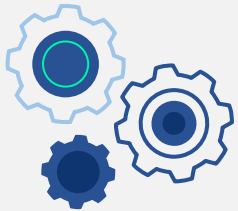
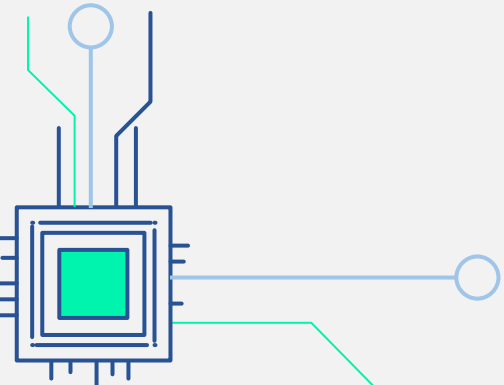


OS TASK 7 - SCHEDULING ALGORITHMS

Farah Kabesh - 900191706
Karim Sherif - 900191474



ROLES OF TEAM MEMBERS

Farah

- ❏ First Come First Served
- ❏ Round Robin
- ❏ Multilevel Feedback Queue
- ❏ Presentation

Karim

- ❏ Shortest Job First
- ❏ Shortest Remaining Time First
- ❏ Round Robin
- ❏ Presentation

PSEUDOCODE FOR NON-PREEMPTIVE SHORTEST JOB FIRST (SJF)

```
Vector <double> OC;


bool AT(const pcb& a, const pcb& b)

    returns a.arrivalTime less than b.arrivalTime


bool BT(const pcb& a, const pcb& b)

    returns a.burstTime less than b.burstTime
```

PSEUDOCODE FOR NON-PREEMPTIVE SHORTEST JOB FIRST (SJF)

```
void SJF(vector <pcb> processes) {

sort(processes.begin(), processes.end(), AT)

initialize j, CycleTime = 0

    for i = 0 to i less processes.size()

     j = i;

while j not equal to processes.size() AND processes[j].arrivalTime
equal to CycleTime)

    j++
```

PSEUDOCODE FOR NON-PREEMPTIVE SHORTEST JOB FIRST (SJF)

```
Initialize:

totalTurnaroundTime = 0, totalWaitingTime = 0, totalResponseTime = 0,
bsum = 0



sort(processes.begin() + i, processes.begin() + j, BT

processes[i].finishTime = CycleTime + processes[i].burstTime

CycleTime += processes[i].burstTime
```

PSEUDOCODE FOR NON-PREEMPTIVE SHORTEST JOB FIRST (SJF)

```
for int i = 0 to i less than processes.size()

processes[i].turnaroundTime = abs(processes[i].finishTime -
processes[i].arrivalTime)

processes[i].waitingTime = abs(processes[i].turnaroundTime -
processes[i].burstTime)

processes[i].responseTime = abs(bsum - processes[i].arrivalTime)

bsum += processes[i].burstTime
```

PSEUDOCODE FOR NON-PREEMPTIVE SHORTEST JOB FIRST (SJF)

```
totalTurnaroundTime += processes[i].turnaroundTime

totalWaitingTime += processes[i].waitingTime

totalResponseTime += processes[i].responseTime

Average Waiting Time= (float)totalWaitingTime / processes.size()

Average Turnaround Time= (float)totalTurnaroundTime /processes.size()

Average Response Time=(float)totalResponseTime / processes.size()
```

PSEUDOCODE FOR PREEMPTIVE SHORTEST JOB FIRST (SRTF)

```
Note: responseTime initialized to -1

Vector <double> OC

bool AT(const pcb& a, const pcb& b)

    returns a.arrivalTime less than b.arrivalTime


void SRTF(vector <pcb> processes)

sort(processes.begin(), processes.end(), AT)

vector <pcb> temp
```

PSEUDOCODE FOR PREEMPTIVE SHORTEST JOB FIRST (SRTF)

```
int cycleTime = 0

int sp = 0

bool process_available = false

int minTime = INT_MAX;

int size = processes.size()
```

