the weights of those blocked processes will be updated according to the given formula:

New_Weight = Old_Weight * (Process_ Weighting/ Avg_Process_Weighting)

In this formula, we try to avoid skew-ness in average waiting time by giving I/O bound processes higher weights on their return to ready queue.

In addition, we purpose a selective preemption technique in which I/O bound processes can preempt currently running process if they have more waiting time than the average waiting time for all the processes. This is because the process has been waiting for I/O and should get higher weight than currently running processes. [3][13]

3.2 Intelligent Time Slice for Round Robin

The proposed architecture focuses on the drawbacks of simple round robin architecture which is context switch. The proposed architecture eliminates the defects of implementing simple round robin architecture in real time operating system by introducing a concept called intelligent time slicing which depends on three aspects they are priority, average CPU burst, and context switch avoidance time. The proposed architecture allows the user is allowed to assign priority to the system. An assumption is made on average CPU burst which are reasonable to the system. A dedicated small processor used to reduce the burden of the main processor is assigned for calculating the time slice. The calculated time slice will be different and independent for each task and the tasks are fed into the ready queue and these tasks execute in the main processor with their individual time slices. The proposed architecture can be implemented in real time operating systems because of greater response time and throughput and the users can allocate priority to every individual task. The intelligent time slice is calculated by the dedicated small processor as shown in figure 3.1.[4]

Figure 3-1 Intelligent Time Slice

The priority, average CPU burst and context switch avoidance time along with original time slice is given to the small dedicated processor which calculates the new time slice dedicated to the corresponding tasks. The output of small dedicated processor will be the task id no, CPU burst and the calculated time slice these criteria are given to the main processor. These dedicated time slice are different and are exclusively allotted for each tasks and the tasks execute on the processor based on these time slicing. Thus the proposed architecture is superior to existing simple round robin architecture and can be implemented in real time operating systems.

Intelligent Time Slice Calculation

A new way of intelligent time slice calculation has been proposed which allocates the frame exclusively for each task based on priority, shortest CPU burst time and context switch avoidance time. Let the original time slice (OTS) is the time slice to be given to any process if it deserves no special consideration.

Intelligent Time Slice = Original Time Slice (OTS) + Priority Component (PC) + Shortness Component for CPU Burst Time + Context Switch Component (CSC)

Priority component (PC) is assigned by the user depending upon the priority which is inversely proportional to the priority number (higher the priority, greater the PC).

Shortness component (SC) is assigned inversely proportional to the length of the next CPU burst for the process. Shortest component should be lesser than assumed CPU burst.

Context Switch Component (CSC) is calculated as follows:

Computed Component (CC) = Priority Component (PC) + Shortness Component (SC) + Original Time Slice (OTS)

CC is deducted from the Assumed CPU burst (ATS). Let the result be called balance CPU burst.

Balanced CPU Burst = Assumed Time Slice (ATS) - Computed Component (CC).

If this Balanced CPU burst is less than OTS, it will be considered as Context Switch Component (CSC). [4]

3.3SRBRR Scheduling Algorithm

Introduction

SRBRR refers to the shortest remaining burst round robin scheduling. It controls and coordinates the use of the hardware among various application programs for various users. In multitasking and multiprocessing environment the way the processes are assigned to run on the available CPUs is called scheduling. Main goal of the scheduling is to maximize the different performance metrics such as CPU utilization, throughput and to minimize response time, waiting time and turnaround time. The scheduling is used in the real time applications like routing of data packets in computer networking, controlling traffic in airways, roadways and railways etc. In Round Robin (RR) every process has equal priority and is given a time quantum after which the process is preempted. Although RR gives improved response time and uses shared resources efficiently, its limitations are larger waiting time, larger turnaround time for processes with variable CPU bursts due to use of static time quantum. This motivates us to implement RR algorithm with sorted remaining burst time with dynamic time quantum concept.[5]

Definitions

A program in execution is called a process. The processes, waiting to be assigned to a processor are put in a queue called ready queue. The time for which a process holds the CPU is known as burst time. The time at which a process arrives is its arrival time.

Turnaround time is the amount of time to execute a particular process, while waiting time is the amount of time a process has been waiting in the ready queue and time from the submission of a request by the process till its first response is the response time. Scheduler selects a process from queues in some manner for its execution. In non-preemption, CPU is assigned to a process; it holds the CPU till its execution is completed. But in preemption, running process is forced to release the CPU by the newly arrived process. In time sharing system, the CPU executes multiple processes by switching among them very fast. The number of times CPU switches from one process to another is called as the number of context switches.

