http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 6 (2): 288-293, 2007
ISSN 1812-5638

Best-Job-First CPU Scheduling Algorithm

Mohammed A.F. Al-Husainy
Department of Computer Science, Faculty of Sciences and Information Technology,
Al-Zaytoonah University of Jordan, Amman, Jordan

Abstract: The major task of an operating system is to manage a collection of processes, in what is known as a CPU scheduling. In this study, a new CPU scheduling algorithm called Best-Job-First is suggested by mixing the functions of some well-known basic scheduling algorithm. When applying the suggested algorithm, the performance measures promised to use this algorithm for CPU scheduling with a good performance and it's introduce an easy way to switch between numbers of different scheduling algorithms to satisfy different goals.

Key words: FCFS algorithm, SJF algorithm, priority algorithm

INTRODUCTION

Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is, of course, one of the primary computer resources. Thus, its scheduling is central to operating-system design. CPU scheduling determines which processes run when there are multiple run-able processes. CPU scheduling is important because it can have a big effect on resource utilization and the overall performance of the system (Sabrina *et al*., 2005).

How do processes behave? First, CPU or I/O burst cycle. A process will run for a while (the CPU burst), perform some I/O (the I/O burst), then run for a while more (the next CPU burst). How long between I/O operations? It is depending on the process.

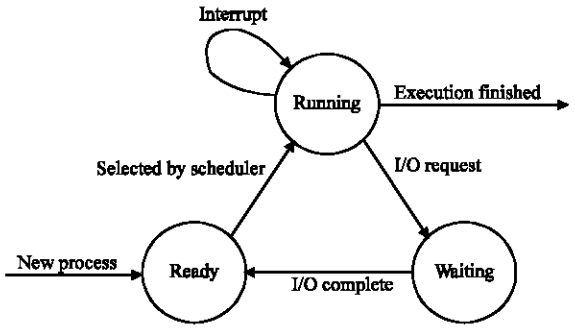
- I/O Bound processes: processes that perform lots of I/O operations. Each I/O operation is followed by a short CPU burst to process the I/O and then more I/O happens.
- CPU bound processes: processes that perform lots of computation and do little I/O. Tend to have a few long CPU bursts.

One of the things a scheduler will typically do is switch the CPU to another process when one process does I/O. Why? The I/O will take a long time and don't want to leave the CPU idle while wait for the I/O to finish (Silberchatz *et al*., 2003).

The state of a process: at any given time, a process is in one of several states. While the set of possible states varies from system to system, the following three comprise a minimal set:

- **Running:** The CPU is currently executing the code belonging to the process. This means that the hardware CPU registers contain the values associated with the particular process; in particular, the hardware program counter is pointing to code belonging to the process.
- **Ready:** The process could be running, but another process has the CPU.
- **Waiting (blocked):** Before the process can run, some external event (normally the completion of an I/O transfer) must occur.

As a process runs, it goes through a series of state transitions, of which the following graph is a simple rendition:



Note that, in this model, parallelism between computation and I/O is on an inter-process basis: that is, computation for one process overlaps I/O for other processes. It is also possible in some systems to have computation and I/O for a single process overlap to some extent: a process may start an I/O burst and continue computing until it reaches a point where further

computation cannot proceed. (e.g., if the operation is a read, computation can proceed until the data read is actually used; if it is a write, computation can proceed until the buffer where the data is stored must be re-used.) If this is the case, then the model must be modified to show the RUNNING → WAITING transition occurring as the result of an explicit WAIT request by the process, rather than as the automatic result of any I/O request.

Another possibility not shown in the diagram is pre-emption: the scheduler may take the CPU away from a process involuntarily either because it has used up its time quantum or because another, higher priority process needs the CPU. This could be shown in the diagram by a line from RUNNING to READY labeled pre-emption.

