A Survey on Different Scheduling Algorithms in Operating System

G. Chitralekha

Abstract Allotting resources for process execution is one of the important procedures to be done in any operating system. Mostly in battery operated systems, where the resources are scarce, it becomes more challenging to have the resource management done, having promised all the processes equal opportunity of resource accesses. There are several algorithms for resource allotment developed and each one of the algorithms thrives to do resource allotment to all the processes dynamically. In this paper, a detailed review of the existing scheduling techniques for resource management has been done. Also, state-of-the-art scheduling mechanisms have been discussed and a comparison has been done among the existing state-of-the-art mechanisms. Finally, the challenges faced by scheduling techniques and how to overcome is also covered.

Keywords Preemptive and non-preemptive scheduling · Fair-share scheduling · Controlled preemptive earliest deadline first (CP-EDF) scheduling · Lottery scheduling · Dynamic heterogeneous earliest finish time (dHEFT) scheduling · RR scheduling

1 Introduction

Resource allotment is crucial in all operating systems where system resources such as processor, RAM/ROM, I/O devices are assigned to processes and/or threads. This allocation of resources is based on a mechanism called scheduling performed by a program called scheduler. There are variants of scheduling algorithms which are derived to yield utmost throughput and the best QoS. These algorithms make sure that the processes/threads are assigned with the right resources at appropriate times having to meet the deadline. Scheduling algorithms play a significant role in systems where multitasks are run simultaneously. Multitasking is a condition where the processor has to switch from one process/task to another in order to perform

G. Chitralekha (✉)
Department of Electronics and Communication Engineering, CMR Institute of Technology, Bengaluru, India
e-mail: chitralekha.g@cmrit.ac.in

S. Smys et al. (eds.), *Computer Networks and Inventive Communication Technologies*, Lecture Notes on Data Engineering and Communications Technologies 58,
https://doi.org/10.1007/978-981-15-9647-6_50

the execution. There have to be some means to guide the processor in choosing any process/task for execution and also to divide the processor's time amongst different processes and tasks. The decision based on which any process/task is taken up for execution is termed as process/task scheduling. Multitasking needs scheduling. OSs have the capability to handle multiple tasks/jobs at the same time; hence, multiprocessor systems are those which have the capacity to operate numerous processes simultaneously. Such systems contain multiple processors and can take up multiple process execution at the same time. An OS being able to carry 'n' processes in storage and toggle the execution from one to other is called multitasking. This is where CPU switching from one to other tasks/job occurs. Several types of multitasking exist such as co-operative, preemptive and non-preemptive multitasking. In first of its type, a new job will get a chance to execute only when the other job which is currently being executed voluntarily suspends the execution and releases the CPU to be used by other jobs. Here, jobs can occupy the CPU for any amount of time. In case the job is non-co-operative, then the other jobs might have to wait for unknown prolonged time waiting for its turn to get the processor. In the second case, that is preemptive multitasking, it is the responsibility of the scheduler to make sure that all jobs get an equal chance of the processor's time for execution. Depending on the implementation of the scheduling algorithm, the amount of the processor's time, a job gets will be decided. Preemption means the temporary stoppage of a job's execution which might be caused due to the priority of the current job and/or the new job that just arrived and also due to the time slot. In the last case, that is non-preemptive multitasking, any job that has occupied the processor is let to utilize all the processor time until it is done with the execution and/or changes the state waiting for a second input like I/O. The job scheduling decision will come into picture when a job changes its state from running to the ready state, or probably from running to blocked/wait state, also may be from blocked/wait to ready and completed state. Based on the following factors, one of the scheduling algorithms will be chosen, utilization of the processor, its throughput, turnaround time, wait and response time. In the next section, let us see different types of scheduling techniques in detail and compare their performances.

2 Background

Scant resources lead to scheduling policies. Scheduling is fundamentally done to diminish the wait time and the time consumed in context switching [1]. Poor resource management makes way to performance degradation of the framework. It is the responsibility of the scheduler to share the resources amongst all processes/tasks in a fair manner [2]. There are different types of scheduling algorithms based on the type of system. For example, in systems where batch processing is made, algorithms like FCFS, SJF and SRT are adopted where the focus is on throughput, turnaround time and utilization of the processor. Frameworks (systems) are intelligent and continuous, embrace calculations, for example, RR planning, fair-share scheduling, priority scheduling and lottery scheduling. Especially for ongoing frameworks, rate monotonic and EDF booking are gone for [3].

