OS Phase1&2 Report

Submitted to:

Dr. Ayman Aboelhassan

TA. Muhammad Alaa

TA. Muhammad Hesham

TA. Ali Haytham

TA. Ahmed Mostafa

Submitted by:

Moaaz Tarek	1200871
Salah Mohamed Salah	1200806
Omar Sherif Elzahar	1200476
Hussein Mostafa Elhawary	1200799

Contents

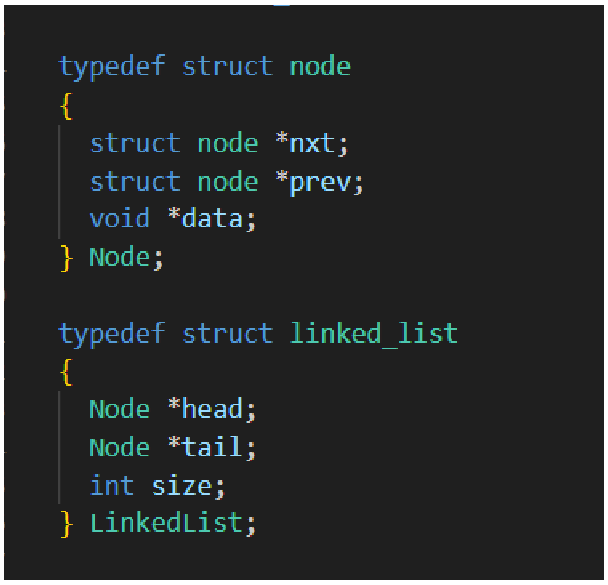
Data Structures

Note: Some code snippets are included for illustration

Linked List:

Dynamic List to be used in RR as a circular queue.

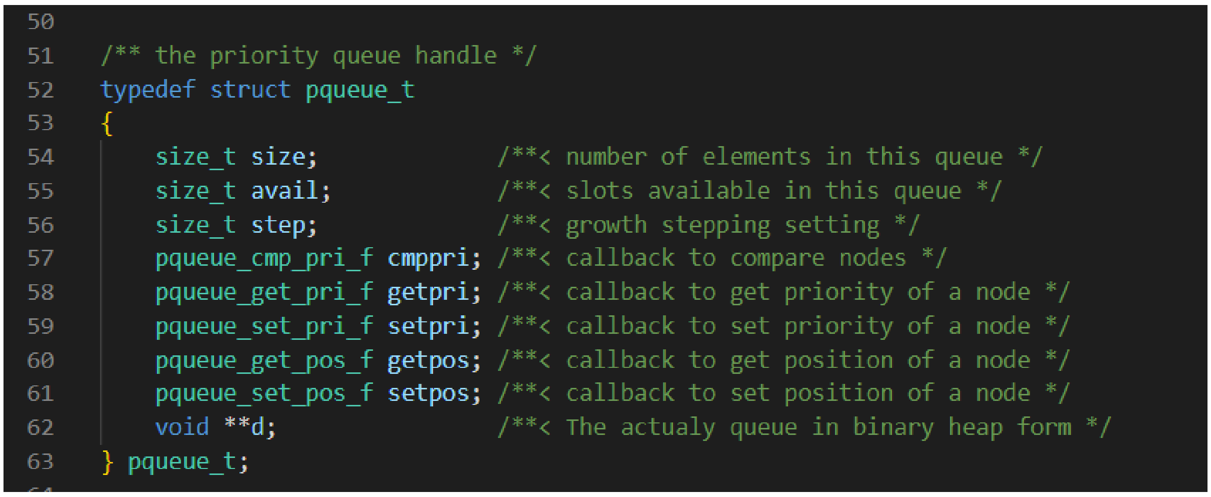
Node in List will save pointer to next and previous nodes. It will also contain PCB pointer which is the process details.



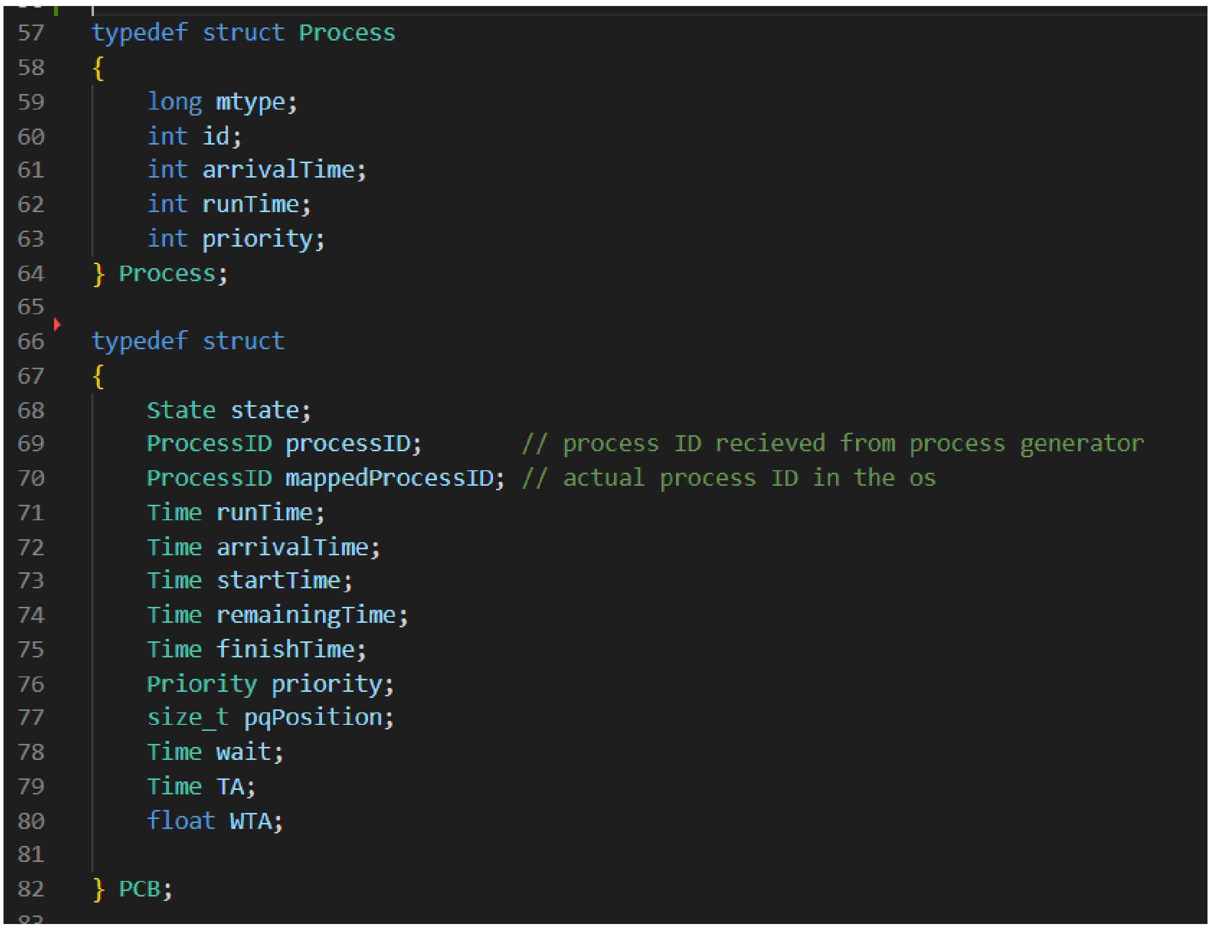
```
typedef struct node
{
    struct node *nxt;
    struct node *prev;
    void *data;
} Node;

typedef struct linked_list
{
    Node *head;
    Node *tail;
    int size;
} LinkedList;
```