Therefore, preemptive and non-preemptive CUP schedulers are different. Preemptive scheduler reruns scheduling decision when process becomes ready. If the new process has priority over running process, the CPU preempts the running process and executes the new process. Non-preemptive scheduler only does scheduling decision when running process voluntarily gives up CPU. In effect, it allows every running process to finish its CPU burst (Wang and Saksena, 1999).

In a simple computer system, the CPU leave to be idle while wait for the I/O to finish. All this waiting time is wasted; no useful work is accomplished. With multiprogramming, we try to use this time productively. Several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. Every time one process has to wait, another process may take over the use of the CPU. The function of CPU scheduling depends on the observed property of processes (Deitel *et al*., 2004; Bic and Shaw, 2003; Silberchatz *et al*., 2003).

The basic assumptions behind most scheduling algorithms are: There is a pool of run-able processes contending for the CPU. The processes are independent and compete for resources. The job of the scheduler is to distribute the scarce resource of the CPU to the different processes fairly (according to some definition of fairness) and in a way that optimizes some performance measures. There is a large number of short CPU bursts and is a small number of long CPU bursts.

The CPU scheduling problem for computer systems distinguishes itself from general scheduling problems (e.g., job shop scheduling) in its variety of requirements of the system and variety of performance metrics. While minimizing the time at which the last job completes execution is usually a natural objective function for many general scheduling problems. A number of different possibilities exist for CPU schedulers in a general purpose multi-user computing environment. Nevertheless, minimizing the mean completion time (the sum of the times at which each job completes, divided by the number of jobs) is a commonly used objective function (Taranovsky, 1999; Barham *et al*., 2003; Gui *et al*., 2000).

Scheduling performance measures: Different CPU scheduling algorithms have different properties and may favor one class of processes over another. Many performance measures have been suggested for comparing CPU scheduling algorithms. Performance measures that are used include the following (Mackerras *et al*., 2005, Silberchatz *et al*., 2003):

- **CPU Utilization:** Keep CPU as busy as possible.
- **Throughput:** Number of processes completed per unit time.
- **Turnaround time:** Mean time from submission to completion of process.
- **Waiting time:** Amount of time spent ready to run but not running.
- **Response time:** Time between submission of requests and first response to the request.
- **Scheduler efficiency:** The scheduler doesn't perform any useful work, so any time it takes is pure overhead. So, we need to make the scheduler very efficient.

A good scheduling algorithm is the one that is able to optimize the above performance measures. The optimization performance measures are:

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time
- Maximize scheduler efficiency

It has also been suggested that, for interactive systems (such as time-sharing systems), it is more important to minimize the variance in the response time than it is to minimize the average response time. A system with reasonable and predictable response time may be considered more desirable than a system that is faster on the average, but is highly variable (Silberchatz *et al*., 2003; Gui *et al*., 2000).

CPU scheduling algorithms: CPU scheduling algorithms receive, for every process submitted to the system, three major factors attached with each process. These factors

are: priority, arrival time and Burst time. Some algorithms might adopt one of these factors to do the scheduling policy on the submitted processes and others may adopt another one. The following paragraphs give a quick survey about the well-known algorithms used in CPU scheduling.

First Come First Serve (FCFS) algorithm: The submitted processes are ordered in the ready queue depend on the submission (arrival) time. Operating system runs the process at head of queue, new processes come in at the end of the queue. A process does not give up CPU until it either terminates or performs I/O. FCFS is a relatively simple algorithm, but it has its problems. If large processes are run first, this can cause a large increase in the waiting time for other, smaller processes. FCFS is non-preemptive. So, it will hold onto the CPU until it has completed. It is important to note that the FCFS algorithm was focusing to order the processes in queue depending on their arrival time, without regarding to the priority or the burst time of these processes.

Short-job-first (SJF) algorithm: The submitted processes are ordered in the ready queue depend on the shortest burst time. Operating system runs the process at head of the queue, new processes come in the order to keep the process of shortest burst time first and the process of longest burst time last. SJF is probably optimal with respect to average waiting time. By moving shorter processes to the front of the queue, this reduces the average waiting time. Also, It is important to note that the SJF algorithm was focusing to order the processes in queue depending on their burst time, without regarding to the priority or the arrival time of these processes.