In first come first serve (FCFS) algorithm, scheduler designates CPU time to the procedures dependent on the request in which they enter the ready line. The first entered process is addressed first. Alike, ticket reservation routine, where individuals need to remain in a line and the primary individual remaining in the line, is addressed first. It is additionally called as first in first out (FIFO) where the procedure which is placed first into the prepared line is adjusted first [4]. Shortest job first (SJF) algorithm sorts the ready line each time a procedure gives up the CPU to pick the procedure with most brief evaluated run time. In SJF, the procedure with the most limited assessed run time is booked first, trailed by the following briefest procedure thus on [5]. In shortest remaining time (SRT) algorithm, scheduler sorts the prepared line when another procedure enters the prepared line and checks whether the execution time of the new procedure is shorter than the staying of the absolute assessed time for them at the present executing process. In the event that the execution time of the new procedure is less, the as of now executing process is acquired and the new procedure is planned for execution [6]. In round -robin (RR) planning, each procedure in the prepared line is executed for a pre-characterized scheduled opening. The execution begins with getting the principal procedure in the prepared line. It is executed for the pre-characterized timeframe, and when the pre-characterized time slips by or the procedure finishes, the following procedure in the prepared line is chosen for execution. This is rehashed for all the procedures in the prepared queue [7]. Priority-based scheduling guarantees that a procedure with high priority is overhauled at the most punctual contrasted with other low need forms in the prepared queue [8]. In fair-share scheduling algorithm, the allocation of system resources is done dynamically [9]. In a lottery scheduling [10] algorithm, the procedures gain admittance to the framework assets on the basis of a lottery ticket. The person who draws the most lotteries for the assets will naturally win the entrance.

3 State-of-the-Art Resource Management Algorithms

The scheduler is the kernel service that implements the scheduling algorithm. The scheduling mechanism will be applied when any job changes its state from running to ready state or when it is preempted. When a job with a high priority which is in the blocked or wait for state completes its input/output operation and changes back to the ready state, the scheduler picks it for execution if the scheduling policy in place is preemptive priority-based scheduling. In preemptive or non-preemptive multitasking, the process gives up the CPU when it enters the blocked/wait state or the completed state and switching of the CPU happens at this stage. Selection of a scheduling criterion/algorithm should consider the following facts, high processor utilization and throughput, minimal turnaround time and wait time and as well least response time. Figure 1 outlines the process transition through various queues that are kept up by OS in accordance with CPU scheduling, such as job/work queue (contains

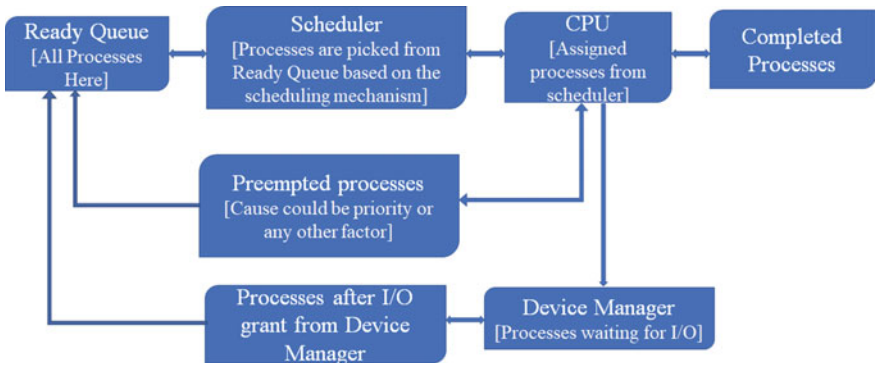

Ready Queue
[All Processes Here]
Scheduler
[Processes are picked from Ready Queue based on the scheduling mechanism]
CPU
[Assigned processes from scheduler]
Completed Processes
Preempted processes
[Cause could be priority or any other factor]
Processes after I/O grant from Device Manager
Device Manager
[Processes waiting for I/O]

