HYBRID SCHEDULING AND DUAL QUEUE SCHEDULING

Syed Nasir Mehmood Shah
Department of Computer and Information Sciences
Universiti Teknologi PETRONAS,
Bandar Seri Iskandar, 31750
Tronoh, Perak, Malaysia.
nasirsyed.utp@gmail.com

Ahmad Kamil Bin Mahmood
Department of Computer and Information Sciences
Universiti Teknologi PETRONAS,
Bandar Seri Iskandar, 31750
Tronoh, Perak, Malaysia.
kamilmh@petronas.com.my

Alan Oxley
Department of Computer and Information Sciences
Universiti Teknologi PETRONAS,
Bandar Seri Iskandar, 31750
Tronoh, Perak, Malaysia.
alanoxley@petronas.com.my

Abstract- **Multiprogramming computer systems execute multiple programs concurrently. An objective of multiprogramming is to optimize resource utilization. Efficient resource utilization is achieved by sharing system resources amongst multiple users and system processes. Optimum resource sharing depends on the efficient scheduling of competing users and system processes for the processor, which renders process scheduling an important aspect of a multiprogramming operating system. As the processor is the most important resource, process scheduling, which is called CPU scheduling, becomes all the more important in achieving the above mentioned objectives.**

Many algorithms have been developed for the CPU scheduling of a modern multiprogramming operating system. Our research work involves the design and development of new CPU scheduling algorithms (the Hybrid Scheduling Algorithm and the Dual Queue Scheduling Algorithm) with a view to optimization. This work involves a software tool which produces a comprehensive simulation of a number of CPU scheduling algorithms. The tool's results are in the form of scheduling performance metrics.

Keywords- Operating System, CPU Scheduling, Process Management, Resource Scheduling, Multiprogramming

I. INTRODUCTION

Scheduling is a fundamental function of an operating system. It is described in every undergraduate textbook on operating systems; see for example [1]. The main concept is to share computer resources among a number of processes. Almost each computer resource is scheduled before use. The CPU is one of the primary computer resources, so its scheduling is essential to an operating system's design. CPU scheduling decides which processes execute when there are multiple run-able processes. CPU scheduling is important because it plays an important role in effective resource utilization and the overall performance of the system.

Scheduling is a branch of the topic of Operational Research. In the work described here, we are considering how best to schedule multiple jobs for processing by a single machine (CPU). It is a relatively easy problem as compared to the problem of scheduling multiple jobs for multiple machines. Some idea of the difficulties associated with this latter problem can be garnered by studying 'job shop' and 'flow shop' scheduling problems, see for example [2, 3].

There are many existing CPU scheduling algorithm simulators. Some are more user-friendly than others. Some are command-line driven whilst others have a GUI (Graphical User Interface). Let us briefly mention some of the simulators that are available.

Suranauwarat [4] developed a simulator which produces a simulation of various scheduling algorithms for a single CPU. A user can run this simulator with predefined scheduling parameters and can also customize parameters for a set of processes. The simulator works in two operating modes. The first mode is 'simulation mode' and the second one is called 'practice mode'. In simulation mode, the user can interact with the simulation during process execution. A user can start and stop the simulation whenever he or she likes. A user can also monitor the simulation straight through from the beginning until the end. By using the simulator in simulation mode, the user could achieve a better conceptual understanding of the CPU scheduling algorithms. In practice mode, the user can predict when and for how long each process is in a particular state. The user is also able to predict why a process is in that state through a very good graphical user interface. The user is also provided with the facility to check whether his or her answer is correct or not at any time during practice. One drawback of this research work is that it is only limited to traditional scheduling policies. Another drawback is that it does not provide any comparative results of different scheduling policies.

[5] provides a scheduling simulator named CSCI 152 CPU Scheduling Algorithm Simulator. This simulator is a server-side program that allows the user to interact with it via its web form. It provides a very good graphical web-based interface. It gives a comparison of the performance of three scheduling algorithms (the FCFS, RR, and SJF scheduling algorithms) for the same set of processes. The limitations of this simulator are that it only works for three scheduling algorithms and that the comparative analysis only relates to response times.