PSEUDOCODE FOR PREEMPTIVE SHORTEST JOB FIRST (SRTF)

```
while temp.size() not equal to size

//check shortest time in accordance with the arrival time

for int i = 0 to i less than processes.size()

if processes[i].arrivalTime less than or equal to cycleTime AND
processes[i].remainingTime less than minTime

    minTime = processes[i].remainingTime

    sp = i

    process_available = true
```

PSEUDOCODE FOR PREEMPTIVE SHORTEST JOB FIRST (SRTF)

```
//if there is available process during this clock cycle
if process_available is equal to false
    cycleTime++
    continue
if processes[sp].responseTime equal to -1
    processes[sp].responseTime = cycleTime - processes[sp].arrivalTime
processes[sp].remainingTime--
minTime = processes[sp].remainingTime
```

PSEUDOCODE FOR PREEMPTIVE SHORTEST JOB FIRST (SRTF)

```
if processes[sp].remainingTime equal to 0
    processes[sp].finishTime = cycleTime + 1
    temp.push_back(processes[sp])
    vector<pcb>::iterator it = processes.begin() + sp
    processes.erase(it)
    minTime = INT_MAX
cycleTime++
processes = temp
```

PSEUDOCODE FOR PREEMPTIVE SHORTEST JOB FIRST (SRTF)

```
Initialize:

totalTurnaroundTime = 0, totalWaitingTime = 0, totalResponseTime = 0,
bsum = 0

for int i = 0 to i less than processes.size()

processes[i].turnaroundTime = abs(processes[i].finishTime -
processes[i].arrivalTime)

processes[i].waitingTime = abs(processes[i].turnaroundTime -
processes[i].burstTime)
```

PSEUDOCODE FOR PREEMPTIVE SHORTEST JOB FIRST (SRTF)

```
totalTurnaroundTime += processes[i].turnaroundTime

totalWaitingTime += processes[i].waitingTime

totalResponseTime += processes[i].responseTime

Average Waiting Time= (float)totalWaitingTime / processes.size()

Average Turnaround Time= (float)totalTurnaroundTime /processes.size()

Average Response Time=(float)totalResponseTime / processes.size()
```

PSEUDOCODE FOR FIRST COME FIRST SERVED (FCFS)

```
const int n=10

process p[n]

bool sortFcfs(process p, process p1)

    returns p.arrivalTime less than p1.arrivalTime

double totalTurnaround = 0

double totalWaiting = 0

double totalResponse = 0

sort(p, p+n, sortFcfs)
```

PSEUDOCODE FOR FIRST COME FIRST SERVED (FCFS)

```
p[0].waitingTime=0, p[0].responseTime=0 , p[0].finishTime = 0 +
p[0].burstTime

For i=1 to i less than n

p[i].waitingTime=0

p[i].responseTime=0

p[i].finishTime = p[i-1].finishTime + p[i].burstTime

    For count=1 to count<=i

    p[i].btSum+=p[i-count].burstTime
```

PSEUDOCODE FOR FIRST COME FIRST SERVED (FCFS)

```
 For int i=0 to i less than n

p[i].turnaroundTime = p[i].finishTime - p[i].arrivalTime

p[i].waitingTime = p[i].turnaroundTime - p[i].burstTime

p[i].responseTime += p[i].btSum - p[i].arrivalTime

totalWaiting += p[i].waitingTime

totalTurnaround += p[i].turnaroundTime

totalResponse += p[i].responseTime
```

PSEUDOCODE FOR FIRST COME FIRST SERVED (FCFS)

```
Average Waiting Time= totalWaitingTime /n

Average Turnaround Time= totalTurnaroundTime /n

Average Response Time=totalResponseTime / n
```

PSEUDOCODE FOR ROUND ROBIN (RR)

```
const int n=10

int timeQuantum = 5

process p[n]

bool sortFcfs(process p, process p1)

    returns p.arrivalTime less than p1.arrivalTime
```

