International Journal of Innovation and Scientific Research
ISSN 2351-8014 Vol. 12 No. 1 Nov. 2014, pp. 180-185
© 2014 Innovative Space of Scientific Research Journals
http://www.ijisr.issr-journals.org/

A Comparative Study of Scheduling Algorithms for Multiprogramming in Real-Time Systems

***Y.A. Adekunle*[1], *Z.O. Ogunwobi*[2], *A. Sarumi Jerry*[3], *B.T. Efuwape*[2], *Seun Ebiesuwa*[1], *and Jean-Paul Ainam*[1]**

[1]Department of Computer Science, Babcock University Ilishan-Remo, Ogun State, Nigeria

[2]Department of Mathematical Sciences, Olabisi Onabanjo University, Ago Iwoye, Ogun State, Nigeria

[3]Department of Computer Science, Lagos State Polytechnic, Ikorodu, Lagos State, Nigeria

ABSTRACT: Scheduling algorithms allow one to decide which threads are given to resource from moment to moment. Various process scheduling algorithms exist and this paper focuses on the scheduling algorithms used for scheduling processes in a multiprogramming system namely First-Come-First-Served (FCFS), Round Robin (RR), Shortest Job First (SJF), Shortest Remaining Time First (SRTF) and Lottery scheduling. Each algorithm has been discussed and a comparison was made on the basis of eight (8) parameters significant in processes scheduling. In fact, compared to other papers, this research made use of more parameters for the analysis. These parameters include CPU utilization, throughput, waiting time, response time, fairness, starvation, predictability and preemption. From this analysis, we showed that there is actually no scheduling algorithm satisfying the conditions of an ideal algorithm and concluded that further studies which improve current scheduling algorithms need to be done.

KEYWORDS: First-come-first-served, Round robin, shortest job first, shortest remaining time first and Lottery scheduling.

1 INTRODUCTION

In multiprogramming the sharing of computing time among programs is controlled by a clock, which interrupts program execution frequently and activates a monitor program. The monitor saves the registers of the interrupted program and allocates the next slice of computing time to another program and so on. Switching from one program to another is also performed whenever a program must wait for the completion of input or output. Thus although the computer is only able to execute one instruction at a time, multiprogramming creates the illusion that programs are being executed simultaneously, mainly because peripherals assigned to different programs indeed operate in parallel. Scheduling is the method by which threads, processes or data flows are given access to system resources. This is usually done to load balance and share system resources effectively or achieve a target quality of service. The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (executing more than one process at a time) and multiplexing (transmitting multiple data streams simultaneously across a single physical channel).

2 BACKGROUND

Before going further into the analysis of algorithms used to schedule processes for multiprogramming systems, a glance at what makes scheduling and multiprogramming systems is indispensable. That simply means a close definition and overview of concepts surrounding scheduling and a multiprogramming system.

Scheduling is a key concept in computer multitasking and multiprocessing operating system design, and in real-time operating system design. It refers to the way processes are assigned priorities in a priority queue. This assignment is carried out by software known as a scheduler. In multiprogramming systems, the running task keeps running until it performs an operation that requires waiting for an external event (e.g. reading from a tape) or until the computer's scheduler forcibly swaps the running task out of the CPU. Multiprogramming refers to multiple processes from multiple programs while multitasking is reserved to mean multiple tasks within the same program that may be handled concurrently by the operating system (OS). In fact, multiprogramming systems are designed to maximize CPU usage. The objective of multiprogramming is to have some process running at all times and to maximize CPU utilization [1], that is, keeps the CPU busy as much as possible by having it work on more than one program at a time. While in a single-processor system, only one process can run at a time; others must wait until the CPU is free and can be rescheduled, in multiprogramming systems, more than one program can run at a time. Therefore, to attain the objective in multiprogramming system, scheduling is a fundamental function for multiprogramming operating-system. The CPU, being one of the primary computer resources used in a multiprogramming system needs to be scheduled. And its scheduling is central to the operating-system design [2]. That is because the CPU scheduling will determine which processes run when there are multiple run-able processes (multiprogramming concept). In real-time systems, some waiting tasks are guaranteed to be given the CPU when an external event occurs. Real time systems are designed to control mechanical devices such as industrial robots, which require timely processing.

