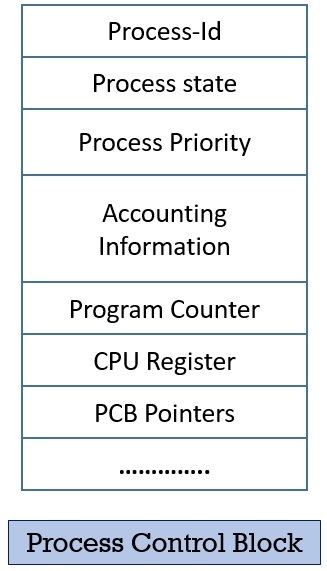
**Operating Systems**

**Process Control Block(PCB)**

For each process, the operating system maintains the **data structure**, which keeps the complete information about

that process. This record or data structure is called **Process Control Block (PCB).**

Whenever a user creates a process, the operating system creates the corresponding PCB for that process. These PCBs of the processes are stored in the memory that is reserved for the operating system.



**Process-id**

Whenever a new process is created by the user, the operating system allots a number to that process. This number becomes the unique

identification of that process and it also helps in distinguishing that process from all other processes existing in the system. This number is

also called as **process-id** of the process.

As we know, the operating system sets a limit on the maximum number of the processes it can deal with at a time. So, let us suppose that there

are **n** number of the processes in the system. Now, the process-id will take on the values between 0 to n-1.

The operating system will allocate the value **0** to the first process that arrives in the system, number **1** to the next process and continues till **n-1**.

At this point when the n-1 value is allocated to some process, and a new process arrives, the operating system wraps around and allocates

value **0** to the newly arrived process. Considering that the process with process-id **0**, would have terminated.

Process-ids are not necessarily allocated in the ascending fashion.

**Process State**

A process in its lifetime undergoes different states. Like, a process may be in **waiting state, running state, ready state, NEW state, TERMINATED state**, and so on.

The PCBs field, **process state** holds the current state of the respective process. For example, if the process is currently executing. So, the

process state will hold the **running** state for that process.

**Process Priority**

The priority of the process is a **numeric** value, lesser the value, greater is the priority of that process. The priority of the process can be

assigned externally by the user or by the operating system itself.

The process is assigned the priority at the time of its creation. The priority of the process may get changed over its lifetime depending on the

various parameter. The parameters for changing the priority of the process can be the age of that process, the resources it consumed and so on.

**Process Accounting Information**

This field of PCB gives the account/description of the resources used by that process. Like, the amount of CPU time, real-time used, connect time.

**Program Counter**

The program counter is the pointer to an instruction in the program or code that is to be executed next. This field contains the address of

the instruction that will be executed next in the process.

**List of Open Files**

It contains the information of all the files that is required by the program during its execution.

This information is also useful for the operating system. Because it helps the operating system to close the all opened files which are not closed explicitly at the termination of the program.

**Process I/O status Information**

Sometimes the process executing in the system require I/o devices. So, this field of PCB contains the list of all the I/O devices allocated to the

process during its execution.

**CPU Registers**

Whenever an interrupt occurs and there is a context switch between the processes, the temporary information is stored in the registers.

So, that when the process resumes the execution from where it leaves. CPU registers are used to hold those temporary values or information.

**PCB Pointer**

In this field, the pointer has an address of the next PCB, whose process state is **ready**.

**Event Information**

This field contains the information of the event. If the event occurred match with this field the process changes its state from

blocked to ready.

So these are the fields of PCBs which contains much information that is associated with the specific process.

**Threads**

If you start Ms Word, it is a process. In Ms Word, you type some thing and it gets automatically saved. Now,

you would have observed editing and saving happens in parallel. These are threads.

If you start Paint, it is a process. In Paint it draw pictures by reading mouse movement.

The program must give its full attention to the mouse input and draw at the same time. To do this two or more threads of the program

will execute at the same time.

.

The thread is something which handles multiple actions. I give a real time example. Any chatting application like **WhatsApp**,

if you create a group in whatsapp and started chatting with multiple people. The group creates a process and it **generates**

**threads to handle messages which comes from multiple people at a time**. At a time multiple action can’t handle by the core

process that is the reason it creates threads.

[Advanced example] A JVM (Java Virtual Machine) runs in a single process and threads in a JVM share the heap belonging to that process.

That is why several threads may access the same object. Threads share the heap and have their own stack space. This is how one

thread’s invocation of a method and its local variables are kept thread safe from other threads. But the heap is not thread-safe

and must be synchronized for thread safety

A thread is a lightweight process that can be managed independently by a scheduler.

It improves the application performance using parallelism. A thread shares information

like data segment, code segment, files etc. with its peer threads while it contains its own

registers, stack, counter etc.

**Difference between Process and Thread:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **PROCESS** | **THREAD** |
| 1. | Process means any program is in execution. | Thread means segment of a process. |
| 2. | Process takes more time to terminate. | Thread takes less time to terminate. |
| 3. | It takes more time for creation. | It takes less time for creation. |
| 4. | It also takes more time for context switching. | It takes less time for context switching. |
| 5. | Process is less efficient in term of communication. | Thread is more efficient in term of communication. |
| 6. | Process consume more resources. | Thread consume less resources. |
| 7. | Process is isolated. | Threads share memory. |
| 8. | Process is called heavy weight process. | Thread is called light weight process. |
| 9. | Process switching uses interface in operating system. | Thread switching does not require to call a operating system and cause an interrupt to the kernel. |
| 10. | If one server process is blocked no other server process can execute until the first process unblocked. | Second thread in the same task could run, while one server thread is blocked. |
| 11. | Process has its own Process Control Block, Stack and Address Space. | Thread has Parents’ PCB, its own Thread Control Block and Stack and common Address space. |

**Solution to the Critical Section Problem**

The critical section problem needs a solution to synchronize the different processes. The solution to the critical section problem must satisfy the following conditions −

* **Mutual Exclusion**

Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

* **Progress**

Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

* **Bounded Waiting**

Bounded waiting means that each process must have a limited waiting time. Itt should not wait endlessly to access the critical section.

**Critical Region : Mutual exclusion and busy wait Techniques**

**Proposal 1 -Disabling Interrupts (Hardware Solution)**

Each process disables all interrupts just after entering in its critical section and re-enable all interrupts just before leaving critical section. With interrupts turned off the CPU could not be switched to other process. Hence, no other process will enter its critical and mutual exclusion achieved.

**Conclusion**Disabling interrupts is sometimes a useful technique within the kernel of an operating system, but it is not appropriate as a general mutual exclusion mechanism for users process. The reason is that it is unwise to give user process the power to turn off interrupts.

**Proposal 2 - Lock Variable (Software Solution)**

In this solution, we consider a single, shared, (lock) variable, initially 0. When a process wants to enter in its critical section, it first test the lock. If lock is 0, the process first sets it to 1 and then enters the critical section. If the lock is already 1, the process just waits until (lock) variable becomes 0. Thus, a 0 means that no process in its critical section, and 1 means hold your horses - some process is in its critical section.

**Conclusion**The flaw in this proposal can be best explained by example. Suppose process A sees that the lock is 0. Before it can set the lock to 1 another process B is scheduled, runs, and sets the lock to 1. When the process A runs again, it will also set the lock to 1, and two processes will be in their critical section simultaneously.

**Proposal 3 - Strict Alteration(Peterson)**

**For Process Pj**

1. **Non - CS**
2. **while (turn ! = i);**
3. **Critical Section**
4. **turn = j;**
5. **Non - CS**

**For Process Pi**

1. **Non - CS**
2. **while (turn ! = j);**
3. **Critical Section**
4. **turn = i ;**
5. **Non - CS**

**The actual problem of the lock variable approach was the fact that the process was entering in the critical section only when the lock variable is 1. More than one process could see the lock variable as 1 at the same time hence the mutual exclusion was not guaranteed there.**

**This problem is addressed in the turn variable approach. Now, A process can enter in the critical section only in the case when the value of the turn variable equal to the PID of the process.**

**There are only two values possible for turn variable, i or j. if its value is not i then it will definitely be j or vice versa.**

**In the entry section, in general, the process Pi will not enter in the critical section until its value is j or the process Pj will not enter in the critical section until its value is i.**

**Initially, two processes Pi and Pj are available and want to execute into critical section.**

In this proposed solution, the integer variable 'turn' keeps track of whose turn is to enter the critical section. Initially, process A inspect turn, finds it to be 0, and enters in its critical section. Process B also finds it to be 0 and sits in a loop continually testing 'turn' to see when it becomes 1.Continuously testing a variable waiting for some value to appear is called the *Busy-Waiting*.

**Conclusion**Taking turns is not a good idea when one of the processes is much slower than the other. Suppose process 0 finishes its critical section quickly, so both processes are now in their noncritical section. This situation violates above mentioned condition 3.

**Using Systems calls 'sleep' and 'wakeup'**

Basically, what above mentioned solution do is this: when a processes wants to enter in its critical section , it checks to see if then entry is allowed. If it is not, the process goes into tight loop and waits (i.e., start busy waiting) until it is allowed to enter. This approach waste CPU-time.

Now look at some interprocess communication primitives is the pair of steep-wakeup.

* Sleep
  + It is a system call that causes the caller to block, that is, be suspended until some other process wakes it up.
* Wakeup
  + It is a system call that wakes up the process.

Both 'sleep' and 'wakeup' system calls have one parameter that represents a memory address used to match up 'sleeps' and 'wakeups' .

**The Bounded Buffer Producers and Consumers**

The bounded buffer producers and consumers assumes that there is a fixed buffer size i.e., a finite numbers of slots are available.

**Statement**  
To suspend the producers when the buffer is full, to suspend the consumers when the buffer is empty, and to make sure that only one process at a time manipulates a buffer so there are no race conditions or lost updates.

