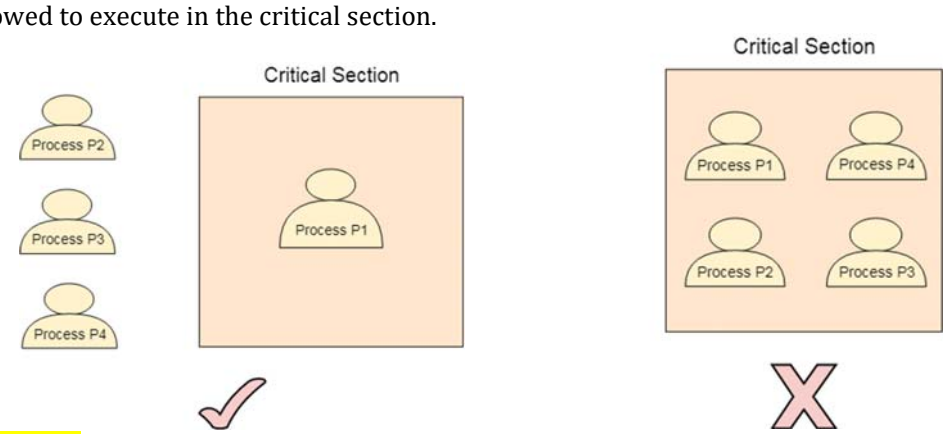
1. Let us consider a scenario where money is withdrawn from the bank by both the cashier (through **cheque**) and the **ATM** at the same time.
2. Consider an account having a balance of **₹10,000**. Let us consider that when a cashier withdraws the money, it takes 2 seconds for the balance to be updated in the account.
3. It is possible to withdraw **₹7000** from the **cashier** and within the balance update time of 2 seconds, also withdraw an amount of **₹6000** from the **ATM**.
4. Thus, the total money withdrawn becomes **greater** than the balance of the bank account.

This happened because of two withdrawals occurring at the same time. In the case of the critical section, only one withdrawal should be possible and it can solve this problem.

Criteria for Solution to Critical Section Problem

Any solution to the critical section problem must satisfy three requirements:

1. **Mutual Exclusion:** If a process is executing in its critical section, then no other process is allowed to execute in the critical section.



2. **Progress:** If there is no process in the critical section and one or more processes want to enter the critical sections then one of them must be permitted to enter the critical section.

That is if one process does not want to enter the critical section then it doesn't stop/ prevent another process from entering the critical section.
Example: *For example, suppose there are two animals horse (process 1), dog (process 2) and a grass field (critical section) with a door through which only one animal can access the field. Dog doesn't want to enter the field because it doesn't eat grass but if it stops horse from entering the field by blocking the door then it violates the principality of progress. In this case there is no progress.*

3. **Bounded Waiting:** it means no process should wait for an infinite amount of time to enter critical section. Every process should at most wait for a finite time after which it must get access to the critical section i.e. the system should be able to tell the process that after a fixed time it will surely get access to the critical section.

Note: Any solution to Critical Section problem must satisfy mutual exclusion and progress conditions, but bounded waiting condition is optional.

Solution to Critical Section Problem

1. Peterson's Solution

- Peterson's Solution is a classical software-based solution to the CS problem.
- Does not require any special hardware.
- Works only between two processes.
- May not work correctly on modern computer architecture.
- It solves the mutual exclusion, progress and bounded waiting conditions of CS.
- It requires two **data items** to be **shared** between the two processes.:

int turn	Indicates whose turn it is to enter its critical section

boolean flag[2]	Used to indicate if a process is ready to enter its critical section

Here the 'turn' and 'flag' are variables which are shared between two processes.

Now consider two processes P_i and P_j.

If turn=i, then it means that it's P_i turn to enter critical section.

If turn=j, then it means that it's P_j turn to enter critical section.

'flag' is an array, which here can store only two values i.e True or False.

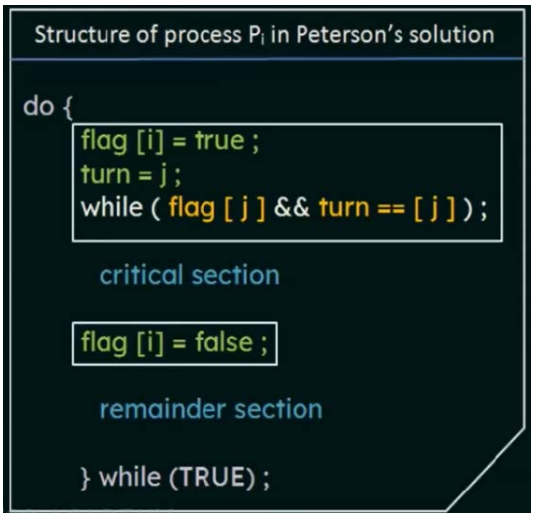
Ex1: if flag[i]==True, it means process P_i is ready to enter critical section.

