Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 6 (9): 2036-2039, 2006
ISSN 1812-5654
© 2006 Asian Network for Scientific Information

A New Multilevel CPU Scheduling Algorithm

[1]Md. Mamunur Rashid and [2]Md. Nasim Akhtar
[1]School of Science and Technology, Bangladesh Open University, Bangladesh
[2]Department of Computer Science and Engineering,
Dhaka University of Engineering and Technology, Bangladesh

Abstract: The productivity of a computer solely depends on the use of CPU scheduling algorithm in a multi-programmed operating system. In this study a new variant of priority scheduling algorithm has been proposed to reduce the average waiting time as well as the average turnaround time of the processes. It is observed that in existing priority scheduling algorithm the average waiting time and average turnaround time is very high for the processes, which have equal priority. But the proposed upgrading of this study can handle priority-scheduling algorithm with much more reduced waiting time and turnaround time for the processes.

Key words: CPU scheduling, priority scheduling, burst time, waiting time, turnaround time

INTRODUCTION

Scheduling is a fundamental operating system function, since 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. When more than one process is run-able, the OS must decide which one to run first. That part of the OS concerned with this decision is called scheduler and the algorithm it uses is called scheduling algorithm. A CPU scheduler is the part of an operating system responsible for arbitrating access to the CPU (Shamim, 1998)

In case of multi-programmed operating systems CPU scheduling plays a fundamental role by witching the CPU among various processes. The intention of an operating system should allow processes as many as possible running at all times in order to maximize the CPU utilization. In a multi-programmed operating system a process is executed until it must wait for the completion of some I/O request. In this case the time has been used proficiently. A number of processes are kept in memory simultaneously and while one process has to wait another process occupy the CPU selected by the operating system (Silverchatz, 2002).

The last thirty years have seen an enormous amount of research in the area of disk scheduling algorithms. The core objective has been to develop scheduling algorithms suited for certain goals, sometimes with provable properties (Seltzer *et al*., 1990).

Being the primary resource of a computer system CPU scheduling has to be designed centrally by an operating system. A successful CPU scheduling depends on process execution and I/O wait. Process execution starts with a CPU burst followed by an I/O burst and this thing happen rapidly. The last CPU burst ends with a system request to expire execution. Figure 1 shows the sporadic progression of CPU and I/O bursts.

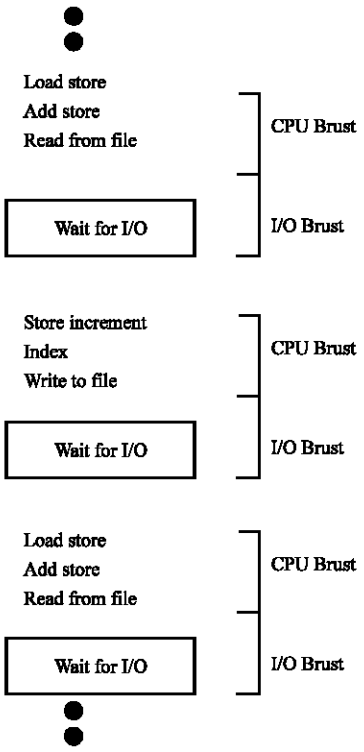


Fig. 1: A typical sporadic progression of CPU and I/O burst

Corresponding Author: Md. Mamunur Rashid, School of Science and Technology, Bangladesh Open University, Bangladesh

The study was done at Software Laboratory of School of Science and Technology of Bangladesh Open University on December 2005.

CONCEPT OF A CPU SCHEDULER

The task of a CPU scheduler is to pick a process among the processes stored in the memory that are ready to execute. For a scheduler the CPU scheduling decision has to take under the following state of affairs:

- When a process switches from the running state to the waiting state.
- When a process switches from the running state to the ready state.
- When a process switches from the waiting state to the ready state.
- When a process terminates.

The success of a CPU scheduler depends on an algorithm. High-quality CPU scheduler algorithms again in terms rely on its CPU utilization rate, throughput, turnaround time, waiting time and response time (Black *et al*., 1985).

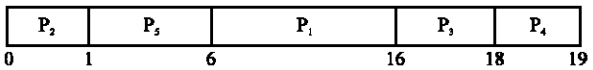
CONTEMPORARY PRIORITY SCHEDULING ALGORITHM

Priority CPU scheduling algorithm decides according to the assigned priority which of the processes (Highest priority process) in the ready queue is to be allocated the CPU. In this case equal-priority processes are scheduled in FCFS (First Come First Served) order. Here the term priority refers to some fixed range of numbers such as 0 to 9 (for 10 processes in the ready queue) or 0 to 4.095 (for 4.046 processes in the ready queue). Generally low numbers are used to represent high priority.

For example, let, P_1, P_2….. P_5 are the set of processes in the ready queue with the following CPU-burst time and priority:

Process	Burst time (milliseconds)	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

So using the contemporary priority scheduling algorithm the processes can be scheduled as the following Gantt chart:



LIMITATIONS OF CONTEMPORARY PRIORITY SCHEDULING ALGORITHM

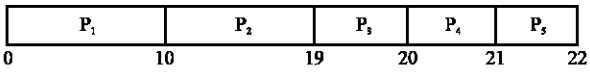
Though priority-scheduling algorithm does an enormous job for allocating the CPU to the processes but still it has some limitations to mention:

- Indefinite blocking or starvation can happen for the low priority processes. And this problem can be solved by the Aging technique.
- If two processes have same priority then the tie can be broken with FCFS (First Come First Serve) basis. So the waiting time gradually increased for the equal priority processes.

