Minimizing Response Time and Effective Utilization of I/O-bound Processes using “Approximate Zero Response Algorithm”

Dinesh Gupta¹, Sakshi Gupta², Neha Jhamb²
¹Asst. Prof., Department of Information Technology, DAVIET, Jalandhar-144008
²Department of Information Technology, DAVIET, Jalandhar-144008
daviet_dineshgupta@yahoo.com, sakshi4200@gmail.com, jhambneha30@gmail.com

Abstract

Various scheduling algorithm are available for the operating system to improve CPU utilization. Different scheduling algorithms have different properties that work on different scheduling criterias and the choice of a particular algorithm may favor one class of processes over another. SJF gives minimum average waiting time for a given set of processes. The Round Robin algorithm decreases the response time. In this paper we have proposed an algorithm which has response time approximately zero and it increases the efficiency of I/O bounded process.

Introduction

A process is a program in execution, and a program is a set of instructions put together to reach a determined state. The time for which a process needs a processor in order to be run is called CPU-burst time and the time of the process for which a process do I/O is called I/O-Burst time (the process frees the processor to perform some I/O). In fact, a process will run for a while (the CPU-burst time), wait in I/O queue, perform some I/O(the I/O-burst time), then run for a while more (the next CPU/burst time). A process always start with A CPU-burst and then it cycles between I/O and CPU bursts during its life time and it eventually terminates with a CPU burst[1].

Types of Processes:

There are two type of processes on the basis of execution:
1) I/O-bounded
2) CPU-bounded

A program is CPU bound if it requires more of CPU than I/O during its life time i.e. it spends the majority of its time simply using the CPU (executing instructions). A program is I/O bound if it needs more of I/O devices than CPU .
program that looks through a huge file for some data will often be I/O bound, since the bottleneck is then the reading of the data from disk. An I/O-bounded process would typically have many very short CPU bursts. A CPU-bound program might have a few very long CPU bursts.

**Sheduling Criteria**

When more than one process is in the ready state and there is only one CPU available, the operating system must decide which process to run first[4]. The part of operating system that makes the choice is called short term scheduler or CPU scheduler that directs the dispatcher to send the selected process for execution[5]. The algorithm that it uses to choose the process is called scheduling algorithm. There are several scheduling algorithms. Different scheduling algorithms have different properties and the choice of a particular algorithm may favor one class of processes over another. Many criteria have been suggested for comparing CPU scheduling algorithms and deciding which one is the best algorithm. Some of the criteria includes:

- **CPU utilization** – it means keeping the CPU as busy as possible. It may range from 40 percent to 90 percent.
- **Throughput** – no. of processes that completed their execution per time unit
- **Turnaround time** – The interval from the time of submission of a process to the time of completion. It is the sum of periods spent waiting to get into memory, waiting in ready queue, executing on the CPU, and doing I/O.
- **Waiting time** – It is the sum of the periods a process spent waiting in the ready queue.
- **Response time** – It is the amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment).

**Optimization Criteria:**

- Max CPU utilization
- Max throughput
- Min turnaround time

In this paper we propose an algorithm to optimize the response time (i.e. to minimize the response time).

**Sheduling Algorithms**

The fundamental scheduling algorithms and its characteristics are described in this section.

**First Come First Serve**

The most intuitive and simplest technique is to allow the first process submitted to run first. This approach is called as first-come, first-served (FCFS) scheduling. In effect, processes are inserted into the tail of a queue when they are submitted. The next process is taken from the head of the queue when each finishes running.

**i. Algorithm**

Step 1: The first requested process is allocated to the CPU first.
Step 2: The new processes are added at the tail of the ready queue.
Step 3: When the process terminates, dequeue the next process from head of ready queue and run it.

**ii. Characteristics**

- The lack of prioritization does permit every process to eventually complete, hence no starvation.
- Turnaround time, waiting time and response time is high.
- One Process with longest burst time can monopolize CPU, even if other process burst time is too short. Hence, throughput is low.

