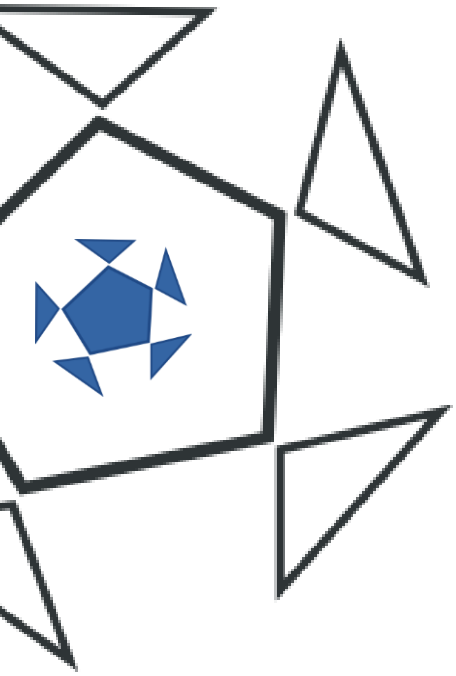
Raza's Simplified

Chapter 2



PROCESS MANAGEMENT

In this Chapter

Process Management

Before starting the core, let us define the computer. It is a machine which solves our problems according to our given instructions (or program). A program controls everything inside the computer. It does calculation, it directs communication and it controls the hardware. But it is only a set of instructions. It can not do anything until it is executed. The program exists on different media such as files, or in the mind of a programmer or even on a piece of paper. But for execution, it must be in computer's memory and then given to the CPU. When a program is executed it is no longer a nonliving entity. It is a process doing jobs on PC. Hence, *a process is an executing program*.

In older batch systems processes were jobs in execution. They were executed sequentially and considered a single object

A process has 4 elements: code, data, resources and execution flows.

Flows of execution are also called threads. A process can have single or multiple threads. A uni-threaded (single thread) process is a **heavy weight process**. The whole process will run and finish in one execution. However, if a process has multiple threads, then it will complete in many executions. Each of these threads is a **light weight process**.

A thread has its own PC, stack and system register values. But they share the same code, data and system resources. So switching of threads within one process takes little time as compared to switching a complete process.

OS are classified as being single-process single-threaded, multi-process single threaded or multi-process multi-threaded

The simplest among them is single-process single-threaded. In such a system, when a process is allocated to CPU for execution, it retains its control till it ends. If there is an I/O, then the CPU waits for its completion, thus wasting the CPU time. Only one process executes at a time and when it is finished, only then any other can be loaded and executed. It is estimated that about fifty percent of CPU time is wasted on such systems. Disk Operating System, DOS, is one such OS.

The multi-processing systems overcome this limitation by running more than one process simultaneously. The speed of I/O is very slow as compared to CPU processing. If an I/O is requested by a running process, the CPU waits for its completion and does nothing, millions of instructions can be executed during that time. Why not to unload this requesting process and load a new one for execution so the performance and speed are improved. This is the scheme in such OS to maximize the CPU utilization. This results in nearly 1oo% CPU utilization. So the CPU is shared among many running processes. If a process has generated an I/O and it becomes idle, then the CPU will not wait and sit idle. Rather, the scheduler will unload the idle process and select a new process for execution. This minimizes CPU idle time. But there is some overhead involved in switching the processes, that's why the CPU utilization remains nearly 100%. UNIX OS is such an example. All resources are allocated on the basis of a process needs. Process scheduling CPU is also based upon process.

However, the multi-process multi-threaded system are little bit different. The resource allocation is done on per process basis, but the CPU allocation is done on thread basis. So, a process is not directly given control of CPU, rather its thread is considered for CPU allocation. Windows from Microsoft, Solaris of Sun and Macintosh of Apple are such examples.

Scheduling Processes Process

Process scheduling is an important responsibility of an OS. The part of OS handling scheduling is called scheduler. The decision of scheduler is to find the process which will be given the control of CPU. There are three phases of scheduling and so three schedulers exist: long term, medium term and short term.

Long term scheduling is also called job scheduling, and long term scheduler is called job scheduler. Its objective is to provide the medium term scheduler with reasonable number of jobs. If the number of jobs is high, then CPU will remain over loaded and system performance will fall. If the number of jobs is too small the CPU will remain idle for much time and resources be wasted. When a job has been created by job scheduler, it remains there in active state till it is somehow terminated. The jobs created by long term scheduler compete for resources on the system and medium term scheduler selects one of them for short term scheduler.