Priority algorithm: The submitted processes are ordered in the ready queue depend on the priority number that is given to each process (we assumed in this work that: 0 is high priority, 1, 2, …, N is low priority). Operating system runs the process at head of queue, new processes come in the order to keep the highest priority process first and the lowest priority process last. If multiple processes with same priority are run-able, use some other performance measures - typically FCFS. Also here, it is important to note that the PRIORITY algorithm was focusing to order the processes in queue depending on their priority, without regarding to the arrival time or the burst time of these processes.

In addition to the above three basic scheduling algorithms, there are some other algorithms derived its policy from the above three basic scheduling algorithms. These algorithm may be in some cases combine two or more policies of the basic scheduling algorithm to done its scheduling policy. Round Robin (RR), Multilevel Queue Scheduling, Multilevel Feedback Queue Scheduling … etc. are some of these algorithms (David and Barr, 2002; Whiteson and Stone, 2004).

Best-job-first (BJF): A new scheduling algorithm: From the above survey of the basic CPU scheduling algorithms, we can extract that there are three major factors attached with each process. These factors are used to order the submitted processes in the ready queue (i.e., arrival time, burst time and priority). Each one of the above three basic scheduling algorithms adopt one of these factors and ignore the others.

Normally, if we try to sort the importance of each factor in respect to the process, the priority factor is more important factor than the arrival and burst time (i.e., usually, any system task has a highest priority than any application/user tasks and must be run before them) and then the burst time is more important than the arrival time (i.e., logically, any task has short CPU burst time must be done by CPU as fast as possible and it must not wait for a long time) this policy helping the system to reduces the average waiting time for the processes in the queue and always keep the queue contains as few as possible processes in it).

Therefore, in the new scheduling algorithm Best-Job-first (BJF), a new factor f is suggested to attach with each submitted process. This factor sums the effects of all three basic factors (priority, arrival time and burst time). The equation summarizes this relation is:

$$f = \text{Priority} + \text{Arrival Time} + \text{BurstTime} \quad (1)$$

Depend on this new factor, the submitted processes are ordered in the ready queue depend on the value of the factor f which is calculated for each process from Eq. 1. Operating system runs the process at head of the queue, new processes comes in the order to keep the process has lowest value of the factor f first and the process has highest value of the factor f last.

Therefore, when adopt using the factor f in scheduling the submitted processes to the CPU. This will lead the CPU start to execute the process that:

- Has highest priority and
- Has shortest burst time and
- Submit to the system early

Now, for adding more control about the effect of each priority, arrival time and Burst time to the factor f in Eq. 1. A percentage ratio, of the effect for each these three basic factors in Eq. 1, will be added to become:

f = (Priority×Pririty_ratio)+(Arrival Time×
Arrival Time_ratio)+(Burst Time×Burst Time_ratio) (2)

By replacing Eq. 1-2 in calculating the factor f, the scheduling algorithm can now has more control to increase/decrease the ratio of effect of the three basic factors on the factor f. Here, we can say that the calculated value of the factor f represents the mixed effect that is produced from the combination of the effects of the three basic factors. In spite of that, the algorithm still has the ability to operate as one of the above well-known scheduling algorithms (i.e., FCFS, SJF and PRIORITY). This is done by set the ratio = 0% to two from the three basic factors, in Eq. 2, for all processes submitted to the system. Experiment 1-3 in Table 2 clarify that.

Periodically, the operating system can make some statistical calculations about the values of the factors (priority, arrival time and burst time) for all submitted process. These statistical calculations enable the operating system to set an appropriate ratio for each of the three basic factors in Eq. 2. This setting helps the system to enforce a suitable scheduling policy for each state in the system at any time. Whenever an appropriate setting is done, a good optimization for the scheduling performance measures will satisfy and the performance of the system will be increase. This is a powerful point introduces to the operating system by the new (BJF) scheduling algorithm.

