INTERNATIONAL JOURNAL OF ENHANCED RESEARCH IN MANAGEMENT & COMPUTER APPLICATIONS

VOL. 1, ISSUE 2, NOV. 2012 ISSN NO: 2319-7471

CPU Scheduling Algorithm with Deadline and Optimize Time Slice for soft real time systems

Vikash Chandra Sharma[1], Priyadarshini[2], Ajay Chaudhary[3], Manindar singh nehra[4]
Dept. of Computer Science & Information Technology , Govt. Engg College of Bikaner, Rajasthan, India.
[1]vikash.cse.iu@gmail.com, [2]cs.priyadarshinisharma@gmail.com, [3]ajay@ecb.ac.in, [4]maninder4unehra@yahoo.com

Abstract: In this paper, a new algorithm proposed for soft real time operating system. In this proposed algorithm response time is better than existing algorithm. Also within deadline all process will complete. In this proposed algorithm if we will follow its then it will give better response time as well as good waiting time and turnaround time. Most important part of proposed algorithm is context switching is very low also range of time quantum is less. Round robin algorithm is not applicable for soft real time operating system. Our proposed algorithm is applicable for soft real time operating system with better response.

Keywords: Real time operating system, Soft operating system, Context switching, Time quantum, Round robin algorithm, time quantum, Response time, Turnaround time, Waiting time.

INTRODUCTION

Operating system is an application program, provide interface between users and computer hardware. Real time system is an system to perform all task for given time interval. Basically three type of real time operating system; hard, firm and soft. In hard real time system, failure to meet deadline or response time constraints leads to system failure. In firm real time system, failure to meet deadline can be tolerated. In soft real time system, failure to meet deadline doesn't lead to system failure, but only performance is degraded[1]. In CPU scheduling mainly schedule by a scheduler for process coming from ready queue to CPU. CPU scheduling is a central concept in real-time OSes (RTOSes) to meet timing requirements. In particular, CPU reservation and multicore scheduling are key techniques to further support concurrently-executing multiple interactive real-time applications. CPU reservation is useful to maintain the QoS for soft real-time applications, and it is also beneficial to provide temporal isolation for hard real-time applications. Multicore technology, meanwhile, supplies additional computing power for heavy workloads, and multicore scheduling algorithms determine how efficiently the system can utilize CPU resources[4]. More advanced algorithms take into account process priority, or the importance of the process. This allows some processes to use more time than other processes. The kernel always uses whatever resources it needs to ensure proper functioning of the system, and so can be said to have infinite priority.

Unfortunately, there is not much study into the question of how well CPU schedulers in existing RTOSes can perform with concurrently-executing multiple inter-active real-time applications on multicore platforms. Consider those in Linux-based RTOSes. Most prior approaches are evaluated by experiments or simulations that simply execute busy-loop tasks [5,6,7,8], or just a few real-time applications on single-core platforms [9,10].

In CPU scheduling different type of algorithm using for its. Basically, all algorithm based on CPU criteria. The criteria for performance evaluation of a CPU scheduling algorithm viz. the performance metrics are as follows[2]:

1). CPU utilization: we want to keep the CPU as busy as possible(3).

2). Throughput: Number of process completed per unit time called throughput.

3) Turnaround time: It is the amount of time from submission to completion process.

4). Waiting time: it is the total time of periods spent waiting in the ready queue.

5). Response time: It is the amount of time starting responding of a process or amount of time it takes from when a request was submitted until the first response is produced.

VOL. 1, ISSUE 2, NOV. 2012 ISSN NO: 2319-7471

6). Context switching: In any CPU scheduling algorithm number of context switching should be less.

7). Fairness - Equal CPU time to each process (or more generally appropriate times according to each process' priority). It is the time for which the process remains in the ready queue.