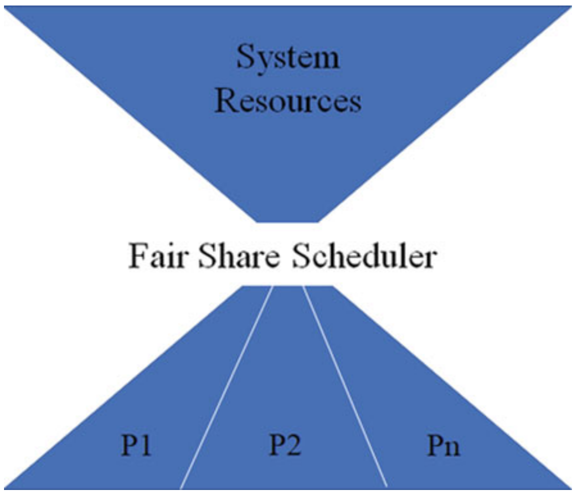
Fig. 1 Outline of process transition through various queues

all the procedures in the framework), ready queue (contains all the procedures, which are ready for execution and counting on CPU for their turn of execution) and device queue (contains the set of procedures, which are looking for an I/O device).

3.1 Fair-Share Scheduler

The scheduler in this algorithm allots the resources to the tasks/processes dynamically [11]. The resources are allotted such that no process/task waits for a long time in the queue for any resources. Each process/task is given a weight of some sort that tells the user's share of system resources as a part of the total usage of those resources. Figure 2 depicts the fair-share scheduler. In this algorithm, the scheduler partitions the

Fig. 2 Fair-share scheduler



system assets into portions, which are then allotted by process schedulers assigned to various fair-share groups.

3.2 Fixed Budget

In this type of scheduling, as the name suggests, there is a budget fixed for each process/task. When any process/task utilizes any resource, it will be charged for the usage and the budget allotted to that particular process/task will reduce by a certain amount. When that process/task runs out of the budget, then it will have no access to the system resources further. So there is a fixed budget for all the processes/tasks of a system, exceeding which they may lose access to the resources. The process/task with more budget will be allowed to use more of the system resources and the one with less budget will have limited access to the system resources.

3.3 VM-Aware Fair Scheduler

Scheduler fairness plays a significant role in heterogeneous systems where multiple processes with variable computational competence run. In order to maintain fairness in virtual machine (VM) systems, virtual machine-aware fair scheduler was proposed. Virtual machine-aware fair scheduler overcomes issues pertaining to preemptive routine that go after assets in a VM condition. The performance of virtual machine-aware fair scheduler for preemptive processes is significant as compared to other VM credit base algorithms [12].

3.4 Practical Fair Scheduler

Practical fair scheduler meets the real-time processes' requirements. Mostly, such algorithm is common in soft real-time processes, where the deadline miss might not turn out to be fatal. Especially for migration and placement techniques, such schedulers are used in multi-core systems. The technique of migration is used to manage resource fairness among the processes which in turn leads to the enhanced performance by default [13].

3.5 Round Robin (Virtual Time)

The algorithm that is a blend of fair sharing and RR scheduling is virtual time RR. According to fair sharing technique, each process will have a definite weightage to utilize system resources and round-robin scheduling will have a fixed time quantum for all processes/tasks. If the tasks exceed the limit of proportional shares of assets, then it is placed back again to the beginning of the queue and its processing will start all over again.

3.6 Lottery Scheduling

Another way of performing fair scheduling is lottery planning that utilizes the assets that are randomized, reservation strategy for proficient response time and gives execution control of the procedures. Fundamentally proportional share resource allocation algorithms are intended to address the issues of applications, indeed real time, in OSs. In lottery scheduling, the asset assignment to the procedures or clients is done based on a lottery ticket. The client gets the portion of framework assets that is relative to the held tickets by the user [14]. A lottery ticket is drawn by a client or procedure that needs assets for execution. Just those procedures can partake in the lottery during run time while the procedures in different states are not qualified for the lottery. Fundamentally the system assets will be allocated for those clients who win the lottery tickets. The likelihood K to win a ticket is essentially n/N where n speaks to the ticket that a client holds and N is the total accessible tickets so the condition becomes $K = n/N$. There have been numerous adjustments in the lottery scheduling algorithms to guarantee its best execution to asset distribution. One of these expansions in lottery planning is the ticket trade technique in which procedures trade their assets with one another through ticket trades.