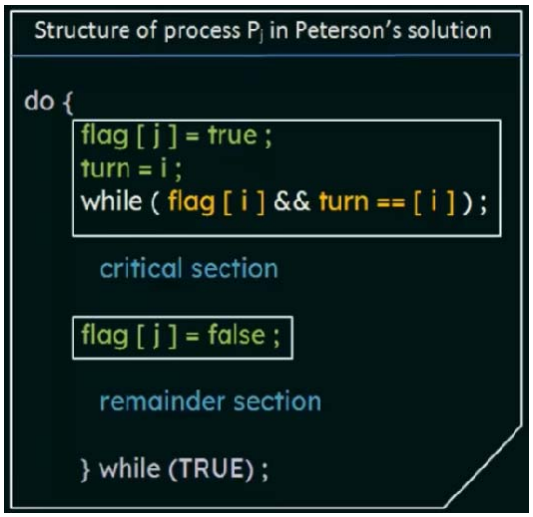
Ex2: if flag[j]==True, it means process P_j is ready to enter critical section.

Brief Working of Peterson's Method:

- If any process wants to enter in critical section, it will first set its flag to True. For example, if P_i wants to enter critical section it will set: **flag[i]=True**
- After this P_i will set the value of variable 'turn' to other process that is P_j as: **turn=P_j**

Peterson's Algorithm





1. Let us suppose P_i wants to enter C.S
2. Process P_i will first set:
   flag[i] = true
   to indicate that P_i is ready to enter C.S
3. Then process P_i will set:
   turn = j
   to indicate that it is allowing process P_j to enter C.S and the process P_i itself will wait for P_j.
4. If suppose P_j also wants to enter C.S, then it will also set:
   flag[j]=true
5. Then 'while' loop conditions are checked. If both conditions are true, then it will not come out of the loop and instruction after that will not be executed. That means P_i has to wait. But if any one condition is false and other is true then it will come out of loop and P_i will now enter the C.S.
6. After successfully entering in C.S, process P_i will set:
   flag[i] = false
7. The same procedure is executed by process P_j also.

Peterson's Solution preserves all three conditions:

- Mutual Exclusion is assured as only one process can access the critical section at any time.
- Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.
- Bounded Waiting is preserved as every process gets a fair chance.

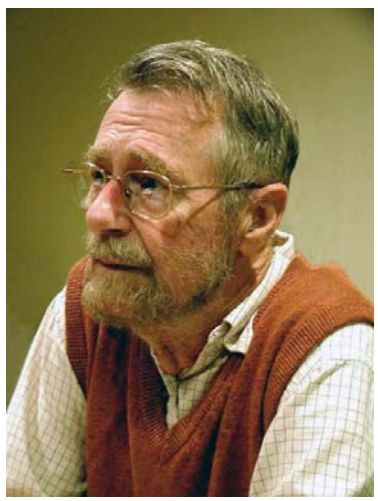
Disadvantages of Peterson's Solution

- It involves <u>busy waiting</u>. (In the Peterson's solution, the code statement- "while(flag[j] && turn == [j]);" is responsible for this. Busy waiting is <u>not favoured</u> because it wastes CPU cycles that could be used to perform other tasks.)
- It is limited to 2 processes.
- Peterson's solution cannot be used in modern CPU architectures.

2. Semaphores

- In multi-programming, if a lot of processes are using a common resource, they need to access the resource in such a way that they do not disturb or interfere with each other. So, semaphore is used to prevent this disturbance and interfere.
- Semaphore is simply a variable which is non-negative and shared between processes.
- This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.
- It is represented as 's' that apart from initialization, accessed only through two standard atomic operations**:
    - wait( ) ----------→ P
    - signal( ) ----------→ V

  *** An atomic operation is a single, indivisible operation that cannot be broken down into smaller operations.*
- Semaphore was proposed by **Edsger W. Dijkstra.**



Edsger W. Dijkstra (1930-2002)

Wait Operation (P operation)

The Wait Operation is used for deciding the condition for the process to enter the critical section for execution of process.

```
wait(S)
{
    while (S <=0);
    S=S-1;
}
```

Signal Operation (V Operation)

The Signal Semaphore Operation is used to update the value of Semaphore.

```
signal(S)
{
    S=S+1
}
```

Note: When one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

Working:

1. Let us consider two process P1, P2.
2. By default, value of semaphore, 'S' is 1.
3. When wait(s) function is called, the value of S is checked in while condition. If S is equal to zero or less than zero, then it will get stuck in while loop itself and it will not come out until the value of S becomes greater than zero.
4. If value of S is greater than zero, then the process will enter the critical section and after processing it will decrement the value of S by one.
5. P1 enters in its critical section then the value of semaphore S becomes 0.
6. Now if P2 wants to enter its critical section then it will wait until S > 0, this can only happen when P1 finishes its critical section and calls signal operation on semaphore S.