The Tran's Scheduling Algorithm Simulator [6] supports a number of scheduling algorithms such as FCFS, RR, SJF, SRTF and HRR. The time quantum is program coded and taken as 1 for each set of processes. It lets the user create a personal set of processes. However, this simulator uses a simple process model, that is, there is only one CPU burst per process. The drawback of this simulator is that the programmed time quantum is one and this causes a number of context switches. This simulator does not produce a comparative performance analysis of the different CPU scheduling policies.

978-1-4244-4520-2/09/$25.00 ©2009 IEEE

Each of the above simulators uses a Gantt chart to animate which process is using the CPU at what time. This approach is fine when the process model is one CPU burst per process.

The authors decided to develop their own scheduling simulator. A new program was necessary as no system was available to offer the desired functionality. The resultant program needed to be an efficient, pictorial and user-friendly simulation to depict a multiprogramming environment on a PC-based platform. There existed a strong need for such a system which would enable a PC-based user to analyze a multiprogramming operating system environment, for the purpose of either analyzing the design of a CPU scheduling system or studying the science of process scheduling. The system described here has been designed with a view to fill the gap, offering a modest simulation of a PC-based multiprogramming environment. It includes a graphical animation of the algorithm. The simulator is unique in a number of respects.

It should be emphasized that there is a valuable 'spin-off' to this part of our work in that the simulator can be used in the classroom. Studies have shown that computer science algorithm simulations, particularly animated ones, have the potential to enhance understanding and learning, see for example [7]. However, care must be taken to integrate a simulator into the curriculum otherwise it may have little effect [8].

The remainder of this paper is organized as follows: Section 2 discusses scheduling algorithms. Section 3 presents evaluation of results and discussion and section 4 concludes the paper.

II. SCHEDULING ALGORITHMS

To begin with we will briefly describe five commonly occurring algorithms [1, 9, 10]. This will be followed by detailed descriptions of the two new CPU scheduling algorithms.

A. First Come First Served Scheduling Algorithm (FCFS)

FCFS is the simplest scheduling algorithm. For this algorithm the ready queue is maintained as a FIFO queue. A PCB (Process Control Block) of a process submitted to the system is linked to the tail of the queue. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. When a process has completed its task it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue.

B. Shortest Job First Scheduling Algorithm (SJF)

For this algorithm the ready queue is maintained in order of CPU burst length, with the shortest burst length at the head of the queue. A PCB of a process submitted to the system is linked to the queue in accordance with its CPU burst length. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. When a process has completed its task it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue.

C. Shortest Remaining Time First Scheduling Algorithm (SRTF)

For this algorithm the ready queue is maintained in order of CPU burst length, with the shortest burst length at the head of the queue. A PCB of a process submitted to the system has its CPU burst length compared with the remaining time of the PCB being executed. If the new process requires less time than that remaining of the 'active' process then preemption occurs and it becomes the new PCB's turn for execution, otherwise it is linked to the queue in accordance with its CPU burst length. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. When a process has completed its task it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue.

D. Round Robin Scheduling Algorithm (RR)

For this algorithm the ready queue is maintained as a FIFO queue. A PCB of a process submitted to the system is linked to the tail of the queue. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. Processes being executed are preempted on expiry of a time quantum, which is a system-defined variable. A preempted process's PCB is linked to the tail of the ready queue. When a process has completed its task, i.e. before the expiry of the time quantum, it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue.

E. Priority Scheduling Algorithm

For this algorithm the ready queue is maintained in the order of system-defined priorities. A PCB of a process submitted to the system is linked to the last PCB in the queue having the same or a higher priority. The algorithm dispatches processes from the head of the ready queue for execution by the CPU. When a process has completed its task it terminates and is deleted from the system. The next process is then dispatched from the head of the ready queue.