PSEUDOCODE FOR ROUND ROBIN (RR)

```
void roundRobin()

double totalTurnaround = 0

double totalWaiting = 0

double totalResponse = 0

int cycleTime = 0, int done = 0

sort(p, p+n, sortFcfs

p[0].waitingTime=0

p[0].responseTime=0
```

PSEUDOCODE FOR ROUND ROBIN (RR)

```
For i=0 to i<n

p[i].remTime = p[i].burstTime

If p[i].remTime greater than 0

If p[i].remTime <= timeQuantum

    done ++;

    cycleTime += p[i].remTime;

    p[i].finishTime = cycleTime;
```

PSEUDOCODE FOR ROUND ROBIN (RR)

```
For int j=1 to j<=i

    p[i].btSum+=p[i-j].remTime

 p[i].turnaroundTime = p[i].finishTime - p[i].arrivalTime

p[i].waitingTime = p[i].turnaroundTime - p[i].burstTime

p[i].responseTime = p[i].btSum - p[i].arrivalTime

 totalWaiting += p[i].waitingTime, totalTurnaround += p[i].turnaroundTime

totalResponse +=p[i].responseTime

p[i].remTime = 0
```

PSEUDOCODE FOR ROUND ROBIN (RR)

```
else if p[i].remTime > timeQuantum

cycleTime += timeQuantum

p[i].remTime = p[i].remTime - timeQuantum

Average Waiting Time= totalWaitingTime /n

Average Turnaround Time= totalTurnaroundTime /n

Average Response Time=totalResponseTime / n
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
Const int n=10

Process Q1[n], Q2[n]

bool sortFcfs(process p, process p1)

    returns p.arrivalTime less than p1.arrivalTime


bool BT(process p, process p1)

    returns p.burstTime less than p1.burstTime
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
Initialize:

double totalTurnaround = 0, double totalWaiting = 0

double totalResponse = 0, double totalTurnaround1 = 0

double totalWaiting1 = 0, double totalResponse1 = 0

double totalTurnaround2 = 0, double totalWaiting2 = 0

double totalResponse2 = 0

int j = 0

int i
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
sort(Q1, Q1 + n, BT);

Q1[0].waitingTime=0

Q1[0].responseTime=0

Q1[0].finishTime = 0 + Q1[0].burstTime

For i=1 to i<n; i++)

Q1[i].waitingTime=0

Q1[i].responseTime=0

Q1[i].finishTime = Q1[i-1].finishTime + Q1[i].burstTime
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
For int count=1 to count<=i

    Q1[i].btSum+=Q1[i-count].burstTime

 int count = 0, int start

For i=0 to i<n

count++

    if(count <=5)

    Q1[i].turnaroundTime = Q1[i].finishTime - Q1[i].arrivalTime

    Q1[i].waitingTime = Q1[i].turnaroundTime - Q1[i].burstTime
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
Q1[i].responseTime += Q1[i].btSum - Q1[i].arrivalTime

totalWaiting1 += Q1[i].waitingTime

totalTurnaround1 += Q1[i].turnaroundTime

totalResponse1 += Q1[i].responseTime

 if(count > 5)

start = Q1[4].finishTime, Q2[j].burstTime = Q1[i].burstTime

Q2[j].pid = Q1[i].pid, Q2[j].arrivalTime = Q1[i].arrivalTime

j++
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
sort(Q2, Q2+j, sortFcfs)

Q2[0].finishTime = start + Q2[0].burstTime

for(i=1; i<j; i++)

Q2[i].waitingTime=0

Q2[i].responseTime=0

Q2[i].finishTime = Q2[i-1].finishTime + Q2[i].burstTime

 For int count=1 to count<=i)

            Q2[i].btSum+=Q2[i-count].burstTime
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
For int i=0 to i<j

Q2[i].turnaroundTime = Q2[i].finishTime - Q2[i].arrivalTime

Q2[i].waitingTime = Q2[i].turnaroundTime - Q2[i].burstTime

Q2[i].responseTime = Q2[i].btSum - Q2[i].arrivalTime

totalWaiting2 += Q2[i].waitingTime

totalTurnaround2 += Q2[i].turnaroundTime

totalResponse2 += Q2[i].responseTime
```