Types of Semaphores

1. **Binary Semaphore:** The value of a binary semaphore can range only between 0 and 1. On some systems, binary semaphores are known as **mutex locks**, as they are locks that provide mutual exclusion. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0.
2. **Counting Semaphore:** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Numerical Questions based on Semaphores:

1. A counting semaphore S is initialized to 10. Then, 6 P operations and 4 V operations are performed on S. What is the final value of S?

Ans: 8

2. A counting semaphore S is initialized to 7. Then, 20 P operations and 15 V operations are performed on S. What is the final value of S?

Ans: 2

3. A counting semaphore S is initialized to 8. Then, 12 P operations and 7 V operations are performed on S. What is the final value of S?

Ans: 3

4. A counting semaphore S is initialized to 8. Then, 3 wait operations and 4 signal operations are performed on S. What is the final value of S?

Ans: 9

Classical Problems of Synchronization

1. Producer-Consumer Problem (Bounded Buffer)

The bounded buffer problem, also known as the producer-consumer problem, is a classic synchronization problem in computer science and concurrent programming. It involves two types of processes, producers and consumers, that share a fixed-size buffer (or queue) to communicate and exchange data.

The key characteristics of the bounded buffer problem are as follows:

1. **Producers**: These are processes that generate data or items and want to add them to the buffer. They produce data faster than the consumers can consume it.
2. **Consumers**: These are processes that want to take items from the buffer and use them. They consume data slower than the producers can produce it.
3. **Buffer**: This is a finite-size data structure that acts as a shared resource between producers and consumers.

4. The producer tries to insert data into an empty slot of the buffer.
5. The consumer tries to remove data from a filled slot in the buffer.
6. The producer must not insert data when the buffer is full.
7. The consumer must not remove data when the buffer is empty.
8. The producer and consumer should not insert and remove data simultaneously.

Common solutions to the bounded buffer problem include:

1. **Semaphore-based Solution**: Using two semaphores, one for tracking the empty slots in the buffer and one for tracking the filled slots. Producers and consumers use these semaphores to coordinate access to the buffer.
2. **Mutex and Conditional Variable Solution**: Using a mutex (or a lock) to protect access to the buffer and conditional variables to signal when the buffer is not empty or not full. Producers and consumers use these constructs to synchronize their actions.

2. Readers Writers Problem

- In this problem there is a shared resource which should be accessed by multiple processes.
- There are two types of processes:

  - Readers – Reader are those processes/users which only read the data
  - Writers – Writers are those processes which also write, that is, they change the data.

- Any number of readers can read from the shared resource simultaneously,
- But only one writer can write to the shared resource.
- When a writer is writing data to the resource, no other process can access the resource.
- A writer cannot write to the resource if some readers accessing the resource at that time.

- Four possibility of reading and writing can arise:

Case	Process 1	Process 2	Allowed / Not Allowed
Case 1	Writing	Writing	Not Allowed
Case 2	Reading	Writing	Not Allowed
Case 3	Writing	Reading	Not Allowed
Case 4	Reading	Reading	Allowed

The solution of readers and writers can be implemented using binary semaphores.

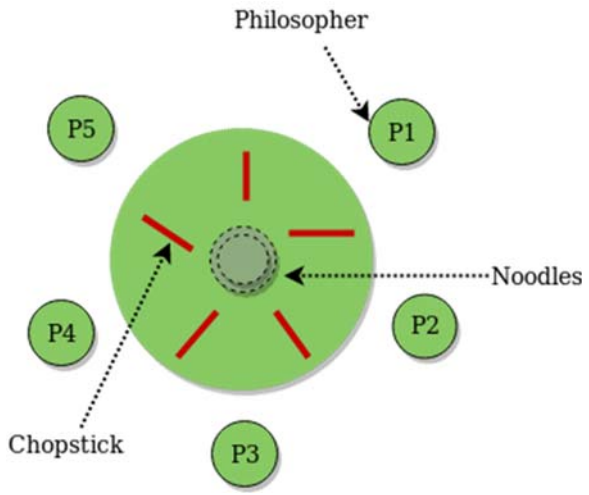
3. Dining Philosophers Problem

The Scenario

- Five silent philosophers sit at a round table with bowls of noodles.
- Forks are placed between each pair of adjacent philosophers.
- Eating is not limited by the amounts of spaghetti left so an infinite supply and an infinite demand are assumed.