Scheduling algorithms

When there are number of processes in the ready queue, the algorithm which decides the order of execution of those processes is called scheduling algorithm. The various well known CPU scheduling algorithms are First Come First Serve (FCFS), Shortest Job First (SJF), Highest Response Ratio Next (HRRN) and Priority. All the above four algorithms are non-preemptive in nature and are not suitable for time sharing systems. Shortest Remaining Time Next (SRTN) and Round Robin (RR) are preemptive in nature. RR is most suitable for time sharing systems.

Related Work

Abielmona provided an analytical overview of a myriad of scheduling algorithms. The Proportional Share Scheduling Algorithm proposed by Helmy and Dekdouk combines low overhead of round robin algorithms and favor shortest jobs. Weight readjustment in the algorithm enables existing proportional share schedulers to significantly reduce, the unfairness in their allocation. The static time quantum which is a limitation of RR was removed by taking dynamic time quantum by Matarneh. The time quantum that was repeatedly adjusted according to the burst time of the running processes are considered to improve the waiting time, turn around time and number of context switches.

SRBRR Algorithm

In our proposed algorithm, the time quantum is taken as the median of the increasingly sorted burst time of all the processes.

Uniqueness of our Approach

In our algorithm, the jobs are sorted in ascending order of their burst time to give better turnaround time and waiting time like SRTN Algorithm. Performance of RR algorithm solely depends upon the size of time quantum. If it is very small, it causes too many context switches. If it is very large, the algorithm degenerates to FCFS. So our algorithm solves this problem by taking a dynamic time quantum where the time quantum is repeatedly adjusted according to the remaining burst time of currently running processes. To get the optimal time quantum, median of the burst time is taken as the time quantum.

Algorithm

In our algorithm, when processes are already present in the ready queue, their arrival times are assigned to zero before they are allotted to the CPU. The burst time and the number of processes (n) are accepted as input. Let TQ be the time quantum.

1. All the processes present in ready queue are sorted in ascending order.
2. While (ready queue != NULL)
3. TQ = median (remaining burst time of all the processes)
4. Assign TQ to process
5. Pi ->TQ
6. If (i<n) then go to step 3
7. If a new process is arrived, Update the counter n and go to step1 End of while
8. Average waiting time, average turnaround time and no. of context switches are calculated
9. End [5]

3.4Self-Adjustment Time Quantum RR

In this research paper present a solution to the time quantum problem by makes the operating systems adjusting the time quantum according to the burst time of the existed set of processes in the ready queue.

When operating system installed for the first time, it begins with a default time quantum value, which is subject to change after a period of time through which the operating system can identify the burst time for a subset of the programs used by the user. So, we assume that the system will not immediately take advantage of this method because it needs a short period of time to learn user behavior through the analysis of the burst time of the new processes. The determined time quantum represents real and optimal value because it based on real burst time unlike the other methods, which depend on fixed or possible time quantum value, determined by a variety methodologies such as guessing, fuzzy logic.

Repeatedly, when a new process loaded to be executed the operating system tests the status of the specified program which can be either 1 or 0.

When the status equals to 0 this means that the process is either being executed for the first time or it has been modified or updated since the last analysis. In this case the operating system assign a counter to find the burst time of the process and continues executing the processes in the ready queue on the current round including the new arrival process using the current time quantum Q, otherwise and when status is equal to 1, then the operating system recalculates the time quantum Q depending on the remaining burst time of all ready processes including the new arrival process.

I have found through experience that the optimal time quantum can be presented by the median for the set of processes in the ready queue, if the median less, than 25 then its value must be modified to 25 to avoid the overhead of the context switch. Formula 1 represents the value of time

quantum Q consequences for the median x:

$$\begin{cases} Y_{(N+1)/2} \text{ if N is odd} \\ 1(Y_{n/2})+(Y_{1+n/2}) \text{ if N is even} \\ 2 \end{cases}$$

Median x =

—

Where, Y is the number located in the middle of a group of numbers arranged in ascending order. Because the value of Q should not be less than 25, we can rewrite formula (1) in more specific form to fit with.

