Operating System Project

Implementation of CPU scheduling algorithms

Fatemeh Rashidi

Basic Concepts

CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the **CPU scheduler**, which selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.

Dispatcher

The **dispatcher** is the module that gives control of the CPU's core to the process selected by the CPU scheduler.

Dispatch latency

The time it takes for the dispatcher to stop one process and start another running.

Basic Concepts

CPU-scheduling decisions may take place under the following four circumstances:

- When a process switches from the running state to the waiting state
- When a process switches from the running state to the ready state
- When a process switches from the waiting state to the ready state
- When a process terminates

Preemptive Scheduling

When a process switches from running state to ready state or from the waiting state to ready state. (circumstances 2 and 3)

Nonpreemptive scheduling

Under nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases it either by terminating or by switching to the waiting state.

Scheduling Criteria

- **CPU utilization:** Keep the CPU as busy as possible.
- **Throughput:** Number of processes that are completed per time unit.
- **Turnaround Time:** The interval from the time of submission of a process to the time of completion
- **Waiting Time:** The amount of time that a process spends waiting in the ready queue.
- **Response Time:** The time from the submission of a request until the first response is produced.

Goal

- Increase CPU utilization and throughput as it's possible.
- Decrease turnaround time, waiting time, and response time as it's possible.

Tool

- CPU scheduling algorithms

Scheduling algorithms

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU's core.

- **FCFS (First Come, First Serve)**
- **SJF (Shortest Job First)**
- **RR (Round-Robin)**
- **Priority**
- **Multilevel Queue**

Compare CPU scheduling algorithms using:

- Average waiting time
- Average turnaround time

Example 1

Assume that processes P1, P2, and P3 arrive at time 0 with the same order with the below information:

Process	Arrival Time	Burst Time
P1	0	10
P2	0	5
P3	0	20

Ready queue at time 0:

P1	P2	P3

- After 10 time units, P1 will be terminated.
  - Ready queue in the interval (0, 10):

P2	P3

- In the interval (10, 15), P2 will run on the CPU.
  - Ready queue in the interval (10, 15):

P3

- At time 15, P3 will start to execute, until it terminates (35) and the queue would be empty.

Processes	Arrival Time	Burst Time	Waiting Time	Turnaround Time
P1	0	10	0	10
P2	0	5	10	15
P3	0	20	15	35

Finally, we have:

$$\frac{\sum_{i=1}^{3} WaitingTime[i]}{3} = \frac{0+10+15}{3} = 8.333$$

$$\frac{\sum_{i=1}^{3} TurnaroundTime[i]}{3} = \frac{10+15+35}{3} = 20$$

Example 2

Assume that processes P1, P2, P3, and P4 arrive at different times, as below:

Process	Arrival Time	Burst Time
P1	0	10
P2	3	5
P3	14	10
P4	14	15

- At time 0, only the process P1 is in the queue:

P1

- P1 will execute on the CPU in the interval (0, 10).
- In the interval (0, 3), the queue would be empty.
- At time 3, while P1 is executing, P2 enters the queue and waits until P1 terminates. So, in the interval (3, 10), the queue would be like below:

P2

- In the interval (10, 15), P2 is running on the CPU.
- At time 14, while P2 is executing, P3 and P4 will enter the queue simultaneously.

P3	P4

- At time 15, when P2 terminates, P3 will start to run on the CPU and P4 is still in the queue.

P4

- After P3 terminates, at time 25, P4 will start to execute on the CPU and the queue would be empty.

Finally, we have:

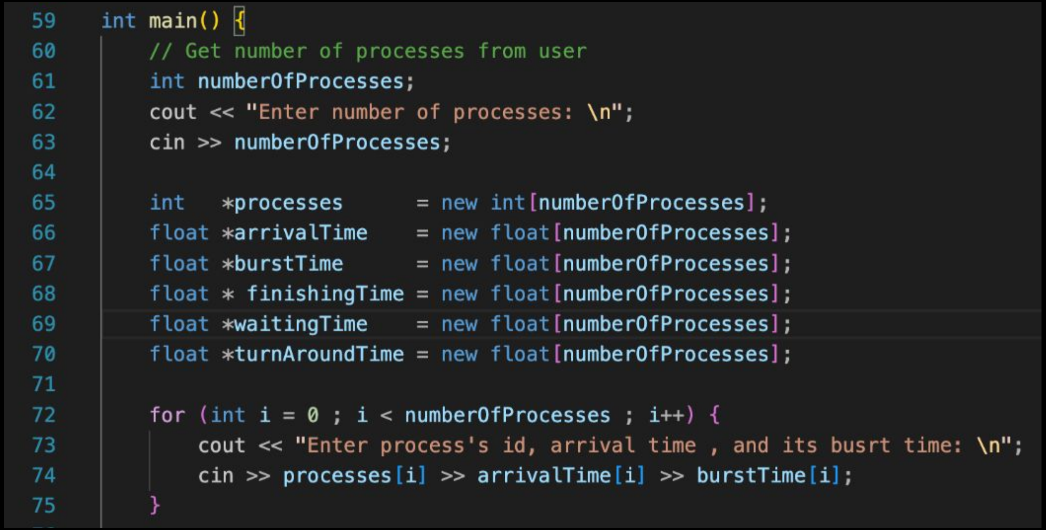
Process	Arrival Time	Burst Time	Waiting Time	Turnaround Time
P1	0	10	0	10
P2	3	5	7	12
P3	14	10	1	11
P4	14	15	11	26

$$\frac{\sum_{i=1}^{4} WaitingTime[i]}{4} = \frac{0+7+1+11}{4} = 4.75$$

$$\frac{\sum_{i=1}^{4} TurnaroundTime[i]}{4} = \frac{10+12+11+26}{4} = 14.75$$

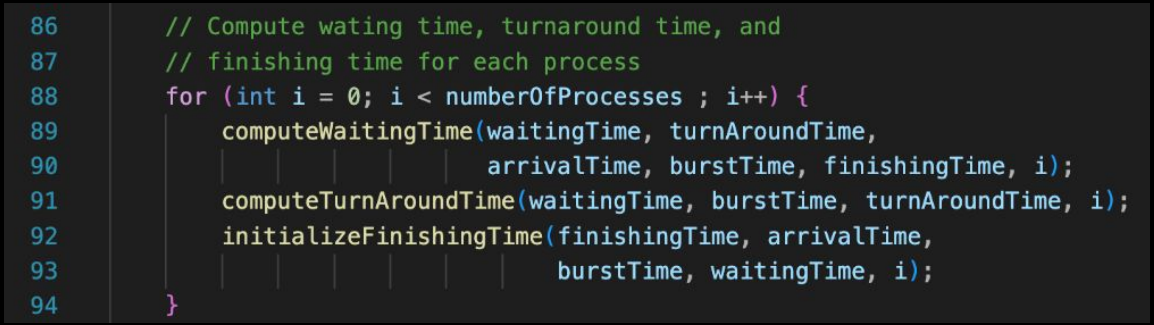
FCFS Implementation

- First of all, get number of processes that are needed to be scheduled from the user.
- Define below arrays:
    - Processes: store process ids.
    - arrivalTime: stores arrival time of processes.
    - busrtTime: stores burst time of processes.
    - finishingTime: stores the finishing time of each process.
    - waitingTime: stores the waiting time for processes.
    - turnAroundTime: stores the turnaround time for each process.
- Get process id, arrival time, and burst time for each process and store them in the corresponding arrays.



```cpp
int main() {
    // Get number of processes from user
    int numberOfProcesses;
    cout << "Enter number of processes: \n";
    cin >> numberOfProcesses;

    int   *processes      = new int[numberOfProcesses];
    float *arrivalTime    = new float[numberOfProcesses];
    float *burstTime      = new float[numberOfProcesses];
    float * finishingTime = new float[numberOfProcesses];
    float *waitingTime    = new float[numberOfProcesses];
    float *turnAroundTime = new float[numberOfProcesses];

    for (int i = 0 ; i < numberOfProcesses ; i++) {
        cout << "Enter process's id, arrival time , and its busrt time: \n";
        cin >> processes[i] >> arrivalTime[i] >> burstTime[i];
    }
```