Some well known CPU scheduling algorithm:

First-Come, First-Served(FCFS): This is the first algorithm for a CPU scheduling. In this algorithm first come first base process assign for completing task in CPU. In this algorithm if process assign then it will complete task that means FCFS algorithm is a non-preemptive. Problem in FCFS convoy effect. The lack of prioritization means that as long as every process eventually completes, there is no starvation. In an environment where some processes might not complete, there can be starvation.

Shortest-Job-First(SJF): In this algorithm shortest job assign first for a CPU.SJF is preemptive as well as non-preemptive. In this algorithm waiting time and turnaround time of process is less than other CPU scheduling algorithm but it is not applicable because in this algorithm always shortest job assign first.SJF is a one of the priority CPU scheduling algorithm. If a shorter process arrives during another process' execution, the currently running process may be interrupted (known as preemption), dividing that process into two separate computing blocks. This creates excess overhead through additional context switching. The scheduler must also place each incoming process into a specific place in the queue, creating additional overhead.

Priority Scheduling: In this scheduling algorithm mainly process assign by a priority. In some operating system using less priority number means higher priority and some operating system using higher priority number means higher priority. In this algorithm higher priority assign first for CPU. In this algorithm, indefinite blocking(or starvation) is a problem. This algorithm is not using in time sharing operating system because response time is not so good. The OS assigns a fixed priority rank to every process, and the scheduler arranges the processes in the ready queue in order of their priority. Lower priority processes get interrupted by incoming higher priority processes.

Round Robin(RR): For time sharing operating system round robin scheduling is better than other scheduling. Because response time is better than other scheduling. But in round robin scheduling waiting time and turnaround time is not better. In round robin scheduling taking time quantum. Time quantum is not fixe in this algorithm. So, if time quantum is small then context switching is high and if time quantum is large then round robin form first-come, first-serve algorithm. Balanced throughput between FCFS and SJF, shorter jobs are completed faster than in FCFS and longer processes are completed faster than in SJF.

Multilevel Queue Scheduling: In this algorithm partitions the ready queue into several separate queue. the process is permanent assign to queue generally based on some property of the process, such as memory size, process priority, or process type[3 Galvin]. Each queue has its own scheduling algorithm. This is used for situations in which processes are easily divided into different groups. For example, a common division is made between foreground (interactive) processes and background (batch) processes. These two types of processes have different response-time requirements and so may have different scheduling needs. It is very useful for shared memory problems.

PROPOSED ALGORITHM

In existing algorithm deadline is not given. Basically priority scheduling using for real time operating system. But in priority scheduling starvation problem is occur. In real time operating system mainly priority scheduling uses but single priority scheduling is not useful for real time operating system because turnaround time and waiting time is not better of priority scheduling also response time is not so good. In our proposed algorithm mainly in this algorithm response time is better as well as turnaround and waiting time is good. And context switching is lesser than existing algorithm. In this proposed algorithm we will take OTS(Optimize Time Quantum) also taken deadline and it is based on burst time.

Algorithm:

1) First of all check ready queue is empty.
2) When ready queue is empty then process assign in ready queue.
3) While(ready queue != NULL)
4) Calculate optimal time slice(OTS):

X=sum of lowest and second lowest burst time in ready queue

Y=average burst time in ready queue

VOL. 1, ISSUE 2, NOV. 2012 ISSN NO: 2319-7471

Z=remaining burst time.
For 1st process arrive for CPU:
If(X<=Y)
OTS(Pi)=X [i=priority no 1 to n]
Else
OTS(Pi)=Y [i=priority no 1 to n]
For 2nd process arrive for CPU:
OTS(Pi)=Z [i=priority no 1 to n]
5) Assign OTS(Pi) to the arriving process for CPU.
6) If(process arrive 1st time for CPU)
Goto step(7)
Else
Goto CPU
7) Calculate deadline of each arrival process:
For 1st process:
[i=1,first priority no]
[i+1=2,second priority no]
[D(Pi)=deadline of process Pi]
[BT(Pi)=burst time of process Pi]
[BT(Pi+1)=burst time of process Pi+1]
[OTS(Pi)=optimize time slice of process Pi]