Priority Queue:

Used to sort in HPF by priority, and in STRN by remaining time.



```c

/** the priority queue handle */
typedef struct pqueue_t
{
    size_t size;             /**< number of elements in this queue */
    size_t avail;            /**< slots available in this queue */
    size_t step;             /**< growth stepping setting */
    pqueue_cmp_pri_f cmppri; /**< callback to compare nodes */
    pqueue_get_pri_f getpri; /**< callback to get priority of a node */
    pqueue_set_pri_f setpri; /**< callback to set priority of a node */
    pqueue_get_pos_f getpos; /**< callback to get position of a node */
    pqueue_set_pos_f setpos; /**< callback to set position of a node */
    void **d;                /**< The actualy queue in binary heap form */
} pqueue_t;
```

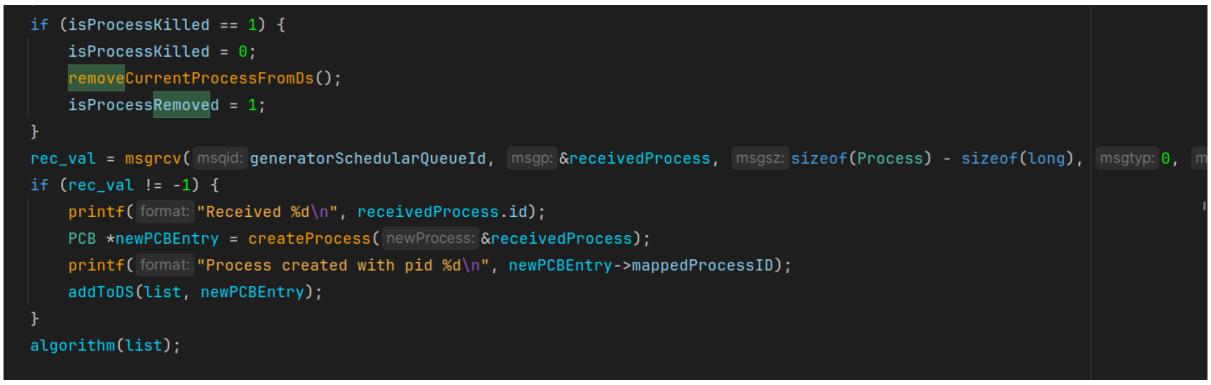

```c
typedef struct Process
{
    long mtype;
    int id;
    int arrivalTime;
    int runTime;
    int priority;
} Process;

typedef struct
{
    State state;
    ProcessID processID;       // process ID recieved from process generator
    ProcessID mappedProcessID; // actual process ID in the os
    Time runTime;
    Time arrivalTime;
    Time startTime;
    Time remainingTime;
    Time finishTime;
    Priority priority;
    size_t pqPosition;
    Time wait;
    Time TA;
    float WTA;

} PCB;
```

This is a process struct that I save data in it from given input file (process.txt)

While PCB struct I save in it all data related to each process when process generator send process to scheduler then any update during running will be modified in PCB struct

Algorithms

All algorithms have near identical skeleton in receiving, deleting and handle the algorithm

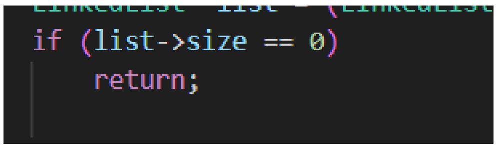


```
if (isProcessKilled == 1) {
    isProcessKilled = 0;
    removeCurrentProcessFromDs();
    isProcessRemoved = 1;
}
rec_val = msgrcv( msqid: generatorSchedularQueueId, msgp: &receivedProcess, msgsz: sizeof(Process) - sizeof(long), msgtyp: 0, m
if (rec_val != -1) {
    printf( format: "Received %d\n", receivedProcess.id);
    PCB *newPCBEntry = createProcess( newProcess: &receivedProcess);
    printf( format: "Process created with pid %d\n", newPCBEntry->mappedProcessID);
    addToDS(list, newPCBEntry);
}
algorithm(list);
```

Algorithm, addToDS are pointer to functions passed to this function to handle different data structures in RR ,HPF and SRTN.