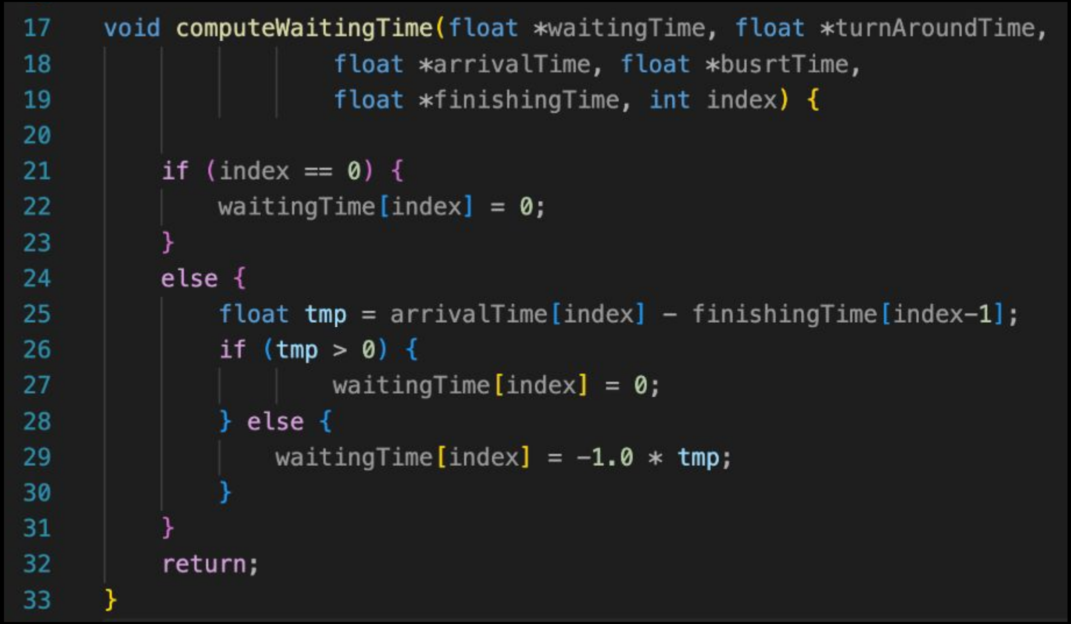
- Compute waiting time, turnaround time, and finishing time for each process.



```
// Compute wating time, turnaround time, and
// finishing time for each process
for (int i = 0; i < numberOfProcesses ; i++) {
    computeWaitingTime(waitingTime, turnAroundTime,
                       arrivalTime, burstTime, finishingTime, i);
    computeTurnAroundTime(waitingTime, burstTime, turnAroundTime, i);
    initializeFinishingTime(finishingTime, arrivalTime,
                            burstTime, waitingTime, i);
}
```

Compute waiting time, using its definition

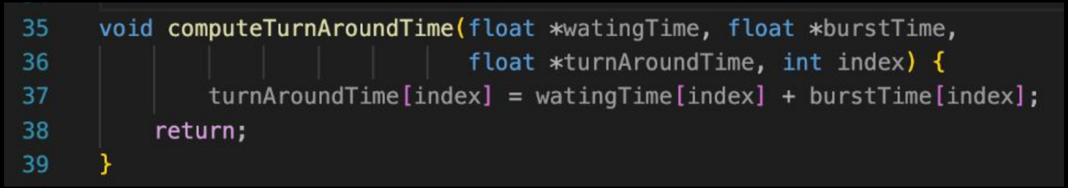
(The amount of time that a process spends waiting in the ready queue.)



```c
void computeWaitingTime(float *waitingTime, float *turnAroundTime,
                        float *arrivalTime, float *busrtTime,
                        float *finishingTime, int index) {

    if (index == 0) {
        waitingTime[index] = 0;
    }
    else {
        float tmp = arrivalTime[index] - finishingTime[index-1];
        if (tmp > 0) {
                waitingTime[index] = 0;
        } else {
            waitingTime[index] = -1.0 * tmp;
        }
    }
    return;
}
```

Compute turnaround time using its definition

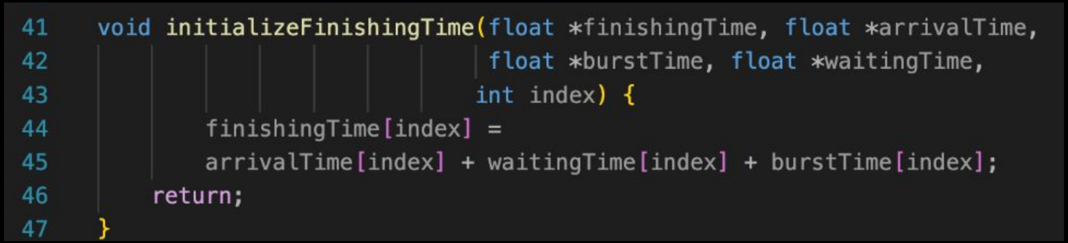
(The interval from the time of submission of a process to the time of completion)



```c
void computeTurnAroundTime(float *watingTime, float *burstTime,
                           float *turnAroundTime, int index) {
        turnAroundTime[index] = watingTime[index] + burstTime[index];
    return;
}
```