Q = { X, if x >=25

This means that 50% of the processes will be terminated through the first round and as time quantum calculated repeatedly for each round then 50% of the remaining processes will be terminated during the second round, with the same manner for the third round, fourth round, which is mean that the maximum number of rounds will be less than or equal to 6 whatever the number of process or their burst time. Figure 1 shows the significant decrease of the number of processes in each round.

The significant decrease of the number of processes, inevitably will lead to significant reduction in the number of context switch, which may pose high overhead on the operating system in many cases. The number of context switch can be represented mathematically as follow:

$$QT = [\sum_{r=1to6} k_r)] - 1$$

Where

QT = the total number of context switch

r = the total number of rounds, r = 1, 2…6

kr = the total number of processes in each round

In other variants of round robin scheduling algorithm the context switch occurs even if there is only a single process in the ready queue, where the operating system assigns the process a specific time quantum Q, when time quantum expires the process interrupted and again assigned the same time quantum Q, regardless,

While (ready queue<>null)

If new process Pi arrived then // check Pi status

If Pi status = 0 then assign new counter C_i for this

Process End if

End if

//Find new quantum

NQ= median (remaining burst time of all process in the ready queue with status = 1)

// continue executing the process using new quantum NQ

for i = 1 to n loop // n = the number of process in ready queue

Pi=NQ // assign the CPU to process Pi and give it slice of time = NQ

If Pi terminated normally and Pi, status = 0 then

Save Ci (Pi)