**Shortest Job First**

The process with least burst time is allocated CPU. A scheduler arranges the processes with the least burst time at head of the queue and longest burst time at tail of the queue. This requires advanced knowledge or estimations about the time required for a process to complete. This algorithm is designed for maximum throughput in most scenarios.

**i. Algorithm**

Step 1: Allocate the CPU to the process that has the shortest
burst time.
Step 2: If one or more process has the same burst time

{ Allocate the CPU to the process according to the FCFS scheduling
 }

ii. Characteristics

• The real difficulty with the SJF algorithm is, to know the length of the next CPU request.
• SJF minimizes the average waiting time because it services small processes before it services large ones. While it minimizes average wait time, it may penalize processes with high service time requests. If the ready list is saturated, then processes with large service times tend to be left in the ready list while small processes receive service. In extreme case, when the system has little idle time, processes with large service time will never be served. This total starvation of large processes is a serious liability of this algorithm.

iii. Shortest Remaining Time First

The SJF algorithm may be preemptive or nonpreemptive. The choice arises when a process arrives at the ready queue while a previous is executing. The new process may have a shorter next CPU burst than what is left of the currently executing process. A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its cpu burst. Preemptive SJF scheduling is sometimes called Shortest-Remaining-Time-First scheduling.

There is criteria to compute the CPU burst in case of SJF or SRTF. Forecasting the next CPU-burst time of a process which is a kind of time series, will be done by history of past CPU-bursts time which were executed in the processor before. Equation (1) shows the exponential average of the next CPU-burst time:

\[ \tau_{n+1} = \alpha t_n + (1-\alpha) \tau_n \]  

(1)

Where \( t_n \) is the time of the nth CPU-burst time, \( \alpha \) is the forgetting coefficient on the range of \([0,1]\) and \( \tau_{n+1} \) is the next CPU-burst time in the exponential average algorithm[2].

Round Robin

The Round Robin (RR) scheduling algorithm assigns a small unit of time, called time slice or quantum. The ready processes are kept in a queue. The scheduler goes around this queue, allocating the CPU to each process for a time interval of assigned quantum. New processes are added to the tail of the queue[3].

i. Algorithm

Step 1: Choose the time quantum and assign it for each process.
Step 2: Allocate the CPU to the process according to the FCFS scheduling.
Step 3: If (burst time of the process < time quantum).
{ Allocate the CPU to that process till it terminates.
}
Else
{ The process will occupy the CPU till the time quantum and it is added to the tail of the ready queue for the next round of execution.
}

ii. Characteristics

• Setting the quantum too short causes too many context switches and lower the CPU efficiency.
• Setting the quantum too long may cause poor response time and approximates FCFS.
• Because of high waiting times, deadlines are rarely met in a pure RR system.

Priority Scheduling

The operating system assigns a fixed priority rank to each process. Lower priority processes get interrupted by incoming higher priority processes.

i. Algorithm

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 and so on.
Step 3: If two or more process has equal-priority then
{
Allocate the CPU to the process according to the FCFS.

ii. Characteristics
- Starvation can happen to the low priority process.
- The waiting time gradually increases for the equal priority processes.
- Higher priority processes have smaller waiting time and response time.

**Proposed algorithm:**

**Approximate Zero Response Algorithm**

In this proposed algorithm the response time is approximately zero. When a new process arrives at the ready queue the CPU-scheduler gives the CPU to the new process for some time say \( N \) ms irrespective of the scheduling algorithm used by the CPU-scheduler. In this algorithm, when a process enters the ready list, a special priority \((\infty)\) will be given to the process. Once the process finishes with the first CPU-burst, the priority is changed according to the original algorithm used by the short-term scheduler.

Assigning special priority \((\infty)\) to the incoming process ensures that the process will get the CPU.

For example:

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

In case of priority algorithm, when a process with priority \((p)\) arrives at the ready list, the CPU will assign it special priority \((\infty)\). In the above example let us suppose that a new process arrives at the ready list with a priority \((2)\). Initially the priority will be set to \((\infty)\), and it will ensure that the process gets the CPU. Once the process finishes with the first CPU-burst, the priority will be reverted back to 2. If the two processes arrive at the same time then one of them get the CPU based on their original priority \((p)\) while the other processes has to wait. It is best for I/O bounded processes as they need less CPU burst.

