Lab8: CPU Scheduling

1. Objective

- Understand CPU scheduling and get familiar with different scheduling algorithms.

2. Syllabus

- Understand the different CPU scheduling algorithms;
- Implement the CPU scheduling with C++.

3. Prerequisite

- C++ language
- Computer which run Linux system (e.g. Ubuntu)

4. Concepts and Principles of CPU Scheduling

Please refer to the chapter 6. The main contents have been lectured in the seventh week, 2017/10/23 and 2017/10/25. The slide of this chapter can be found in the course website: http://www.thinkmesh.net/~jiangxl/teaching/106F14A/.

5. Experimental Contents

5.1 Program for FCFS Scheduling

Description:

Given n processes with their burst times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm.

First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.

In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

Here we are considering that arrival time for all processes is 0.

Note: In this post, we have assumed arrival times as 0, so turn around and completion times are same.

```
// C++ program for implementation of FCFS
// scheduling
#include<iostream>
using namespace std;

// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
                          int bt[], int wt[])
{
    // waiting time for first process is 0
    wt[0] = 0;
    // calculating waiting time
    for (int   i = 1; i < n ; i++ )
        wt[i] =   bt[i-1] + wt[i-1] ;
}

// Function to calculate turn around time
void findTurnAroundTime( int processes[], int n,
                         int bt[], int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int   i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}

//Function to calculate average time
void findavgTime( int processes[], int n, int bt[])
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    //Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt);

    //Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);

    //Display processes along with all details
    cout << "Processes   "<< " Burst time   "
         << " Waiting time   " << " Turn around time\n";

    // Calculate total waiting time and total turn
    // around time
    for (int   i=0; i<n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << "      " << i+1 << "\t\t" << bt[i] <<"\t       "
            << wt[i] <<"\t\t   " << tat[i] <<endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
```

```
              << (float)total_tat / (float)n;
}

// Driver code
int main()
{
        //process id's
        int processes[] = { 1, 2, 3};
        int n = sizeof processes / sizeof processes[0];

        //Burst time of all processes
        int    burst_time[] = {10, 5, 8};

        findavgTime(processes, n,    burst_time);
        return 0;
}
```

File Name: FCFS_Scheduling.cpp

How to run: g++ -o FCFS_Scheduling FCFS_Scheduling.cpp

```
g++ -o FCFS_Scheduling FCFS_Scheduling.cpp
./ FCFS_Scheduling
```

5.2 Program for Shortest Job First (SJF) scheduling

Description: Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm. Shortest Job first has the advantage of having minimum average waiting time among all scheduling algorithms.

- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.
- It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.

Note: In this post, we have assumed arrival times as 0, so turn around and completion times are same.

```
// C++ program to implement Shortest Job first
#include<bits/stdc++.h>
using namespace std;

struct Process
{
      int pid; // Process ID
      int bt;    // Burst Time
```

```
};

// This function is used for sorting all
// processes in increasing order of burst
// time
bool comparison(Process a, Process b)
{
        return (a.bt < b.bt);
}

// Function to find the waiting time for all
// processes
void findWaitingTime(Process proc[], int n, int wt[])
{
    // waiting time for first process is 0
    wt[0] = 0;

    // calculating waiting time
    for (int i = 1; i < n ; i++ )
        wt[i] = proc[i-1].bt + wt[i-1] ;
}

// Function to calculate turn around time
void findTurnAroundTime(Process proc[], int n,
                        int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = proc[i].bt + wt[i];
}

//Function to calculate average time
void findavgTime(Process proc[], int n)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    // Function to find waiting time of all processes
    findWaitingTime(proc, n, wt);

    // Function to find turn around time for all processes
    findTurnAroundTime(proc, n, wt, tat);

    // Display processes along with all details
    cout << "\nProcesses "<< " Burst time "
         << " Waiting time " << " Turn around time\n";

    // Calculate total waiting time and total turn
    // around time
    for (int i = 0; i < n; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        cout << " " << proc[i].pid << "\t\t"
             << proc[i].bt << "\t " << wt[i]
             << "\t\t " << tat[i] <<endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    Process proc[] = {{1, 6}, {2, 8}, {3, 7}, {4, 3}};
    int n = sizeof proc / sizeof proc[0];

    // Sorting processes by burst time.
    sort(proc, proc + n, comparison);

    cout << "Order in which process gets executed\n";
    for (int i = 0 ; i < n; i++)
        cout << proc[i].pid <<" ";

    findavgTime(proc, n);
    return 0;
}
```