Experimental evaluation: To clarifies the performance of the new suggested scheduling algorithm and compares its performance with the performance of the other three basic algorithms. An assumption of 50 processes with different values for their three factors (priority, arrival time and burst time) is submitted to the system. Firstly, the three basic scheduling algorithms are implemented and the results for their performance are recorded in Table 1.

Then the new scheduling algorithm (BJF) is implemented by setting different percentage ratios for the (priority, arrival time and burst time) in Eq. 2. The performance of the new scheduling algorithm can be noted from the recorded results in Table 2. Here, we want to focus on the Exp. 4 in the Table 2. This experiment realizes the perfect use of the percentage ratios, for (priority, arrival time and burst time), of the processes. This mean that the percentage 100% is divided into three percentage ratios in order to sort the processes in the queue firstly on the priority factor, secondly on the burst time factor and thirdly on the arrival time. These ratios keep the order of importantly of the three basic factors (priority, arrival time and burst time) as in the right normal order.

Table 1: FCFS, SJF, priority and RR scheduling algorithms

Experiments		Exp. 1	Exp. 2	Exp. 3	Exp. 4
Scheduling algorithm		FCFS	SJF	Priority	R.R.
Number of processes		50	50	50	50
FCFS ratio		100%	0%	0%	-
SJF ratio		0%	100%	0%	-
Priority ratio		0%	0%	100%	-
Wait time	Min	0	0	0	0
	Max	156	147	363	230
	Mean	34.7	23.4	28.52	41.12
	Sdev.	43.230	33.15	57.439	52.274
Response time	Min	0	0	0	0
	Max	156	147	363	4
	Mean	34.7	23.4	28.52	0.36
	Sdev.	43.230	33.15	57.439	0.8426
Turnaround time	Min	3	3	3	3
	Max	196	246	386	329
	Mean	65.32	54.02	59.14	71.74
	Sdev.	49.253	48.45	65.074	69.499

Table 2: BJF scheduling algorithms

Experiments		Exp. 1	Exp. 2	Exp. 3	Exp. 4
Scheduling algorithm		BJF	BJF	BJF	BJF
Number of processes		50	50	50	50
FCFS rate		100%	0%	0%	15%
SJF rate		0%	100%	0%	30%
Priority rate		0%	0%	100%	55%
Wait time	Min	0	0	0	0
	Max	156	147	363	147
	Mean	34.7	23.4	28.52	23.66
	Sdev.	43.230	33.15	57.439	32.011
Response time	Min	0	0	0	0
	Max	156	147	363	147
	Mean	34.7	23.4	28.52	23.66
	Sdev.	43.230	33.15	57.439	32.011
Turnaround time	Min	3	3	3	3
	Max	196	246	386	246
	Mean	65.32	54.02	59.14	54.28
	Sdev.	49.253	48.456	65.074	47.516

Now, to summarizes the performance of the FCFS, SJF, PRIORITY, RR and BJF scheduling algorithms. Figure 1-6 may be help us make a performance comparison, between the three basic algorithms and the BJF new algorithm, by looking at the mean and standard deviation (Sdev.) of the waiting time, response time and turnaround time of these algorithms.

Lastly, from Fig. 1-6 we can note that the BJF is a stable scheduling algorithm. This is clear from its keeping low level of mean and Sdev. values for the waiting time, response time and turnaround time. This is important point for any scheduling algorithm, whereas some other basic algorithms have high/low level, for the waiting time, response time and turnaround time values, in different figures.