3 Related Works

Developing scheduling algorithms and understanding their impact in practice have been of interest since the early 1970s. Researchers have shown that, combining more than one scheduling technique improves performance [3]. In their paper [2], Neetu et al. presented a state diagram that depicts the comparative study of various scheduling algorithms for a single CPU and showed which algorithm is best for the particular situation. Their paper gave a representation on what is going on inside the system and why a different set of processes is a candidate for the allocation of the CPU at different times. In a nutshell, the objective of their paper was to analyse the high efficient CPU scheduler on design of the high quality scheduling algorithms which suits the scheduling goal. However, they were not able to arrive at a definite conclusion. Besides this work, Edwin et al. in [4] studied the scheduling of tasks in computer systems which utilize imprecise (partial) computations. In their system, tasks arrive randomly during run-time. Each task has two levels of computation time requirement: the full level computation requirement and the reduced level computation requirement. They finally devoted their attention to develop an explicit formula for the practically important performance metrics such as the normalized mean task waiting time, the mean task served computation time, and the fraction of tasks fully processed. Another paper closely related to our paper is [6]. In their paper, Ankur et al. made a comparison of existing scheduling algorithm on the basis of seven parameters namely preemption, complexity, allocation, application, waiting time, usability, and type of system. They also observed that there is no one algorithm satisfying all the seven basis parameters chosen for analysis and suggested needs for improvement. However, no direction for the improvement was provided and has been referred to as future works. In addition to these related papers, K.M. et al., in [7] surveyed research on scheduling algorithms, reviewed previous classifications of scheduling problems and presented a classification scheme. Using a uniform terminology for scheduling strategies and the new classification scheme, they identified trends in scheduling research and finally concluded their work by presenting a methodology for developing scheduling strategies. This and others papers [8, 9, 10, 11, 12], though not related to our paper, presented scheduling algorithms used in other computing fields.

4 Methodology

4.1 Scheduling Criteria

Scheduling criteria are used to compare scheduling algorithms in multiprogramming systems.

The criteria used in this paper include:

CPU utilization: This is the percentage of time that the CPU is busy. CPU utilization refers to a computer's usage of processing resources, or the amount of work handled by a CPU. Actual CPU utilization varies depending on the amount and type of managed computing tasks. Certain tasks require heavy CPU time, while others require less because of non-CPU resource requirements. CPU utilization may be used to gauge system performance. For example, a heavy load with only a few running programs may indicate insufficient CPU power support, or running programs hidden by the system monitor - a high indicator of viruses and/or malware.

Throughput: This is the number of processes completed in a unit of time. It is the average rate of successful processes completed over a specified period of time. It is measured in bits per second (bps).

Waiting time: This is the total amount of time that a process is in the ready queue waiting in order to be executed. Sometimes the waiting time could be short and at other times it could be long.

Response time: This is the time between when a process is ready to run and its next I/O request. It is the time from the submission of a request until the first response is produced (i.e., the amount of time it takes to start responding, but not the time that it takes to output that response).

Fairness: This is the criterion that ensures that each process has an equal chance of being executed as the other. With fairness brought to the fore no particular process has a so-called preference or unfair advantage over the others.

Starvation: This is a problem encountered in multitasking where a process is perpetually denied necessary resources. Without those resources, the program can never finish its task.

Starvation is usually caused by an overly simplistic scheduling algorithm. For example, if a (not very well designed) multi-tasking system always switches between the first two tasks while a third never gets to run, then the third task is being starved of CPU time. The scheduling algorithm, which is part of the kernel, is supposed to allocate resources equitably; that is, the algorithm should allocate resources so that no process perpetually lacks necessary resources.

Predictability: This is the criterion used to determine the next process to be executed after the process currently being executed.