3.7 Stomp Scheduling

One of the scheduling approaches that are like lottery planning is stomp (stride) scheduling [15]. This scheduling is fundamentally intended to accomplish relatively high all through rate and lower response time. In this planning, clients hold various tickets that are in an extent of assets of contending clients and have a period stretch called stride which is contrarily relative to the distribution of tickets that serves to choose how quick it arrives in a usable state. Likewise, a pass is related with each client and the client with the least pass is planned for that time stretch and increased by work stride. Its assessment should be possible by two different ways: one is by utilizing reenactments and another is by actualizing models for Linux Kernel.

3.8 Round-Robin Scheduler

The RR algorithm is one of the additional ways to deal with deciding faster turnaround time, having unknown task completion times. In this algorithm, a predefined time slice is dedicated to all tasks. If any task can complete its execution before the time given within the time slice, then it has to let go of the resources which the task had occupied for execution. Otherwise, the task will be placed in the ready queue again but in the last waiting for all the earliest tasks to get the benefit of the time slices dedicated to each one of them [16]. RR essentially utilizes the priority that is a blend of run time and arrival time of procedure. The RR relies upon the time slice that is dispensed to each procedure and there are two circumstances of the time slice. On the off chance that the time quantum is interminable, and at that point, there will be FCFS strategy, yet if the time slice is zero, at that point would be in limit and the processor has no holding up line so that all procedures execute [17].

3.9 CP-EDF Scheduling

CP-EDF stands for the controlled preemptive earliest deadline first. In this type of scheduling algorithm [18], the number of preemptions that may take place in a system is limited. Preemptions are temporary halt of execution of any process/task when that particular process/task switches to wait for state because of input from I/O device or it may also be a condition where the preemption happens because a process/task having a higher priority than the currently executing task comes to the ready state. No matter what is the condition for preemption, whenever it occurs, context switching happens which incur processor's time and energy. In order to avoid the number of preemptions that may occur in a system, CP-EDF was proposed. When the number of preemptions is less, then CP-EDF behaves as a fully preemptive EDF algorithm. Thus, the context switching is less and the processor's time and energy are saved. When the number of preemptions is high, then the algorithm behaves as non-preemptive EDF. In non-preemptive EDF scheduling, the deadline of all tasks will be calculated, based on which a process/task will be chosen which has the least deadline to finish. Further processes/tasks will be chosen based on their deadlines respectively in ascending order. In case of fully preemptive EDF scheduling, preemption will happen when the deadline of other task is less than the deadline of the currently executing task.

3.10 Save Energy Scheduling

When a process/task completely executes, it fetches some utility and that is run time-dependent. Shorter the running time, better the utility. Hence, faster the processes/tasks are executed, got maximum utility for which a time function proposed by Jensen et al. [19] describes the interlink between the two, the run time and the task utility.

3.11 Harvest Energy Scheduling

The lazy scheduling algorithm (LSA) modifies the worst-case processing time by modifying recurrence of the processor as indicated by tasks' energy utilization. This algorithm depends on a solid presumption that the most pessimistic scenario execution time of an undertaking is connected with its vitality utilization legitimately and that this supposition is unrealistic. Another scheme called as late as possible, which is a fixed-need scheduling strategy that defers the execution of employments as far as might be feasible. Another scheme called as soon as possible plans jobs as quickly as time permits when there is accessible vitality in the battery and suspended execution to renew the vitality expected to execute one-time unit. Under requirements of energy and time, ASAP ends up being ideal. Moreover, so as to show signs of improvement framework execution and less energy utilization, some mechanisms take dynamic voltage recurrence scaling into thought. The HADVFS mechanism [20] streamlines the framework execution and enhances the energy usage further dependent on the EADVFS calculation. Nonetheless, in some continuously installed frameworks with high unwavering quality, DVFS and related calculations can expand the run time of tasks, influence the run time trait of the framework and in this way decline its dependability.