Let Pi status = 1

End if

End for

End while

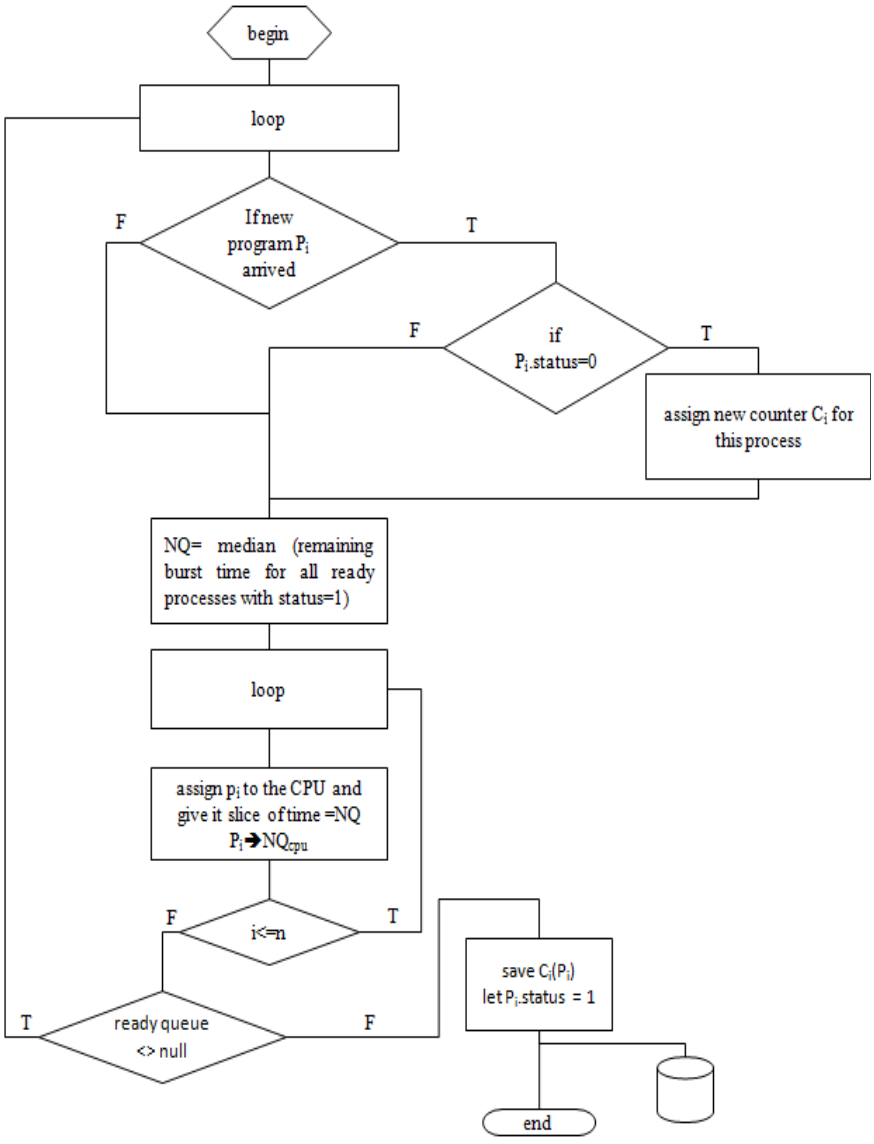

Figure 3-2 Flow chart for Self Adjustment Time Quantum

whether the process alone in the ready queue or not, which means that, there will be additional unnecessary context switches, while this problem does not occur at all in the new proposed algorithm; because in this case the time quantum will equal to the remaining burst time of the process. The proposed algorithm was designed to meet all scheduling criteria such as max CPU utilization, max throughput, min turnaround time, min waiting time and min response time. To evaluate the proposed method with regard to the above criteria, for the purpose of simplicity I will take a group of four processes in four different cases with random burst time and what should be mentioned here that the

number of processes does not change the result because the algorithm works effectively even if it used with a very large number of processes. [6]

3.5 Deficit Round Robin (DRR)

Also called as Deficit Weighted Round Robin (DWRR), is a modified weighted round robin scheduling discipline. DRR was proposed by M. Shreedhar and G. Varghese in 1995. It can handle packets of variable size without knowing their mean size. A maximum packet size number is subtracted from the packet length, and packets that exceed that number are held back until the next visit of the scheduler.

WRR serves every non-empty queue whereas DRR serves packets at the head of every non-empty queue whose deficit counter is greater than the packet's size at the head of the queue (HoQ). If the deficit counter is lower, then the queue is skipped and its credit is increased by some given value called quantum. This increased value is used to calculate the deficit counter the next time around when the scheduler examines this queue for serving its head-of-line packet. If the queue is served, then the Credit is decremented by the size of packet being served.

Scheduling is a key concept in computer multitasking, multiprocessing operating system and real-time operating system designs. Scheduling refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs. This assignment is carried out by software's known as a scheduler and dispatcher. The scheduler is concerned mainly with:

- **CPU utilization:** To keep the CPU as busy as possible.
- **Throughput:** Number of processes that complete their execution per time unit.
- **Turnaround:** Total time between submission of a process and its completion.
- **Waiting time:** Amount of time a process has been waiting in the ready queue.
- **Response time:** Amount of time it takes from when a request was submitted until the first response is produced.
- **Fairness:** Equal CPU time to each thread

In real-time environments, such as mobile devices for automatic control in industry (for example robotics), the scheduler also must ensure that processes can meet deadlines; this is crucial for keeping the system stable. Scheduled tasks are sent to mobile devices and managed through an administrative back end. [7][8]

3.6Weighted Round Robin (WRR)

The Weighted Round Robin Algorithm has been used for scheduling real time traffic in high speed switched network. It builds on the basic Round Robin scheme. Rather than giving all the ready jobs equal shares of the processor, different jobs may be given different weights. Here, the weight of a job refers to the fraction of processor time allotted to the job. Specifically, a job with weight wt gets wt time slices every round and the round is equal to the sum of the weights of all the jobs. By adjusting the weights of jobs, we can speed up or retard the progress of each job toward its completion. According to the weight every job decides the time quantum that means suppose a job assigns a 5ms weight then it uses time quantum 5ms. Each packet flow or connection has its own packet with some weight in queue in a network interface card. It is the simplest approximation of generalized processor sharing (GPS). While GPS serves infinitesimal amounts of data from each nonempty queue, WRR serves a number of packets for each nonempty queue.

Algorithm

WRR mechanism:

Step 1: Low = find smallest weight // calculate number of packets to be served each round.

Step 2: For each flows F

Step 3: F.packets_to_be_served = (F.weight) / low;

Step 4: Loop

For each non-empty flow queue F

Low (F.packets_to_be_served, F.packets_waiting) .times do

Serve Packet F.getPacket and goto step 3

Step 5: Exit[9][21]

Weaknesses

For the normalization of weights, required, a mean of packet size must be known. In IP networks with their variable packet size the mean has to be estimated which makes good GPS approximation hard to achieve in practice. Another weakness of WRR is that it cannot guarantee fair link sharing. Deficit Round Robin (DRR) is a variation of WRR that achieves better GPS approximation without knowing the mean packet size of each connection in advance.

In the review papers the problem is related to time quantum ,context switches, turn around time and average waiting time .The approach followed is dynamic approach for selecting time quantum but most real time systems built today are static.