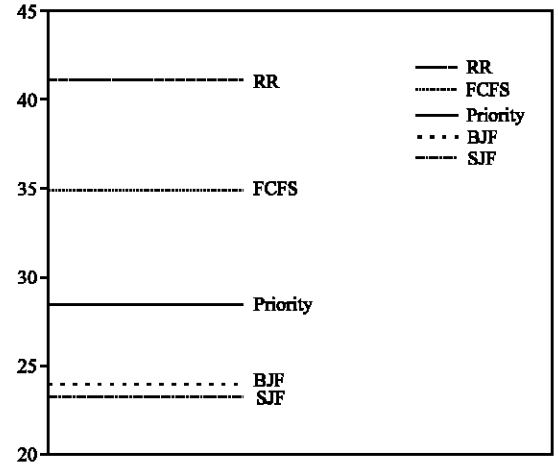


Fig. 1: Mean of waiting time

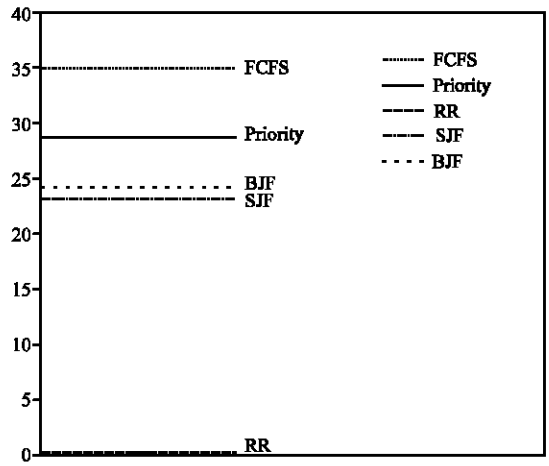


Fig. 2: Mean of response time

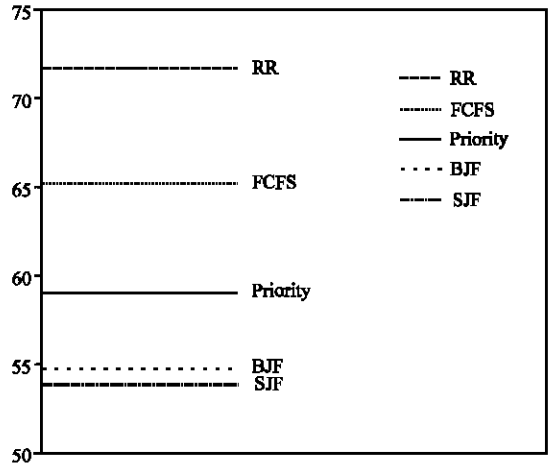


Fig. 3: Mean of turnaround time

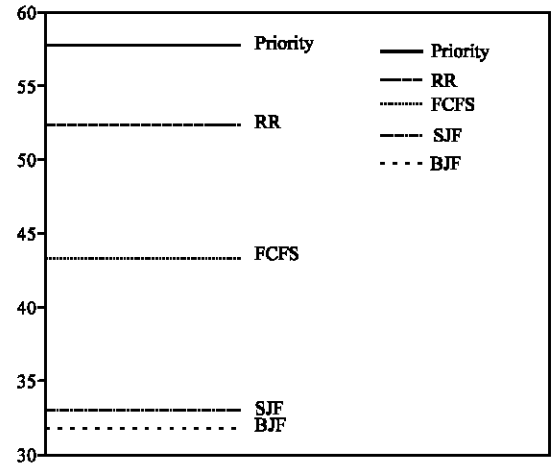


Fig. 4: Sdev. of waiting time

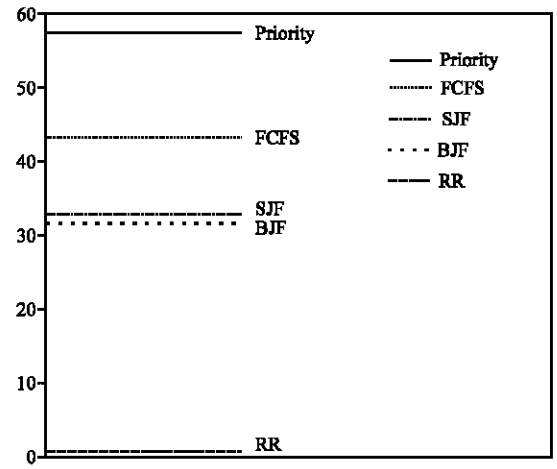


Fig. 5: Sdev. of response time

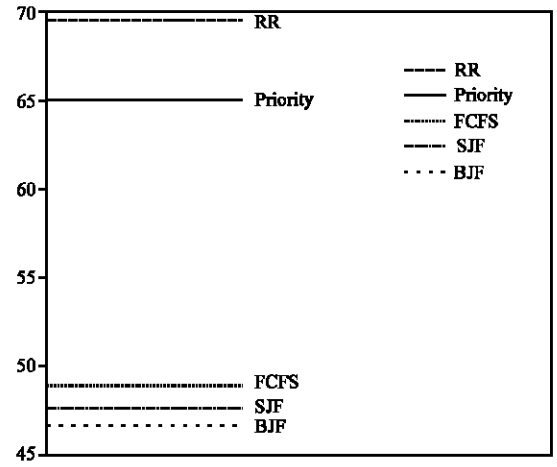


Fig. 6: Sdev. of turnaround time conclusion

CONCLUSIONS

From the comparison of the results that are recorded in the above tables. We can say here, the new scheduling algorithm (BJF) succeeded in satisfy a good performance in the scheduling policy. And enable the operating system to change its scheduling policy whenever it is required. This change is done without need to change the scheduling algorithm itself, but through changing the ratios of the three basic factors (priority, arrival time and burst time) which are effect on the factor f in the Eq. 2. In addition to that, the (BJF) takes in its consideration all the three basic factors of each process to determine the scheduling policy for the CPU. This consideration satisfies some type of balance in the dependency on all factors without ignoring the effects of others. Obviously, this is done through the sum of the three basic factors values for each process. This summation value is the factor f in the Eq. 2 that is used in this new scheduling algorithm. Obviously, to ensure the (BJF) scheduling algorithm work efficiently. The operating system must support this algorithm by accurate measurements for the next CPU burst time for each process and applying judicious policy in assigning a priority factor for each process.