PSEUDOCODE FOR MULTILEVEL FEEDBACK QUEUE (MLFQ)

```
totalWaiting = totalWaiting1+totalWaiting2

totalTurnaround = totalTurnaround1 + totalTurnaround2

totalResponse = totalResponse1 + totalResponse2


Average Waiting Time= totalWaitingTime /n

Average Turnaround Time= totalTurnaroundTime /n

Average Response Time=totalResponseTime / n
```

PROCESS ATTRIBUTES CHOSEN & WHY

- **PID**: ID of each process.
- **Arrival Time**: arrival time of each process.
- **Burst Time**: burst time of each process.
- **Finish Time**: completion time of each process.
- **btSum**: sum of previous burst times before current process. Used in calculating finish time and response time.

PROCESS ATTRIBUTES CHOSEN & WHY

- **Turnaround Time**: turnaround time for each process. Equal to finish time - arrival time.
- **Waiting Time**: waiting time for each process. Equal to turnaround time - burst time.
- **Response Time**: difference between arrival time of process and when it first got the CPU. Equal to sum of previous burst times - arrival time.
- **Remaining Time**: used in RR and SRTF to calculate remaining burst time of each process.

HOW ALGORITHMS WERE TESTED?

- Changing n (number of processes) each time.
- Range from 10 till 200 with increments of 10.
- Randomly Generated Burst Times Range: 1 to 10.
- Randomly Generated Arrival Times Range: 0 to 10.

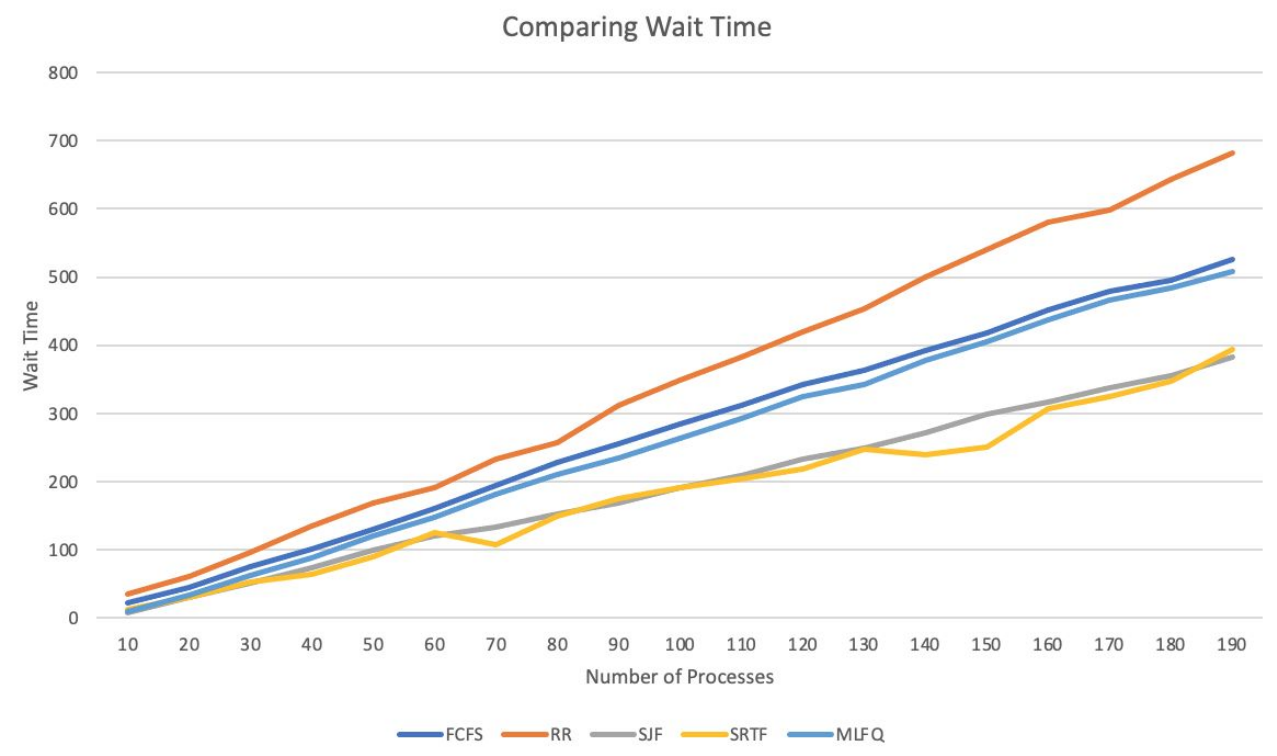
```
int main()
{
    for (int i=0; i<n; i++)
    {
        p[i].pid = i;

        p[i].burstTime = 1 + (rand() % 10);

        p[i].arrivalTime = 0 + (rand() % 10);
    }
    fcfs();
    return 0;
}
```