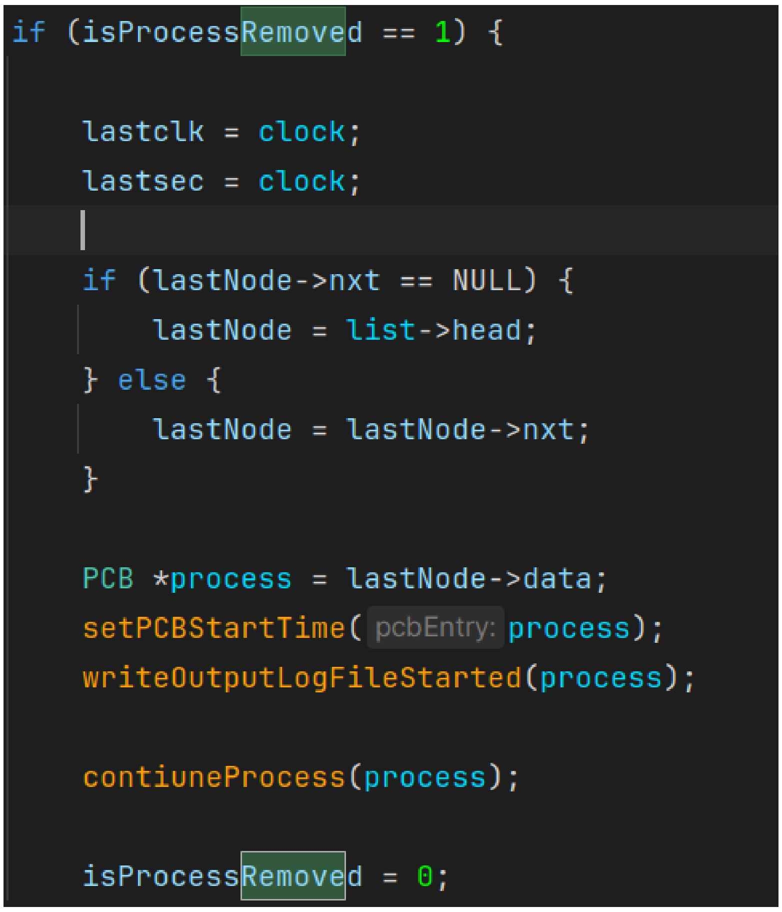
Round Robin

Start by checking if the list is empty, if it is then return from handling RR.



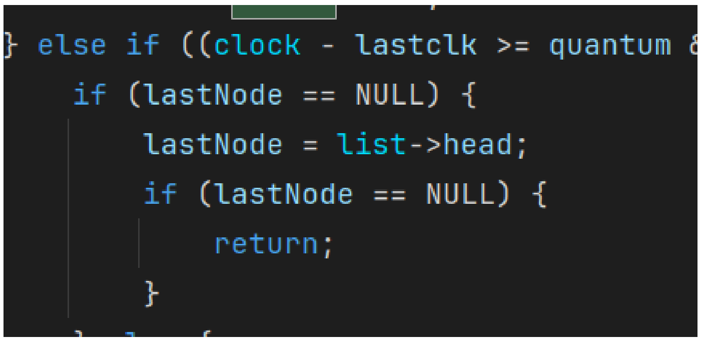
```
if (list->size == 0)
    return;
```

Then check if there was a PCB that finished and removed form LL if yes goes to next pointer and start the process



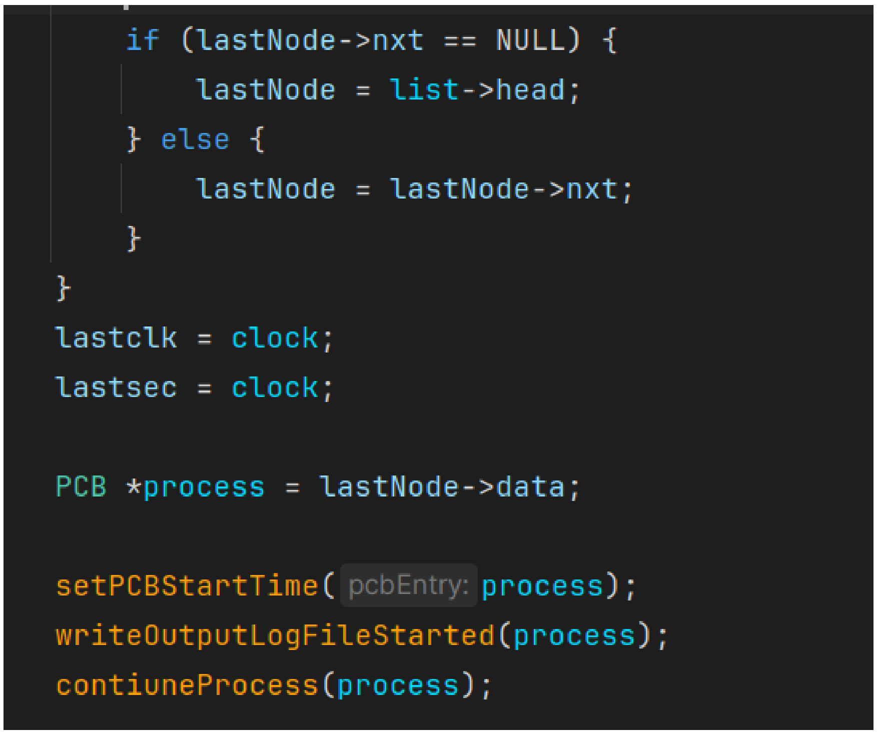
```
if (isProcessRemoved == 1) {

    lastclk = clock;
    lastsec = clock;

    if (lastNode->nxt == NULL) {
        lastNode = list->head;
    } else {
        lastNode = lastNode->nxt;
    }

    PCB *process = lastNode->data;
    setPCBStartTime( pcbEntry: process);
    writeOutputLogFileStarted(process);

    contiuneProcess(process);

    isProcessRemoved = 0;
```

Else it checks if it is time for quantum and sets node if null



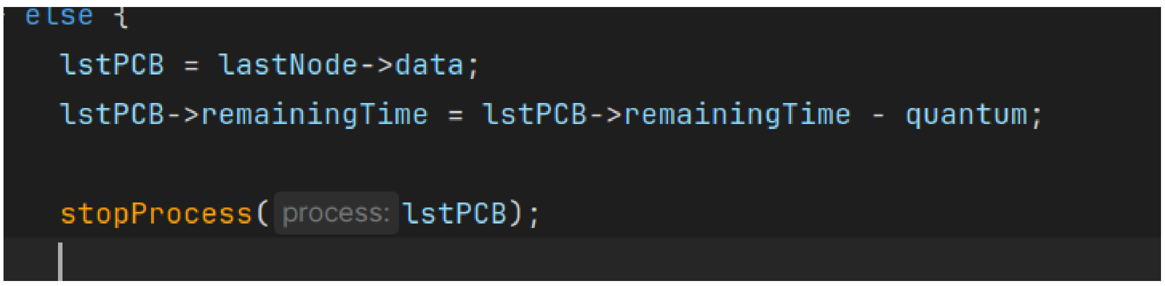
```
} else if ((clock - lastclk >= quantum &
    if (lastNode == NULL) {
        lastNode = list->head;
        if (lastNode == NULL) {
            return;
        }
```

or iterate circulary and start the process.