The medium term scheduler is called swapper. It decides which process would stay in memory and which would be put on some disk or backing store to free memory for new processes. Doing so will result in increased number of processes running at a time in memory. But some memory management schemes, which are designed to increase memory performance like paging, may remove a complete process from memory.

Short term scheduler is the last scheduler. It is also known as dispatcher. Its responsibility is to look in the pool of processes in memory, select one of them and then allocate CPU to it. Dispatcher allocates the CPU to a process for a fixed and preset amount of time. Then the process must release the CPU even if it is not finished execution and return to the process pool. Dispatcher will select a new process for execution. To do this, the dispatcher must save the state of the outgoing process, including registers status so it would be reloaded again for the next term. The amount of time for which a process gets CPU is called time quantum, time slice or time slot. Dispatcher also schedules threads. It is done on multi-threaded systems. Since a process has flows of execution called threads, and every thread within a process shares resources and memory, therefore it is easier to switch threads instead of processes. The overhead is low if the threads of same process are switched, but if the threads of different processes are switched, the overhead is again high.

Process states can be defined for long, medium and short term schedulers. The simplest of which is short term scheduler, which defines the following simple states:

Blocked

When a process is executing, it may need some input or output. When this happens, the process must wait until this need is satisfied. Such a waiting process is unloaded from CPU and put in blocked state. I/O takes time to complete. CPU can use this time by executing another process, hence enhancing the system performance. Many processes can be in blocked state.

Ready

A process in ready state is not running but it is ready to run. A process which is ready to get the CPU is placed in ready state. Such a process can run if it gets CPU, and then its state will become running. Many processes can be in ready state.

Running

A process which is loaded on to the CPU and is executing is said to be in running state. Such a process comes from ready state. If there is *n* number of CPUs on a system then, at most, n processes can be executed at any given moment.

Terminated

A process enters the system and it is executed by the CPU for a specific time. A process eventually finishes execution of all of its code. At this stage the process is removed from system and its resources are released.. Its data and memory locations are marked as clear or invalid. Sometimes, data about a terminated process is kept for future use. This record can be used by its parent process that created it, or its child would need it, or it has an I/O in progress which would request it. A terminated process with its record still maintained is called zombie process. This term is coined with UNIX OS.

Swapping

In swapping a process is taken away from memory and stored on disk. It is done when the RAM runs short and we need space to load some new process or data in it. This swapped out process is placed at a specific location on disk. A process can be selected from ready or blocked processes for swapping. This swapping is done by medium term scheduler called swapper. It adds two more states to the scheme.

Swapped Ready

A process in ready state and waiting for CPU allocation can be selected and removed to the secondary storage. It will not run until it is brought back to memory.

Swapped Blocked

The swapper can select a blocked process for swapping.

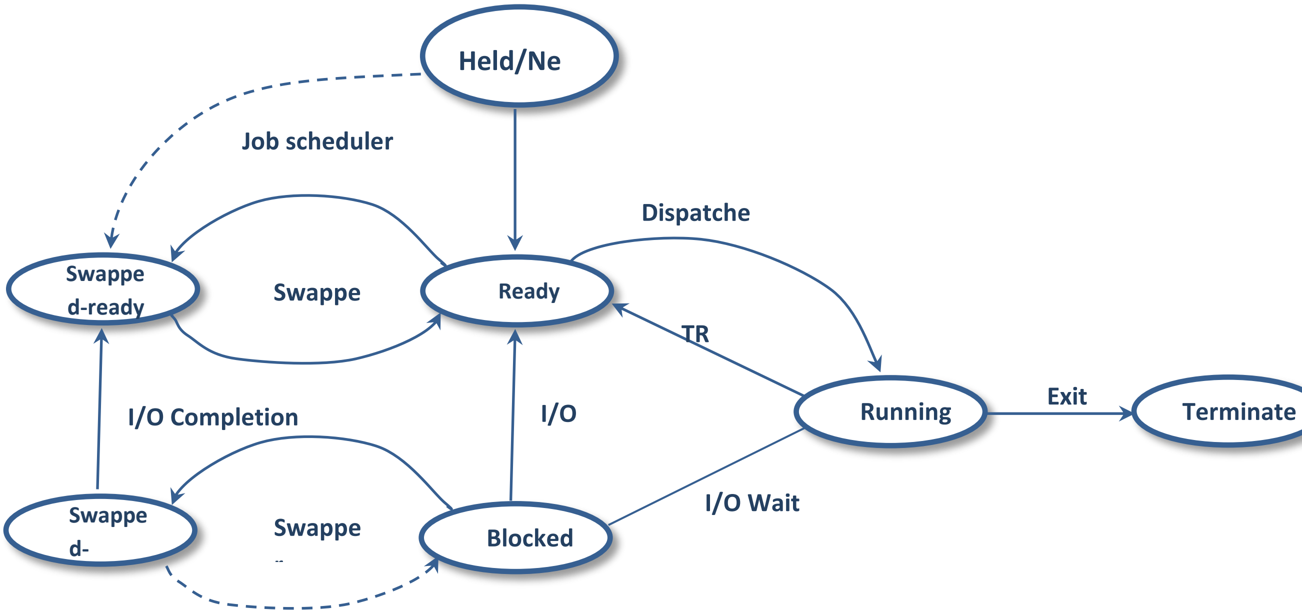
A process is said to be swapped if it is not in the ready processes queue. Two schemes need special attention here: demand paging and demand segmentation. In these schemes a process is divided in small parts called pages and segments. They can remove some or all pages/segments of a process to a backing store, yet such a process will not be called a swapped process. This is because it is not removed from ready queue and dispatcher still considers them.