The Rules

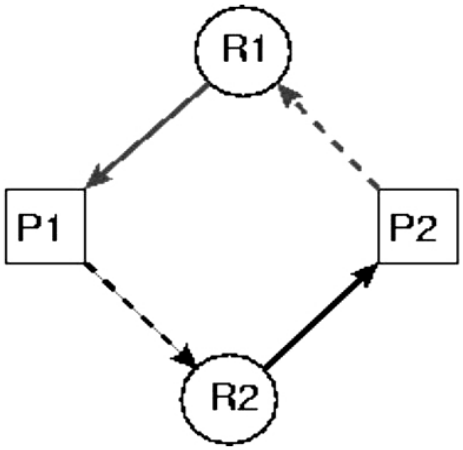
- Each philosopher must alternately think and eat.
- However, a philosopher can only eat spaghetti when they have both left and right forks.
- Each fork can be held by only one philosopher
- After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others.
- A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both forks.



The solution of the Dining Philosophers Problem

- We use a semaphore to represent a chopstick and this acts as a solution of the Dining Philosophers Problem.
- Wait and Signal operations will be used for the solution of the Dining Philosophers Problem

- For picking a chopstick wait operation can be executed while for releasing a chopstick signal semaphore can be executed.

DEADLOCK	**DEADLOCK**
The deadlock is the situation in which two computer processes share only one resource, so that no process can access the resource. That means the deadlock is the condition in which two or more processes only fight (fight) for one resource, due to which no resource can access it. For example, there are two friends, and both of them want to play computer games, because of which both fight. One has a remote control then the other has the cd of the games. Which makes none of the two friends play, but neither of them is ready to cooperate. This situation is called **deadlock**.	Deadlock वह स्थिति है जिसमें दो कंप्यूटर processes केवल एक resource को share (साझा) करते है जिसके कारण कोई भी प्रोसेस resource को access नही कर पाता है. अर्थात डेडलॉक वह कंडीशन है जिसमें दो या दो से अधिक processes केवल एक resource के लिए fight (लड़ाई) करते हैं जिसके कारण कोई भी resource को access नही कर पाता है. उदहारण के लिए माना दो दोस्त है और दोनों कंप्यूटर गेम्स खेलना चाहते हैं जिसके कारण दोनों लड़ते है. एक पास रिमोट कण्ट्रोल है तो दुसरे के पास गेम्स की cd है. जिसके कारण दोनों दोस्तों में से कोई भी नही खेल पाता है लेकिन दोनों में से कोई भी cooperate करने के लिए तैयार नही है. यह स्थिति ही **डेडलॉक** कहलाती है.
	
According to the picture we have two processes p1 and p2 and two resources are r1 and r2. In this there is resource r1 which is allocated to process p1, and resource r2 which is allocated to process p2. But the process requires R2 to complete the process, so he requests for R2, but R2 is already allocated to P2. Similarly process p2 requires r1 to complete its execution, but r1 is already allocated to p1.	चित्र के अनुसार हमारे पास दो processes p1 तथा p2 हैं तथा दो resources r1 तथा r2 हैं. इसमें resource r1 जो है वह process p1 के साथ allocate है, और resource r2 जो है वह process p2 के साथ allocate है. परन्तु प्रक्रिया को पूरी होने के लिए resource r2 की आवश्यकता है इसलिए वह r2 के लिए request करता है, परन्तु r2 पहले से ही p2 के साथ allocate है. इसी तरह process p2 को अपनी execution को पूरी करने के लिए r1 की आवश्यकता है, परन्तु r1 तो पहले

Both p1 and p2 are always waiting for each other's resources. This process is called DEADLOCK.	से ही p1 के साथ allocate है. दोनों p1 तथा p2 एक दुसरे के resources के लिए हमेशा wait (इन्तजार) करते रहते है. यही प्रक्रिया ही DEADLOCK कहलाती है.
CONDITIONS OF DEADLOCK The four conditions of the deadlock are as follows: - (1) **Mutual exclusion**: In this situation, there is one such resource that can not be shared between processes. (2) **Hold & wait**: In this, processes hold a resource while waiting for another resource. (3) **No-preemption**: The resource that has been allocated for a process can not be forcibly assigned to any other process. (4) **Circular wait**: In this situation, each process waits for a resource which has another process held.	**CONDITIONS OF DEADLOCK (डेडलॉक की शर्तें)** Deadlock की चार कंडीशन होती है जो निम्नलिखित है:- (1) **mutual exclusion**: इस स्थिति में, कोई न कोई एक ऐसा resource होता है जिसे processes के मध्य share (साझा) ना किया जा सकें. (2) **Hold & wait**: इसमें, processes एक resource को होल्ड किये रहते है जबकि दुसरे resource के लिए wait (इन्तजार) करते है. (3) **No-preemption**: वह resource जो किसी process के लिए allocate हो चूका हो, उसे जबरन किसी दुसरे process को allocate नही किया जा सकता है. (4) **Circular wait:** इस स्थिति में, प्रत्येक process एक resource के लिए wait करता है जिसे दुसरा process held किये हुए होता है.