F. Hybrid Scheduling Algorithm (H)

For this algorithm the ready queue is maintained in order of CPU burst length, with the shortest burst length at the head of the queue. Two numbers are maintained. The first number, t_{large} , represents the burst length of the largest PCB in the queue while the second one, t_{exec} , represents a running total of the execution time of all processes (since a reset was made). A PCB of a process submitted to the system is linked to the queue in accordance with its CPU burst length.

The algorithm dispatches processes from the head of the ready queue for execution by the CPU. Processes being executed are preempted on expiry of a time quantum, which is a system-defined variable. Following preemption, t_{exec} is updated as follows:

$$t_{exec} = t_{exec} + quantum$$

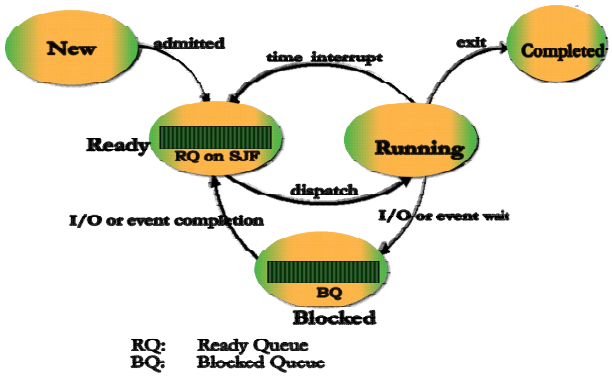

Figure 1. Process State Diagram of H.

The numbers are then compared.

If $t_{exec} < t_{large}$ then the preempted process's PCB is linked to the tail of the ready queue. The next process is then dispatched from the head of the ready queue.

If $t_{exec} \geq t_{large}$ then the PCB with the largest CPU burst length is given a turn for execution. Upon preemption, the ready queue is sorted on the basis of SJF. The value of t_{large} is reset to the burst length of the largest PCB, which is lying at the tail of the queue, and t_{exec} is reset to 0. The next process is then dispatched from the head of the ready queue.

When a process has completed its task it terminates and is deleted from the system. t_{exec} is updated as follows:

$$t_{exec} = t_{exec} + \textit{time to complete}$$

The numbers are then compared and the actions taken are the same as those for a preempted process.

G. Dual Queue Scheduling Algorithm (DQ)

For this algorithm the ready queue comprises two queues – the waiting queue and the execution queue. The waiting queue is maintained as a FIFO queue. A PCB of a process submitted to the system is linked to the tail of the waiting queue. Whenever the execution queue is empty, all PCBs in the waiting queue are moved to the execution queue, leaving the waiting queue empty. The execution queue is maintained in order of CPU burst length, with the shortest burst length at the head of the queue. Two numbers are maintained. The first number, t_{large} , represents the burst length of the largest PCB in the ready queue (waiting queue and execution queue combined) while the second one, t_{exec} , represents a running total of the execution time of all processes (since a reset was made). The algorithm dispatches processes from the head of the execution queue for execution by the CPU. Processes being executed are preempted on expiry of a time quantum, which is a system-defined variable. Following preemption, t_{exec} is updated as follows:

$$t_{exec} = t_{exec} + \textit{quantum}$$

The numbers are then compared.

If $t_{exec} < t_{large}$ then the preempted process's PCB is linked to the tail of the execution queue. The next process is then dispatched from the head of the execution queue.

If $t_{exec} \geq t_{large}$ then the PCB with the largest CPU burst length is given a turn for execution. Upon preemption, all PCBs in the waiting queue are moved to the execution queue, leaving the waiting queue empty. The execution queue is then sorted on the basis of SJF. The value of t_{large} is reset to the burst length of the largest PCB and t_{exec} is reset to 0. The next process is then dispatched from the head of the execution queue.