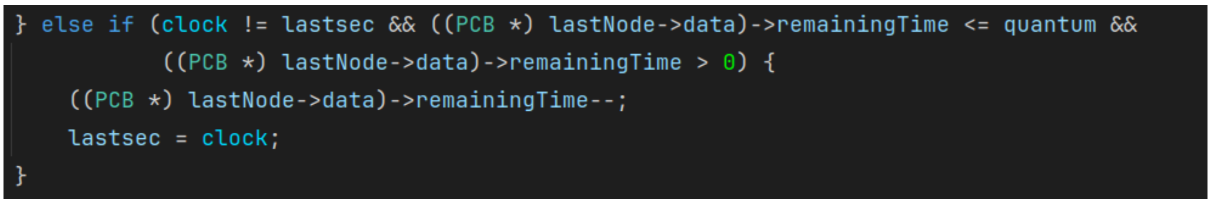
```
        if (lastNode->nxt == NULL) {
            lastNode = list->head;
        } else {
            lastNode = lastNode->nxt;
        }
    }
    lastclk = clock;
    lastsec = clock;

    PCB *process = lastNode->data;

    setPCBStartTime( pcbEntry: process);
    writeOutputLogFileStarted(process);
    contiuneProcess(process);
```

It also ends the process if not finished at quantum time and sets remaining time to synchronize.



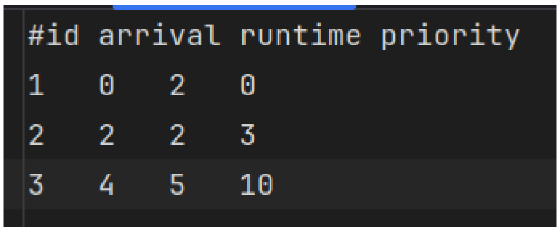
```
 else {
    lstPCB = lastNode->data;
    lstPCB->remainingTime = lstPCB->remainingTime - quantum;

    stopProcess( process: lstPCB);
```

Then if remaining time is less than the quantum it will decrease remaining time each second to synchronize between process and scheduler. Because a process can finish before a quantum time has passed.



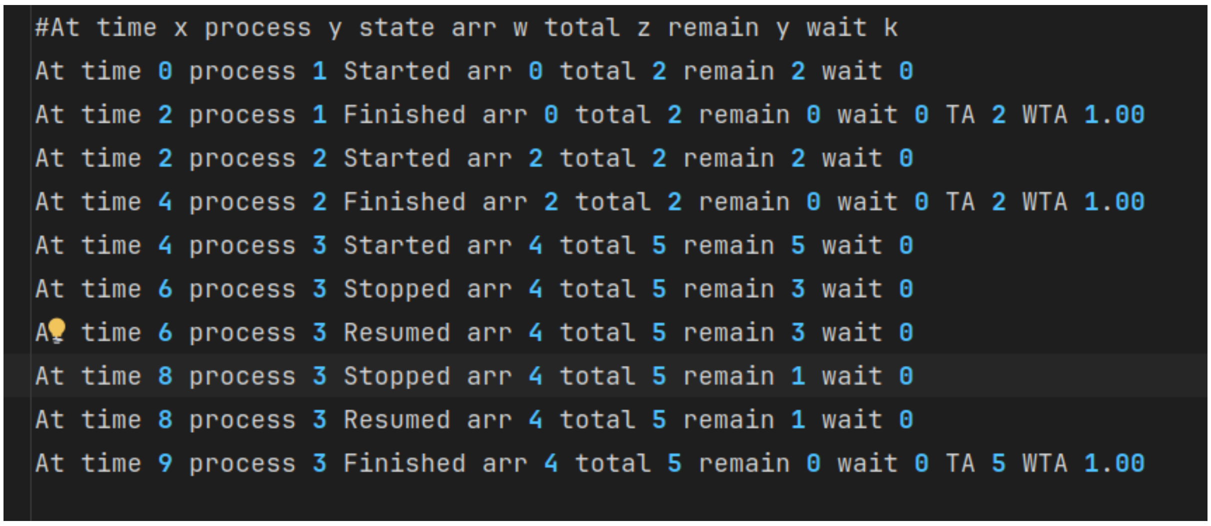
```
} else if (clock != lastsec && ((PCB *) lastNode->data)->remainingTime <= quantum &&
           ((PCB *) lastNode->data)->remainingTime > 0) {
    ((PCB *) lastNode->data)->remainingTime--;
    lastsec = clock;
}
```

Testcase:

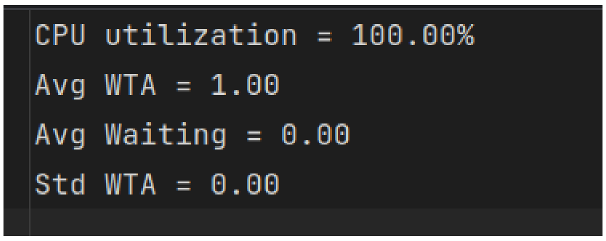


```
#id arrival runtime priority
1   0   2   0
2   2   2   3
3   4   5   10
```