Long term scheduling also creates following states.

New/Held

A new process is in this state before entering any other state. A process once removed from held state does not return to it. A process can have this state only once in its lifetime.

Process States Transitions



Transition means change. Process state transitions represent how a process switches from one state to another. It is also about the situations under which a process makes a transition.

1. Process creation to held

Sometimes it is called new. A new process is put in held state when it is created.

2. Held to ready

A process is put in ready state and it competes there for CPU allocation with other processes. In this state, the process is not executing but it will be executed if selected by the scheduler

3. Held to swapped ready

If the memory is low, then the swapper may decide to swap out a process from memory to disk. In this state, it is not considered for execution by the CPU. Later when the available memory is sufficient, the swapped out process can be swapped in and put in the ready state.

4. Swapped ready to ready

This transition occurs when a swapped out process is swapped back in memory. It is done when memory is sufficient to load this process. Space in memory is made available by following techniques:

a. Medium term scheduler selects and swaps out a process. As a result memory is released and old swapped out process can be loaded in that space.
b. Currently running processes release some occupied memory that is enough to swap in the process.
c. Some running processes have finished their execution and are terminated. So their memory is available for the swapped in process.
d. The OS has decided to reduce memory allocation to all the running processes. As a result, some memory is freed from every running process and made available for the swapped out process. This decision by OS is taken by the overall system load and processing needs.

There is exception to systems using demand segmentation and demand paging. A process may not have any part of it in memory, yet it will be considered swapped in as ready process. The page fault processing mechanism handles this situation. When such a process is accessed for CPU allocation, a page fault or segment fault occurs. It results in an I/O operation and the process will be blocked, waiting for I/O to complete.

5. Blocked to swapped blocked

A process which is waiting for an I/O to complete is said to be blocked. It is removed from CPU and put in blocked queue. If the swapper sees that there is not enough room in memory, it would swap out a blocked process to free some memory. A blocked process when swapped out is said to be in swapped blocked state. The swapper will do this if the memory needed for current processes is low, or if the current processes have requested more memory due to their changed needs and memory is not readily available.

6. Ready to swapped ready

A process which is waiting for the allocation of the CPU is in ready state. If the swapper sees that there is not enough room in memory, it would swap out a process in ready state to free some memory. A ready process when swapped out is said to be in swapped ready state. The swapper would do this if the memory needed for current processes is low or if the current processes have requested more memory due to their changed need and memory is not readily available.

7. Swapped blocked to blocked

A swapped blocked can be swapped in when:

a. sufficient unused memory is available,
b. or the process is a high priority process
c. or the system is under loaded.

A blocked process, wherever may it reside, can not execute. That's why, on many systems this transition is not supported, as it seems to be a waste of time. But in some cases it saves time. After this transition the process is in memory. When the blocking condition is satisfied, the process gets CPU allocation sooner than the process still swapped out.

8. Blocked to ready

When the pending I/O of a blocked process is completed, it can run on CPU once again. It is now put in the ready state.

9. Swapped blocked to swapped ready

When the pending I/O of a swapped blocked process is completed, it is put in swapped ready queue. It still cannot run on CPU as it is not in the ready state.

10. Running to ready

Every process is allocated CPU for a specific duration of time called time slice. If during this time, the process has generated an I/O it will be blocked. But if the time slice has finished and the process is still running, a TRO (time run out) interrupt will occur. It will transfer the control from the process to the dispatcher. Dispatcher will save the state of running process with its data and resources, registers and alike. The process will be put in ready queue, and a new process will be selected to run on CPU.