3.12 Hybrid Scheduling Algorithm

This algorithm comprises of two phases [21]. The primary stage to isolate the approaching errand chart into two almost equivalent size subtask diagrams dependent on the otherworldly parceling calculation. This stage attempts to accomplish load adjusting and limit information conditions among subtasks. The second stage at that point plans each parcel applying hereditary algorithm (GA), so as to limit the makespan without abusing priority imperatives. For limiting the absolute force utilization, the scheduler attempts to relegate a chosen subtask to a processor that expends lower power. On the off chance that the chose subtask cannot be booked on the objective lower power processor; in view of infringement of the cutoff time, the proposed technique chooses an elective processor that expends higher capacity to execute the subtask inside the cutoff time.

3.13 Heterogeneous Earliest Finish Time (HEFT) Scheduler

Indeed in heterogeneous systems, tasks that are waiting to be executed by numerous processors are given weightage according to their priorities; hence, this phase is called as prioritization phase. In relation to the first phase output, the most efficient processor is selected for task execution; thus, this phase is called processor selection phase.

The weights assigned to the tasks are actually done with respect to the execution time of processors, cost incurred in the task communication and the prioritization value of the previous task. In the second phase of the processor, the algorithm makes a randomized decision based on a threshold value to cross over among processors. In addition, the cost incurred during task communication is also accounted for, in calculating the threshold. In case of static heterogeneous scheduler, it consists of two compile-time phases, job prioritization phase and processor selection phase. Firstly, the tasks/processes will be assigned priorities depending on the critical path length of a given task to exit task, computation and as well communication cost included. The scheduler will arrange the tasks according to their priorities in descending order. During the next phase, an appropriate task is selected for execution. dHEFT [22] assumes both fast and slow cores. And these cores are used to maintain information about the task costs. During run time, dHEFT learns about the task costs, the mean cost of each task for each core type is computed and then the core that finishes the task at the earliest is found. To find the efficient processor which can complete execution at the earliest, dHEFT maintains a list per core and the tasks in the wait queue are also included. When any task enters the ready queue, dHEFT first places that task in the re-arranged ready queue, then the task with high priority is selected and dHEFT checks if there are execution time records for this task. If the records number is enough, then the estimated cost of the task is said to be stable. Using that estimated execution time, task is assigned to the fastest executor. In the event that the quantity of records does not get the job done, at that point the assignment is booked to the soonest agent of this center sort to get another record of that task-type and center sort execution time. In all cases, dHEFT refreshes the historical backdrop of records on each undertaking execution to adjust for stage changes in the application.

3.14 User-Centric Resource Management Framework

In this algorithm [23], charging patterns of the user's battery and state of charge (SOC) are considered. Figure 3 shows the outlook of the proposed approach that can be categorized into three phases such as (i) offline training for state of charge, (ii) online inferring for asset allocation decisions based on the trained user, and (iii) virtual user and state of charge simulation for the purpose of evaluation. Firstly, analytical models are built in order to predict user's charging pattern that determines the probability of plugin/out, charging/discharging behavior of battery which estimates current state of charge. These models are inferred during run time to make absolute asset assignment decisions that has positive influence on performance and energy consumption of the device.