Output:



```
#At time x process y state arr w total z remain y wait k
At time 0 process 1 Started arr 0 total 2 remain 2 wait 0
At time 2 process 1 Finished arr 0 total 2 remain 0 wait 0 TA 2 WTA 1.00
At time 2 process 2 Started arr 2 total 2 remain 2 wait 0
At time 4 process 2 Finished arr 2 total 2 remain 0 wait 0 TA 2 WTA 1.00
At time 4 process 3 Started arr 4 total 5 remain 5 wait 0
At time 6 process 3 Stopped arr 4 total 5 remain 3 wait 0
At time 6 process 3 Resumed arr 4 total 5 remain 3 wait 0
At time 8 process 3 Stopped arr 4 total 5 remain 1 wait 0
At time 8 process 3 Resumed arr 4 total 5 remain 1 wait 0
At time 9 process 3 Finished arr 4 total 5 remain 0 wait 0 TA 5 WTA 1.00
```



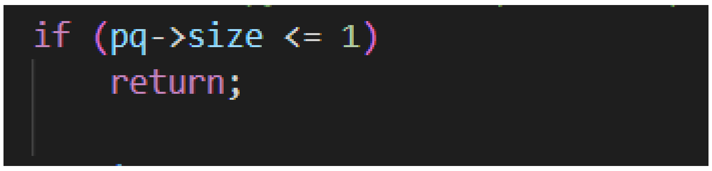
```
CPU utilization = 100.00%
Avg WTA = 1.00
Avg Waiting = 0.00
Std WTA = 0.00
```

Assumption:

when process is received at quantum time it should start if last running was the previous process.

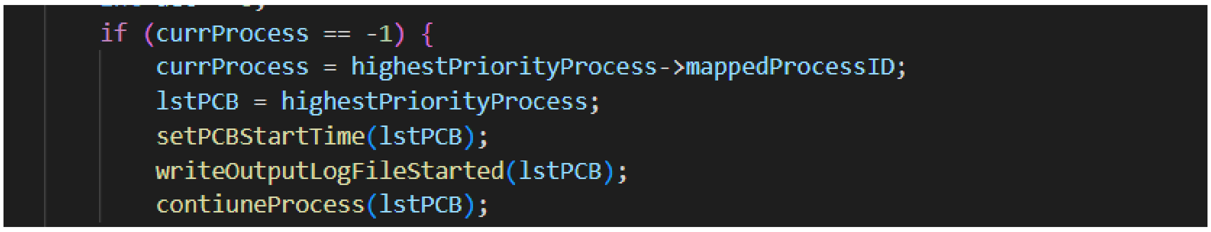
Shortest Remaining Time Next

Start by checking if the Priority Queue is empty, if it is then return from handling SRTN.



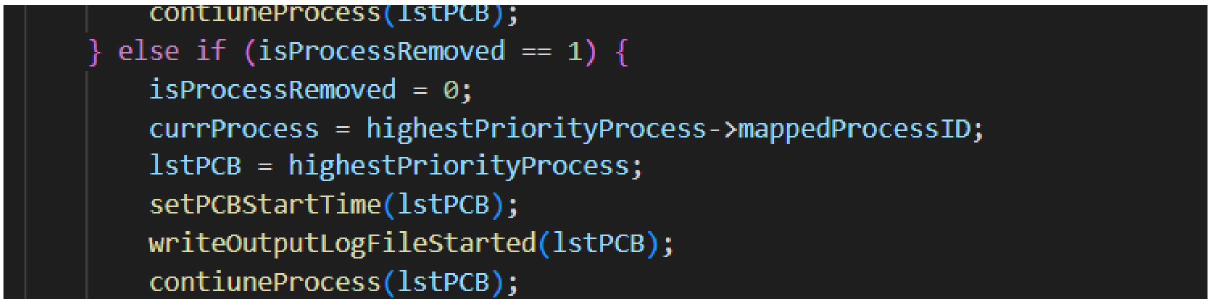
```
if (pq->size <= 1)
    return;
```

If no process is running and a new process arrives start it and process its data



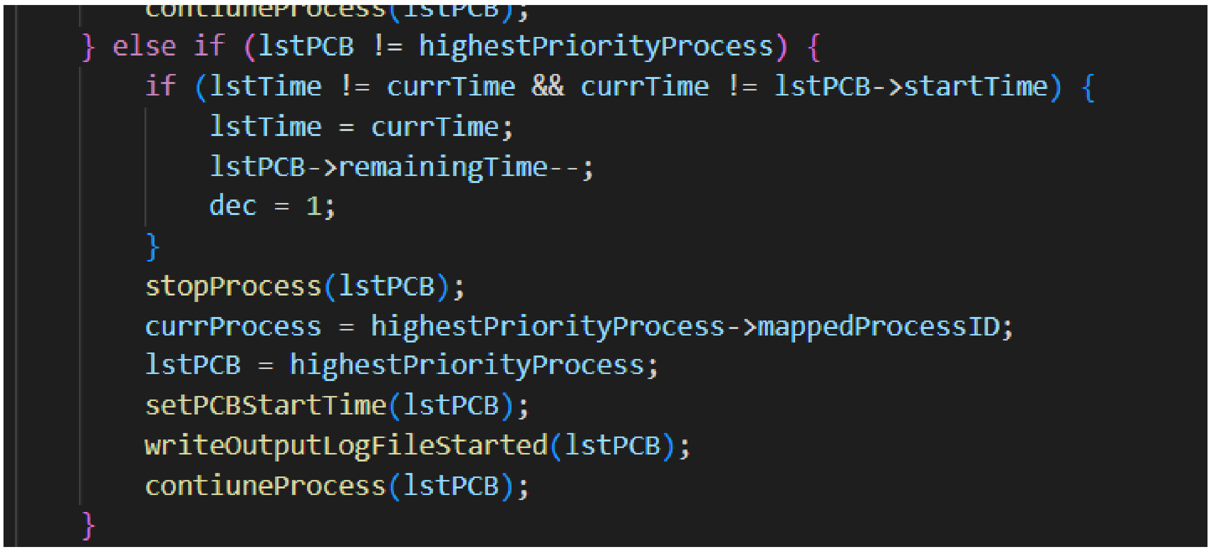
```
    if (currProcess == -1) {
        currProcess = highestPriorityProcess->mappedProcessID;
        lstPCB = highestPriorityProcess;
        setPCBStartTime(lstPCB);
        writeOutputLogFileStarted(lstPCB);
        contiuneProcess(lstPCB);
```