When a process has completed its task it terminates and is deleted from the system. t_{exec} is updated as follows:

$$t_{exec} = t_{exec} + \textit{time to complete}$$

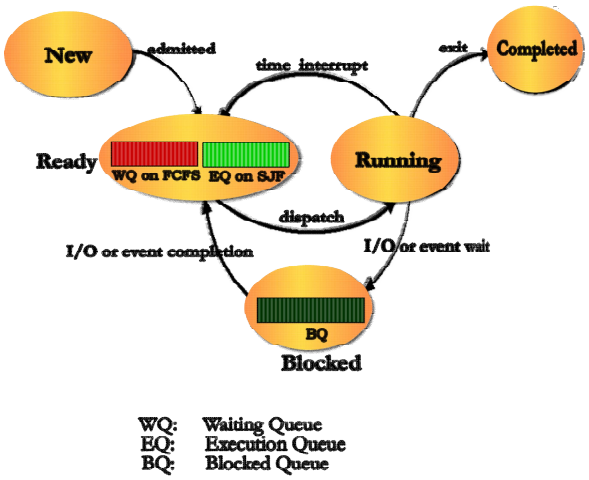

Figure 2. Process State Diagram of DQ.

The numbers are then compared and the actions taken are the same as those for a preempted process.

III. Evaluation of Results and Discussion

The simulator has been used to carry out extensive experimentation using the Windows Vista operating system on an Intel Core2 Duo. The simulations of the algorithms have generated useful data that has been analyzed.

To check the performance of the proposed algorithms, i.e. H and DQ, we took twenty process sets, each with different characteristics, and ran these through the simulator. Each process set consists of ten processes. Each process is specified by its CPU burst length, arrival time and priority number. Each process set has been given a time quantum for simulation.

Performance metrics for the CPU scheduling algorithms are based on three factors - Average Waiting Time, Average Turn Around Time, and Average Response Time. Table I shows the average time, for each process set, for each performance factor and algorithm combination.

	Performance Factors

Scheduling Algorithm	Average Waiting Times	Average Turn around Times	Average Response Times
FCFS	11224.00	13604.90	11224.02
SJF	6544.87	8925.70	6544.84
SRTF	4937.59	4653.89	3170.69
Priority	8657.02	11182.80	8803.51
Priority(P)	7577.11	9957.95	6860.31
RR	6186.28	8443.62	2553.90
H	**3136.23**	**5517.06**	**2614.49**
DQ	**4744.32**	**7125.15**	**1937.31**

TABLE I EXPERIMENTAL RESULTS

Below is graph derived from Table I, and this is followed by a discussion. Fig. 3 shows graphs of the Average Waiting Times, Average Turnaround Times, and Average Response Times, respectively.

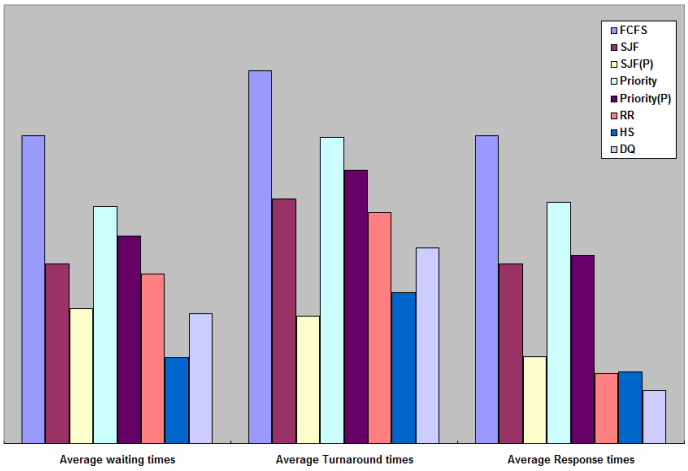

Figure 3. Graph showing the comparison of all performance factors

A. FCFS

Results obtained by simulation of FCFS have confirmed that the execution of FCFS produces smaller computational overheads. It gives poor performance, lower throughput and longer average waiting times. The relationship between average CPU burst length and average waiting time is non-linear.