MAIN RESULTS

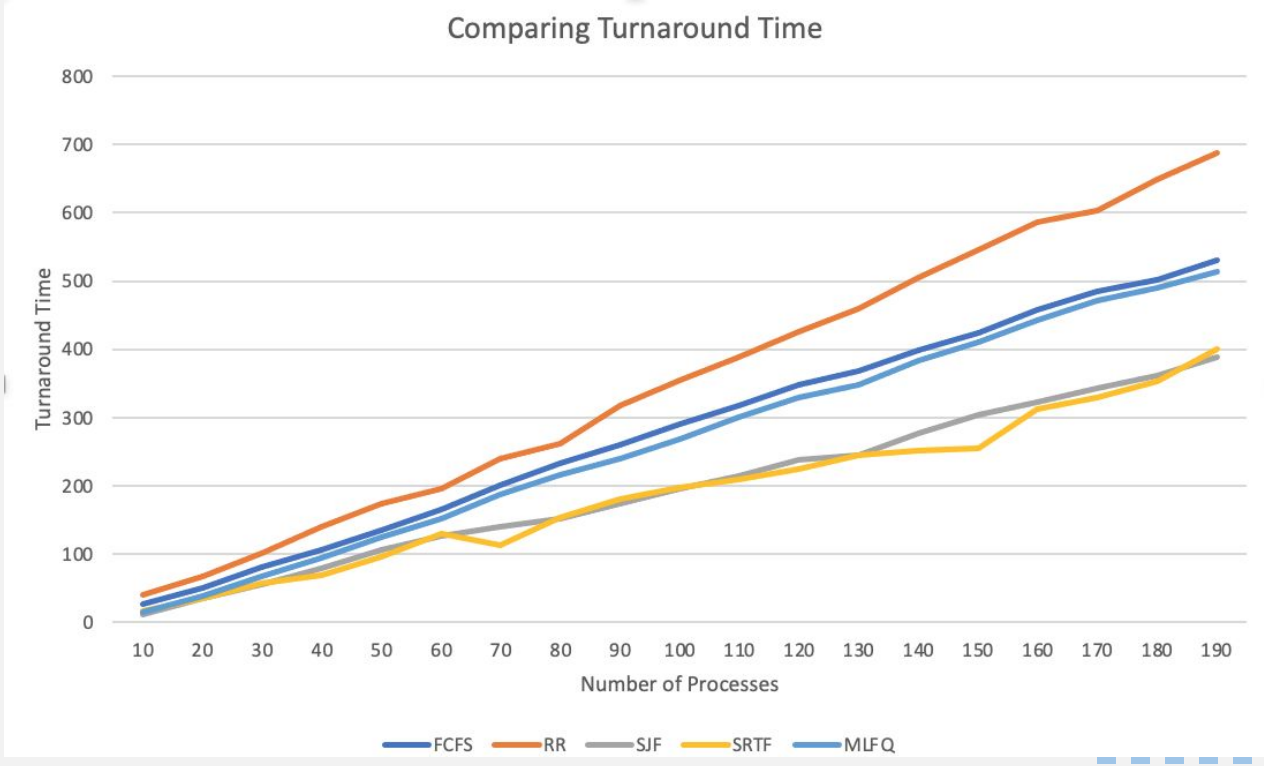
WAITING TIME



CONCLUSION - WAIT TIME

- SJF and SRTF have better waiting times than rest of algorithms (Winners).
- RR has the longest waiting time (Loser).

