**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:\_Affan Ahmad \_\_\_\_\_\_\_\_\_ Enroll No: \_03-134221-003\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 <vector>

#include <algorithm>

using namespace std;

struct Process {

int process\_id;

int arrival\_time;

int burst\_time;

int priority;

int waiting\_time;

int turnaround\_time;

};

bool compare\_priority(const Process &a, const Process &b) {

return a.priority < b.priority;

}

void calculate\_waiting\_time(vector<Process> &processes) {

int total\_waiting\_time = 0;

processes[0].waiting\_time = 0; // First process has 0 waiting time

for (size\_t i = 1; i < processes.size(); i++) {

processes[i].waiting\_time = processes[i - 1].waiting\_time + processes[i - 1].burst\_time;

total\_waiting\_time += processes[i].waiting\_time;

}

}

void calculate\_turnaround\_time(vector<Process> &processes) {

int total\_turnaround\_time = 0;

for (size\_t i = 0; i < processes.size(); i++) {

processes[i].turnaround\_time = processes[i].waiting\_time + processes[i].burst\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

}

}

void display\_results(const vector<Process> &processes) {

cout << "Process ID\tArrival Time\tBurst Time\tPriority\tWaiting Time\tTurnaround Time\n";

for (size\_t i = 0; i < processes.size(); i++) {

cout << processes[i].process\_id << "\t\t" << processes[i].arrival\_time << "\t\t"

<< processes[i].burst\_time << "\t\t" << processes[i].priority << "\t\t"

<< processes[i].waiting\_time << "\t\t" << processes[i].turnaround\_time << endl;

}

}

void priority\_scheduling(vector<Process> &processes) {

// Sort processes based on priority (higher priority gets executed first)

sort(processes.begin(), processes.end(), compare\_priority);

// Calculate waiting time and turnaround time

calculate\_waiting\_time(processes);

calculate\_turnaround\_time(processes);

// Display results

display\_results(processes);

}

int main() {

int n;

cout << "Enter the number of processes: ";

cin >> n;

vector<Process> processes(n);

for (int i = 0; i < n; i++) {

cout << "Enter details for process " << i + 1 << ":\n";

cout << "Process ID: ";

cin >> processes[i].process\_id;

cout << "Arrival Time: ";

cin >> processes[i].arrival\_time;

cout << "Burst Time: ";

cin >> processes[i].burst\_time;

cout << "Priority: ";

cin >> processes[i].priority;

}

priority\_scheduling(processes);

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;

}

## **C++ Version:**

#include <iostream>

using namespace std;

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;

cout << "Enter the Total Number of Processes: ";

cin >> limit;

cout << "Enter Details of " << limit << " Processes\n";

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

{

cout << "Enter Arrival Time: ";

cin >> arrival\_time[i];

cout << "Enter Burst Time: ";

cin >> 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;

cout << "\nAverage Waiting Time: " << average\_waiting\_time << endl;

cout << "Average Turnaround Time: " << average\_turnaround\_time << endl;

return 0;

}

**OUTPUT**