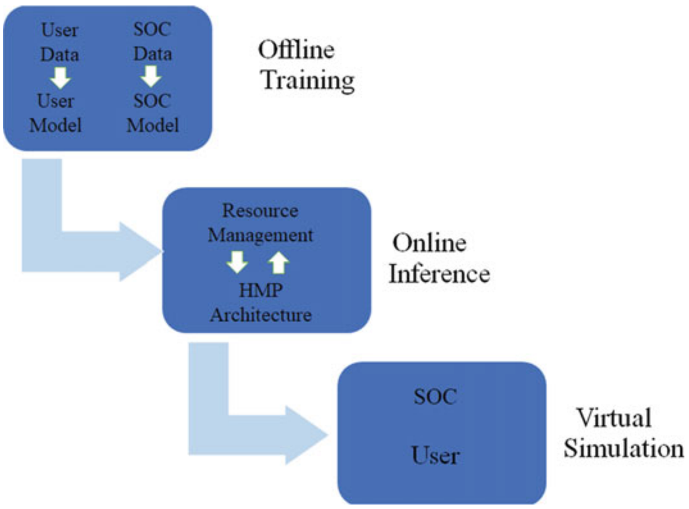


Fig. 3 Outline of the user-centric framework

3.15 Load Balancing-Based Algorithm

In this algorithm [24], there are two phases. Initially, it is assumed that there are some virtual machines available to which already some number of tasks are assigned. In Phase I, the time needed by a task to be executed on a virtual machine is calculated. During which another factor is also taken into consideration and that is the ready time of any virtual machine that would host the task. This is a recursive process as the ready times of virtual machines keeps changing. After these calculations, the task with the highest completion time will be chosen to be paired with a virtual machine. Once a task gets assigned to a virtual machine, it will be deleted from the pool of tasks. This recursion will take place as long as all the tasks have been assigned to virtual machines. In Phase II, the emphasis is laid upon load balancing, where the workload of the resources is taken into consideration. This phase also occurs recursively. Firstly, a virtual machine Va that bears the highest workload is considered. Then the tasks assigned to this virtual machine are looked at and the one whose completion time is smallest is chosen. This is the same task Ta that got assigned to the virtual machine via Phase I. During Phase I, before pairing up of the task Ta with any virtual machine, Ta's completion time on all the virtual machines would have been calculated. Here, machine Vb would be selected for pairing with Ta that has the highest completion time. Out of all the tasks assigned to Va, the last task's completion time is noted for further analysis. A comparison is conducted with the following, the maximum completion time of Ta on Vb if found to be less than the last assigned task's completion time to Va, then the task Ta is removed from Va and paired with Vb. This change in the pairing will not affect the completion of the other tasks on either Va or Vb. However, the available times for both Va and Vb needs to be recalculated. This reassignment of a pairing of tasks and the virtual machines are repeated until there is no chance of any more rescheduling of the tasks that were originally assigned to Va. Thus, no resources will be neither overloaded nor idle.

4 Analysis Work

Table 1 lists out most of the scheduling mechanisms along with their advantages and disadvantages.

5 Results

Here, the scheduling patterns are shown for some of the algorithms like first come first serve (Fig. 4), round robin (Fig. 5), virtual round robin (Fig. 6), shortest job first (Figs. 7 and 8) and shortest remaining time (Figs. 9 and 10) scheduling algorithms. These algorithms were run on AnimOS CPU scheduling, an animated tool for CPU scheduling. Five processes are considered, Process1, Process2, Process3, Process4 and Process5 with arrival times 2, 3, 1, 5 and 3, CPU burst times 4, 2–6, 5, 6 and 3–6 and IO burst times 5, 1, 3–6, 3, respectively. Time quantum for round robin is assumed as 3.

6 Conclusion

In this paper, an overview is directed in which the working of various scheduling algorithms has been talked about and how asset management challenges in OS is solved through these algorithms. These scheduling algorithms function admirably in various conditions and situations. A correlation has been done against all algorithms dependent on their exhibition standards, for example, throughput, holding uptime and response time. Fundamentally, all calculations are intended to accomplish reasonableness among forms; however consistently, there lies some distinction in throughput, hold up and response time of various scheduling mechanisms. All the mechanisms discussed excel in different scenarios based on the type of system in which it is deployed. Yet the research is still on in finding out the best scheduling mechanism (suitable) for a system. A similar survey can also be extended to mobile operating systems.

Table 1 Analysis of scheduling algorithms

Scheduling mechanism	How it works?	Advantage	Drawback
Fair-share scheduling	Proportional resource allotment for all processes	All processes get equal opportunity to claim resources	Big sized processes might have to wait for a long time to complete
Fixed budget scheduling	A fixed number of accesses is given to all processes, which reduces by a certain amount after every access	Processes with high budget will have more access to resources	Processes with low budget will have limited access to resources
VM-aware fair scheduling	Each virtual machine is assumed to have some amount of processes assigned already	Shorter processes will get paired with virtual machines quicker	Some processes might have to wait for a long time to get paired with a virtual machine
Practical fair scheduling	Proportional resource allotment for all processes in real-time environment	Best suited in soft real-time systems where missing deadline does not turn out to be fatal	Not suitable for hard real-time systems
Virtual round-robin scheduling	Blend of fair-share scheduling and round-robin scheduling	Processes with limited access to resources will get another chance to complete as they are put back in the ready queue after they exceed their proportion of resources	As priority of the processes are not given consideration, higher priority processes may miss deadline
Lottery scheduling	Processes in run time have to win lottery to get access to resources	Lottery trading is allowed between processes	If priority not considered then deadline misses may happen for sure
Round-robin scheduling	All processes will have equal time quantum at every iteration	All processes are treated equal irrespective of their other attributes	Processes with high CPU bursts will have to go through too many iterations to complete