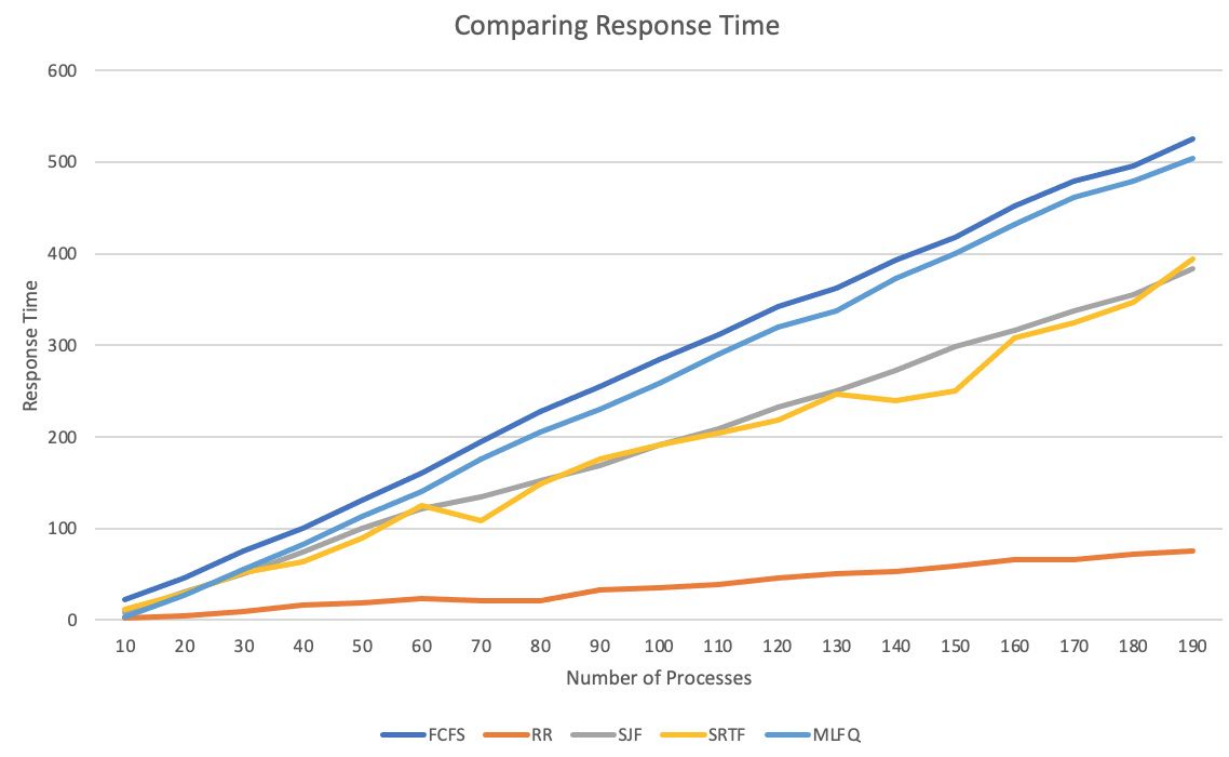
TURNAROUND TIME



CONCLUSION - TURNAROUND TIME

- SJF and SRTF have better turnaround times than rest of algorithms (Winners).
- RR has the longest turnaround time (Loser).

RESPONSE TIME



CONCLUSION - TURNAROUND TIME

- RR has best response times than rest of algorithms (Winner).
- FCFS has the longest response time (Loser).