Operating systems may feature up to three distinct types of schedulers: a long-term scheduler, a mid-term scheduler and a short-term scheduler. This simply suggests the relative frequency with which these functions are performed and are not the aim of this paper. Thus, more details about these three types of schedulers can be found in [1]. Besides, an extensive study on scheduling algorithms used in a multiprogramming system was carried out and described in the subsequent sections.

Preemption: This is the act of temporality interrupting a task being carried out by the system, without requiring its cooperation, and with the intention of resuming the task at a later time. It is normally carried out by a privileged task or part of the system known as a preemptive scheduler, which has the power to preempt, or interrupt, and later resume, other tasks in the system.

4.2 SCHEDULING ALGORITHMS

4.2.1 FIRST COME FIRST SERVED (FCFS)

First-Come-First-Served algorithm is known as the simplest scheduling algorithm. Here, processes are dispatched according to their arrival time on the ready queue. Classified as non-preemptive scheduling algorithm, it runs to completion once admitted to the CPU. Therefore, it refers to as « run until done » and uses FIFO (First-In-First-Out) algorithm to carry out the task. In other term, it means that one program runs non-preemptively until it is finished (including any blocking for I/O operation). That is, keep the CPU until done. Therefore, the FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. Simple to write and understand; its major drawback is that the average time is often quite long. Also, the FCFS is unfair in the sense that long jobs make short jobs wait and unimportant long jobs make important short jobs wait. However, it is more predictable than most of the other schemes since it offers time. As a matter of fact, the FCFS is rarely used as a scheme in modern operating systems but it is often embedded within other schemes.

4.2.2 ROUND ROBIN SCHEDULING (RR)

As noted above, though FCFS scheme is potentially bad for short jobs; RR scheme solves this problem by providing each process a small unit of CPU time known as quantum time which varies from 10 – 100 milliseconds. RR is also one of the simplest scheduling algorithms for processes in an operating system. In fact, after quantum expires, the process is preempted and added to the end of the ready queue. The major advantage of RR is fairness whereby each job gets an equal amount of the CPU. And its drawback is the average waiting time. The average waiting time can be bad especially when the number of processes is large. For example, let N be the number for processes in ready queue and time quantum is Q milliseconds, therefore each process gets 1/N of the CPU time. RR offers better performance for short jobs but context-switching time adds up for long jobs. Also, if the chosen quantum is: too large, response time suffers and if too small, throughput suffers and percentage overhead grows.

4.2.3 SHORTEST JOB FIRST (SJF)

Shortest Job First (SJF) is a scheduling algorithm that selects the waiting process with the smallest execution time to execute next. Another name for SJF is Shortest Process Next Algorithm. Shortest job first is advantageous because of its simplicity and because it maximizes process throughput (in terms of the number of processes run to completion in a given amount of time). However, it has the potential for process starvation for processes which will need a long time to complete if short processes are added continuously. Highest response ratio next is similar but provides a solution to this problem. Shortest job next scheduling is rarely used outside of specialized environments because it requires accurate estimations of the runtime of all processes that are waiting for execution. It is provably optimal as regards the minimization of the average waiting time. It works for both preemptive and non-preemptive schedulers.

The SJF is also called Shortest Next-CPU-Burst First. Unlike other scheduling algorithms, this algorithm assumes that information about a process's burst length is stored between the times when it is ready. In keeping with the need for efficiency, only a small amount of information is stored and only a simple calculation is performed. We can weigh the previous expectation (representing all previous bursts) and the most recent burst with any two weights that add up to 1, e.g., say 0.5 and 0.5, or 0.9 for previous expectations and 0.1 for actual time for most recent CPU burst.

