**Bahria University, Lahore Campus**

Department of Computer Sciences

Lab Journal 8

**(Spring 2024)**

|  |  |  |
| --- | --- | --- |
| Course: | **Operating System Lab** | Date: 05/09/2024 |
| Course Code: | CSL – 320 | Max Marks: 20 |
| Faculty’s Name: | **ABDULLAH** |  |

Name: **Muhammad Hammad** Enroll No: **03-134221-024**

Objective(s) :

To write a C program to implement CPU scheduling algorithm for Priority Scheduling and Shortest Remaining Time First.

## Lab Tasks :

## Task 01: Write a C program to implement Priority Scheduling algorithm.

**Task 02:** Write the output of a C program for Shortest Remaining Time First.

**Lab Grading Sheet :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Max Marks** | **Obtained Marks** | **Comments(*if any*)** |
| 1. | 10 |  |  |
| 2. | 10 |  |  |
| **Total** | **20** |  | **Signature** |

**Note : Attempt all tasks and get them checked by your Lab. Instructor**.

# Lab 08: Priority and Shortest Remaining Time First Scheduling

**Objective(s):**

To write a C program to implement CPU scheduling algorithm for Priority Scheduling and Shortest Remaining Time First.

**Tool(s) used:**

Ubuntu, VIM Editor

CPU scheduler will decide which process should be given the CPU for its execution. For this its use different algorithm to choose among the processes. One of the scheduling algorithms is priority scheduling. It is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Moreover, another scheduling algorithm is Shortest Remaining Time First. Shortest remaining time (SRTF) is the preemptive version of the SJF algorithm. The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion. Impossible to implement in interactive systems where required CPU time is not known. It is often used in batch environments where short jobs need to give preference.

## Task 01: Write a C program to implement Priority Scheduling algorithm.

Algorithm:

**Step 1:** Take number of process from user.  
**Step 2:** Initialize all the structure elements with 0. Receive inputs from the user to fill process id, burst time and Priority.  
**Step 3:** Sort the array in the ascending order based on priority.  
**Step 4:** Calculate the Turnaround time and Waiting time for the remaining processes.  
**Step 5:** Waiting time of one process is the total service time of all the previous  
processes.  
**Step 6:** Total time of process is calculated by adding its Waiting time and its Service  
time.  
**Step 7:** Calculate the average waiting time and average turnaround time  
**Step 8:** Display the result in the following format for each process.

printf("Id \t Burst Time \t Priority \t Waiting Time \t Total Time");

|  |
| --- |
| Code: #include <iostream>#include <algorithm>using namespace std;struct Process {int id;int burst\_time;int priority;int waiting\_time;int total\_time;};bool comparePriority(const Process &a, const Process &b) {return a.priority < b.priority;}int main() {int n;cout << "Enter the number of processes: ";cin >> n;Process processes[n];// Input process detailsfor (int i = 0; i < n; i++) {cout << "Enter Process ID, Burst Time, and Priority for process " << i + 1 << ": ";cin >> processes[i].id >> processes[i].burst\_time >> processes[i].priority;processes[i].waiting\_time = 0;processes[i].total\_time = 0;}// Sort processes based on prioritysort(processes, processes + n, comparePriority);// Calculate waiting time and total timeprocesses[0].waiting\_time = 0;processes[0].total\_time = processes[0].burst\_time;for (int i = 1; i < n; i++) {processes[i].waiting\_time = processes[i - 1].total\_time;processes[i].total\_time = processes[i].waiting\_time + processes[i].burst\_time;}// Calculate average waiting time and average turnaround timedouble avg\_waiting\_time = 0, avg\_turnaround\_time = 0;for (int i = 0; i < n; i++) {avg\_waiting\_time += processes[i].waiting\_time;avg\_turnaround\_time += processes[i].total\_time;}avg\_waiting\_time /= n;avg\_turnaround\_time /= n;// Display process detailscout << "\nId \t Burst Time \t Priority \t Waiting Time \t Total Time\n";for (int i = 0; i < n; i++) {cout << processes[i].id << "\t" << processes[i].burst\_time << "\t\t" << processes[i].priority << "\t\t"<< processes[i].waiting\_time << "\t\t" << processes[i].total\_time << endl;}// Display average waiting time and average turnaround timecout << "\nAverage Waiting Time: " << avg\_waiting\_time << endl;cout << "Average Turnaround Time: " << avg\_turnaround\_time << endl;return 0;} |
| **OUTPUT** |