D(Pi)= BT(Pi)+BT(Pi+1)
i=1(first priority)
if(BT(Pi)<=OTS(Pi))
complete total task
goto step(2)
else
switch on OTS(Pi)
goto step(7)
for next process except last process:
[BT(Pi)=burst time of process Pi,i denote priority no,2 to n-1]
[OTS(Pi)=optimize time slice of process Pi,I denote priority no 2 to n-1]
[SPBT(Pi)= sum of previous burst time completion in CPU of process Pi]
[PPD(Pi)=preemptive process deadline of process Pi-1]

D(Pi)=D(Pi-1)+BT(Pi+1)
If(BT(Pi)<=OTS(Pi)&&(SPBT(Pi)+BT(Pi)<=PPD(Pi-1))
Pi complete task
Goto step(2)
Else
Pi switch on PPD(Pi)
Goto step(7)
Otherwise
Pi switch on OTS(Pi)
Goto step(7)
For last process:
D(Pi)=D(Pi-1)
If(BT(Pi)<=OTS(Pi)&&(SPBT(Pi)+BT(Pi)<=PPD(Pi-1))

VOL. 1, ISSUE 2, NOV. 2012 ISSN NO: 2319-7471

Pi complete task
Goto step(2)
Else
Pi switch on PPD(Pi)
Goto step(7)
Otherwise
Pi switch on OTS(Pi)
Goto step(7)
8) Calculate remaining burst time(RBT):
[RBT(Pi)=remaining burst time of process Pi,i denote priority no 1 to n]
[EBT(Pi)=total execution burst time of Pi,i denote priority no 1 to n]
[BT(Pi)=burst time of process Pi,i denote priority no 1 to n]

RBT(Pi)=BT(Pi)-EBT(Pi)
Goto step(9)
9) Calculate deadline of preemptive process :
[PPD(Pi)=preemptive process deadline of process Pi,i denote priority no 1 to n]
[RBT(Pi)=remaining burst time of process Pi,i denote priority no, 1 to n]

for arrival in CPU:
PPD(Pi)=D(Pi)-RBT(Pi)
If(PPD(Pi+1)<=PPD(Pi)) ||(PPD(Pi+1)<D(Pi))
Then
PPD(Pi)=PPD(Pi+1)-RBT(Pi)
Goto step(2)
10) If new process coming then goto step(2).
11) End while()
12) End.

PERFORMANCE EVALUTION:

We assume four process with burst time and priority no given below table:

Process	Burst time	Priority no
P1	24	1
P2	3	3
P3	5	4
P4	30	2

Table1: process burst time with priority

CALCULAION OF DEADLINE BY PROPOSED ALGORITHM:

In below table show deadline values of each process using above table.

VOL. 1, ISSUE 2, NOV. 2012 ISSN NO: 2319-7471

Process	Burst time	Priority no	Deadline
P1	24	1	54
P2	3	3	62
P3	5	4	62
P4	30	2	57

Table2:process burst time, priority no, with deadline.

Grant chart by proposed algorithm:

P1	P4	P1	P2	P4	P3

0 8 16 32 35 57 62

Grant chart by existing priority scheduling algorithm:

P1	P4	P2	P3

0 24 54 57 62

COMPARISON:

In our proposed algorithm response time is better than priority scheduling algorithm. As shown below table:

Process	Response time by proposed algorithm	Response time by priority algorithm
P1	0	0
P2	32	54
P3	57	57
P4	8	24

Table: comparison of response time.

In above table we can check response time of individual process is better than existing scheduling algorithm.

Average response time: Average response time calculated by proposed algorithm is 24.75 and average response time by existing algorithm is 33.75. So response time of individual processes is less by proposed algorithm. Also average response time is less than existing algorithm.

Turnaround time: Comparison of turnaround time of individual processes.