DEADLOCK PREVENTION	DEADLOCK PREVENTION (डेडलॉक रोकथाम)
We can prevent the condition of deadlock to some degree. Most of us can not prevent them because our resources are limited. Some resources are sharable and some are non-sharable. If we can prevent one of these 4 conditions then the deadlock will not be in the system.	Deadlock होने की जो conditions होती है उनको हम कुछ हद तक prevent कर सकते है. ज्यादातर हम उनको prevent नहीं कर सकते है क्योंकि हमारे resources limited होते है. कुछ रिसोर्स sharable होते है तथा कुछ non sharable होते है. यदि हम इन 4 conditions में से किसी एक को prevent कर सकते हो तो सिस्टम में Deadlock नहीं होगा.
(1) Mutual Exclusion: If we can share resources, we can prevent mutual exclusion. If we have non-sharable resources that can not be shared, then in that situation we can not prevent mutual exclusion from occurring. For example, the Printer is a non-sharable resource. In this situation, mutual exclusion can not be prevented. But if you have a sharable resource, you can stop mutual exclusion by sharing it. Like: We can share memory.	**(1) Mutual Exclusion:** अगर हम resources को share कर सकते है तो हम mutual exclusion को prevent कर सकते है. अगर हमारे पास non-sharable resources होते है जिन्हें share नहीं किया जा सकता है तो उस स्थिति में हम mutual exclusion को होने से नहीं रोक सकते. उदाहरण के लिए Printer एक non-sharable resource है. इस स्थिति में mutual exclusion को prevent नहीं कर सकते. लेकिन अगर आपके पास sharable resource है तो आप उसे share करके mutual exclusion को रोक सकते है. जैसे: - मैमोरी को हम share कर सकते है.
(2) Hold & Wait: If resources are processed before its execution, then we can prevent hold & wait condition. But its loss is that it reduces the utilization of the system.	**(2) Hold & Wait:** अगर resources को processes को, इसके execution से पहले ही, allocate कर दिया जाएँ तो हम hold & wait कंडीशन को होने से रोक सकते है. परन्तु इसकी हानि यह है कि इससे सिस्टम की utilization कम हो जाती है.
For example, a process needs a printer afterwards. And we already allocate the printer to it. So till the execution is over, this printer will remain blocked and the other process can not use it in its spare time. And its second loss is that it leads to starvation.	उदाहरण के लिए किसी प्रोसेस को प्रिंटर की बाद में जरुरत है. और हम उसे पहले ही प्रिंटर allocate कर देते है. तो जब तक इसकी execution समाप्त नहीं हो जाती तब तक यह प्रिंटर block रहेगा और दूसरे प्रोसेस इसका प्रयोग इसके खाली समय में नहीं कर सकते. तथा इसकी दूसरी हानि यह है कि इसमें starvation होता है.

(3) No Preemption: If a process has held resources and is waiting for other resources, then all resources are released from that process, so that other processes can complete their execution. But some resources like: - printer, tape drivers can not be preempted.	**(3) No Preemption:** अगर किसी प्रोसेस ने resources को hold किया है और वह दूसरे resources का wait कर रहा है तो उस प्रोसेस से सभी resources को release कर दिया जाता है जिससे दूसरे प्रोसेस अपने execution को पूरा कर सकें. लेकिन कुछ resources जैसे: - printer, tape drivers को preempt नहीं किया जा सकता है.
(4) Circular Wait: To prevent circular wait, the resources can only be assigned resources in incremental order. That is, there is a sequence of resources, according to which processes are allocated resources.	**(4) Circular Wait:** circular wait को रोकने के लिए, processes को resources केवल बढ़ते हुए क्रम में ही allocate कर सकते है. अर्थात् resources का एक क्रम होता है जिसके अनुसार ही processes को resources allocate किया जाता है.
For example: If the processing P2 is allocated to the R6 resource, the resources for R2, R4 or below will not be allocated to P2 the next time. Only resources from R6 will be allocated to P2. But its disadvantage is that resource usage is rarely reduced by this.	**उदाहरण के लिए:** यदि प्रोसेस P2 को R6 रिसोर्स allocate किया गया है तो अगली बार P2 को R5, R4 या इससे नीचे के रिसोर्स allocate नहीं किये जायेंगे. केवल R6 से उपर के रिसोर्स ही P2 को allocate किये जायेंगे. परन्तु इसकी हानि यह है कि इससे रिसोर्स utilization बहुत ही कम हो जाता है.
Effects of Deadlock prevention: The effects of deadlock prevention are the following. • It reduces the utilization of the device very much. • It reduces the effect of the system throughput.	Deadlock prevention **के प्रभाव**: Deadlock prevention से होने वाले प्रभाव निम्नलिखित है. • इससे डिवाइस की utilization बहुत कम हो जाती है. • इससे सिस्टम की प्रभाव क्षमता (throughput) बहुत कम हो जाता है.