As an example how sleep-wakeup system calls are used, consider the producer-consumer problem also known as bounded buffer problem.

Two processes share a common, fixed-size (bounded) buffer. The producer puts information into the buffer and the consumer takes information out.

Trouble arises when

1. The producer wants to put a new data in the buffer, but buffer is already full.  
   Solution: Producer goes to sleep and to be awakened when the consumer has removed data.
2. The consumer wants to remove data the buffer but buffer is already empty.  
   Solution: Consumer goes to sleep until the producer puts some data in buffer and wakes consumer up.

Problem Statement – We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.

To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

Initialization of semaphores –

mutex = 1

Full = 0 // Initially, all slots are empty. Thus full slots are 0

Empty = n // All slots are empty initially

Solution for Producer –

do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)

When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed and consumer can access the buffer.

Solution for Consumer –

do{

wait(full);

wait(mutex);

// consume item from buffer

signal(mutex);

signal(empty);

}while(true)

As the consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producer can access the buffer now.

**Conclusion**This approaches also leads to same race conditions we have seen in earlier approaches. Race condition can occur due to the fact that access to 'count' is unconstrained. The essence of the problem is that a wakeup call, sent to a process that is not sleeping, is lost.

**Binary Semaphore**

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows −

* **Wait**

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

wait(S)

{

   while (S<=0);

   S--;

}

* **Signal**

The signal operation increments the value of its argument S.

signal(S)

{

   S++;

}

**Bounded Buffer (Producer and Consumer Problem)**

**What is CPU Scheduling?**

**CPU Scheduling** is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution.

**Preemptive Scheduling**

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

**Non-Preemptive Scheduling**

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

**When scheduling is Preemptive or Non-Preemptive?**

To determine if scheduling is preemptive or non-preemptive, consider these four parameters:

1. A process switches from the running to the waiting state.
2. Specific process switches from the running state to the ready state.
3. Specific process switches from the waiting state to the ready state.
4. Process finished its execution and terminated.

**Only conditions 1 and 4 apply, the scheduling is called non- preemptive.**

**All other scheduling are preemptive.**

**Maximize:**

**CPU utilization:**CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible. It can range from 0 to 100 percent. However, for the RTOS, it can be range from 40 percent for low-level and 90 percent for the high-level system.

**Throughput:**The number of processes that finish their execution per unit time is known Throughput. So, when the CPU is busy executing the process, at that time, work is being done, and the work completed per unit time is called Throughput.

**Minimize:**

**Waiting time:**Waiting time is an amount that specific process needs to wait in the ready queue.

**Response time:**It is an amount to time in which the request was submitted until the first response is produced.

**Turnaround Time:**Turnaround time is an amount of time to execute a specific process. It is the calculation of the total time spent waiting to get into the memory, waiting in the queue and, executing on the CPU. The period between the time of process submission to the completion time is the turnaround time.

**Interval Timer**

Timer interruption is a method that is closely related to preemption. When a certain process gets the CPU allocation, a timer may be set to a specified interval. Both timer interruption and preemption force a process to return the CPU before its CPU burst is complete.

Most of the multi-programmed operating system uses some form of a timer to prevent a process from tying up the system forever.

**What is Dispatcher?**

It is a module that provides control of the CPU to the process. The Dispatcher should be fast so that it can run on every context switch. Dispatch latency is the amount of time needed by the CPU scheduler to stop one process and start another.

Functions performed by Dispatcher:

* Context Switching
* Switching to user mode
* Moving to the correct location in the newly loaded program.

**Types of CPU scheduling Algorithm**

There are mainly six types of process scheduling algorithms

1. First Come First Serve (FCFS)
2. Shortest-Job-First (SJF) Scheduling
3. Shortest Remaining Time
4. Priority Scheduling
5. Round Robin Scheduling
6. Multilevel Queue Scheduling

**First Come First Serve**

First Come First Serve is the full form of FCFS. It is the easiest and most simple CPU scheduling algorithm. In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

As the process enters the ready queue, its PCB (Process Control Block) is linked with the tail of the queue. So, when CPU becomes free, it should be assigned to the process at the beginning of the queue.

**Characteristics of FCFS method:**

* It offers non-preemptive and pre-emptive scheduling algorithm.
* Jobs are always executed on a first-come, first-serve basis
* It is easy to implement and use.
* However, this method is poor in performance, and the general wait time is quite high.

**Shortest Remaining Time**

The full form of SRT is Shortest remaining time. It is also known as SJF preemptive scheduling. In this method, the process will be allocated to the task, which is closest to its completion. This method prevents a newer ready state process from holding the completion of an older process.

**Characteristics of SRT scheduling method:**

* This method is mostly applied in batch environments where short jobs are required to be given preference.
* This is not an ideal method to implement it in a shared system where the required CPU time is unknown.
* Associate with each process as the length of its next CPU burst. So that operating system uses these lengths, which helps to schedule the process with the shortest possible time.

**Priority Based Scheduling**

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.

Priority scheduling also helps OS to involve priority assignments. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority can be decided based on memory requirements, time requirements, etc.

**Round-Robin Scheduling**

Round robin is the oldest, simplest scheduling algorithm. The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turn. It is mostly used for scheduling algorithms in multitasking. This algorithm method helps for starvation free execution of processes.

**Characteristics of Round-Robin Scheduling**

* Round robin is a hybrid model which is clock-driven
* Time slice should be minimum, which is assigned for a specific task to be processed. However, it may vary for different processes.
* It is a real time system which responds to the event within a specific time limit.

**Shortest Job First**

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.

**Characteristics of SJF Scheduling**

* It is associated with each job as a unit of time to complete.
* In this method, when the CPU is available, the next process or job with the shortest completion time will be executed first.
* It is Implemented with non-preemptive policy.
* This algorithm method is useful for batch-type processing, where waiting for jobs to complete is not critical.
* It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

**Multiple-Level Queues Scheduling**

This algorithm separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property of the process, like the process priority, size of the memory, etc.

However, this is not an independent scheduling OS algorithm as it needs to use other types of algorithms in order to schedule the jobs.

**Characteristic of Multiple-Level Queues Scheduling:**

* Multiple queues should be maintained for processes with some characteristics.
* Every queue may have its separate scheduling algorithms.
* Priorities are given for each queue.

**The Purpose of a Scheduling algorithm**

Here are the reasons for using a scheduling algorithm:

* The CPU uses scheduling to improve its efficiency.
* It helps you to allocate resources among competing processes.
* The maximum utilization of CPU can be obtained with multi-programming.
* The processes which are to be executed are in ready queue.

**Summary:**

* CPU scheduling is a process of determining which process will own CPU for execution while another process is on hold.
* In Preemptive Scheduling, the tasks are mostly assigned with their priorities.
* In the Non-preemptive scheduling method, the CPU has been allocated to a specific process.
* Burst time is a time required for the process to complete execution. It is also called running time.
* CPU utilization is the main task in which the operating system needs to make sure that CPU remains as busy as possible
* The number of processes that finish their execution per unit time is known Throughput.
* Waiting time is an amount that specific process needs to wait in the ready queue.
* It is an amount to time in which the request was submitted until the first response is produced.
* Turnaround time is an amount of time to execute a specific process.
* Timer interruption is a method that is closely related to preemption,
* A dispatcher is a module that provides control of the CPU to the process.
* Six types of process scheduling algorithms are:
* First Come First Serve (FCFS), 2) Shortest-Job-First (SJF) Scheduling 3) Shortest Remaining Time 4) Priority Scheduling 5) Round Robin Scheduling 6) Multilevel Queue Scheduling
* In the First Come First Serve method, the process which requests the CPU gets the CPU allocation first.
* In the Shortest Remaining time, the process will be allocated to the task, which is closest to its completion.
* In, Priority Scheduling the scheduler selects the tasks to work as per the priority.
* In, this Round robin scheduling works on principle, where each person gets an equal share of something in turn
* In Shortest job first the shortest execution time should be selected for execution next
* In Multilevel scheduling, method separates the ready queue into various separate queues. In this method, processes are assigned to a queue based on a specific property
* The CPU uses scheduling to improve its efficiency.

FCFS Scheduling

**First come first serve** (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 if the burst time of the first process is the longest among all the jobs.

Advantages of FCFS

* Simple
* Easy
* First come, First serv

Disadvantages of FCFS

1. The scheduling method is non preemptive, the process will run to the completion.
2. Due to the non-preemptive nature of the algorithm, the problem of starvation may occur.
3. Although it is easy to implement, but it is poor in performance since the average waiting time is higher as compare to other scheduling algorithms.

Example

Let's take an example of The FCFS scheduling algorithm. In the Following schedule, there are 5 processes with process ID **P0, P1, P2, P3 and P4**. P0 arrives at time 0, P1 at time 1, P2 at time 2, P3 arrives at time 3 and Process P4 arrives at time 4 in the ready queue. The processes and their respective Arrival and Burst time are given in the following table.

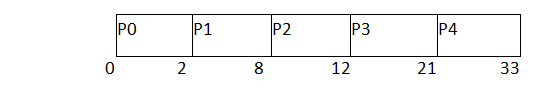
The Turnaround time and the waiting time are calculated by using the following formula.