### Task 02: Write the output of a C program for Shortest Remaining Time First.

## Code

#include <stdio.h>

int main()

{

      int arrival\_time[10], burst\_time[10], temp[10];

      int i, smallest, count = 0, time, limit;

      double wait\_time = 0, turnaround\_time = 0, end;

      float average\_waiting\_time, average\_turnaround\_time;

      printf("nEnter the Total Number of Processes:t");

      scanf("%d", &limit);

      printf("nEnter Details of %d Processesn", limit);

      for(i = 0; i < limit; i++)

      {

            printf("nEnter Arrival Time:t");

            scanf("%d", &arrival\_time[i]);

            printf("Enter Burst Time:t");

            scanf("%d", &burst\_time[i]);

            temp[i] = burst\_time[i];

      }

      burst\_time[9] = 9999;

      for(time = 0; count != limit; time++)

      {

            smallest = 9;

            for(i = 0; i < limit; i++)

            {

                  if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

                  {

                        smallest = i;

                  }

            }

            burst\_time[smallest]--;

            if(burst\_time[smallest] == 0)

            {

                  count++;

                  end = time + 1;

                  wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

                  turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

            }

      }

      average\_waiting\_time = wait\_time / limit;

      average\_turnaround\_time = turnaround\_time / limit;

      printf("nnAverage Waiting Time:t%lfn", average\_waiting\_time);

      printf("Average Turnaround Time:t%lfn", average\_turnaround\_time);

      return 0;

}

|  |
| --- |
| #include <iostream>  using namespace std;  int main() {  int arrival\_time[10], burst\_time[10], temp[10];  int count = 0, limit;  // Input the number of processes  cout << "Enter the Total Number of Processes: ";  cin >> limit;  // Input arrival time and burst time for each process  cout << "Enter Details of " << limit << " Processes\n";  for (int i = 0; i < limit; i++) {  cout << "Enter Arrival Time for Process " << i+1 << ": ";  cin >> arrival\_time[i];  cout << "Enter Burst Time for Process " << i+1 << ": ";  cin >> burst\_time[i];  temp[i] = burst\_time[i]; // Store burst time in temporary array  }  // Simulate the scheduling algorithm  int time = 0;  while (count != limit) {  int smallest = -1; // Index of the process with the smallest burst time  for (int i = 0; i < limit; i++) {  if (arrival\_time[i] <= time && burst\_time[i] > 0) {  if (smallest == -1 || burst\_time[i] < burst\_time[smallest]) {  smallest = i;  }  }  }  // If no process is ready to execute, move time forward  if (smallest == -1) {  time++;  continue;  }  burst\_time[smallest]--; // Execute the process for one time unit  time++; // Increment time  if (burst\_time[smallest] == 0) { // If process completes execution  count++; // Increment completed process count  int end = time;  int wait = end - arrival\_time[smallest] - temp[smallest];  int turnaround = end - arrival\_time[smallest];  cout << "Process " << smallest+1 << " Completed.\n";  cout << "Wait Time: " << wait << ", Turnaround Time: " << turnaround << endl;  }  }  // Calculate and output average waiting time and turnaround time  double total\_wait\_time = 0, total\_turnaround\_time = 0;  for (int i = 0; i < limit; i++) {  int end = arrival\_time[i] + temp[i];  total\_wait\_time += end - arrival\_time[i] - temp[i];  total\_turnaround\_time += end - arrival\_time[i];  }  double average\_waiting\_time = total\_wait\_time / limit;  double average\_turnaround\_time = total\_turnaround\_time / limit;  cout << "Average Waiting Time: " << average\_waiting\_time << endl;  cout << "Average Turnaround Time: " << average\_turnaround\_time << endl;  return 0;  } |

**OUTPUT**

|  |
| --- |
|  |