The dispatcher may do the same things if a high priority process enters the system, or a high priority process returns from blocked state to ready state. It is called preempting a low priority process.

11. Ready to running

If the CPU is not running a process, the dispatcher selects a process from the ready queue and loads it on CPU.

12. Running to blocked

A process may execute an instruction that is beyond its control. Such cases include:

a. I/O request
b. Communication to another process
c. Wait for special condition in system to occur

When anyone of these occurs, the process must wait as its further execution is suspended. So it is not desirable to keep such a process loaded on CPU as it will keep it idle, thus wasting the resources. To avoid this wastage, the process is put in blocked state and dispatcher selects a new process to run on CPU.

13. Running to terminated

A process which has completed its execution is unloaded from CPU and memory, all data is cleared, and its resources are released. In special cases its record is maintained for future use.

14. Any state of process to terminated

OS or some other high priority and privileged processes can decide to terminate a process. The terminating process can be in any state. In this case the process has not finished its execution. This method of terminating a process is called process killing or process abort.

Criteria of Scheduling for Processes

A scheduling algorithm is a technique selected by dispatcher to decide which process to run next. Every algorithm is developed to possibly maximize system performance. The best performance is achieved by making a combination of many scheduling criteria of different relative importance.

A scheduling policy decides the importance of each criterion. Many different algorithms have been developed and used to apply a scheduling policy.

Every criterion use different method to measure the system performance. An algorithm cannot optimize performance to meet all criteria. Sometimes algorithms measure the same aspect of performance in different way. We call them complementary algorithms. Other times they use contradictory techniques. So, improving performance according to one criterion reduces it as seen by the other.

Some criteria are as follows.

a. CPU Utilization
b. Balanced utilization
c. Throughput
d. Turnaround time
e. Wait time
f. Response time
g. Predictability
h. Fairness
i. Priorities

CPU Utilization

The percentage of time the CPU remains busy. This is dependent upon the load on system. If the system is heavily loaded, the CPU utilization will be high and vice versa.

Also, this factor is more considerable while working with high end time shared systems as compared to single user systems where this criterion is unimportant.

Balanced Utilization

The percentage of time all resources remain busy. It includes not only the utilization of CPU, but also memory, I/O devices, and other system resources.

Throughput

The number of processes executed in a given time. It involves the average time required for a process to complete its execution. Throughput will be high if the processes’ execution times are short, and low if they are long.

Turnaround Time

It is the interval between entry of a process and its termination. It is found by calculating the difference between the time the process is created and when it is terminated. Turnaround time is the total time the process remains in the system. It is the reciprocal of throughput.

Wait Time

It is the time a process spends waiting in the system, doing nothing. It is obtained by subtracting execution time of a process from turnaround time. Using wait time in calculation is better than using turnaround time as turnaround time contains both waiting and execution times. Turnaround time does not define wait and processing time separately and becomes a less accurate measure. On the other hand, the wait time, represents the performance more accurately by defining only one single aspect.

Response Time

The time taken by the system to produce first output after a user has given some input, especially on interactive systems. On such systems, the turnaround time is not much important, as it might be dependent upon speed of user’s input. More important in such system is the quick response to inputs.

Predictability

Predictability is the consistency in the measures of performance. It is the extent to which the performance measures won’t vary. Think of a system X, which always responds in two seconds, and another system Y, which mostly responds in half second but sometimes in 5 seconds. Y will be considered less predictable as compared to X. Unpredictable system behaviour is not liked by the users.

Fairness

Fairness is the extent to which all processes are given equal chance to execute. A process should never suffer from starvation, which is a situation in which a process remains stuck in a scheduling queue and never gets a chance to execute on CPU.

Priorities

Sometimes a process is given preference over others due to some reasons. In that case, a priority number is associated with processes and system decides when to execute them according to their priority number.

Algorithms for Process Scheduling

There are two categories of scheduling: non preemptive and preemptive.

Non Preemptive Algorithms

It is the old one. In this scheduling, when a process gains control of the CPU, it keeps CPU occupied till process termination or blocking. Such a scheme is good for batch processing systems as they do not care much about response time.

Preemptive Algorithms

Today's computer users are much concerned about system interactiveness and the systems cannot adapt the non-preemptive scheme to meet their needs. Instead they use **preemptive** scheme.