If a process has finished start a new process with the shortest remaining time



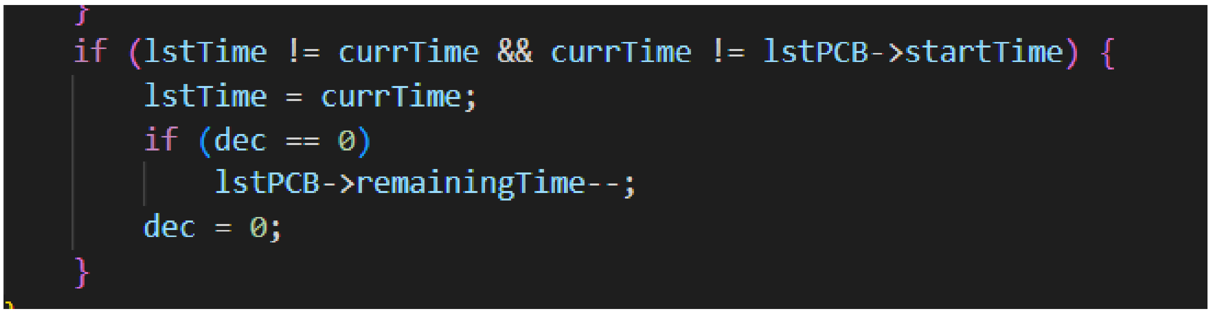
```
        contiuneProcess(lstPCB);
    } else if (isProcessRemoved == 1) {
        isProcessRemoved = 0;
        currProcess = highestPriorityProcess->mappedProcessID;
        lstPCB = highestPriorityProcess;
        setPCBStartTime(lstPCB);
        writeOutputLogFileStarted(lstPCB);
        contiuneProcess(lstPCB);
```

If a new process arrives that has a shorter remaining time than all other processes stop the current process and start the one that just arrived



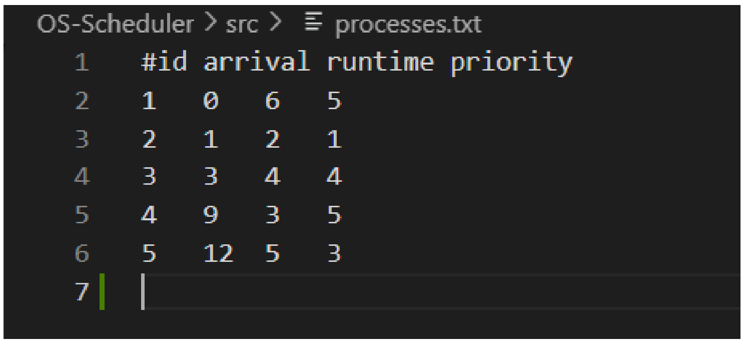
```
        contiuneProcess(lstPCB);
    } else if (lstPCB != highestPriorityProcess) {
        if (lstTime != currTime && currTime != lstPCB->startTime) {
            lstTime = currTime;
            lstPCB->remainingTime--;
            dec = 1;
        }
        stopProcess(lstPCB);
        currProcess = highestPriorityProcess->mappedProcessID;
        lstPCB = highestPriorityProcess;
        setPCBStartTime(lstPCB);
        writeOutputLogFileStarted(lstPCB);
        contiuneProcess(lstPCB);
    }
```

To keep track of the remaining time the scheduler for the Priority Queue to work correctly



```
    }
    if (lstTime != currTime && currTime != lstPCB->startTime) {
        lstTime = currTime;
        if (dec == 0)
            lstPCB->remainingTime--;
        dec = 0;
    }
```

Testcase:

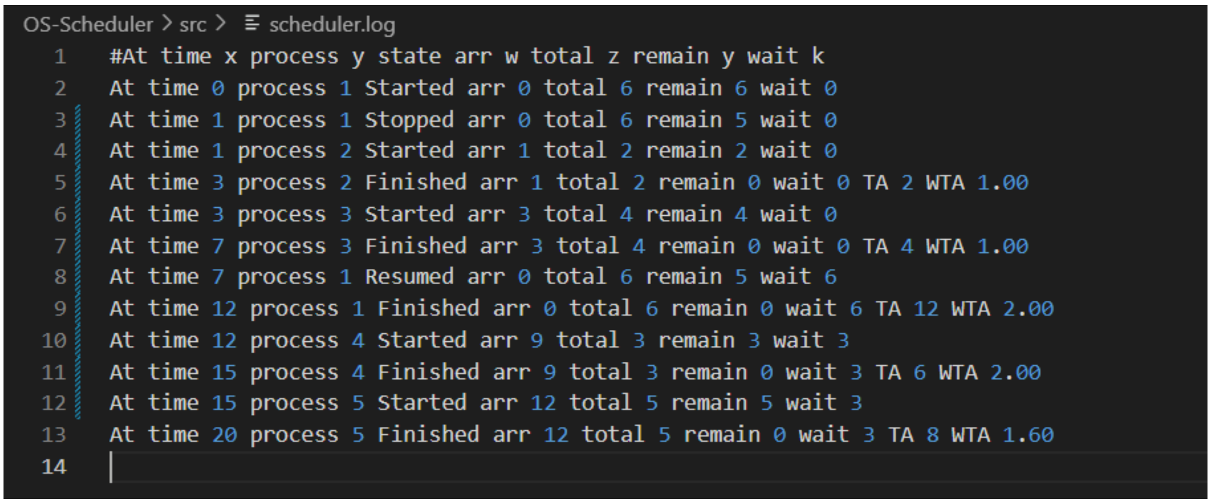


OS-Scheduler > src > processes.txt

```
#id arrival runtime priority
1   0   6   5
2   1   2   1
3   3   4   4
4   9   3   5
5   12  5   3

```

Output:

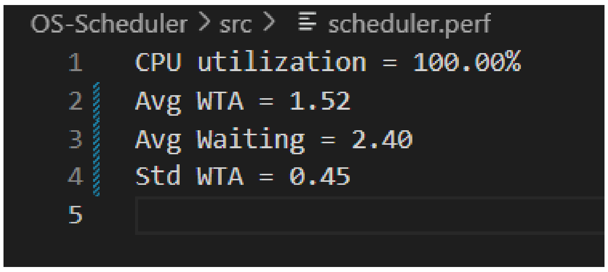
scheduler.log file



OS-Scheduler > src > scheduler.log

```
#At time x process y state arr w total z remain y wait k
At time 0 process 1 Started arr 0 total 6 remain 6 wait 0
At time 1 process 1 Stopped arr 0 total 6 remain 5 wait 0
At time 1 process 2 Started arr 1 total 2 remain 2 wait 0
At time 3 process 2 Finished arr 1 total 2 remain 0 wait 0 TA 2 WTA 1.00
At time 3 process 3 Started arr 3 total 4 remain 4 wait 0
At time 7 process 3 Finished arr 3 total 4 remain 0 wait 0 TA 4 WTA 1.00
At time 7 process 1 Resumed arr 0 total 6 remain 5 wait 6
At time 12 process 1 Finished arr 0 total 6 remain 0 wait 6 TA 12 WTA 2.00
At time 12 process 4 Started arr 9 total 3 remain 3 wait 3
At time 15 process 4 Finished arr 9 total 3 remain 0 wait 3 TA 6 WTA 2.00
At time 15 process 5 Started arr 12 total 5 remain 5 wait 3
At time 20 process 5 Finished arr 12 total 5 remain 0 wait 3 TA 8 WTA 1.60

```

Scheduler.perf

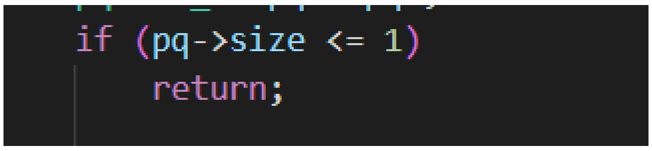


OS-Scheduler > src > scheduler.perf

```
CPU utilization = 100.00%
Avg WTA = 1.52
Avg Waiting = 2.40
Std WTA = 0.45

```

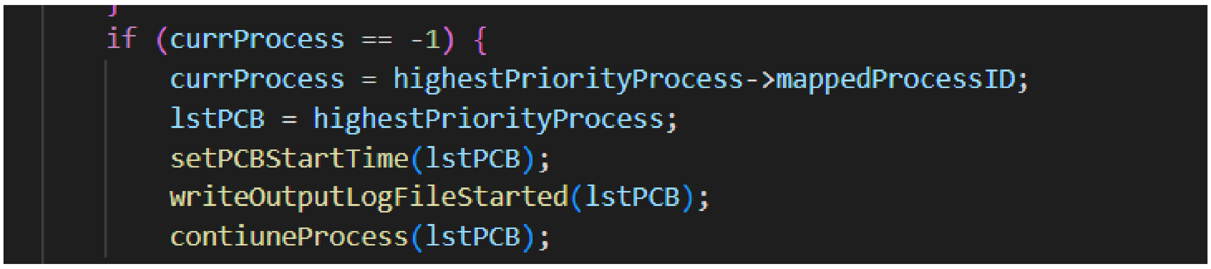
HPF

Start by checking if the Priority Queue is empty, if it is then return from handling HPF.



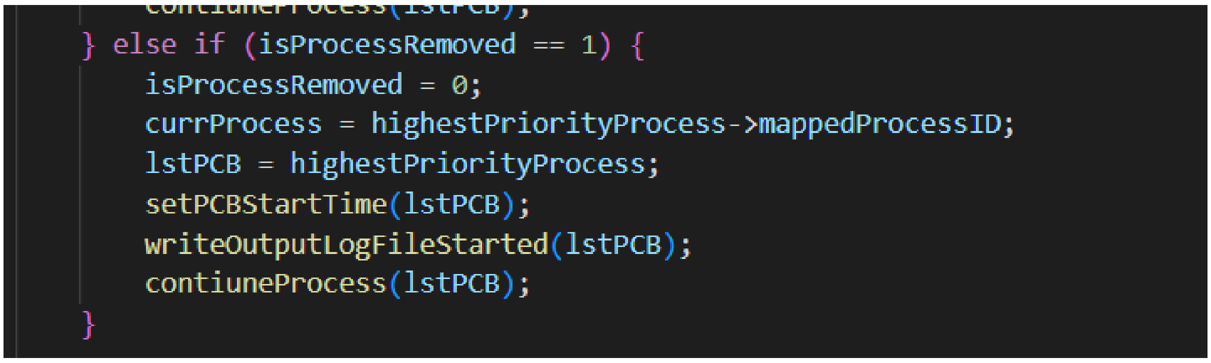
```
    if (pq->size <= 1)
        return;
```

If no process is running and a new process arrives start it and process its data



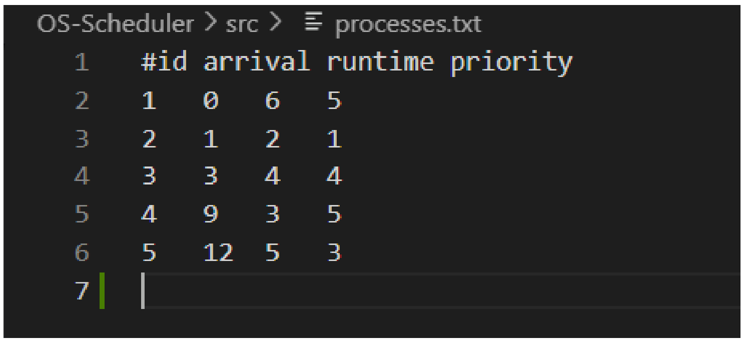
```
        if (currProcess == -1) {
            currProcess = highestPriorityProcess->mappedProcessID;
            lstPCB = highestPriorityProcess;
            setPCBStartTime(lstPCB);
            writeOutputLogFileStarted(lstPCB);
            contiuneProcess(lstPCB);
```