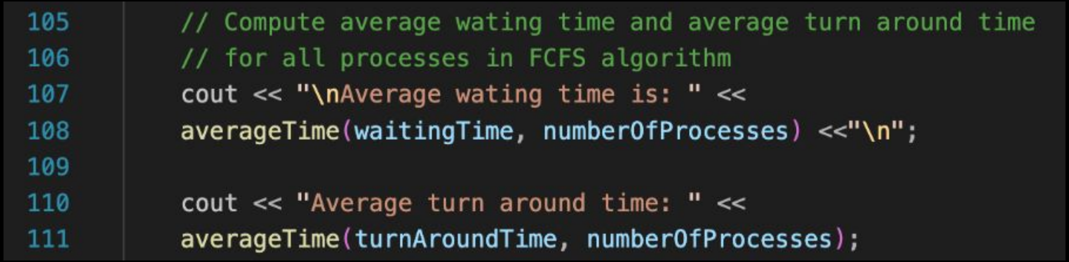
Compute finishing time for each process

(The time that each process terminates)

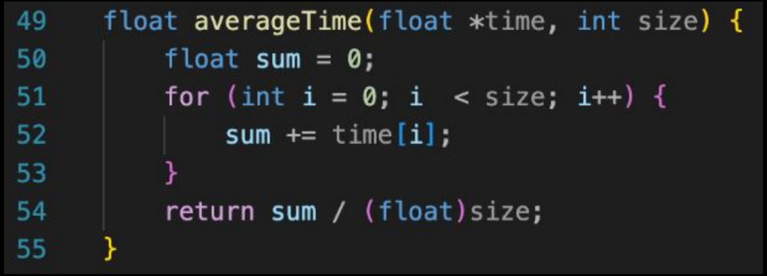


```c
void initializeFinishingTime(float *finishingTime, float *arrivalTime,
                             float *burstTime, float *waitingTime,
                            int index) {
        finishingTime[index] =
        arrivalTime[index] + waitingTime[index] + burstTime[index];
    return;
}
```

Compute average waiting time and turnaround time; using the function averageTime().



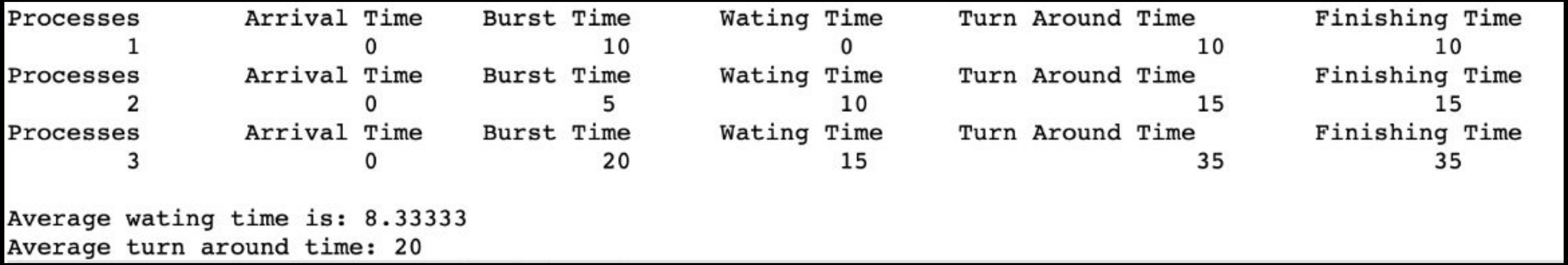
```
// Compute average wating time and average turn around time
// for all processes in FCFS algorithm
cout << "\nAverage wating time is: " <<
averageTime(waitingTime, numberOfProcesses) <<"\n";

cout << "Average turn around time: " <<
averageTime(turnAroundTime, numberOfProcesses);
```



```
float averageTime(float *time, int size) {
    float sum = 0;
    for (int i = 0; i  < size; i++) {
        sum += time[i];
    }
    return sum / (float)size;
}
```