Before a running process ends or blocks, it may be unloaded from the CPU and some other process is executed instead. Interactive systems are doing this all the time and it is extremely necessary. On these systems, a lengthy, CPU bound process must not occupy the CPU for a long time. Whenever a user gives an input, it must be responded to quickly.

CPU Bound and I/O Bound Processes

A process which produces minimum I/O request or does not produce them at all is a CPU bound process. It is executed by the CPU and gets blocked very less.

A process which keeps on generating I/O requests is called I/O bound process. It is executed by the CPU for a short period and then it blocks soon to wait for an I/O to complete. Such a process causes the CPU switch to some other process quite often.

Algorithms

Following algorithms are used for selecting a process from ready queue to load on and run by the CPU:

FCFS

It means first come first served. It is a non-preemptive algorithm in nature. All incoming processes are arranged in a process queue in the order they arrive. The timestamp of every entering process is maintained. The dispatcher will select only that process for execution which is the oldest (meaning the value in timestamp is lowest).

FCFS gives favor to the CPU bound processes.

Suppose a system is executing a mix of I/O bound and CPU bound processes. An I/O bound process will run for a short time and then block. Now the CPU will load another process. This new process will block soon if it is also an I/O bound process.

Soon a CPU bound process will be executing. It will not block until it terminates. FCFS is non-preemptive so this process will not be unloaded from CPU by the system itself. Meanwhile if a requested I/O of an I/O bound process is completed, then that process will wait until the running CPU bound process ends. As a result, the I/O device utilization decreases and wait time for I/O bound processes increases.

There are some situations in which CPU will sit doing nothing, even by having a mix of CPU and I/O bound processes. Consider the case when a CPU bound process generates I/O. Next the CPU will load another process from ready queue, and it can also be an I/O bound one. Now it will soon block and all I/O bound processes will run shortly and then block. This may be the case that the CPU bound process is still waiting for its I/O to complete. So, the CPU is idle now.

Use of a suitable combination of CPU and I/O bound processes is better than mere FCFS.

SJF

It means Shortest Job First. It is non-preemptive algorithm. It was implemented first on batch processing systems. An expected duration to run the process till its completion is estimated. When the dispatcher is to select a process to load on the CPU, it will select the one with lowest expected time to complete. Sometimes, more than one process can have equal processing time, so a tie will result. FCFS algorithm is used in such a case.

A modified form of SJF is implemented on interactive systems. On such systems, the algorithm uses a CPU burst time. To find the next expected burst time, the average of previous CPU burst times for the process is used or exponential average is used. To find exponential average, following equation is used:

$$e_n+1=\alpha\ t_n+(1-\alpha)e_n$$

E_n	= the nth expected burst time
t_n	= the time taken for nth burst

α = a constant to give weight to the most recent burst utilized.

If $\alpha=0$, then the most recent burst is not used.

If $\alpha=1$, the only most recent burst is to be considered.

SJF favors short jobs and ignores the long ones. If many short jobs arrive constantly, then a long job will never be executed. This situation is called **starvation.**

SRT

It means shortest remaining time. It is an algorithm which combines SJF and preemption. When a process enters the ready queue, its remaining time to execute is compared with the remaining time of the process being executed. If the execution time of the running process is greater than that of the arriving, the running process is unloaded from the CPU and the new one is loaded.

It also favors short jobs and continuous arrival of short jobs causes the long jobs to starve.

RR

It means Round Robin. It is also a preemptive algorithm. It uses the concept of time quantum, time slice or time slot. A time quantum is a small period of time for which a process is run and then suspended.

This algorithm selects a job whose wait time is largest and runs it, setting an **interval timer**. After the time quantum, the timer generates an interrupt and the process is unloaded and a new selection is made by the algorithm.

This algorithm is used heavily on time shared systems, such systems need to be much responsive and a small time quantum gives smaller response time. But if the time quantum is very small then the CPU will spend most of its time switching between the processes and it will become inefficient. Hardware plays a vital role here and the algorithm acts much efficiently if hardware supports fast process switching.

Priority Scheduling

It comes in both preemptive and non-preemptive versions.

Every running process is given a value. The algorithm selects the process with highest priority. If more than one process have the same priority then FCFS is sued to resolve the conflict.