REFERENCES

Barham, P., B. Dragovic, K. Fraser and S. Hand, 2003. Xen and the Art of Virtualization, SOSP'03, October 19.22.

Bic, L.F. and A.C. Shaw, 2003. Operating Systems Principles, (Prentice Hall).

David, S. and M. Barr, 2002. Rate Monotonic Scheduling, Embedded Systems Programming, pp: 79-80.

Deitel, H.M., P.J. Deitel and D.R. Choffnes, 2004. Operating Systems, (Prentice Hall, 3rd Edn).

Gui, X.N., T. Brecht and K. Lu, 2000. Preemptive scheduling of parallel jobs on multiprocessors, SIAM J. Comput. Soc. Indust. Applied Mathematics, 30: 145-160.

Mackerras, P., T.S. Mathews and R.C. Swanberg, 2005. Operating system exploitation of the POWER5 system. IBM J. Res. Develop. POWER5 and Packaging, pp: 49.

Sabrina, F., C.D. Nguyen, S. Jha, D. Platt and F. Safaei, 2005. Processing resource scheduling in programmable networks. Computer Commun., 28: 676-687.

Silberchatz, Galvin and Gagne, 2003. Operating Systems Concepts, (6th Edn., John Wiley and Sons).

Taranovsky, D.A., 1999. CPU Scheduling in Multimedia Operating Systems Research Report.

Whiteson, S. and O.P Stone, 2004. Adaptive job routing and scheduling. Engineering Applications of Artificial Intelligence, 17: 855-869.

Wang, Y. and M. Saksena, 1999. Scheduling Fixed-Priority Tasks with Preemption Threshold, RTCSA'99 by the IEEE Computer Society Press.