Output

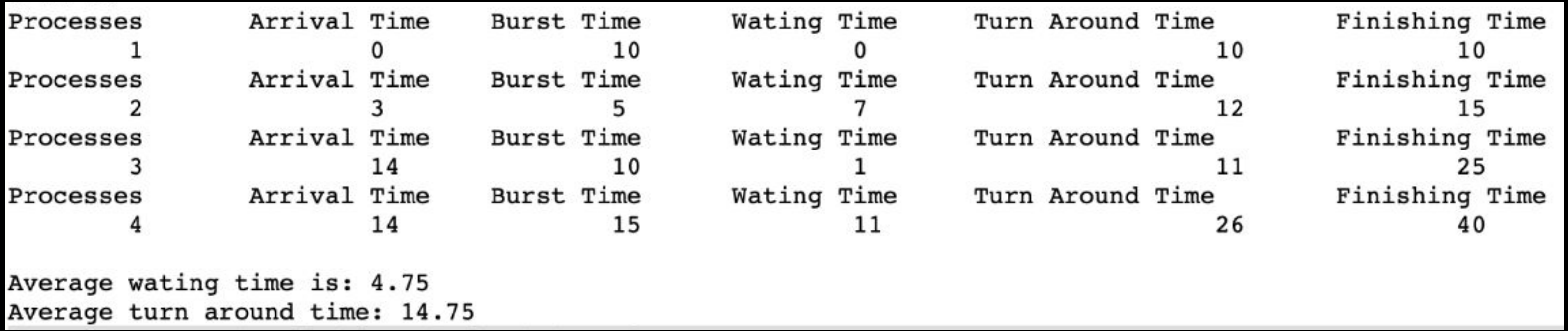
Output of the previous implementation for the first example



```
Processes        Arrival Time     Burst Time       Wating Time      Turn Around Time         Finishing Time
        1                0                10                0                       10                10
Processes        Arrival Time     Burst Time       Wating Time      Turn Around Time         Finishing Time
        2                0                 5               10                       15                15
Processes        Arrival Time     Burst Time       Wating Time      Turn Around Time         Finishing Time
        3                0                20               15                       35                35

Average wating time is: 8.33333
Average turn around time: 20
```

→ The above output is what we expected to be for its corresponding input data.

Output of the previous implementation for the second example



```
Processes        Arrival Time     Burst Time       Wating Time      Turn Around Time        Finishing Time
        1                0                10                0                      10                10
Processes        Arrival Time     Burst Time       Wating Time      Turn Around Time        Finishing Time
        2                3                 5                7                      12                15
Processes        Arrival Time     Burst Time       Wating Time      Turn Around Time        Finishing Time
        3                14               10                1                      11                25
Processes        Arrival Time     Burst Time       Wating Time      Turn Around Time        Finishing Time
        4                14               15               11                      26                40

Average wating time is: 4.75
Average turn around time: 14.75
```

→ The above output is what we expected to be for its corresponding input data.

Problem of FCFS algorithm

FCFS algorithm is non-preemptive in nature, that is, once CPU time has been allocated to a process, other processes can get CPU time only after the current process has finished. This property of FCFS scheduling leads to the situation called **Convoy Effect**.

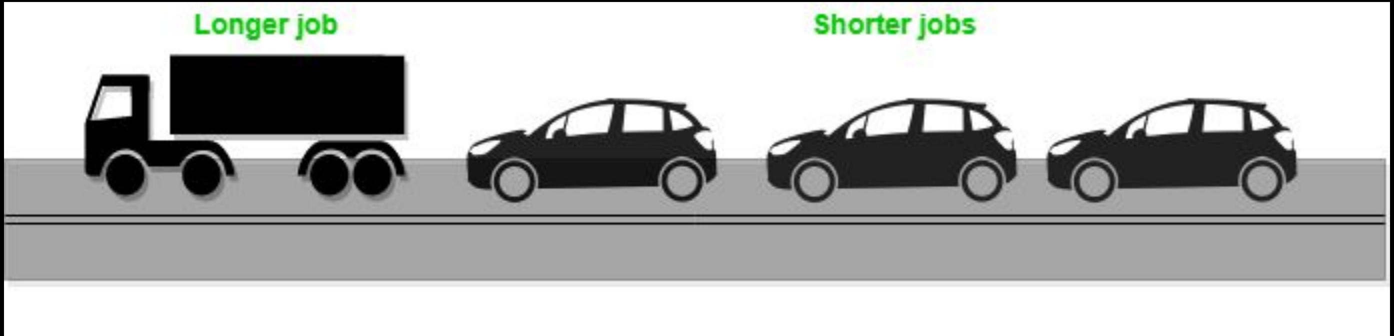


Figure - The Convey Effect, Visualized

A solution for the above problem

Use SJF instead of FCFS such that when there is a process with shorter burst time, a context switch occurs and the shorter process will run on the CPU first.