If a process has finished start a new process with the highest priority



```
    } else if (isProcessRemoved == 1) {
        isProcessRemoved = 0;
        currProcess = highestPriorityProcess->mappedProcessID;
        lstPCB = highestPriorityProcess;
        setPCBStartTime(lstPCB);
        writeOutputLogFileStarted(lstPCB);
        contiuneProcess(lstPCB);
    }
```

Testcase:

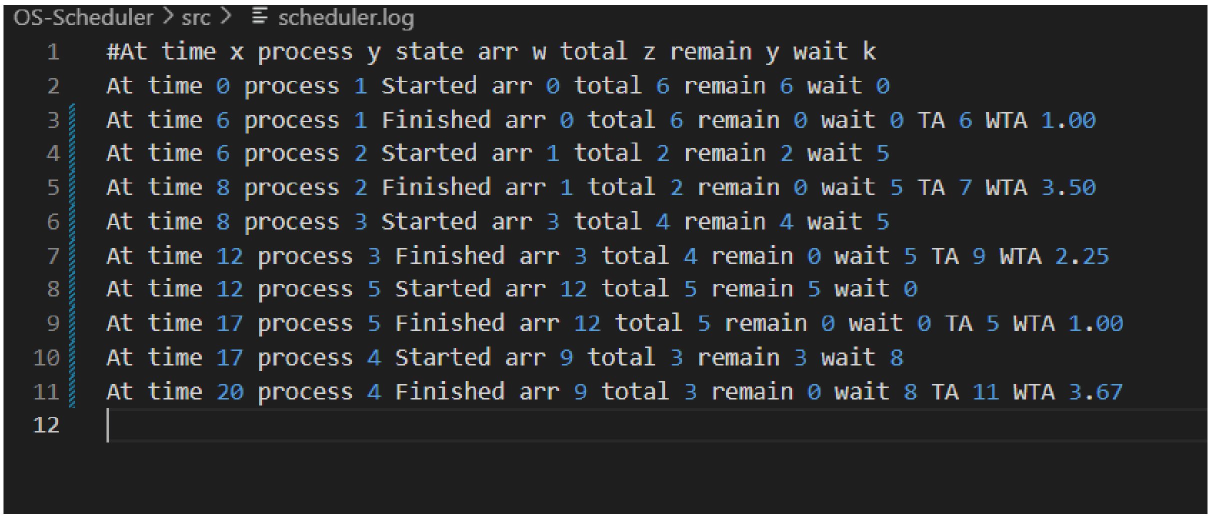


OS-Scheduler > src > processes.txt

```
#id arrival runtime priority
1   0   6   5
2   1   2   1
3   3   4   4
4   9   3   5
5   12  5   3
```

Output:

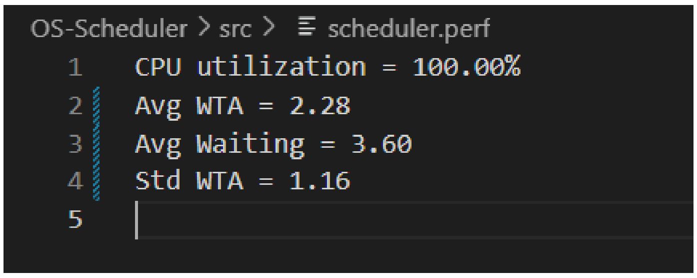
scheduler.log file



OS-Scheduler > src > scheduler.log

```
#At time x process y state arr w total z remain y wait k
At time 0 process 1 Started arr 0 total 6 remain 6 wait 0
At time 6 process 1 Finished arr 0 total 6 remain 0 wait 0 TA 6 WTA 1.00
At time 6 process 2 Started arr 1 total 2 remain 2 wait 5
At time 8 process 2 Finished arr 1 total 2 remain 0 wait 5 TA 7 WTA 3.50
At time 8 process 3 Started arr 3 total 4 remain 4 wait 5
At time 12 process 3 Finished arr 3 total 4 remain 0 wait 5 TA 9 WTA 2.25
At time 12 process 5 Started arr 12 total 5 remain 5 wait 0
At time 17 process 5 Finished arr 12 total 5 remain 0 wait 0 TA 5 WTA 1.00
At time 17 process 4 Started arr 9 total 3 remain 3 wait 8
At time 20 process 4 Finished arr 9 total 3 remain 0 wait 8 TA 11 WTA 3.67
```

Scheduler.perf



OS-Scheduler > src > scheduler.perf

```
CPU utilization = 100.00%
Avg WTA = 2.28
Avg Waiting = 3.60
Std WTA = 1.16
```

Assumptions

No process should have runtime equal to Zero.

In SRTN if process 2 was running and @ time 11 its remaining time was 4 and process 5 came with a runtime 4 process 2 will remain running then process 5 will run after process 2 finishes.

In HPF if two processes came at same time with same priority it will run according to ascending order of #id.

In RR when process is received at quantum time it should start if last running was the previous process.

Memory Management

Assumptions

When a process arrives the code checks if there is sufficient space for it or not if no sufficient space for all algorithms the process will be inserted in a queue (First Come First Serve).

Algorithm

Memory will be represented by linked list whose head is the first block in memory

In allocation, searches for first block with smallest available size then try to insert. If a split can occur, then calls the split function and and recurse again to check split and so on.

In deallocation, sets the the block as merge then tries to merge. Merge function is a binary search on block by using start and end pointer, then when it finds it, it checks if last operation changed left or right pointer and this will tell if it is a left or right child to merge with correct buddy.

The waiting list queue is checked whenever a process is deallocated from memory

Work Loads:

Moaaz Tarek	RR	Process Generator	20% Skeleton
Salah Mohamed	HPF	80% Skeleton	Process.c
Omar Sherif	SRTN	OuputFiles	
Hussein Mostafa	SRTN, HPF	headers	

HPF	3 hours + 10 debugging
SRTN	40 min + 12 hours debugging
RR	1 hour + 8 debugging
Skeleton	5 hours
Process Generator	1~2 hours
Process	30 mins + 2 hours debugging
Headers	1 hour
OutputFiles	2~3 hours + 4 debugging