(continued)

Table 1 (continued)

Scheduling mechanism	How it works?	Advantage	Drawback
Controlled preemptive EDF scheduling	When preemptions are less, fully preemptive conditions are considered otherwise fully non-preemptive	Lesser the number of preemptions, lesser is the context switching, hence saved processor time	EDF scheduling does not perform well under over load conditions
HEFT scheduling	Two phases are involved. In the first phase, process with highest priority is chosen, and in the second phase, the most efficient processes is chosen based on the task communication cost	Efficient fast processors are chosen for execution	Runs on single core type
LBB scheduling	Resources are allotted based on the virtual machine ready time and the workload of the resources	Resources are not overloaded	Virtual machine ready times keeps changing as and when new tasks gets added

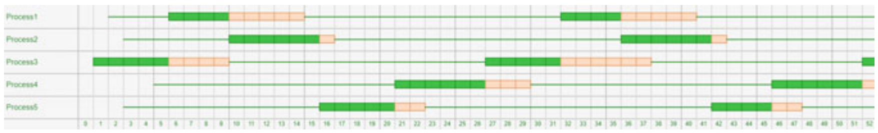


Fig. 4 First come first serve scheduling

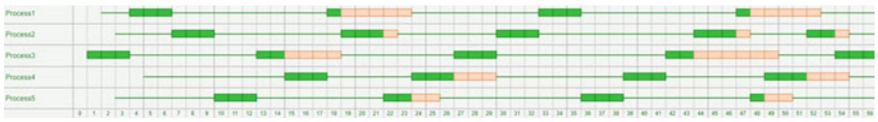


Fig. 5 Round-robin scheduling

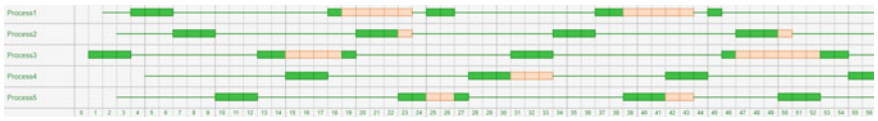


Fig. 6 Virtual round-robin scheduling

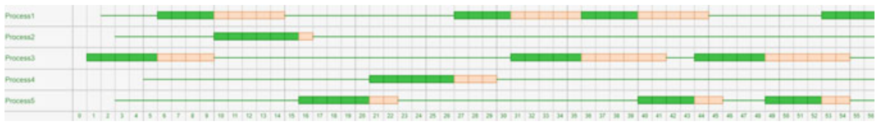


Fig. 7 Shortest job first scheduling (with different arrival times)

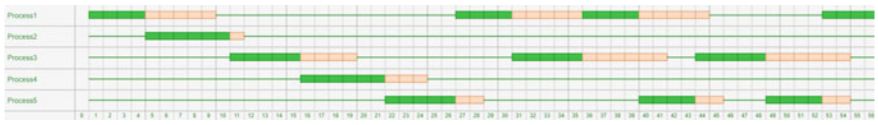


Fig. 8 Shortest job first scheduling (with same arrival times)

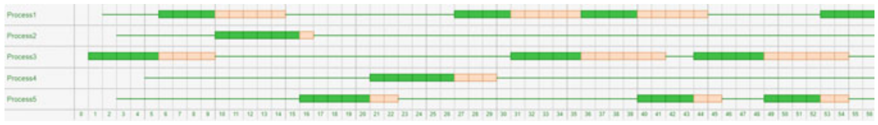


Fig. 9 Shortest remaining time scheduling (with different arrival times)

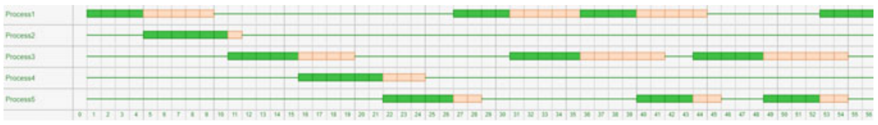


Fig. 10 Shortest remaining time scheduling (with same arrival times)