Deadlock Avoidance

Deadlock prevention algorithms are not good, thereby reducing resource utilization and system throughput. But we can avoid deadlock.

Deadlock avoidance algorithm ensures that the processes will never go into unsafe state.

There are two states:

• **Safe State:** safe state is the state in which we execute processes in a safe sequence. In this process exist in the safe sequence in such a way that the first process has enough resources to execute. And when the execution is over, after the release of the resources of this process, there are enough resources for the execution of the next process.

• **Unsafe state:** If processes are not in a safe sequence then it is in the state of the unsafe state and can be deadlock in the unsafe state.

For example, we have 11 tapes drive.

processes	maximum required	current hold
P0	9	5
P1	4	2
P2	8	2

The current hold of our tape drivers:

5 + 2 + 2 = 9
We have the remaining tape drive: 11-9 = 2.
First of all, if we look at P0 then we can not execute P0 first because it requires 4 and a tape

Deadlock Avoidance

Deadlock prevention की algorithms अच्छी नहीं है जिससे resource utilization तथा system throughput कम हो जाता है. लेकिन हम डेडलॉक को avoid कर सकते है.

Deadlock avoidance अल्गोरिथम यह सुनिश्चित करती है कि processes कभी भी unsafe state में नहीं जाएगी.

इसमें दो states होती है:-

- **Safe State:** safe state वह state होती है जिसमें हम processes को एक safe sequence में execute करते है. इसमें प्रोसेस safe sequence में इस प्रकार exist होती है कि पहली प्रोसेस के पास execute होने के लिए पर्याप्त resources हों. और जब execution समाप्त हो जाएँ तो इस प्रोसेस के resources release होने के बाद अगली प्रोसेस के execution के लिए भी पर्याप्त resources हों.

- **Unsafe state:** अगर processes एक safe sequence में नहीं है तो वह unsafe state में होती है और unsafe state में डेडलॉक हो सकता है.

उदाहरण के लिए माना हमारे पास 11 tapes drive है.

processes	maximum required	current hold
P0	9	5
P1	4	2
P2	8	2

हमारी जो tape drivers की current hold है- 5+2+2=9
हमारे पास बची हुई हुए tape drive है: 11-9=2.
सबसे पहले हम P0 को देखते है तो हम P0 को पहले execute नहीं कर सकते है क्योंकि उसे 4 और tape drive की जरुरत है परन्तु हमारे पास 2 ही है.

drive but we have only 2. Now let's see P1. P1 requires 2, and then we execute it. Now after the completion of its execution it will release 4 tape drives. Now we will execute P0 because it requires 4 tape drives, after its execution it will release 9 tape drive. And then we will execute P2, P2 needs 6 and we have 9 tape drive then P2 will also execute. Then our safe sequence will be: - **P1, P0, P2**	अब P1 को देखते है. P1 को 2 की जरुरत है तो हम इसे execute कर लेते है. अब इसके execution के पूरा होने के बाद यह 4 tape drive को release करेगा. अब हम P0 को execute करेंगे क्योंकि इसे 4 tape drive की जरुरत है तो इसके execution के बाद यह 9 tape drive को release करेगा. और फिर हम P2 को execute करेंगे, P2 को 6 की जरूरत है और हमारे पास 9 tape drive है तो P2 भी execute हो जायेगा. तो हमारी safe sequence होगी: - **P1, P0, P2**

Banker's Algorithm	**Banker's Algorithm**
The Banker's algorithm was developed by **Edger Dijkstra**. Banker's algorithm is sometimes called detection algorithm. Banker's algorithm is a resource allocation and deadlock avoidance algorithm.	Banker's algorithm को Edger Dijkstra ने विकसित किया था. Banker's algorithm को कभी कभी detection algorithm भी कहते है. Banker's algorithm एक resource allocation तथा deadlock avoidance अल्गोरिथम है.
The name used to be called banker's algorithm because it can be used to determine whether the loan can be used in the banking system.	इसका नाम banker's algorithm इसलिए पड़ा क्योंकि इसका प्रयोग बैंकिंग सिस्टम में loan को दिया जा सकता है या नहीं, यह निर्धारित करने के लिए किया जाता है.
The security of allocation of the maximum possible resources already prescribed in it is tested. And after that, it creates the resource allocation state, from which it decides whether the allocation can be done or not.	इसमें पहले से निर्धारित अधिकतम संभावित resources के allocation की safety को टेस्ट किया जाता है. तथा इसके बाद यह resource allocation state को बनाता है, जिससे वह यह निर्णय लेता है कि allocation किया जा सकता है या नही.
Resource allocation state is defined as the available number of resources and the basis of allocated resources and processes.	Resource allocation state वह है resources की उपलब्ध संख्या तथा allocated resources और processes की जरुरत के आधार पर निर्धारित होता है.
When the process requests for an available resource, then the system decides whether the system will be in safe state from the allocation of these resources.	जब process किसी उपलब्ध resource के लिए request करता है तब सिस्टम निर्णय लेता है इस resources के allocation से सिस्टम safe state में रहेगा या नहीं.