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turnaround time - Burst Time

The average waiting Time is determined by summing the respective waiting time of all the processes and divided the sum by the total number of processes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 0 | 0 | 2 | 2 | 2 | 0 |
| 1 | 1 | 6 | 8 | 7 | 1 |
| 2 | 2 | 4 | 12 | 10 | 6 |
| 3 | 3 | 9 | 21 | 18 | 9 |
| 4 | 4 | 12 | 33 | 29 | 17 |

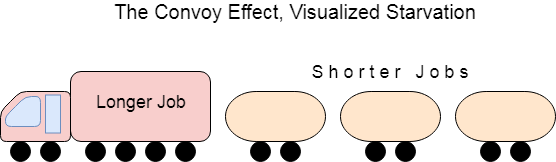
               Avg Waiting Time=31/5



**(Gantt chart)**

FCFS may suffer from the **convoy effect** if the burst time of the first job is the highest among all. As in the real life, if a convoy is passing through the road then the other persons may get blocked until it passes completely. This can be simulated in the Operating System also.

If the CPU gets the processes of the higher burst time at the front end of the ready queue then the processes of lower burst time may get blocked which means they may never get the CPU if the job in the execution has a very high burst time. This is called **convoy effect** or **starvation**.



Example

In the Example, We have 3 processes named as **P1, P2 and P3**. The Burt Time of process P1 is highest.

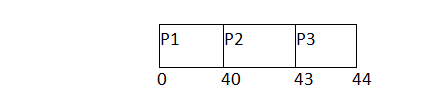
The Turnaround time and the waiting time in the following table, are calculated by the formula,

1. Turn Around Time = Completion Time - Arrival Time
2. Waiting Time = Turn Around Time - Burst Time

In the First scenario, The Process P1 arrives at the first in the queue although; the burst time of the process is the highest among all. Since, the Scheduling algorithm, we are following is FCFS hence the CPU will execute the Process P1 first.

In this schedule, the average waiting time of the system will be very high. That is because of the convoy effect. The other processes P2, P3 have to wait for their turn for 40 units of time although their burst time is very low. This schedule suffers from starvation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 1 | 0 | 40 | 40 | 40 | 0 |
| 2 | 1 | 3 | 43 | 42 | 39 |
| 3 | 1 | 1 | 44 | 43 | 42 |

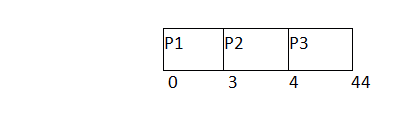


         Avg waiting Time = 81/3

In the Second scenario, If Process P1 would have arrived at the last of the queue and the other processes P2 and P3 at earlier then the problem of starvation would not be there.

Following example shows the deviation in the waiting times of both the scenarios. Although the length of the schedule is same that is 44 units but the waiting time will be lesser in this schedule.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** | **Completion Time** | **Turn Around Time** | **Waiting Time** |
| 1 | 1 | 40 | 44 | 43 | 3 |
| 2 | 0 | 3 | 3 | 3 | 0 |
| 3 | 0 | 1 | 4 | 4 | 3 |



**Avg Waiting Time=6/3**

FCFS with Overhead

In the above Examples, we are assuming that all the processes are the CPU bound processes only. We were also neglecting the context switching time.

However if the time taken by the scheduler in context switching is considered then the average waiting time of the system will be increased which also affects the efficiency of the system.

Context Switching is always an overhead. The Following Example describeshow the efficiency will be affected if the context switching time is considered in the system

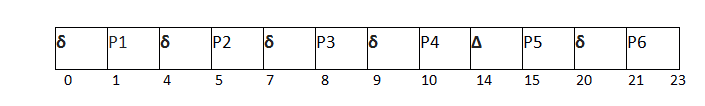
Example

In the following Example, we are considering five processes P1, P2, P3, P4, P5 and P6. Their arrival time and Burst time are given below.

|  |  |  |
| --- | --- | --- |
| **Process ID** | **Arrival Time** | **Burst Time** |
| 1 | 0 | 3 |
| 2 | 1 | 2 |
| 3 | 2 | 1 |
| 4 | 3 | 4 |
| 5 | 4 | 5 |
| 6 | 5 | 2 |

If the context switching time of the system is 1 unit then the Gantt chart of the system will be prepared as follows.

Given **δ=1 unit;**



The system will take extra 1 unit of time (overhead) after the execution of every process to schedule the next process.

1. Inefficiency= (6/23) X 100 %
3. Efficiencyͷ = (1-6/23) X 100 %

**Shortest Job First (SJF) Scheduling**

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

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.

Advantages of SJF

Maximum throughput

Minimum average waiting and turnaround time

Disadvantages of SJF

May suffer with the problem of starvation

It is not implementable because the exact Burst time for a process can't be known in advance.

There are different techniques available by which, the CPU burst time of the process can be determined. We will discuss them later in detail.

Example

In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

PID Arrival Time Burst Time Completion Time Turn Around Time Waiting Time

1 1 7 8 7 0

2 3 3 13 10 7

3 6 2 10 4 2

4 7 10 31 24 14

5 9 8 21 12 4

Since, No Process arrives at time 0 hence; there will be an empty slot in the Gantt chart from time 0 to 1 (the time at which the first process arrives).

According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.

Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.

This will be executed till 8 units of time. Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.

Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.

So that's how the procedure will go on in shortest job first (SJF) scheduling algorithm.

os SJF scheduling algorithm

Avg Waiting Time = 27/5

Prediction of CPU Burst Time for a process in SJF

The SJF algorithm is one of the best scheduling algorithms since it provides the maximum throughput and minimal waiting time but the problem with the algorithm is, the CPU burst time can't be known in advance.

We can approximate the CPU burst time for a process. There are various techniques which can be used to assume the CPU Burst time for a process. Our Assumption needs to be accurate in order to utilize the algorithm optimally.

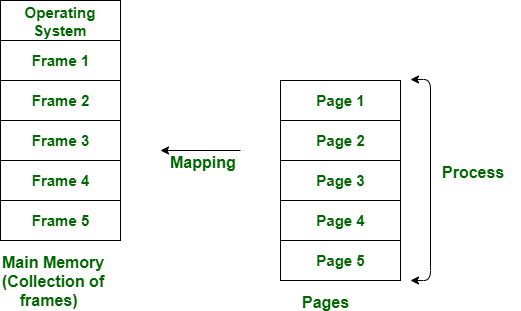
There are the following techniques used for the assumption of CPU burst time for a process.

**MEMORY MANAGEMENT**

Difference Between Paging and Segmentation

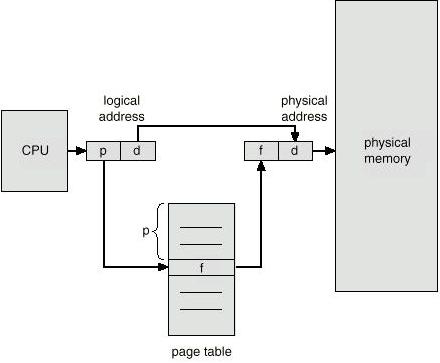
[**Paging**](https://www.geeksforgeeks.org/paging-in-operating-system/)**:**  
Paging is a method or techniques which is used for non-contiguous memory allocation. It is a fixed size partitioning theme (scheme). In paging, both main memory and secondary memory are divided into equal fixed size partitions. The partitions of secondary memory area unit and main memory area unit known as pages and frames respectively.

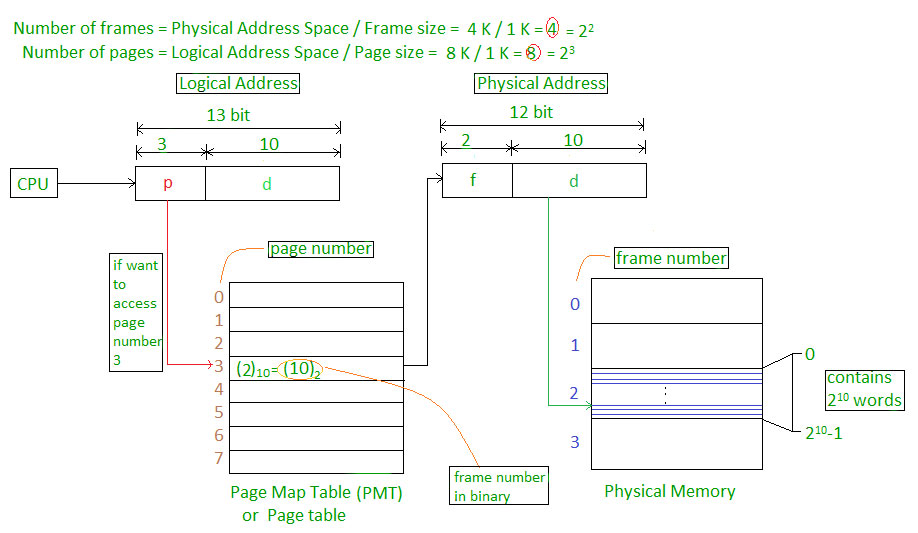
Paging is a memory management method accustomed fetch processes from the secondary memory into the main memory in the form of pages. In paging, each process is split into parts wherever size of every part is same as the page size. The size of the last half could also be but the page size. The pages of process area unit hold on within the frames of main memory relying upon their accessibility.



Address generated by CPU (logical address) is divided into:

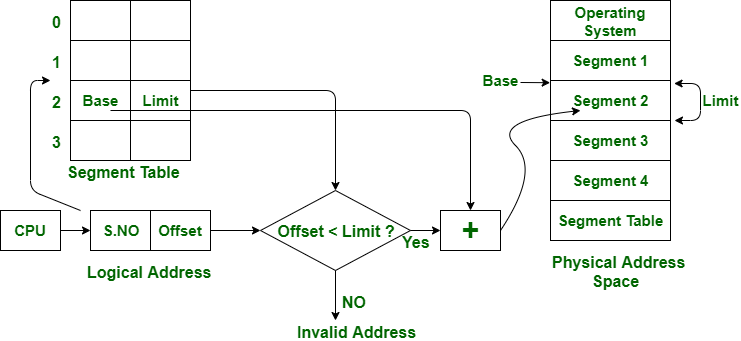
1. Page number **p** is an index into a *page table* that contains base address of each page in physical memory.
2. Page offset **d** is a *displacement*, combined with base address to define the physical memory address that is sent to the memory unit.