B. SJF

SJF is an optimal scheduling discipline in terms of minimizing the average waiting time of a given workload. However, preferred treatment of short processes in SJF tends to result in increased waiting times for long processes. Thus, if there is a continuous inflow of short processes in the system there is a possibility that long processes may get stranded in the ready queue forever. Let us now compare FCFS with SJF. In cases where a batch is processed with FCFS, the resulting waiting/turnaround times experienced by the individual processes will be in accordance with their relative positions in the batch. For a batch processed with SJF the resulting waiting/turnaround times will be such that short jobs will be executed earlier than long jobs. Short jobs experience shorter waiting times with SJF in comparison with FCFS. Medium burst length processes experience almost equal waiting times in both algorithms. Long processes, however, result in shorter waiting times with FCFS when compared to SJF. Overall, the average waiting time of the batch has shortened in comparison with the FCFS.

C. SRTF

The jobs are submitted to the system and executed on the basis of SJF. It is a preemptive variant of SJF. It produces more optimized results as compared to FCFS and SJF.

D. RR

All processes get equitable response time and share of the processor's time, making this algorithm attractive for time sharing systems. Shorter processes complete before the expiry of the time quantum, so such processes, if present in the batch, are completed earlier. Medium burst length processes consume a few time slices before completion. Long processes take a greater number of time slices for completion resulting in longer completion time.

The relationship between the time slice and performance is nonlinear. Too short a time slice may result in significant overhead due to the frequent timer interrupts and context switches. On the other hand, too long a time slice reduces the preemption overhead but increases response time.

E. Preemptive Priority Scheduling Algorithm

It is a preemptive version of priority scheduling. The processes are prioritized in accordance with their operational significance. It produces optimized results as compared to Priority scheduling and FCFS for most of the cases.

F. Hybrid Scheduling Algorithm(H)

H is based on RR and SJF. Fig. 3 shows that average waiting times computed for H is optimum. It produces the least average waiting time as compared to the other CPU scheduling policies. Fig. 3 also shows that average turnaround time calculated for H is greater than the computed average turnaround time by SRTF but less than the computed average turnaround times for all other CPU scheduling policies. Furthermore, Fig. 3 shows that the average response time calculated for H is slightly greater than the computed values for RR and DQ but less than the computed average response times for other CPU scheduling policies.

G. DQ

DQ is an extended form of H. Fig. 3 shows that the average waiting time computed for DQ is slightly greater than the computed value for H. However, it shows the least average waiting time as compared to the other CPU scheduling policies. Fig. 3 also shows that the average

turnaround time for DQ is greater than the computed average turnaround times for SRTF and H but less than the computed average turnaround times for all other CPU scheduling policies. Furthermore, Fig. 3 shows that the average response time calculated for DQ is optimum. It produces the least average response time as compared to all other CPU scheduling policies.

The experimental results are according to the established facts and principles of the science of process scheduling. Performance metrics have been derived from the experimental results. They are presented in Table II.

IV. Conclusions

Results have shown that the execution of FCFS produces smaller computational overheads because of its simplicity, but it gives poor performance, lower throughput and longer Average Waiting Times. SJF is an optimal scheduling discipline in terms of minimizing the average waiting time of a given workload. However, the preferred treatment of short processes in SJF tends to result in increased waiting times for long processes in comparison with FCFS. Thus, there is a possibility that long processes may get stuck in the ready queue because of the continuous arrival of shorter processes in the queue. RR achieves fair sharing of the CPU. Short processes execute within a single time quantum and thus exhibit good response time. It tends to subject long processes to relatively longer turn around and waiting times. In the case of priority-based scheduling there is a possibility that low-priority processes will, in effect, be locked out by the higher priority ones. In other words, completion of a process within a finite time of its creation cannot be guaranteed with this scheduling policy. In RR there is fairness across the jobs, i.e. the jobs get equal time. However, upon completion of the time quantum, the PCB is linked to the tail of the ready queue and waits in the ready queue until it again gets the time quantum (after a complete circle). The processes with very small burst lengths also wait for a long time in the ready queue. H provides optimized results for turn around time, response time and waiting time. There is no starvation for any jobs when using the H. There is a little overhead due to the sorting involved with H. In order to avoid the overhead of sorting, DQ was introduced. DQ gives optimized results with minimum overhead and best response times for all nature of process sets. And again, there is no starvation for any process when using DQ.