The expected burst length is calculated as follows:

$a(t)$ = actual amount of time required during CPU burst t

$e(t)$ = amount of time that was expected for CPU burst t

$e(t+1)$ = expected time during the next CPU burst

then... $e(t+1) = 0.5 * e(t) + 0.5 * a(t)$

4.2.4 SHORTEST REMAINING TIME FIRST (SRTF)

Shortest remaining time first is a method of CPU scheduling that is a preemptive version of shortest job first scheduling. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected for execution. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time. Shortest remaining time is advantageous because short processes are handled very briskly. The system also requires very little overhead since it only makes a decision when a process completes or a new process is added, and when a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute. However, it has the potential for process starvation for processes which will require a long time to complete if short processes are continually added, though this threat can be minimal when process times follow a heavy-tailed distribution. Synonymous to shortest job first scheduling, shortest remaining time first scheduling is rarely used outside of specialized environments because it requires accurate estimations of the runtime of all processes that are waiting for execution.

With the SRTF when a process arrives at the ready queue with an expected CPU-burst-time that is less than the expected remaining time of the running process, the new one preempts the running process. A process may mislead the scheduler if it previously ran quickly but now may be CPU intensive (the SRTF algorithm fails very badly for such a case). The SRTF algorithm provably gives the highest throughput (number of processes completed) of all scheduling algorithms if the estimates are exactly correct.

4.2.5 LOTTERY SCHEDULING (LOT)

Lottery scheduling is a probabilistic scheduling algorithm for processes in an operating system. Processes are each assigned some number of lottery tickets, and the scheduler draws a random ticket to select the next process to be executed. The distribution of tickets need not be uniform; granting a process more tickets provides it a relative higher chance of selection. This technique can be used to approximate other scheduling algorithms, such as shortest job next and Fair-share scheduling. Lottery scheduling solves the problem of starvation.

Giving each process at least one lottery ticket guarantees that it has non-zero probability of being selected at each scheduling operation. On average, CPU time is proportional to the number of tickets given to each job. For approximation in SJF, most tickets are assigned to short running jobs, and fewer to longer running jobs. To avoid starvation, every job gets at least one ticket. Implementations of lottery scheduling should take into consideration that there could be a large number of tickets distributed among a large pool of threads. To have an array of tickets with each ticket corresponding to a thread may be highly inefficient.

5 ANALYSIS OF SCHEDULING ALGORITHMS FOR MULTIPROGRAMMING SYSTEMS

The assignment of the software, known as scheduler, used to assign priorities in a priority queue consists of mainly, CPU utilization – to keep the CPU as busy as possible. Throughput – number of process that completes their execution per time unit. Waiting time – amount of time a process has been waiting in the ready queue. Response time – amount of time a process takes from when a request was submitted until the first response is produced.

An ideal scheduling algorithm is one which:

i. Minimizes response Time that is elapsed time to do an operation (job); response time is what the user sees (e.g. time to echo keystroke in editor, time to compile a program, real-time tasks). It must meet deadlines imposed by the World.

ii. Maximizes throughput: refers to jobs per second; throughput relates to response time, but are not identical. Minimizing response time will only lead to more context switching than if you maximized only throughput.

iii. Minimizes overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.) and;

iv. Maximizes fairness that is sharing CPU among users in some equitable way; not just minimizing average response time.

The following table shows the analysis of the different algorithms described in section 5.2

	Algorithms	**FCFS**	**RR**	**SJF**	**SRTF**	**LOT**
1	CPU Utilization	Low	Medium	Higher	Requires very little overhead since it only makes a decision when a process completes or a new process is added	CPU time proportional to the number of tickets given to each job
2	Throughput	Traded off for better Response Time	Bad when the chosen quantum is small	Traded off for better Response Time	Gives the highest throughput of all scheduling algorithms.	Probabilistic guarantee of throughput proportional to ticket allocation.
3	Waiting Time	The average waiting time is large	The average waiting time is large as compared to others algorithms	The average waiting time is small compared to other algorithms	The average waiting time is large	Low waiting time
4	Response Time	Good	Bad when the number of processes is large	Good	Good	Good
5	Fairness	Unfair for long jobs make short jobs wait and unimportant long jobs make important short jobs wait	Fair	Unfair	Unfair	Fair
6	Starvation	No potential for starvation	Free-starvation	Long running CPU bound jobs can starve	Has the potential for process starvation for processes	Solve the problem of starvation
7	Predictability	More predictable	Predictable	Impossible to predict the amount of CPU time a job has left	Impossible to predict the amount of CPU time a job has left	Less predictable
	Preemption	Nonpreemptive	Preemptive	Handles preemptive and nonpreemptive	Preemptive	Can be preemptive or non-preemptive.