Example:

सिस्टम में 5 processes (P0, P1, P2, P3, P4) है.

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

Q.1. इसकी need matrix क्या है.

Q.2. अगर यह safe state में है तो इसकी safety sequence क्या है?

Ans. Need matrix का फार्मूला है:

NEED MATRIX= MAX – AVAILABLE

NEED MATRIX

A	B	C
7-0	5-1	3-0
3-2	2-0	2-0
9-3	0-0	2-2
2-2	2-1	2-1
4-0	3-0	3-2

NEED MATRIX

A	B	C
7	4	3
1	2	2
6	0	0
0	1	1
4	3	1

Safety Algorithm: **(इसका सूत्र है)**

If
need <=available
then
execute process
new available = available + allocation
else
do not execute go ahead

यदि need जो है वह available से छोटी या बराबर है तो प्रोसेस execute होगा और नया available बनेगा. नहीं तो आगे की प्रोसेस को देखेंगे.

पद 1:-

प्रोसेस P0 के लिए

need = (7,4,3)
if need <= available
if (7,4,3) <= (3,3,2)

यह execute नहीं होगा.

पद 2: -

प्रोसेस P1 के लिए

need = (1,2,2)

if need <= available
if (1,2,2,) <= (3,2,2)
P1 execute होगा.

new available = available + allocation
= (3,3,2) + (2,0,0)
= (5,3,2)

पद 3: -

प्रोसेस P2 के लिए

need = (6,0,0)
if (6,0,0) <= (5,3,2)

P2 execute नहीं होगा.

पद 4: -

प्रोसेस P3 के लिए

need = (0,1,1)
if need <= available
(0,1,1) <= (5,3,2)

P3 को execute करेंगे.
new available = available + allocation
= (5,3,2) + (2,1,1)
= (7,4,3)

पद 5: -

प्रोसेस P4 के लिए

need = (4,3,1)
if need <= available
(4,3,1) <= (7, 4, 3)
P4 को execute करेंगे.

new available = available + allocation
= (7,4,3) + (0,0,2)
= (7,4,5)

पद 6:-
P0 पहले execute नहीं हुआ था तो उसे देखते है-

P0 के लिए

need = (7,4,3)
if (7,4,3) <= (7,4,5)
P0 को execute करेंगे.

new available = available + allocation
= (7,4,5) + (0,1,0)
= (7,5,5)

पद 7:-
P2 भी execute नहीं हुआ था उसे देखते है.

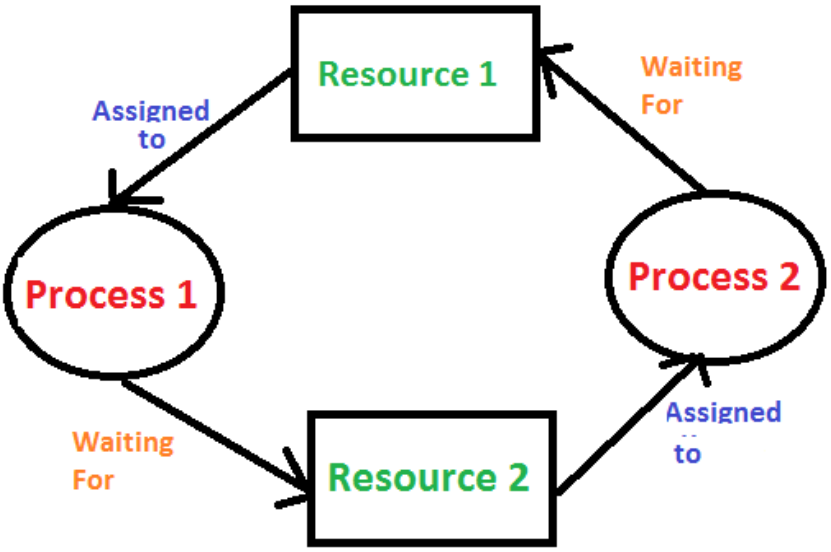
प्रोसेस P2 के लिए

need = (6,0,0)
if need <= available
(6,0,0) <= (7,5,5)

P2 execute होगा.
new available = available + allocation
= (7,5,5) + (3,0,2)
= (7,5,5) + (3,0,2)
= (10,5,7)