In case of I/O bounded processes this algorithm can initialize the process and then they will get busy with I/O burst this will increase the throughput of the CPU and will increase its efficiency.

i. Algorithm

Step 1: Allocate the special priority (according to the original algorithm) to the process that has arrived so that the newly arrived process will get CPU for example:

In case of priority algorithm, highest priority is assigned to the process

Step 2: If two or more process arrives at the same time then

{ Allocate the CPU to the process according to the original algorithm and other process will wait. }

ii. Characteristics
- It reduces the response time to approximately zero.
- In case of I/O bounded process it increases the efficiency to a great extent and increases the efficiency of the systems that has high rate of I/O-bounded process

iii. Example:

<table>
<thead>
<tr>
<th>Process ID</th>
<th>Arrival Time (ms)</th>
<th>Burst Time (ms)</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>500</td>
<td>4</td>
</tr>
<tr>
<td>P2</td>
<td>25</td>
<td>620</td>
<td>5</td>
</tr>
<tr>
<td>P3</td>
<td>55</td>
<td>350</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>55</td>
<td>450</td>
<td>3</td>
</tr>
<tr>
<td>P5</td>
<td>115</td>
<td>300</td>
<td>1</td>
</tr>
<tr>
<td>P6</td>
<td>145</td>
<td>410</td>
<td>8</td>
</tr>
<tr>
<td>P7</td>
<td>175</td>
<td>436</td>
<td>4</td>
</tr>
<tr>
<td>P8</td>
<td>250</td>
<td>330</td>
<td>9</td>
</tr>
</tbody>
</table>

Here if the CPU-scheduler is following priority algorithm and value of \( N \) is 20 ms. When P1 will arrive it will get CPU. At 25 ms P2 will arrive and will get CPU for 20 ms irrespective of FCFS algorithm and will use CPU. At 55 ms P3 and P4 will arrive. Both the processes have same arrival time so original algorithm(priority algorithm) is used. P4 will get CPU as it has higher priority relative to P4 and P3 has to wait. When P5 will arrive at 115 ms, it will get the CPU for next 20 ms. At 145 ms, when P6 will
come it will get CPU for 20 ms. At 175 ms, P7 will get the CPU for 20 sec. At 205 ms when P7 will leave CPU, the CPU will be assigned to P6 as it has the highest priority. At 250 ms, when P8 will arrive it will get the CPU for 20ms and will keep the CPU as it has the highest priority at that time and will leave the cpu when the new process will enter the CPU.

It reduces the Response time to zero (approximately). Suppose P2 is an I/O bounded process and need CPU for first 10 ms and after that has I/O burst. So when P2 will arrive it will get the CPU and then will get burst will I/O burst and hence increases the Throughput and reduces the response time approximately to zero.

**NOTE:**
The value of N has to be taken carefully. It should not be so large and not so small. If N will be large the algorithm will be of no use and if N will be small it will not give the start needed by the I/O burst. So, value of N should be decided carefully.

**iv. Comparative Results of Response time:**

<table>
<thead>
<tr>
<th>PID</th>
<th>Arr. Time</th>
<th>Burst time</th>
<th>Response Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SJF</td>
<td>SRTF</td>
<td>RR</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>15</td>
<td>55</td>
<td>5</td>
</tr>
<tr>
<td>P3</td>
<td>25</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>P4</td>
<td>25</td>
<td>70</td>
<td>130</td>
</tr>
<tr>
<td>P5</td>
<td>45</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

In this the average response time is:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Average Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SJF</td>
<td>46</td>
</tr>
<tr>
<td>SRTF</td>
<td>41</td>
</tr>
</tbody>
</table>

**Graph:**

In this paper we have concluded that use of Approximately Zero response Algorithm will decrease the response time and will help in increasing the throughput of system where frequency of I/O-bound processes is much.

**References**