****



[**Segmentation**](https://www.geeksforgeeks.org/segmentation-in-operating-system/)**:**  
Segmentation is another non-contiguous memory allocation scheme like paging. like paging, in segmentation, process isn’t divided indiscriminately into mounted(fixed) size pages. It is variable size partitioning theme. like paging, in segmentation, secondary and main memory are not divided into partitions of equal size. The partitions of secondary memory area unit known as segments. The details concerning every segment are in a table known as segmentation table. Segment table contains two main data concerning segment, one is Base, which is the starting address of the segment and another is Limit, which is the length of the segment.

In segmentation, CPU generates logical address that contains Segment number and segment offset. If the segment offset is a smaller amount than the limit then the address called valid address otherwise it throws miscalculation because the address is invalid.



The above figure shows the translation of logical address to physical address.

**Difference between Paging and Segmentation:**

| **S.NO** | **PAGING** | **SEGMENTATION** |
| --- | --- | --- |
| 1. | In paging, program is divided into fixed size pages. | In segmentation, program is divided into variable size sections. |
| 2. | For paging operating system is accountable. | For segmentation compiler is accountable. |
| 3. | Page size is determined by hardware. | Here, the section size is given by the user. |
| 4. | It is faster in the comparison of segmentation. | Segmentation is slow. |
| 5. | Paging could result in internal fragmentation. | Segmentation could result in external fragmentation. |
| 6. | In paging, logical address is split into page number and page offset. | Here, logical address is split into section number and section offset. |
| 7. | Paging comprises a page table which encloses the frame no of every page. | While segmentation also comprises the segment table which encloses base address and limit. |
| 8. | Page table is employed to keep up the page data. | Section Table maintains the section data. |
| 9. | In paging, operating system must maintain a free frame list. | In segmentation, operating system maintain a list of holes in main memory. |
| 10. | Paging is invisible to the user. | Segmentation is visible to the user. |
| 11. | In paging, processor needs page number, offset to calculate absolute address. | In segmentation, processor uses segment number, offset to calculate full address. |

Segmentation in Operating System

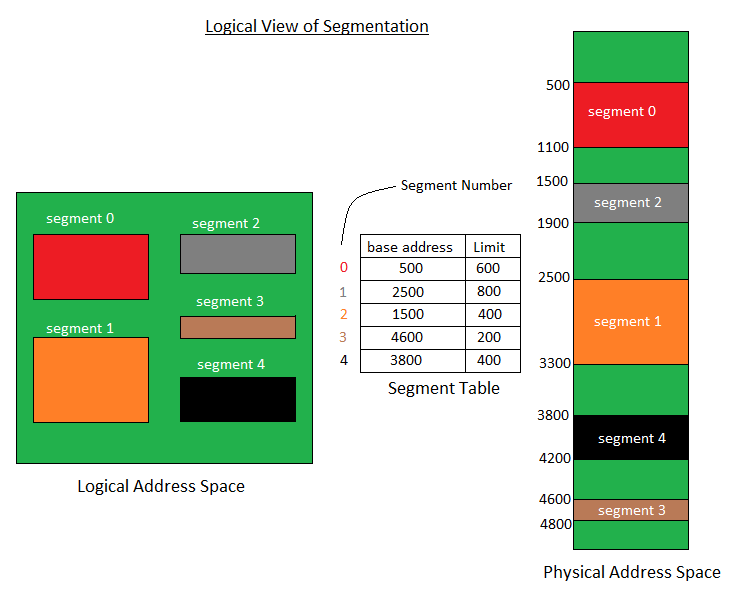
A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the same sizes are called segments. Segmentation gives user’s view of the process which paging does not give. Here the user’s view is mapped to physical memory.

**segmentation –**  
Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

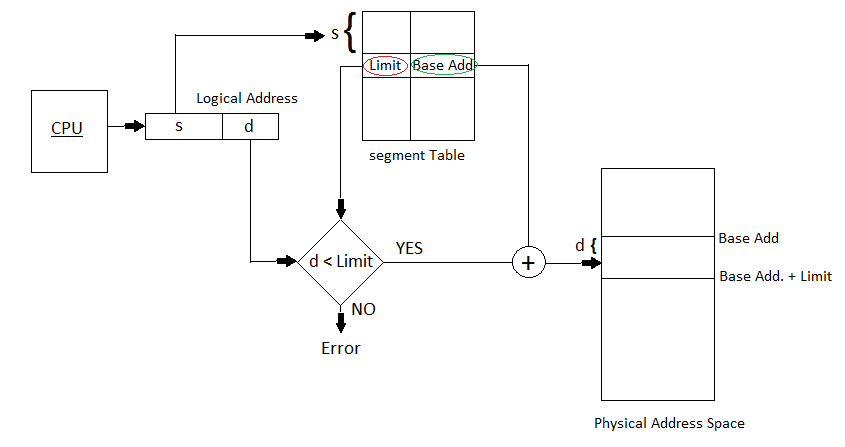
There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

**Segment Table –** It maps two-dimensional Logical address into one-dimensional Physical address. It’s each table entry has:

* **Base Address:**Itcontains the starting physical address where the segments reside in memory.
* **Limit:** It specifies the length of the segment.



Translation of Two dimensional Logical Address to one dimensional Physical Address.



Address generated by the CPU is divided into:

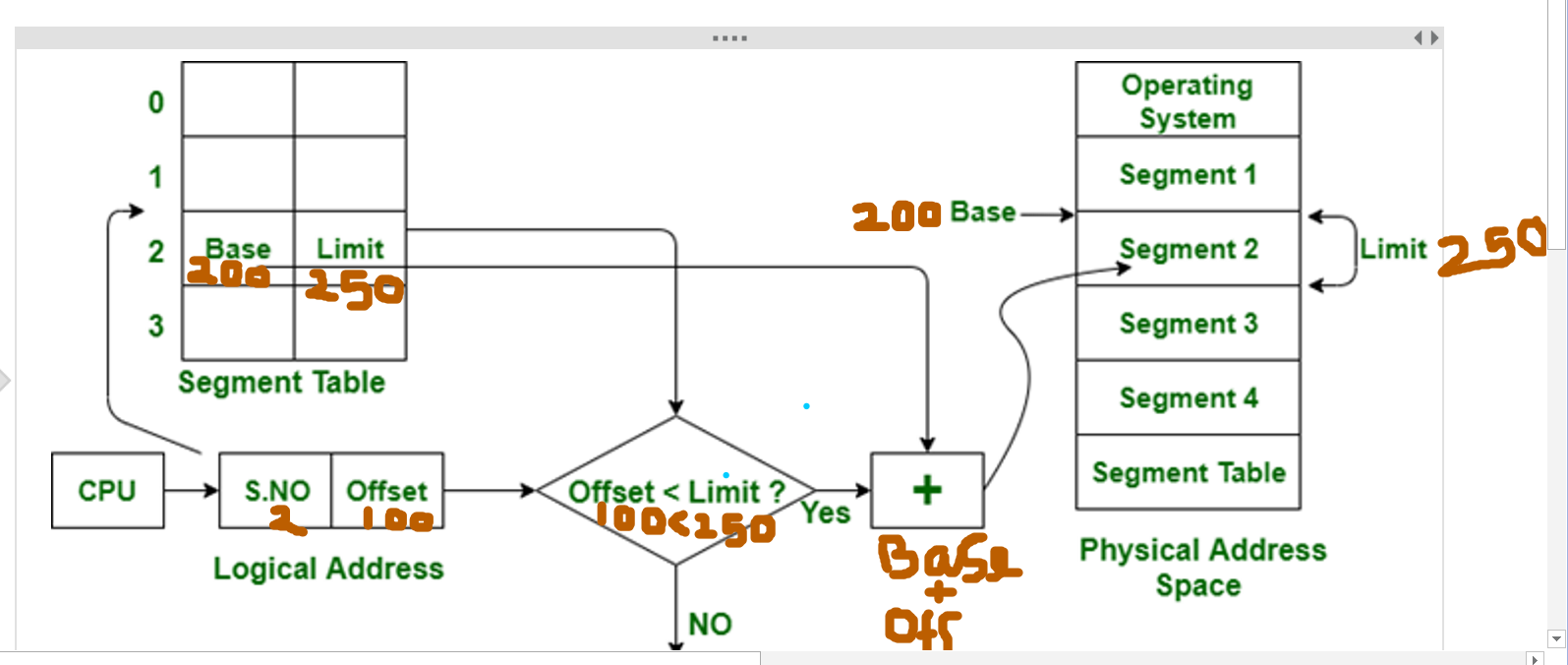
* **Segment number (s):** Number of bits required to represent the segment.
* **Segment offset (d):** Number of bits required to represent the size of the segment.

**Advantages of Segmentation –**

* No Internal fragmentation.
* Segment Table consumes less space in comparison to Page table in paging.

**Disadvantage of Segmentation –**

* As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.



**PAGE REPLACEMENT ALGORITHMS**

We have discussed-

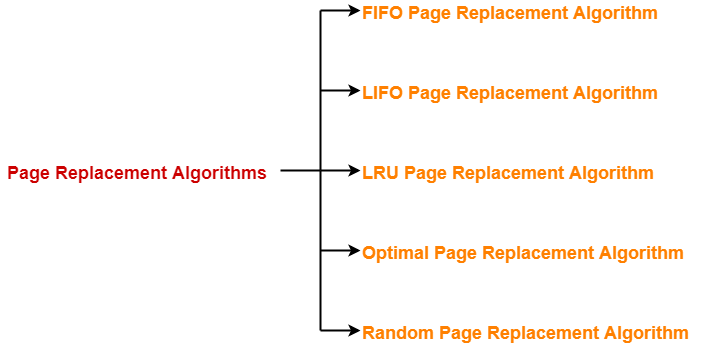
* A page fault occurs when a page referenced by the CPU is not found in the main memory.
* The required page has to be brought from the secondary memory into the main memory.
* A page has to be replaced if all the frames of main memory are already occupied.