References

1. Helmy T, Dekdouk A (2007) Burst round robin as a proportional-share scheduling algorithm. IEEEGCC, 2007
2. Shreedhar M, Varghese G (1996) Efficient fair queuing using deficit round-robin. IEEE/ACM Trans Netw., Jun 1996
3. Silberschatz A, Galvin PB, Gagne G (1998) Operating system concepts, vol 4. Addison-Wesley, Reading
4. Uuganbaatar N (2010) Optimality of first-come-first-served: a unified approach. (2010, December)
5. Chung J, Chatterjee D (2013, August) Enhanced shortest-job-first memory request scheduling
6. Alsadeh A, Yahya A (2008) Shortest remaining response time scheduling for ımproved web server performance. In: 4th International Conference, web ınformation systems and technologies, WEBIST 2008, Funchal, Madeira, Portugal, 4–7 May 2008
7. Trick MA, Rasmussen RV (2008) Round robin scheduling—a survey. Eur J Oper Res 188(3):617–636. https://doi.org/10.1016/j.ejor.2007.05.046
8. Son SH, Park S (1994) A priority-based scheduling algorithm for real-time databases. J Inf Sci Eng 11:233–248
9. Mana SC (2012) Recourse management using a fair share scheduler. Int J Comput Sci Secur (IJCSS) 6:29–33

10. Waldspurger C, Weihl W (1994) Lottery scheduling: flexible proportional-share resource management. In: OSDI '94: Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation, November 1994
11. Kimbrel T, Schieber B, Sviridenko M (2006) Minimizing migrations in fair multiprocessor scheduling of persistent tasks. J Sched 9:365–379. https://doi.org/10.1007/s10951-006-7040-0
12. Choffnes D, Astley M, Ward MJ (2008) Migration policies for multi-core fair-share scheduling. ACM SIGOPS Oper Syst Rev 42(1):92–93
13. M. Cluster Suite (2008). https://www.clusterresources.com/pages/products/moabcluster-, and M. website. Suite.php
14. Topcuoglu H, Hariri S, Wu M-Y (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distrib Systems. https://doi.org/10.1109/71.993206
15. Jeffay K, Smith FD, Moorthy A, Anderson J (1998) Proportional share scheduling of operating system services for real-time applications. In: The 19th IEEE Proceedings ın Real-time Systems Symposium, pp 480–491
16. Mohanty PR, Behera PHS, Patwari K, Dash M, Prasanna ML (2011) Priority based dynamic round robin (PBDRR) algorithm with ıntelligent time slice for soft real time systems. Int J Adv Comput Sci Appl (IJACSA) 2(2):46–50
17. Isard M, Prabhakaran V, Currey J, Wieder U, Talwar K, Goldberg A (2009) Quincy: fair scheduling for distributed computing clusters. In: Proceedings of 22nd ACM Symposium on Operating Systems Principles, October 2009, Published by Association for Computing Machinery, Inc
18. Keerthanaa C, Poongothai M (2016) Improved priority based scheduling algorithm forreal time embedded systems. In: 2016 International Conference on Circuit, Power and Computing Technologies [ICCPCT]
19. Jensen ED, Locke CD, Tokuda H (1985) A time-driven scheduling model for real-time operating systems. RTSS 85:112–122
20. Abdeddaïm Y, Chandarli Y, Masson D (2013) Toward an optimal fixed-priority algorithm for energy-harvesting real-time systems. In: RTAS 2013 WiP, pp 45–48
21. Taheri G, Khonsari A, Entezari-Maleki R, Sousa L (2020) A hybrid algorithm for task scheduling on heterogeneous multiprocessor embedded systems. https://doi.org/10.1016/j.asoc.2020.106202
22. Chronaki K, Rico S, Casas M, Moreto M, Badia RM, Ayguade E, Labarta J, Valero M (2016) Task scheduling techniques for asymmetric multi-core systems. https://doi.org/10.1109/TPDS.2016.2633347
23. Shamsa E, Taherinejad N, Kanduri A, Pröbstl A (2020) User-centric resource management for embedded multi-core processors. https://doi.org/10.13140/RG.2.2.27763.89121
24. Shi Y, Qian K (2019) LBMM: a load balancing based task scheduling algorithm for cloud. In: IEEE Future of Information and Communication Conference