तो हमने अब सभी प्रोसेस को execute कर लिया तो अब हमारा safe sequence होगा-

Safe sequence: (P1, P3, P4, P0, P2)

Deadlock Detection and Recovery In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques. The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph. **Deadlock Detection** **1. If resources have single instance**: In this case for Deadlock detection we can run an algorithm to check for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock.	**डेडलॉक डिटेक्शन और रिकवरी** इस दृष्टिकोण में, OS डेडलॉक से बचने या रोकने के लिए किसी भी तंत्र को लागू नहीं करता है। इसलिए यह प्रणाली मानती है कि डेडलॉक निश्चित रूप से होगा। डेडलॉक्स से छुटकारा पाने के लिए, O.S समय-समय पर किसी भी डेडलॉक के लिए सिस्टम की जांच करता है। अगर, यह किसी भी डेडलॉक को पाता है तो OS कुछ रिकवरी तकनीकों का उपयोग कर सिस्टम को पुनर्प्राप्त करेगा। OS का मुख्य कार्य डेडलॉक्स का पता लगा रहा है। OS संसाधन आवंटन ग्राफ की मदद से डेडलॉक्स का पता लगा सकता है। **डेडलॉक डिटेक्शन** **1. यदि संसाधनों की संख्या एक ही हो:** इस मामले में डेडलॉक पहचान के लिए हम संसाधन आवंटन ग्राफ में चक्र की जांच के लिए एक एल्गोरिदम चला सकते हैं। यदि ग्राफ में चक्र की उपस्थिति पायी जाती है तो डेडलॉक का होना निश्चित है।
	
In the above diagram, resource 1 and resource 2 have single instances. There is a cycle R1–>P1–>R2–>P2. So Deadlock is Confirmed.	उपरोक्त आरेख में, संसाधन 1 और संसाधन 2 में एकल उदाहरण हैं। एक चक्र R1–>P1–>R2–>P2 है। तो डेडलॉक का होना निश्चित है।

2. If there are multiple instances of resources: Detection of cycle is necessary but not sufficient condition for deadlock detection, in this case system may or may not be in deadlock and varies according to different situations.	**2. यदि संसाधनों की संख्या एक से अधिक हैं:** ग्राफ में चक्र का पता लगाना आवश्यक है लेकिन डेडलॉक पहचान के लिए पर्याप्त शर्त नहीं है, इस मामले में सिस्टम डेडलॉक में हो भी सकता है या नहीं भी हो सकता है और विभिन्न परिस्थितियों के अनुसार यह भिन्न होता है।
Deadlock Recovery	**डेडलॉक रिकवरी**
1. Preempt the resource We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.	**1. संसाधन को मुक्त करें** हम संसाधन (प्रक्रिया) के मालिक से संसाधनों में से एक को इस उम्मीद के साथ छीन कर किसी अन्य प्रक्रिया को दे सकते हैं कि वह निष्पादन को पूरा करेगा और जल्द ही इस संसाधन को system में फ्री कर देगा । खैर, एक संसाधन चुनना जो छीन लिया जाएगा थोड़ा मुश्किल काम है।
2. Rollback to a safe state System passes through various states to get into the deadlock state. The operating systems can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state. The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.	**2. एक सुरक्षित स्थिति में रोलबैक** सिस्टम डेडलॉक स्थिति में जाने के लिए विभिन्न स्तिथियो के माध्यम से गुजरता है। ऑपरेटिंग सिस्टम सिस्टम को पिछले सुरक्षित स्थिति में रोलबैक कर सकते हैं। इस उद्देश्य के लिए, OS को हर स्थिति में चेक पॉइंटिंग लागू करने की आवश्यकता है। जिस पल, हम डेडलॉक में आते हैं, हम पिछले आवंटन में आने के लिए सभी आवंटन को रोलबैक करेंगे।
3. Kill a process Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.	**3. एक प्रक्रिया को मार(kill) डालो** एक प्रक्रिया को मारना हमारी समस्या को हल कर सकता है लेकिन बड़ी चिंता यह तय करना है कि कौन सी प्रक्रिया को मारना है। आम तौर पर, ऑपरेटिंग सिस्टम ऐसी प्रक्रिया को मारता है जिसने अब तक कम से कम काम किया है।
4. Kill all process This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to	**4. सभी प्रक्रियाओं को मार(kill) डालो** यह कोई सुझाव नहीं है लेकिन अगर समस्या बहुत गंभीर हो जाती है तो इसे कार्यान्वित किया जा सकता है। सभी प्रक्रियाओं को मारने से सिस्टम में अक्षमता हो जाएगी क्योंकि सभी प्रक्रियाएं शुरू होने से फिर से

inefficiency in the system because all the processes will execute again from starting.	निष्पादित होंगी।