Page replacement is required when-

* All the frames of main memory are already occupied.
* Thus, a page has to be replaced to create a room for the required page.

Page replacement algorithms help to decide which page must be swapped out from the main memory to create a room for the incoming page.

Various page replacement algorithms are-



## **FIFO Page Replacement Algorithm-**

* As the name suggests, this algorithm works on the principle of “**First in First out**“.
* It replaces the oldest page that has been present in the main memory for the longest time.
* It is implemented by keeping track of all the pages in a queue.

## **LIFO Page Replacement Algorithm-**

* As the name suggests, this algorithm works on the principle of “**Last in First out**“.
* It replaces the newest page that arrived at last in the main memory.
* It is implemented by keeping track of all the pages in a stack.

## **LRU Page Replacement Algorithm-**

* As the name suggests, this algorithm works on the principle of “**Least Recently Used**“.
* It replaces the page that has not been referred by the CPU for the longest time.

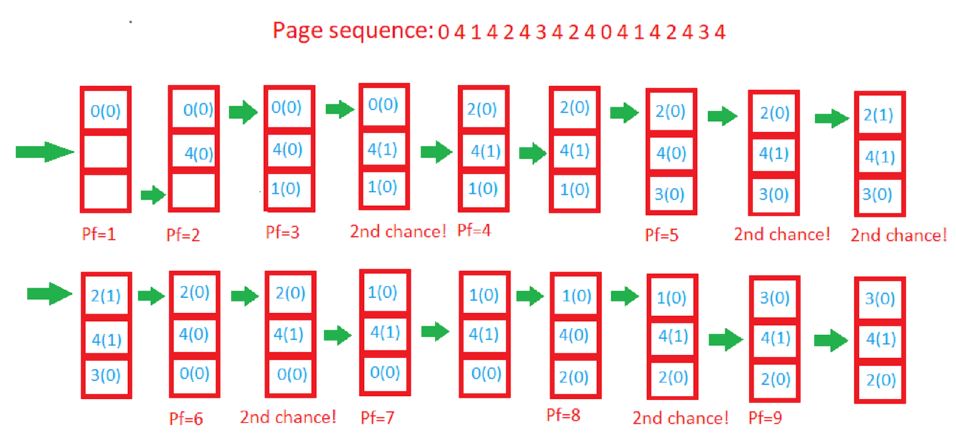
## **Optimal Page Replacement Algorithm-**

* This algorithm replaces the page that will not be referred by the CPU in future for the longest time.
* It is practically impossible to implement this algorithm.
* This is because the pages that will not be used in future for the longest time can not be predicted.
* However, it is the best known algorithm and gives the least number of page faults.
* Hence, it is used as a performance measure criterion for other algorithms.

## **Second Chance Page Replacement Algorithm**

**Example –**  
Let’s say the reference string is **0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4** and we have **3** frames. Let’s see how the algorithm proceeds by tracking the second chance bit and the pointer.

* Initially, all frames are empty so after first 3 passes they will be filled with {0, 4, 1} and the second chance array will be {0, 0, 0} as none has been referenced yet. Also, the pointer will cycle back to 0.
* **Pass-4:** Frame={0, 4, 1}, second\_chance = {0, 1, 0} [4 will get a second chance], pointer = 0 (No page needed to be updated so the candidate is still page in frame 0), pf = 3 (No increase in page fault number).
* **Pass-5:** Frame={2, 4, 1}, second\_chance= {0, 1, 0} [0 replaced; it’s second chance bit was 0, so it didn’t get a second chance], pointer=1 (updated), pf=4
* **Pass-6:** Frame={2, 4, 1}, second\_chance={0, 1, 0}, pointer=1, pf=4 (No change)
* **Pass-7:** Frame={2, 4, 3}, second\_chance= {0, 0, 0} [4 survived but it’s second chance bit became 0], pointer=0 (as element at index 2 was finally replaced), pf=5
* **Pass-8:** Frame={2, 4, 3}, second\_chance= {0, 1, 0} [4 referenced again], pointer=0, pf=5
* **Pass-9:** Frame={2, 4, 3}, second\_chance= {1, 1, 0} [2 referenced again], pointer=0, pf=5
* **Pass-10:** Frame={2, 4, 3}, second\_chance= {1, 1, 0}, pointer=0, pf=5 (no change)
* **Pass-11:** Frame={2, 4, 0}, second\_chance= {0, 0, 0}, pointer=0, pf=6 (2 and 4 got second chances)
* **Pass-12:** Frame={2, 4, 0}, second\_chance= {0, 1, 0}, pointer=0, pf=6 (4 will again get a second chance)
* **Pass-13:** Frame={1, 4, 0}, second\_chance= {0, 1, 0}, pointer=1, pf=7 (pointer updated, pf updated)
* **Page-14:** Frame={1, 4, 0}, second\_chance= {0, 1, 0}, pointer=1, pf=7 (No change)
* **Page-15:** Frame={1, 4, 2}, second\_chance= {0, 0, 0}, pointer=0, pf=8 (4 survived again due to 2nd chance!)
* **Page-16:** Frame={1, 4, 2}, second\_chance= {0, 1, 0}, pointer=0, pf=8 (2nd chance updated)
* **Page-17:** Frame={3, 4, 2}, second\_chance= {0, 1, 0}, pointer=1, pf=9 (pointer, pf updated)
* **Page-18:** Frame={3, 4, 2}, second\_chance= {0, 1, 0}, pointer=1, pf=9 (No change)