Hence H works well from a system perspective. We can say that H is a scheduling policy from the system point of view; it satisfies the system requirements (i.e. minimum Average Waiting Time and minimum Turn Around Time). DQ works excellently from a system perspective. We can say that DQ is a scheduling policy from the user's point of view because it meets the user requirements (i.e. minimum Average Response Time). These results are extremely useful for the designing and development of modern operating systems.

Let us now consider the new simulator. This has been put through extensive experimentation. Various possible input patterns have been experimented with, from the above-mentioned CPU scheduling algorithms. The overall response of the system has been monitored accordingly. Behaviour of the system and the experimentation results, thereafter, has been in conformity with the established facts and principles of the science of process scheduling and we believe that the simulator is a valuable contribution to the understanding of modern operating systems.

References

[1] A. Silberschatz, P. B. Galvin, G. Gagne, "Operating System Concepts", 7th ed., John Wiley & Sons, 2005.

[2] Jacek Blazewicz, Wolfgang Domschke, Erwin Pesch, "The job shop scheduling problem: Conventional and new solution techniques",European Journal of Operational Research, Volume 93, Issue 1, 23 August 1996, pp.1-33.

[3] Chandrasekharan Rajendran, Oliver Holthaus, "A comparative study of dispatching rules in dynamic flow shops and job shops", European Journal of Operational Research Vol. 116, No. 1 (1999), pp. 156-170.

[4] Sukanya Suranauwarat, "A CPU Scheduling Algorithm Simulator", 37th ASEE/IEEE Frontiers in Education Conference, October 2007, pp. F2H19-F2H24.

[5] http://www.capricorn.org/~akira/cgi-bin/scheduler/index.html.

[6] http://www.utdallas.edu/~ilyen/animation/cpu/program/prog.html

[7] Andrea W. Lawrence, Albert Badre, John Stasko, "Empirically Evaluating the Use of Animations to Teach Algorithms", Proceedings of the 1994 IEEE Symposium on Visual Languages, 1994, pp. 48-54.

[8] John Stasko, Albert Badre , Clayton Lewis, "Do Algorithm Animations Assist Learning? An Empirical Study and Analysis", Proceedings of the SIGCHI conference on Human factors in computing systems, 1993, pp. 61-66.

[9] Milan Milenkovic, "Operating System Concepts and Design", Second Edition McGraw Hill International, 1992.

[10] Leland L. Beck, "System Software", 3rd Ed., Addison Wesley, 1997

	FCFS	RR	SJF	SRTF	Priority	Priority (P)	H	DQ
Selection Function	max[w]	constant	min[s]	min[s-e]	min[p]	min[p]	SJF	SJF
Decision Mode	non pre-emptive	pre-emptive	non pre-emptive	pre-emptive	non pre-emptive	pre-emptive	RR	RR
Through-put	low	low for small quantum	high	high	low	average	high	high
Response Time	high	less	less for short processes	less	average	average	less	least
Turn Around Time	high	less for short processes	less for short processes	least	average	average	less/ slightly higher than SRTF	less
Waiting Time	high	less	less for short processes	less	average	average	least	less
Overhead	minimal	low	can be high	can be high	can be high	can be high	can be high	low
Starvation	no	no	possible	possible	possible	possible	no	no
Where w = waiting time e = execution time p= priority number s = total service time required by the process including execution time(e)								

TABLE II Performance Metrics