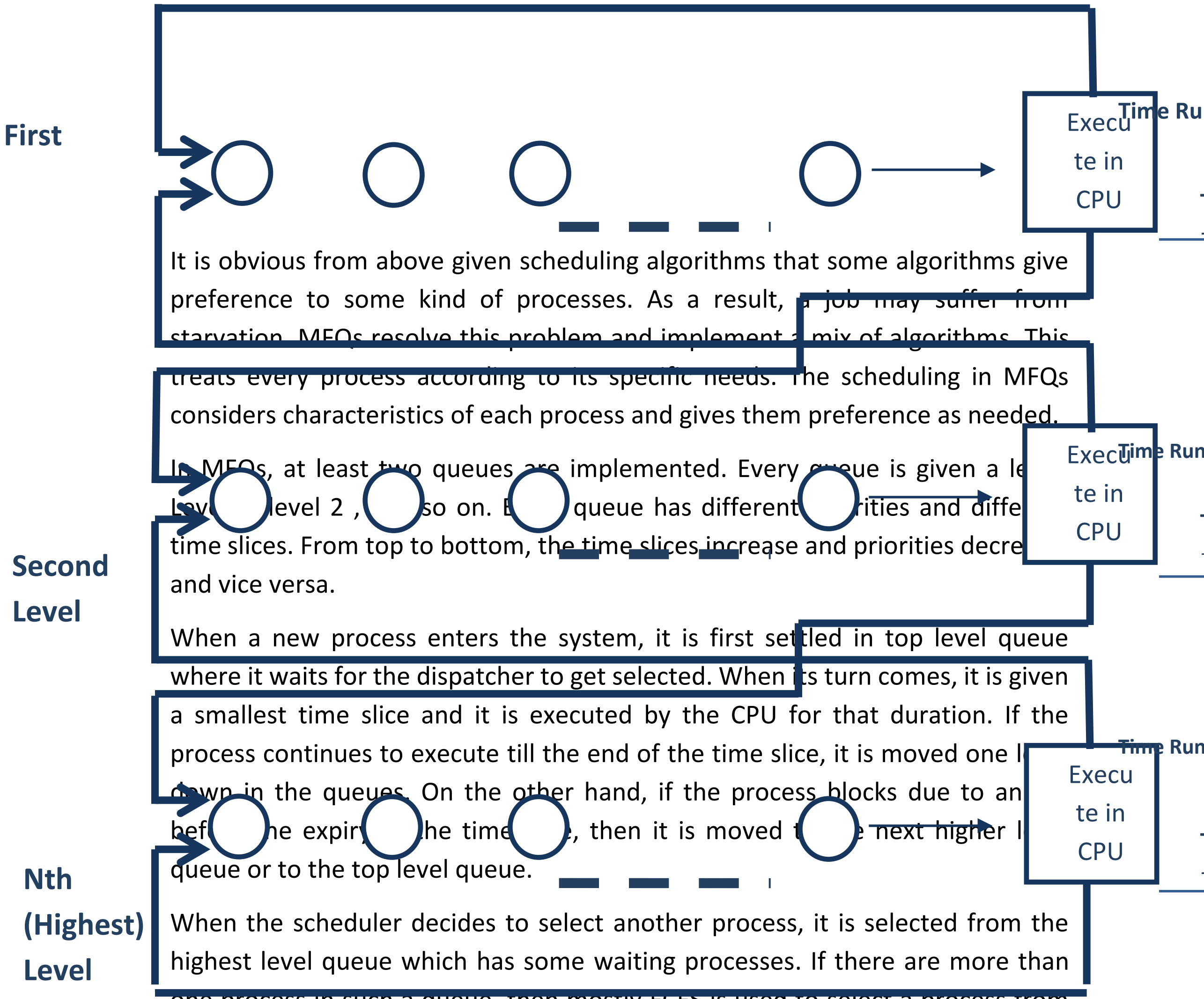
A process gets priority on the basis of different varying factors. The assignment can be based upon:

1. Process characteristics
    a. memory usage
    b. I/O frequency
2. Characteristics of user executing the process
3. Usage Cost
    a. more CPU time increases the cost
4. User assigned priorities

Some priorities are dynamic, some are static. Dynamic priorities change with system situations, like amount of running time. While static do not change, like user running a process.

On some systems, the high priority number means high selection priority. But on others, a lower priority number means high selection priority.

MFQs



It is obvious from above given scheduling algorithms that some algorithms give preference to some kind of processes. As a result, a job may suffer from starvation. MFQs resolve this problem and implement a mix of algorithms. This treats every process according to its specific needs. The scheduling in MFQs considers characteristics of each process and gives them preference as needed.

In MFQs, at least two queues are implemented. Every queue is given a level: level 1, level 2, and so on. Each queue has different priorities and different time slices. From top to bottom, the time slices increase and priorities decrease and vice versa.