File name: SJF_Scheduling.cpp

How to run:
```
g++ -o SJF_Scheduling SJF_Scheduling.cpp
./SJF_Scheduling
```

5.3 Program for FCFS Scheduling | set two(Processes with different arrival times)

Description: We have already discussed FCFS Scheduling of processes with same arrival time. In this post, scenario when processes have different arrival times are discussed. Given n processes with

their burst times and arrival times, the task is to find average waiting time and average turn around time using FCFS scheduling algorithm.
FIFO simply queues processes in the order they arrive in the ready queue. Here, the process that comes first will be executed first and next process will start only after the previous gets fully executed.
Implementation: 1- Input the processes along with their burst time(bt) and arrival time(at)
2- Find waiting time for all other processes i.e. for a given process i: wt[i] = (bt[0] + bt[1] +...... bt[i-1]) - at[i]
3- Now find turn around time = waiting_time + burst_time for all processes
4- Average waiting time = total_waiting_time / no_of_processes
5- Average turn around time = total_turn_around_time / no_of_processes

---

```
// C++ program for implementation of FCFS
// scheduling with different arrival time
#include<iostream>
using namespace std;

// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n, int bt[],
                                    int wt[], int at[])
{
    int service_time[n];
    service_time[0] = 0;
    wt[0] = 0;

    // calculating waiting time
    for (int i = 1; i < n ; i++)
    {
        // Add burst time of previous processes
        service_time[i] = service_time[i-1] + bt[i-1];

        // Find waiting time for current process =
        // sum - at[i]
        wt[i] = service_time[i] - at[i];

        // If waiting time for a process is in negative
        // that means it is already in the ready queue
        // before CPU becomes idle so its waiting time is 0
        if (wt[i] < 0)
            wt[i] = 0;
    }
}
```

```
// Function to calculate turn around time
void findTurnAroundTime(int processes[], int n, int bt[],
                                                int wt[], int tat[])
{
    // Calculating turnaround time by adding bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}

// Function to calculate average waiting and turn-around
// times.
void findavgTime(int processes[], int n, int bt[], int at[])
{
    int wt[n], tat[n];

    // Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt, at);

    // Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details
    cout << "Processes " << " Burst Time " << " Arrival Time "
         << " Waiting Time " << " Turn-Around Time "
         << " Completion Time \n";
    int total_wt = 0, total_tat = 0;
    for (int i = 0 ; i < n ; i++)
    {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        int compl_time = tat[i] + at[i];
        cout << " " << i+1 << "\t\t" << bt[i] << "\t\t"
             << at[i] << "\t\t" << wt[i] << "\t\t "
             << tat[i]    <<    "\t\t " << compl_time << endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}

// Driver code
int main()
```

```
{
    // Process id's
    int processes[] = {1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    // Burst time of all processes
    int burst_time[] = {5, 9, 6};

    // Arrival time of all processes
    int arrival_time[] = {3, 3, 6};

    findavgTime(processes, n, burst_time, arrival_time);

    return 0;
}
```

File name: FCFS_Scheduling_Set2.cpp

How to run:

```
g++ -o FCFS_Scheduling_Set2 FCFS_Scheduling_Set2.cpp
./FCFS_Scheduling_Set2
```

6. Conclusion:

In this chapter, three experiments have been completed. Now we know how the basic algorithm runs and how to describe them in C++.