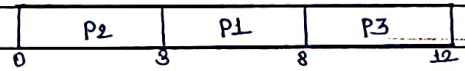
Experiment No. 3

Aim:	To study & implement the first come, first serve (FCFS) algorithm of CPU scheduling & it's full program code.

Aim:	To study & implement the first come first serve (FCFS) algorithm of CPU scheduling & it's full program code.
Theory:	First come, first serve (FCFS) is one of the simplest types of CPU scheduling algorithms. It is exactly what it sounds like processess are attended in the order which they arrive in the ready queue. FCFS CPU scheduling algo. is a non-preemptive algorithm, meaning once a process starts running it cannot be stopped until it voluntarly requires typically performing / waiting for I/O operations.
Example of FCFS:	Consider the following table consisting of processes P1, P2 & P3 along with their details :

Process	Burst Time (B_T)	Arrival Time (A_T)
P1	5 ms	2 ms
P2	3 ms	0 ms
P3	4 ms	4 ms

Gantt Chart :



Process	Completion Time (C_T)	Turn-around Time (A_T)	Waiting Time (W_T)
P2	3 ms	3 ms	0 ms
P1	8 ms	6 ms	1 ms
P3	12 ms	8 ms	4 ms

Average : TAT = 1.67

WAT = 5.67

Code :

```
using namespace std;
class process {
    private :
        int at;
        int bt;
        int ct;
        int tat;
        int wt;
        int pid;
    public :
        int & operator [] (string var) {
            if (var == "at")
                return at;
            if (var == "bt")
                return bt;
            if (var == "ct")
                return ct;
```

```
if (var == "tat")
        return tat;
if (var == "wt")
        return wt;

return pid;
}
void update_after_ct() {
        tat = ct - at;
        wt = tat - bt;
}
void display() {
    printf("%d \t %d \t %d \t %d \t %d \t %d \t",
            pid, at, bt, ct, tat, wt);
}
};
float average(vector<process> p, string var) {
    int total = 0;
    for (auto temp : p) {
            total += temp[var];
        }
        return (float) total / p.size();
}
```

```
int main () {
    int n;
    cin >> n;
    int counter = 0;
    vector<Process> P(n);
    for(Process& temp : P) {
        temp["id"] = counter++;
        cin >> temp["at"] >> temp["bt"];
    }
    sort(P.begin(), P.end(), [](Process first,
    Process second) {
        return first["at"] < second["at"];
    });
    printf("pid \t at \t bt \t ct \t tat \t wt \n");

    P[0]["ct"] = P[0]["at"] + P[0]["bt"];
    P[0].update_after_ct();
    P[0].display();

    for(int i = 1; i < P.size(); i++) {
        if(P[i]["at"] < P[i-1]["ct"]) {
            P[i]["ct"] = P[i-1]["ct"] +
                         P[i]["bt"];
    }
```

```
else {
      printf("avn [%d]: %d, per [%d]: %d
    \n",
          P[0]["at"], P[i-1]["ct"]);
P[i]["ct"] = P[0]["at"] + P[0]["bt"];
}
P[i].update_after_ct();
P[i].display();
}

printf("Average waiting time : %f \n"
              average(P, "wt"));
return 0;

}
```

Output:

Process	Bt	Wt	At
1	10	0	10
2	5	10	15
3	8	15	23

Average waiting time : 8.333
Average Turn around time = 16.

Conclusion: We have successfully studied & implemented the FCFS CPU scheduling algorithm.

Conclusion: We have successfully studied & implemented the FCFS CPU scheduling algorithm.