When a new process enters the system, it is first settled in top level queue where it waits for the dispatcher to get selected. When its turn comes, it is given a smallest time slice and it is executed by the CPU for that duration. If the process continues to execute till the end of the time slice, it is moved one level down in the queues. On the other hand, if the process blocks due to an I/O before the expiry of the time slice, then it is moved to the next higher level queue or to the top level queue.

When the scheduler decides to select another process, it is selected from the highest level queue which has some waiting processes. If there are more than one process in such a queue, then mostly FCFS is used to select a process from the queue. If a process arrives in a higher level queue and there is an executing process from lower level queue, then lower level executing process will be preempted as higher level processes have high priority.

All CPU bound processes remain in lower level queues and all I/O bound processes remain in high level queues. I/O bound processes get high priority but small time slice as they will soon generate an I/O and then block. CPU bound processes get lower priority and high time slice as they need more CPU time to complete. Sometimes, a CPU bound process can suffer from starvation as continuous arrival and/or completion of I/O of I/O bound processes will not let a CPU bound process to get executed. To overcome this, a CPU bound process can be transferred to a high level queue to get increase its priority if it has been in the lower queue long enough.

MFQs are complex and flexible. Following factors are considered while designing and perceiving an MFQ mechanism.

1. Total number of queues to be implemented
2. Size of time slice set for each queue
3. The selection of algorithm to select a process from within every queue.
4. The set of conditions which will cause a process to move to a higher level queue.
5. The set of conditions which will cause a process to move to a lower level queue.
6. The scheme to use: preemptive or non-preemptive. In other words, is it needed to preempt an executing lower level queue process if a higher level queue has an incoming process?
7. Decide the placement of processes in a queue when they arrive.

Process Attributes

The PCB

PCB means process control block. It is a data structure maintained by OS for every running process. It contains data and information about a process necessary for the OS. This information includes:

1. Run state and scheduling information
2. Memory management

3. Hardware state
4. Process signaling
5. Access control
6. I/O information
7. Allocated resources
8. Accounting
9. And more

1-Run state and Scheduling Information

Every process in computer undergoes many states. The PCB of each process records the current state. It allows the OS to quickly determine the state of a single process and number of total processes in a particular state. The OS also maintains individual lists for each state. Each list contains the processes in that particular state. These lists are also called queues. PCB of a process contains a pointer to the corresponding queue. If a process is in blocked state, then the information about its blocking event is also maintained.

PCB also contains information about:

1. CPU allocation
2. last allocation to the CPU
3. priority level
4. and more

2-Memory Management

A process is loaded in memory and then it is loaded in CPU to run. There can be many processes present together in memory. A process's data must be protected from other processes. A process must not interfere with the memory in control of another process. To do this, the record of a process's allocated memory is maintained in PCB. When a process is loaded, a register in CPU points to this record. Now this process must stay within the defined limits. In some cases, these records are copied into CPU's special memory management registers.

3-Hardware State Information (context switch)

The PCB also contains hardware state information. This information is used when an outgoing process is reloaded. This allows the process to resume. This process is also known as context switch.

This information includes:

1. Program counter
2. Accumulator registers
3. General registers
4. Stack pointer Registers
5. ALU condition codes
6. Execution priority levels

All this and many other pieces of information are stored for every process in memory, in any state except running.

4-Signaling Information

A signal is similar to an interrupt coming from OS. A process or the OS itself would create it and one or more processes receive it. OS must provide some method to facilitate signalling. On the receiving side, the receiving process would ignore it, or execute a routine as a response to it or it would terminate.

Some signals are set to be sent in future. For example, a timer interrupt is sent only when its set time period expires.

OS implements a mechanism on how a process responds to a signal. OS also keeps track of the signals which are sent to a process and are still waiting for the response.

5-Access Control

The following access control information is stored:

1. information about the access rights of a process
2. information about the rights assigned to any object created by that process

This information includes:

1. Passwords
2. Capabilities
3. ACLs[access control lists]

Its contents vary with the type of access control mechanism in use.

6-Input and Output Information

In most OSs, every device or file must be opened before it is used. A process does this for the sake of some I/O. Every file or device accessed and opened by a process should be maintained in a list by the OS. For every process, a table called Open File Table is used to maintain information about each opened file and I/O device. This table includes:

1. List of all opened files
2. List of all opened devices
3. Process's root directory
4. Working directory

7-Other Attributes

Process Id

It is a unique number assigned to a process. its purpose is identification of the process

Parent Process Id

Sometimes, a process is created by another process. In this case, to maintain proper hierarchy; all the children contain there parent's id.