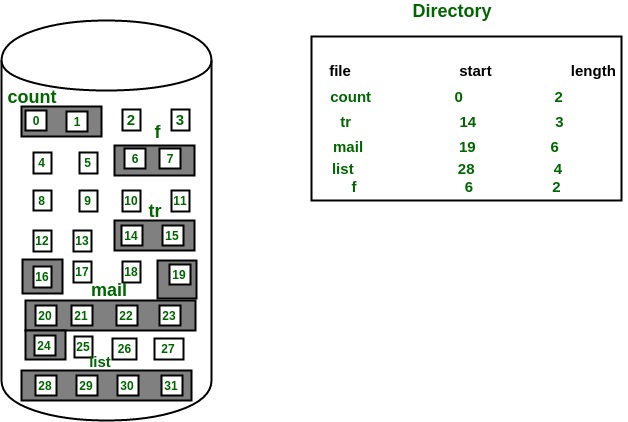
Disk space allocation algorithms

* 1. Contiguous

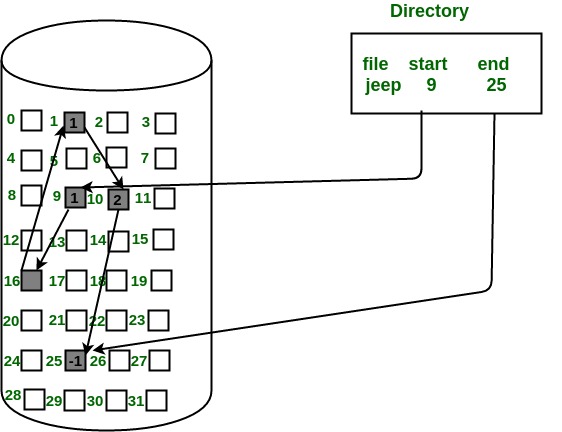
2. linked

3. I-Node

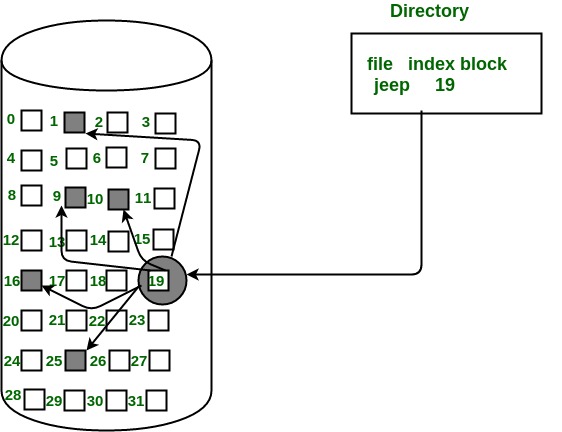
* 1. **Contiguous Allocation**

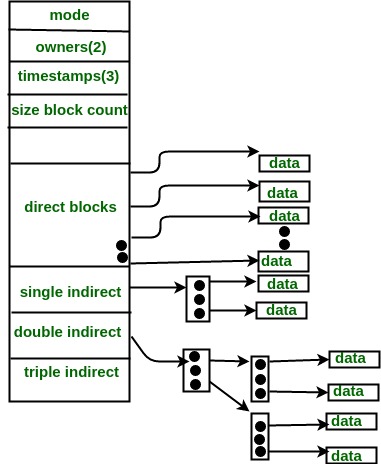


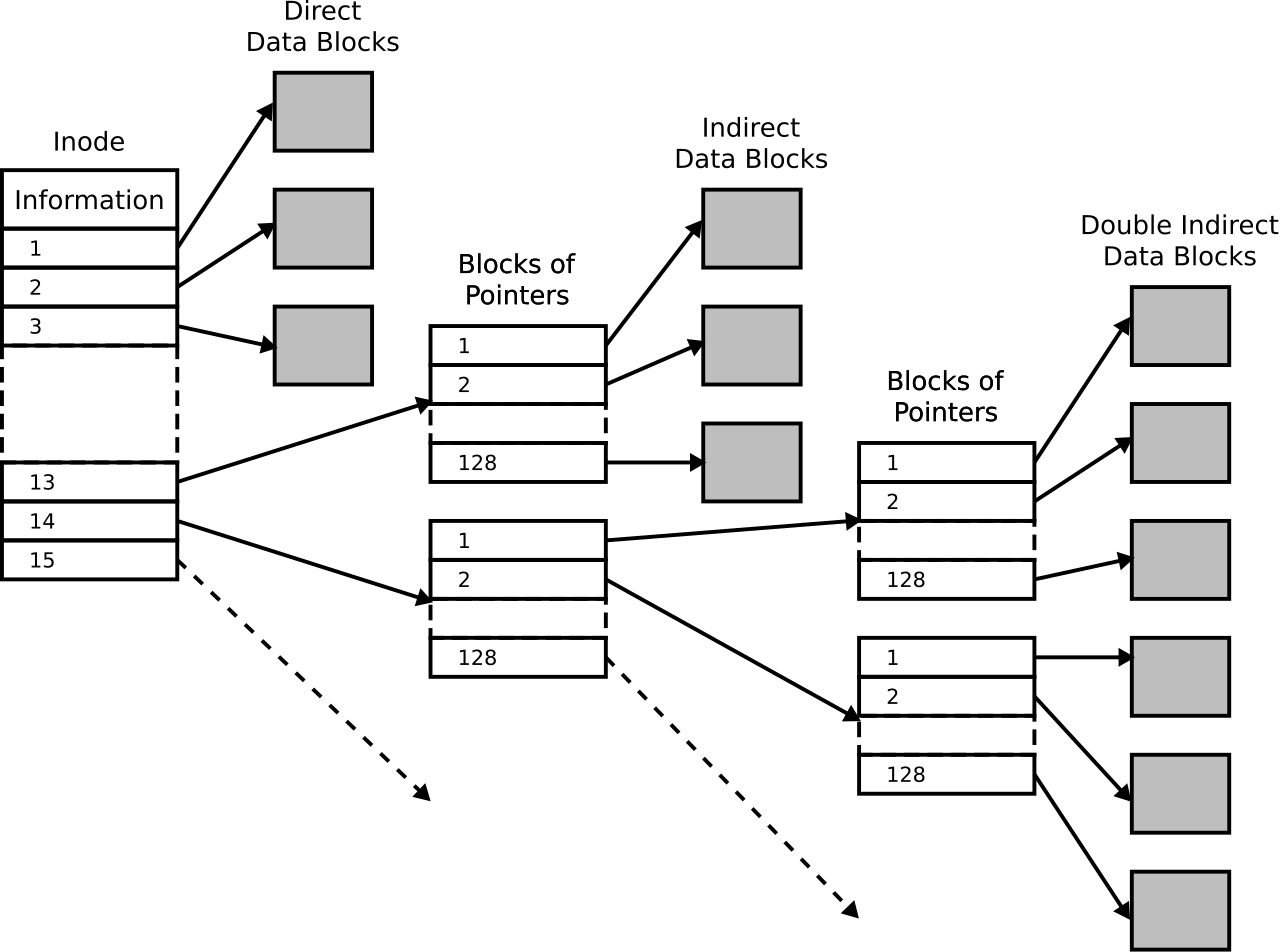
2.**Linked Allocation**



3.Indexed Allocation (I-node)





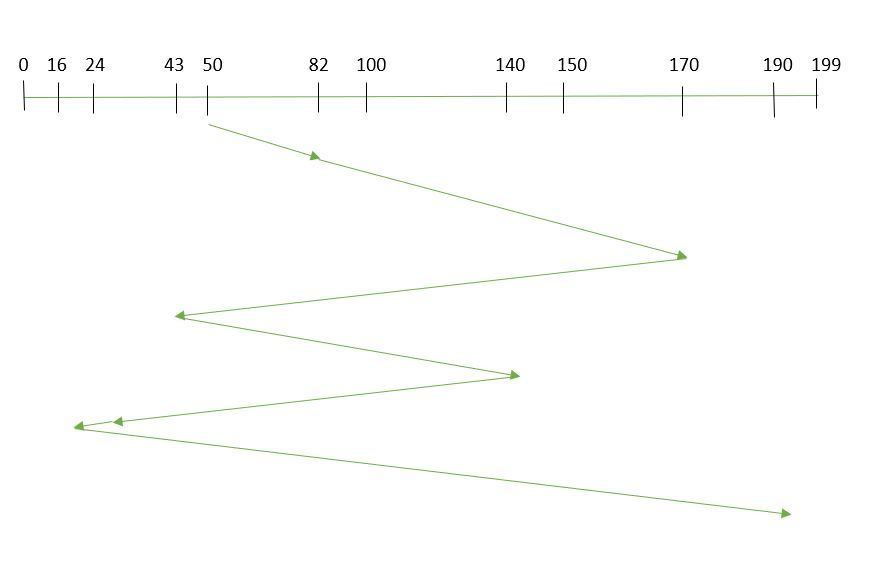


<https://www.geeksforgeeks.org/file-allocation-methods/>

**Disk Scheduling Algorithms (I/O Scheduling, Disk displacement algorithms)**

Example:

1. **FCFS** :Suppose the order of request is- (82,170,43,140,24,16,190)  
   And current position of Read/Write head is : 50

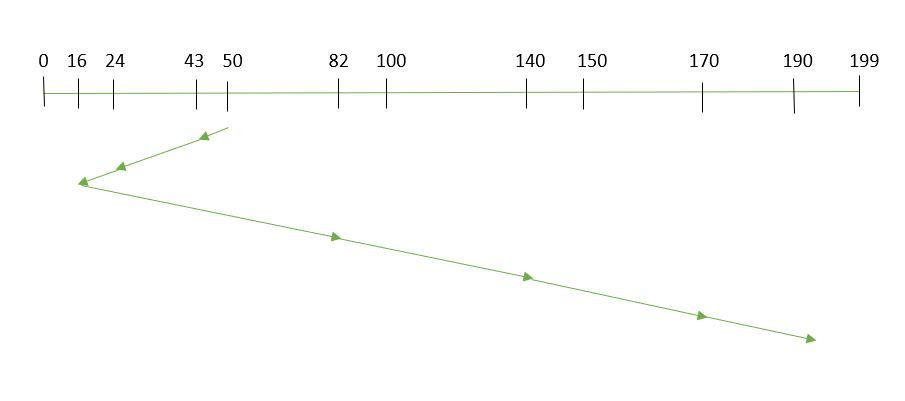
****

So, total seek time:   
=(82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16)**=642**

**SSTF(Shortest Seek Time First)**

#### **Example:**

1. Suppose the order of request is- (82,170,43,140,24,16,190)  
   And current position of Read/Write head is : 50





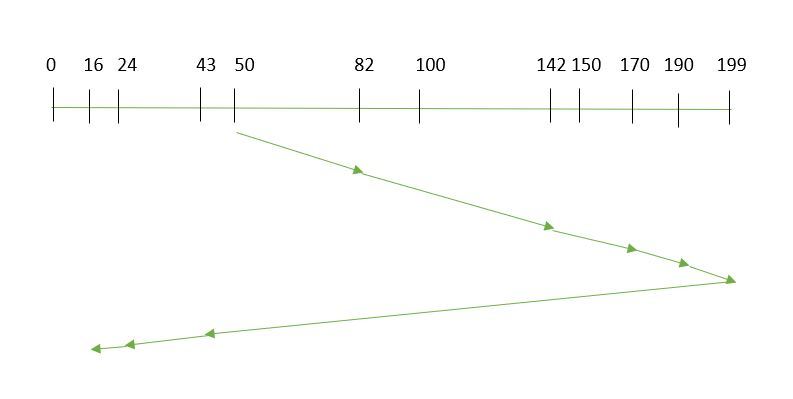
So, total seek time:

1. =(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170)   
   =**208**

**SCAN (Elevator Algorithm)**

Example:

1. Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”.

****

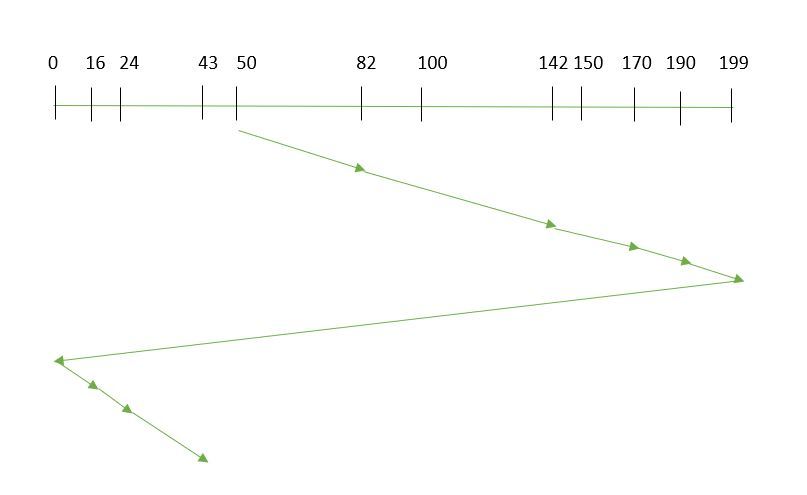
Therefore, the seek time is calculated as:

1. =(199-50)+(199-16)**=332**

**C-SCAN (Circular SCAN)**

**Example:**

Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”. 