Let us consider the following processes with equal priority:

Process	Burst time (milliseconds)	Priority
P_1	10	2
P_2	9	2
P_3	1	2
P_4	1	2
P_5	1	2

So the Gantt chart for the above mentioned processes:



The waiting time for the processes are as follows and the average waiting time is (0+10+19+20+21)/5 = 14 ms.

Process	Waiting time (milliseconds)
P_1	0
P_2	10
P_3	19
P_4	20
P_5	21

Again the turnaround time for the processes is as follows and the average turnaround time is (10+19+20+21+22)/5 = 18.4 ms. according to algorithm described in literature (Silverchatz *et al*., 2002).

Process	Turnaround time (milliseconds)
P_1	0+10 = 10
P_2	10+9 = 19
P_3	19+1 = 20
P_4	20+1 = 21
P_5	21+1 = 22

PROPOSED PRIORITY SCHEDULING ALGORITHM

In our proposed priority-scheduling algorithm we have combined the working principle of SJF (Shortest Job First) scheduling algorithm along with the contemporary priority-scheduling algorithm. Here one thing should mention that SJF is a special case of the general priority-scheduling algorithm. Our proposed algorithm works as follows:

Step 1: Assign priority to each of the processes in the ready queue.

Step 2: Allocate the CPU to the process that has the highest priority.

Step 3: If two or more process has equal-priority then
{
Allocate the CPU to the process that has shortest job (minimum burst time).
}

Step 4: If two or more process has equal-priority and equal burst time then
{
Allocate the CPU to the processes according to FCFS (First Come First server) basis.
}

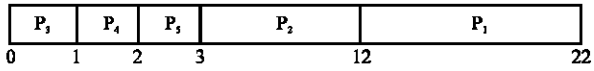
IMPLEMENTATION AND COMPARISON OF PROPOSED ALGORITHM

To implement the proposed algorithm we have used a multilevel stable sorting technique. Some common stable sorting technique like Bubble sort, Selection sort etc. can be used for this purpose (Zahorjan and McCann, 1990). We have also designed a compare function and swapping function so that our algorithm can work with multiple criteria.

Now to show the comparison let consider the same processes, which have been implemented for the existing algorithm:

Process	Burst time (milliseconds)	Priority
P_1	10	2
P_2	9	2
P_3	1	2
P_4	1	2
P_5	1	2

So the Gantt chart for the above mention processes using propose algorithm:



The revised waiting time for the processes are as follows and the average waiting time is (12+3+0+1+2)/5 = 3.6 ms.

Process	Waiting time (milliseconds)
P_1	12
P_2	3
P_3	0
P_4	1
P_5	2

Again the revised turnaround time for the processes is as follows and the average turnaround time is (22+12+1+2+3)/5 = 8 ms. according to our proposed algorithm.

Process	Turnaround time (milliseconds)
P_1	12+10 = 22
P_2	3+9 = 12
P_3	0+1 = 1
P_4	1+1 = 2
P_5	2+1 = 3

Thus if we consider our proposed priority scheduling algorithm to that of the existing one then we find that our algorithm reduce ((14-3.6)/14) * 100 = 74.29% waiting time for the above mentioned example. The comparison can be understood well from the following Fig. 2.

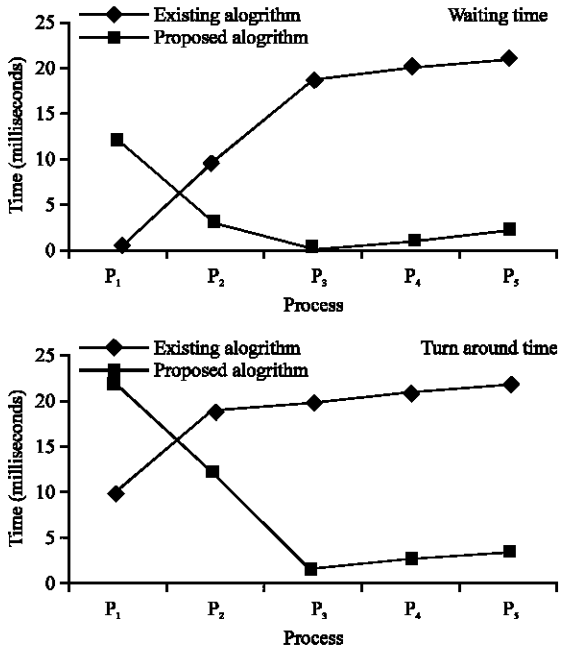


Fig. 2: Comparison of proposed algorithm with existing algorithm

CONCLUSIONS

From the above experiment and comparison of proposed algorithm with existing algorithm it is clear to us that the exiting algorithm tasks significantly more time to execute the more processes listed above 18.5 ms. But using our algorithm the waiting time as well as the execution of processes decrease drastically that is, average turnaround time is 8 ms.

RECOMMENDATIONS

It is recommended that any kind of simulation for any CPU scheduling algorithm has limited accuracy. The only accurate way to evaluate a scheduling algorithm to code it and has to put it in the operating system. Then a proper working capability of the algorithm can be measured in real systems. As our proposed algorithm has been justified with code (In C++ environment) so it can be an innovative move in case of CPU scheduling.

REFERENCES

Black, D.L., 1985. Scheduling support for concurrency and parallelism in the mach operating system. Computer, 23: 123-126.

Seltzer, M., P. Chen and J. Ousterhout, 1990. Disk scheduling revisited in USENIX. Winter Technical Conference.

Shamim, H.M., 1998. Operating system, DCSA-2302. School of Science and Technology. Bangladesh Open University, Gazipur-1705.

Silverchatz, G.G., 2002. Operating System Concepts. 6th Edn., John Wiley and Sons, INC.

Zahorjan and C. McCann, 1990. Processor scheduling in shared-memory multiprocessors. Proceedings of the Conference on Measurement and Modeling of Computer Systems, pp: 12-18.