Child Process Ids

The parent contains ids of all of its children.

Process Group Id(s)

Sometimes, some processes become a part of a group other than their ancestral group. Then a number, group id, is used to identify their group.

User Id

This is the number identifying the user who is executing the process.

Effective User Id

Sometimes a user executes a program which is not written or owned by itself. Then this user uses someone else's access rights to run this process. In this case, the effective user id determines which user access rights are actually in action.

User Group Id(s)

Sometimes some users have similar access rights on a common object. In this case, the access is controlled by using user group ids. A single process can belong to one or more user groups

Effective User Group Id(s)

When many user groups access the same object, then effective user group id determines which user group rights will be enforced.

Account Id

Mostly users and accounts are the same for an OS. But in some cases, users are the human beings accessing some object. And an account means to record usage of a resource.

Priority Level

It is a number used to record process' scheduling preference. The scheduler will select a process with highest priority level.

Elapsed CPU Time

The total amount of CPU's time utilized by the process till now.

Start Time

It is the timestamp recording the time when process was created and was put in the new/held state.

Scheduled Start Time

The timestamp when a process will go out of its held state and loaded on CPU for execution. A process will not remain in the held state when its scheduled start time has started.

Maximum CPU Time

The maximum amount of CPU time which a process will use till it is terminated.

Memory Allocation
Total memory assigned to the process.

Command
When a user executes a process through a terminal device, the terminal number identifies it. It is used in remote processing situations. A terminal device can be: monitor and keyboard, terminals, or telnet sessions and alike.

Termination Status
When a process has finished execution and it is terminated, the termination status is stored. This status tells how the process terminated; with error, abort call or natural exit.

Process Supervisor Calls

Process supervisor calls are used when:

1. A process is created
2. A process is terminated
3. Signals

1-Process Creation
While creating a process, PCB is created. It contains data structures needed for the execution of the process. The PCB is then populated with the required information. This information is loaded in PCB using:

1. The parameters are passed with the call. The PCB gets information from these parameters.
2. If a process is a child of some other process, the properties of the parent can be used to populate PCB.
3. The characteristics of the user who created the process, can also be used to load the PCB.
4. The OS can also populate PCB by considering the overall system load at that time.

Before executing the process for the first time, during its creation, an initial hardware state for the process must be created, we call it initial ***hardware runtime state***.

Along with this initial hardware runtime state, a memory image for the process is also created. CPU registers are also initialized.

The process creation and loading the PCB with appropriate values can be done in two ways:

First method: This approach is used when creating a totally new process, which is not the child of an existing process.

It is done using a supervisor call designed specifically for this purpose. This call is the **system** supervisor call. When this call is executed, a file containing the process code is specified. This file is an executable file. It can be in different formats and an OS may support many formats for such files. When this file is executed, the process's memory is loaded based upon this file. This system call can also set the initial hardware runtime state.

Second method: it is to separate the process creation and loading the PCB from each other. It is suitable for a process which is a child of another process. Two commands, **fork and exec**, are used separately.

Fork is a call which creates a duplicate of the parent process. The child is a clone of the parent, except a special value. This value is the **return value** of the process. The OS checks this value and determines the hierarchy of processes.

After executing the fork, the *exec* is executed later on when needed. Mostly it is executed immediately after the fork call to load the child with its own program. In this way the system call discussed earlier is replaced. But this second method is more flexible because the exec can be executed more than once. So, without creating new processes, an old process can be loaded with new code.

A process also needs **resources** in addition to the memory and CPU. These resources are mostly a subset of the parent process's resources or the user's resources.

2-Process Termination

A process is terminated using one of two ways.

a) Exit

When a process has fully executed its code it is terminated by the OS. This is called exit.

b) Abort

Means that a process still has some code to execute, but due to some reasons it is desirable to terminate its execution and clear its memory. For example, in case of deadlocks. This abort signal is issued from either from

- OS
- system administrator
- Another process.
    - parent process
    - process in the same group
    - specific processes of the user
    - Processes of the administrator can abort a process.

Most systems allow the parent to run concurrently with the child.

On some systems, when there is a child, the parent may wait for its completion. When child finishes, the parent resumes execution. This is done using a *wait* call. This call also has a mechanism to send back the terminating status of the child to the parent.

Some OS terminate all the children of a terminating process. If there are grand children under the children, then they are also terminated before terminating the children. We call this cascading termination.

3-Process Signaling

These calls are used to send and receive signals between two communicating processes for data sharing and synchronization. We present details about them in chapter 3.