****

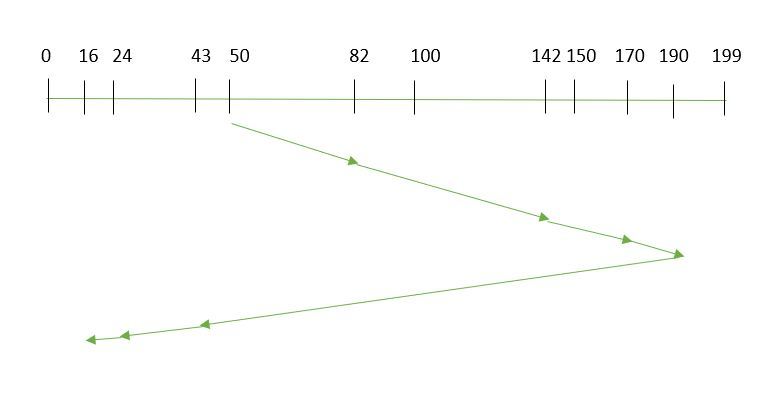
Seek time is calculated as:

=(199-50)+(199-0)+(43-0)**=391**

**LOOK:**

Example:

1. Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towards the larger value”.

****

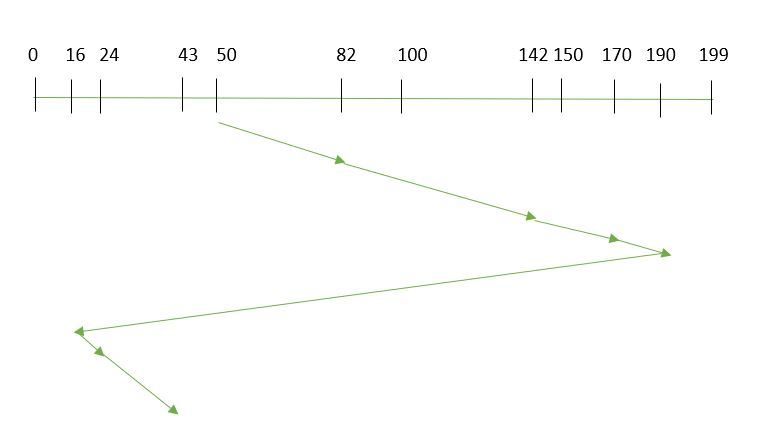
So, the seek time is calculated as:

=(190-50)+(190-16**)   
=314**

**CLOOK:**

Example:

1. Suppose the requests to be addressed are-82,170,43,140,24,16,190. And the Read/Write arm is at 50, and it is also given that the disk arm should move “towardsthe larger value”

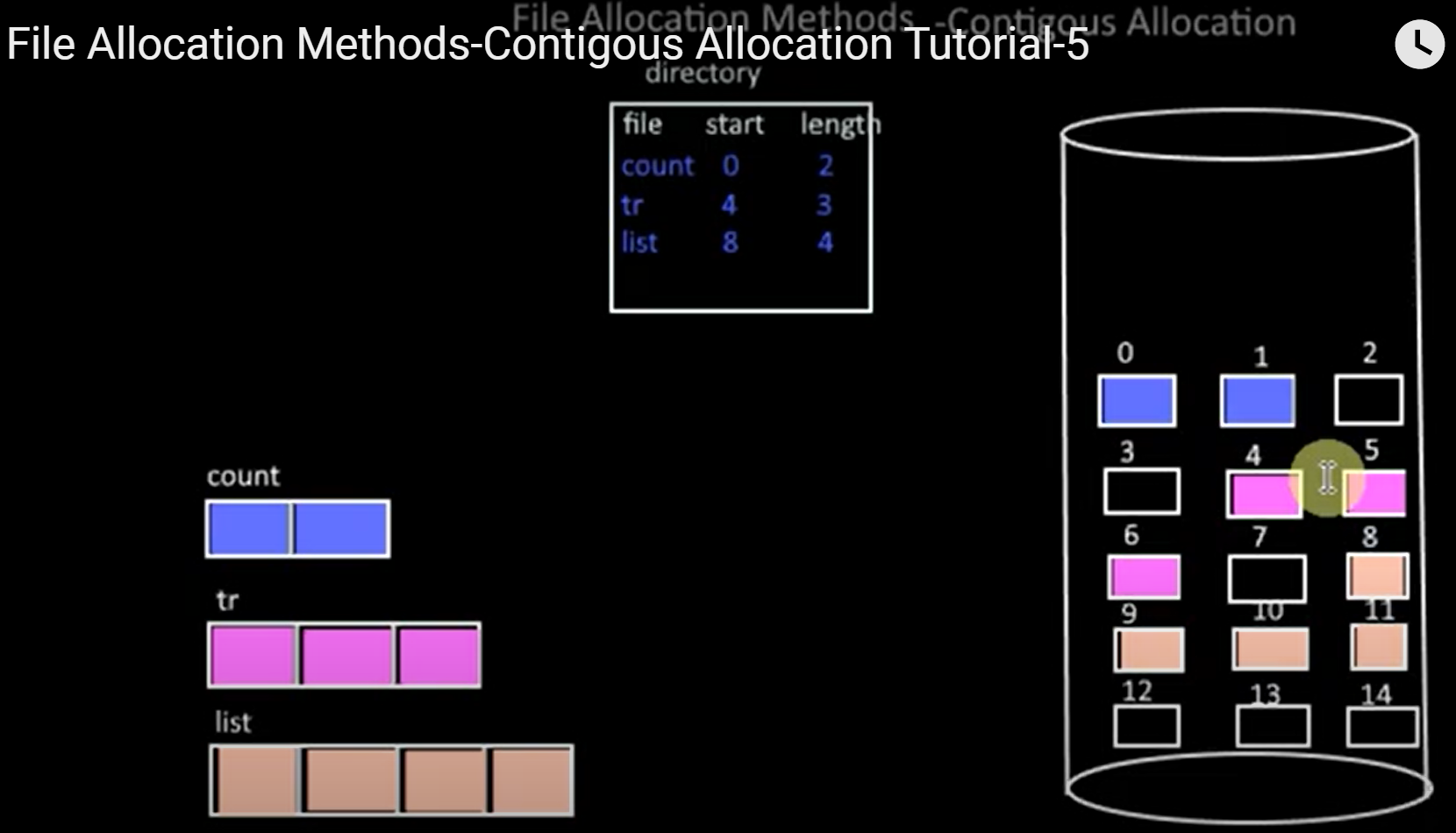
****

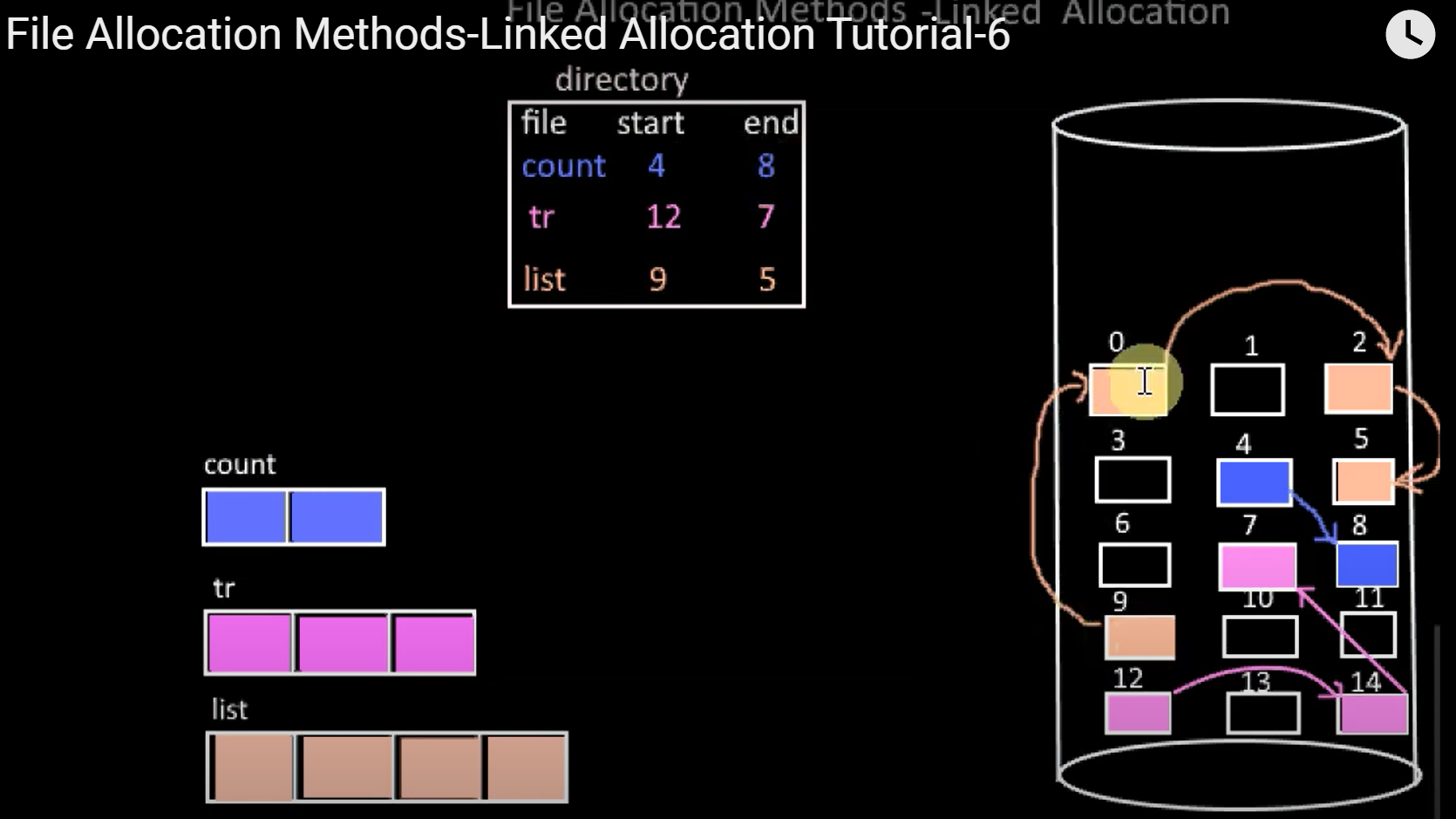
So, the seek time is calculated as:

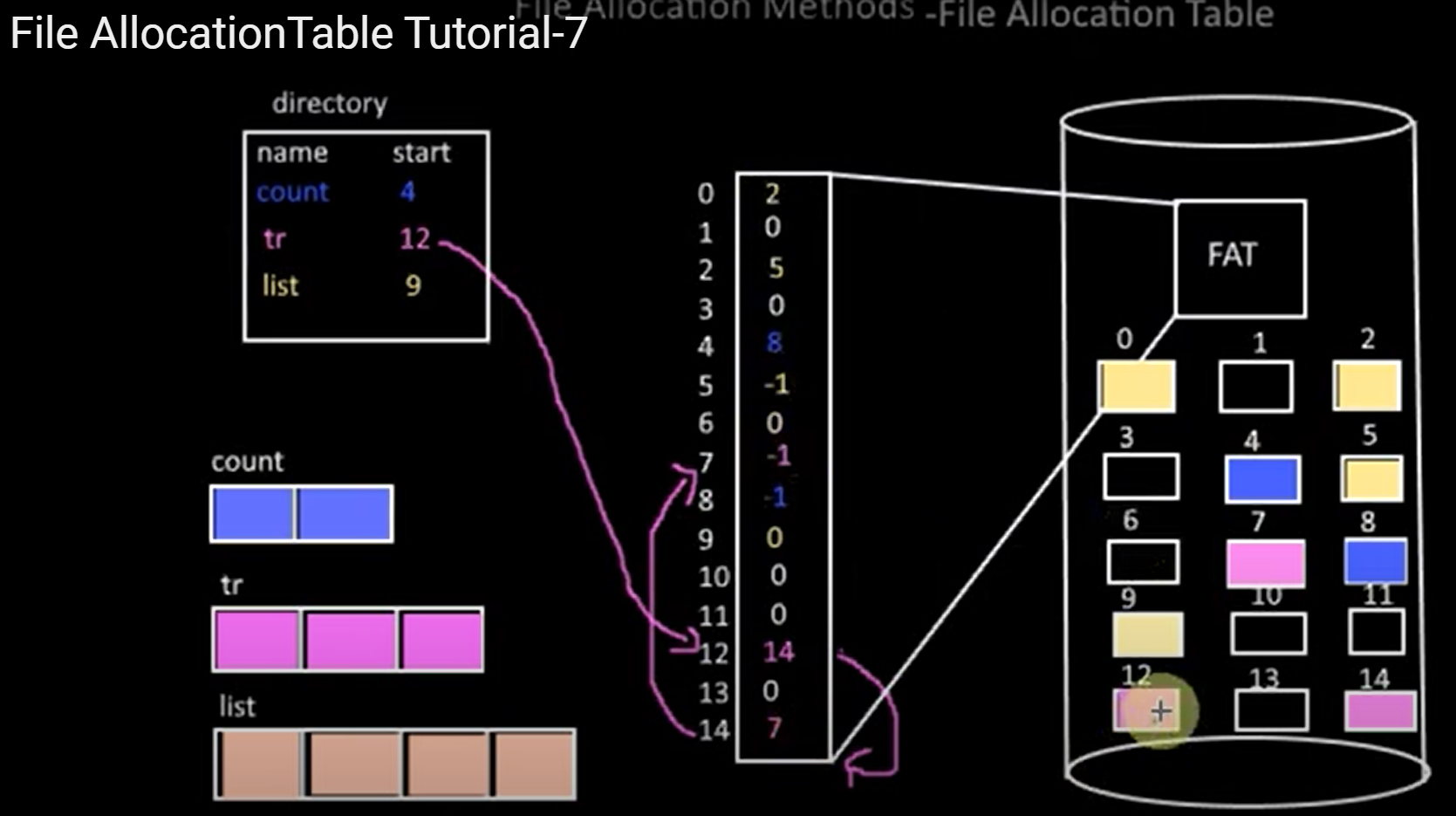
=(190-50)+(190-16)+(43-16)**=341**

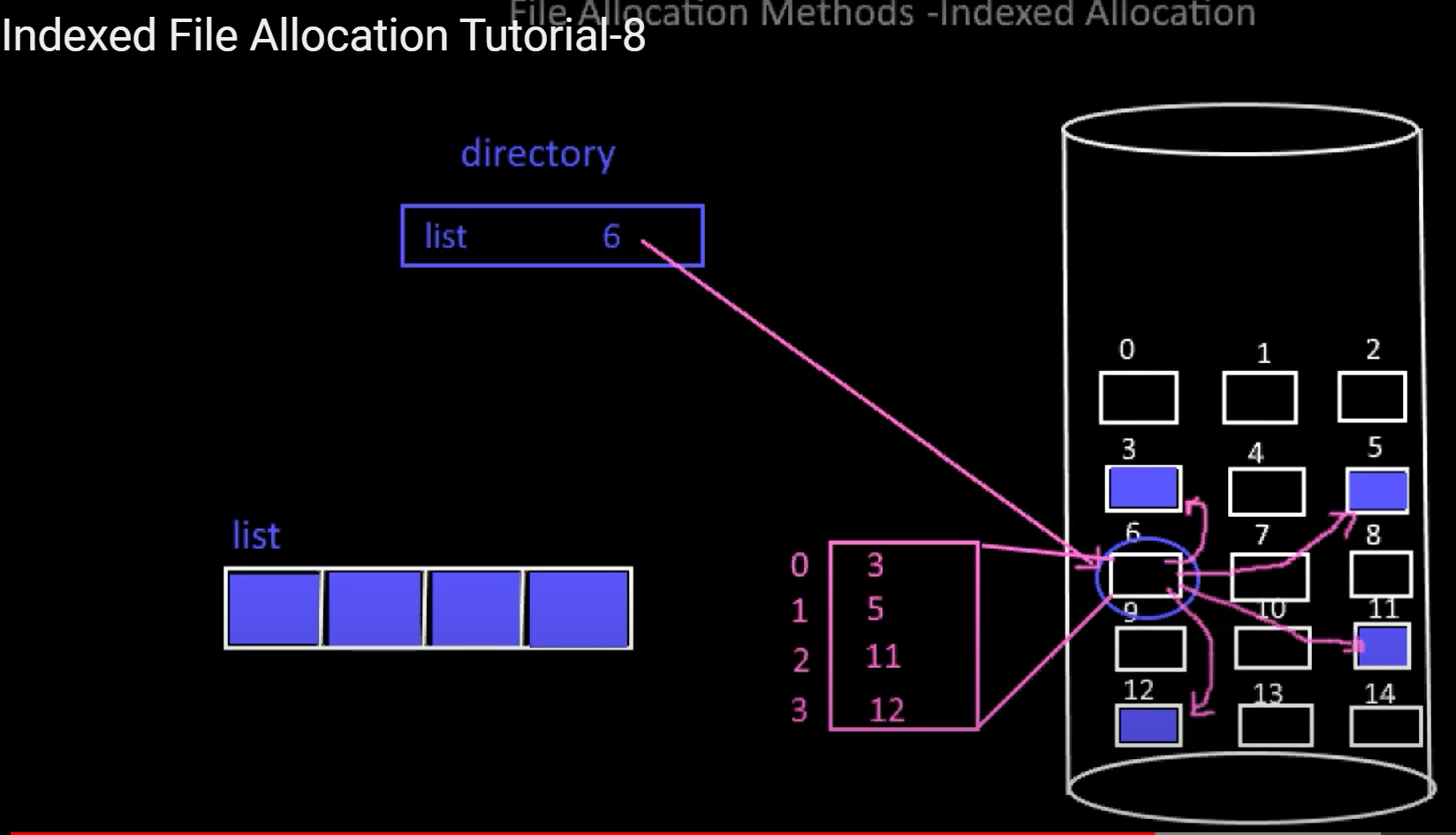
[**https://www.geeksforgeeks.org/disk-scheduling-algorithms/**](https://www.geeksforgeeks.org/disk-scheduling-algorithms/)

**File Allocation Methods**

****

****





|  |
| --- |
| Which of the following is not true in the context of the operating system?  Interface  Extended machine  Application  Resource manager |
| CPU cycle is not included this step.  Fetch  Sort  Execute  Decode |
| Which one of the following memory is the fastest one?  Main memory  Cache memory  Secondary memory  Register memory |
| Which of the following holds the address of the next instruction?  Stack pointer  Program counter  Address space  process |
| \_\_\_\_\_ is a system call to create a duplicate process in Linux.  fork  create  load  exec  \_\_\_ is a system call to load the selected program into the memory.  exec |
| MULTICS operating system is an example of \_\_\_\_\_\_ structure.  Monolithic  Microkernel  Layered  Client-Server  Which one of the following operating system structures has a Reincarnation server?  Microkernel |
| POSIX is an API of Windows system calls.  TRUE  FALSE |
| TYPE1 Hypervisor is called as Bare metal system.  TRUE  FALSE |
| In which type of operating system has high memory constraint?  Personal computer operating system  Embedded system operating system  Server operating system  Smart card operating system |
| Process is a  system call  program on execution  thread  code |
| Which one of the following is not a parameter of the Read system call?  fd  name  nbytes  buffer |
| Type 1 hypervisor has guest and host operating systems.  TRUE  FALSE |
| A process can go from Ready state to \_\_\_ state.  Running  Waiting  Terminate  new |
|  |
|  |
|  |
|  |