VOL. 1, ISSUE 2, NOV. 2012 ISSN NO: 2319-7471

Process	Turnaround time of proposed algorithm	Turnaround time of priority algorithm
P1	32	24
P2	35	57
P3	62	62
P4	57	54

Table :Turnaround time

Average turnaround time:- Average turnaround time calculated by proposed algorithm is 46.50 and average turnaround time by existing algorithm is 49.25. So in this illustration turnaround time better is we apply proposed algorithm but some time turnaround time is not better than existing algorithm .Only our focus is less response time by using proposed algorithm.

Waiting time:-comparison of waiting time of individual process.

Process	Waiting time by proposed algorithm	Waiting time by priority algorithm
P1	8	0
P2	32	54
P3	57	57
P4	27	24

Table :comparison of waiting time

Average waiting time:- Average waiting time calculated by proposed algorithm is 31 and average waiting time by existing algorithm is 33.25. So in this illustration average waiting time is less than existing algorithm. But sometimes average waiting time is larger than existing algorithm. important thing is response time is always better. In given illustration waiting time of individual process is less than existing algorithm.

CONCLUSION

In our proposed algorithm response time is better than existing algorithm. Also most important part of our proposed algorithm is process completing task within deadline. So, proposed algorithm is applicable in soft real time operating system. In proposed algorithm turnaround time and waiting time is also good. In our proposed algorithm response time of individual process is better than existing algorithm. And in CPU scheduling individual is always better than average. From the above comparison we observe that proposed algorithm is better than existing algorithm for soft real time operating system.

FUTURE WORK

This proposed algorithm is only for a soft real time system. So we can improve this proposed algorithm for real time(hard and firm) system using better deadline. Also we can improve turnaround time and waiting time for using some technique.

VOL. 1, ISSUE 2, NOV. 2012 ISSN NO: 2319-7471

REFERENCES

[1]. R. Mohantry and K. Patwari, "Priority Based Dynamic Round Robin Algorithm with Intelligent Time Slice for Soft Real Time Systems", IJACSA, Vol.2, No.2,February2011.

[2]. H. S. Behera, S .Sahi and s. k. bhoi, "Weighted Mean Priority Based Scheduling for Interactive Systems", Journal of Global Research in Computer Science,Vol2,No.5,may2011.

[3]. Silberschatz, Galvin and Gagne, "operating system concept", ISBN 9812-53-055-X,John Wiley & sons(ASIA) Pte Ltd. 6Ed 2004.

[4]. Shinpei Kato, Yutaka Ishikawa, Ragunathan (Raj) Rajkumar, "CPU Scheduling and Memory Management for Interactive Real-Time Applications".

[5]. Brandenburg B, Anderson J, "On the implementation of global real-time schedulers", in Proc. of the IEEE Real-Time Systems Symposium, pp 214–224.

[6]. Block A, Brandenburg B, Anderson J, Quint S (2008), "Adaptive multiprocessor real-time scheduling with feedback control", in Proc. of the Euromicro Conference on Real-Time Systems, pp 23–33.

[7]. Calandrino J, Leontyev H, Block A, Devi U, Anderson J (2006) LITMUS RT, "A testbed for empirically comparing real-time multiprocessor schedulers", In: Proc. of the IEEE Real-Time Systems Symposium, pp 111–123.

[8].Oikawa S, Rajkumar R (1999) Portable RT, "A portable resource kernel for guaranteed and enforced timing behavior", In: Proc. of the IEEE Real-Time Technology and Aplications Symposium, pp 111–120.

[9]. Palopoli L, Cucinotta T, Marzario L, Lipari G (2009), "AQuoSA: Adaptive quality of service architecture. Software–Practice and Experience 39:1–31.

[10]. Yang T, Liu T, Berger E, Kaplan S, Moss JB (2008), "Redline: First class support for in-teractivity in commodity operating systems", In: Proc. of the USENIX Symposium on Operating Systems Design and Implementation, pp 73–86 .