6 CONCLUSION

This paper establishes the fact that an ideal scheduling algorithm should minimize the response time, maximize the throughput, minimize the overhead (in terms of CPU utilization, disk and memory) and maximize fairness. However, it is palpable from this paper that none of the scheduling algorithms considered satisfies all of the aforementioned requirements. Hence, the contribution to knowledge made by this paper is that based on the most prioritized requirement of a particular user, the scheduling algorithm that best satisfies that requirement as outlined in this paper can be selected for use thereby allowing the user get awesome results without having to waste valuable time.

In future, another researcher can do further research in order to discover another scheduling algorithm that will be ideal thereby satisfying all the aforementioned requirements and consequently offering optimum satisfaction to the user.

REFERENCES

[1]. Silberschats, A, P.B. Galvin and G. Gagne (2012), Operating System Concepts, 8th Edition, Wiley India.

[2]. Neetu Goel, R.B. Garg; A Comparative Study of CPU Scheduling Algorithms; Internal Journal of Graphics & Image Processing, Vol. 2, Issue 4, November 2012.

[3]. C.L. LIU, JAMES W. LAYLAND, Scheduling Algorithm for Multiprogramming in a Hard-Real-Time Environment, Journal of the ACM, Vol. 20, Issue 1, Jan. 1973, PP. 46 – 61.

[4]. Edwin K.P Chong, Wei Zhao, Performance Evaluation of Scheduling Algorithms for Imprecise Computer Systems; Journal of Systems and Software, 1991.

[5]. K. L. Krausse, V. Y. Shen, H. D. Schwetman, A task-scheduling algorithm for a multiprogramming computer system: published in Proceeding SOSP ’73 Proceedings of the fourth ACM symposium on Operating system principles, PP. 112 – 118; ACM New York, NY, USA 1973

[6]. Ankur B., Rachhpal S., Gaurav, Comparative Study of Scheduling Algorithms in Operating System; International Journal of Computers and Distributed Systems, Vol. No. 3, Issue I, April – May 2013.

[7]. K. M. Baumgartner, B. W. Wah, Computer Scheduling Algorithms: Past, Present, and Future, Journal Information Science; Elsevier Science Inc.:, Vol. 57 – 58; Sept./Dec. 1991; PP. 319 – 345.

[8]. Aarti Singh, Manisha Malhotra; A Comparative Analysis of Resource Scheduling Algorithms in Cloud Computing, American Journal of Computing Science and Engineering Survey, Pubicon Open Journal, 2013, PP. 14 – 32.

[9]. Tong Li, Dan Baumberger, Scott Hahn, Efficient and Scalable Multiprocessor Fair Scheduling Using Distributed Weighted Round-Robin, Published in Proceeding, PPoPP’ 9 Proceedings of the 14th ACM SIGPLAN symposium on Principles and Practice of parallel programming, PP. 65 – 74.

[10]. Junchao Xiao, Leon J. Osterweil, Qing Wang, Mingshu Li, Dynamic Scheduling in Systems with Complex Resource Allocation Requirments;

[11]. Kartik Gopalan and Tzi-Cker Chiueh, Multi-Resource Allocation and Scheduling for Periodic Soft Real-Time Application, In Proceedings of Multimedia Computing and Networking 2002.

[12]. Krithi Ramamritham and John A. Stankovic, Scheduling Algorithm and Operating Systems Support for Real-time Systems, published in Proceedings of the IEEE (Vol. 82, Issue 1; Jan 1994, PP. 55-67.