Operating System Assignment

Bermal ARATOĞLU

B201202039

Project topic: Process scheduling with memory allocation simulation in C

In this project, first several data structures are created so that the program can better simulate process scheduling.

Data Structures ⇒

Process: Process struct is for simulating the CPU process as it can be understood by the name. A process' all relevant information is saved like arrival time, priority, memory requirements etc. Process can be identified by the name attribute.

MemoryBlock: MemoryBlock struct is for better simulating memory blocks in a computer 's memory so that memory allocation and de-allocation can be easier to manage. It holds process' name that is going to be stored, start address which is the first address that the process is going to be stored and size which is the memory size that is needed for the process.

CpuQueue: CpuQueue is a struct that holds the line of process are going to be simulated. So that the algorithms can be easily applied.

With making use of these structures, a couple of algorithms in accordance to processes' priorities.

Scheduling Algorithms ⇒

First Come First Serve (FCFS): First come first serve is an algorithm with a basic mentality behind. Process are scheduled in a way where the first process that is to arrive, is to be processed earlier. While this algorithm is easy-to-understand and processes each process, it also weak in performance and has longer waiting times.

With process with priorities that are equal to "0" are queued into a CpuQueue named CPU-1 queue.

Shortest Job First (SJF): Shortest job first algorithm does also have a basic mentality where the burst times are compared. This algorithms minimizes the waiting time but also in scenarios where new processes with shorter burst times arrive continuously process with longer burst time will be starved.

This algorithms is used with process' priorities equal to "1" being put in a CpuQueue name CPU-2 queue_high.

Round Robin (RR): Round robin is a preemptive scheduling algorithm where each process is assigned a quantum, fixed time slice. After this time slice is over, process goes back to the queue. This algorithm gives a chance to each process to be executed but also can cause poor performance due to high switching rates.

In this program, Round Robin algorithm is used with process that have priority values both "2" and "3". Priority value "2" processes are queued into CPU-2 queue_medium with quantum value 8 and priority value "3" processes are queued into CPU-2 queue_low quantum value 16.

Because of the limited memory space that is given in the instruction, in order to be able to execute each process memory management become necessary. A basic memory management design is implemented into the program with an allocation and de-allocation functions.

Memory management ⇒

allocateMemory: allocateMemory method has a basic function of allocating memory for the process that is given as a parameter. This method has three parameters: process name array, required memory and priority.

The function iterates memory blocks to find an available one with sufficient memory according to process priority. Once the memory block found, the memory address is returned or else "-1" value is returned.

If the memory block has more memory than required, than the block is split so that the excess memory can still be available.

deallocateMemory: deallocateMemory method de-allocates the memory that has been used by a process that has been simulated. So that memory space can be used repeatedly, making use of a little space to execute processes.

The method searches the memory blocks that has been allocated to a specific process. Once found, it resets the process name making the space available.

Also if the adjacent memory block are free, the deallocated memory is merge with them to create one larger memory space.

The process are read from an input text file and the results are written to an output text file. So to take care of these functionalities, file I/O algorithms are necessary.

File I/O ⇛

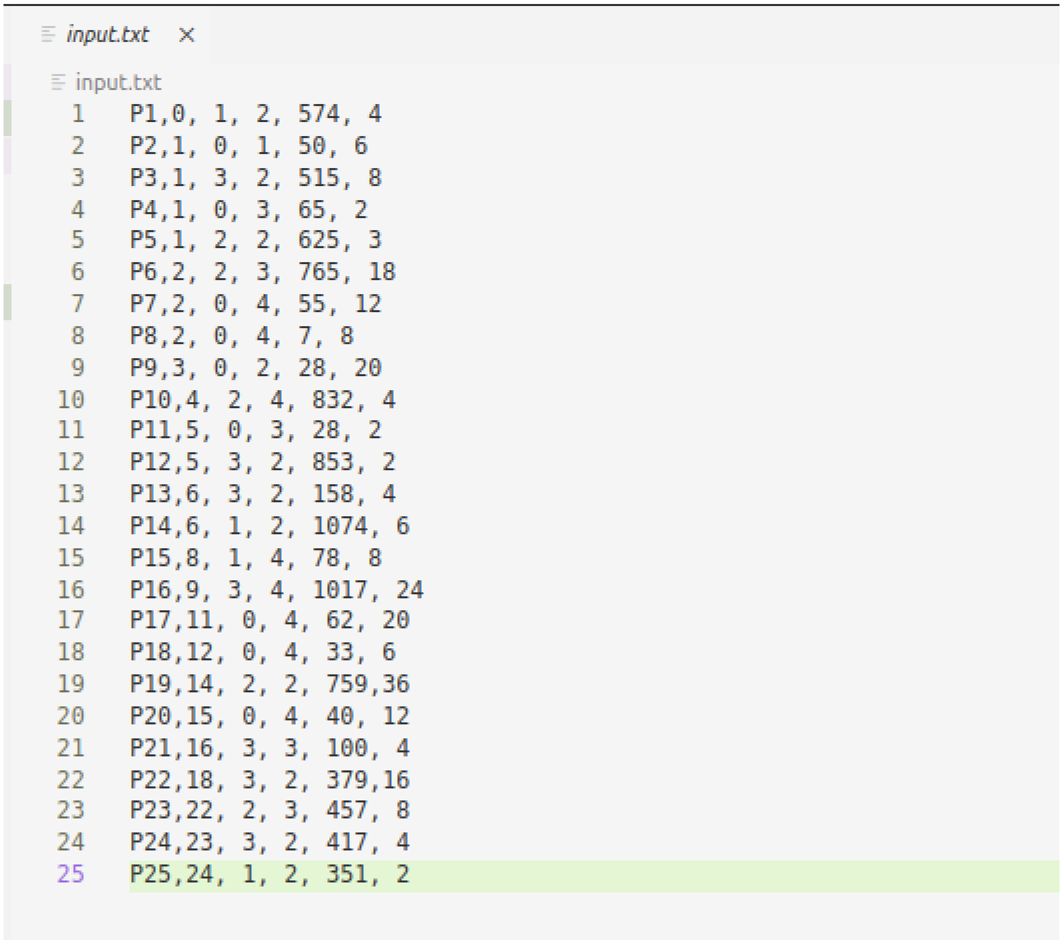
read_process: read_process function is for reading the processes from an input file and initializing the process as Process struct array. This processes array is taken as a parameter and after a successful execution returned from the method. For this method **'sscanf'** function is made use of.

write_to_output: This function is responsible for creating and writing the necessary information to the output file with **'fprintf'** function. Also for making the debug process easier, the necessary information written to console.

All of these algorithms are called inside the main function to execute the application successfully.

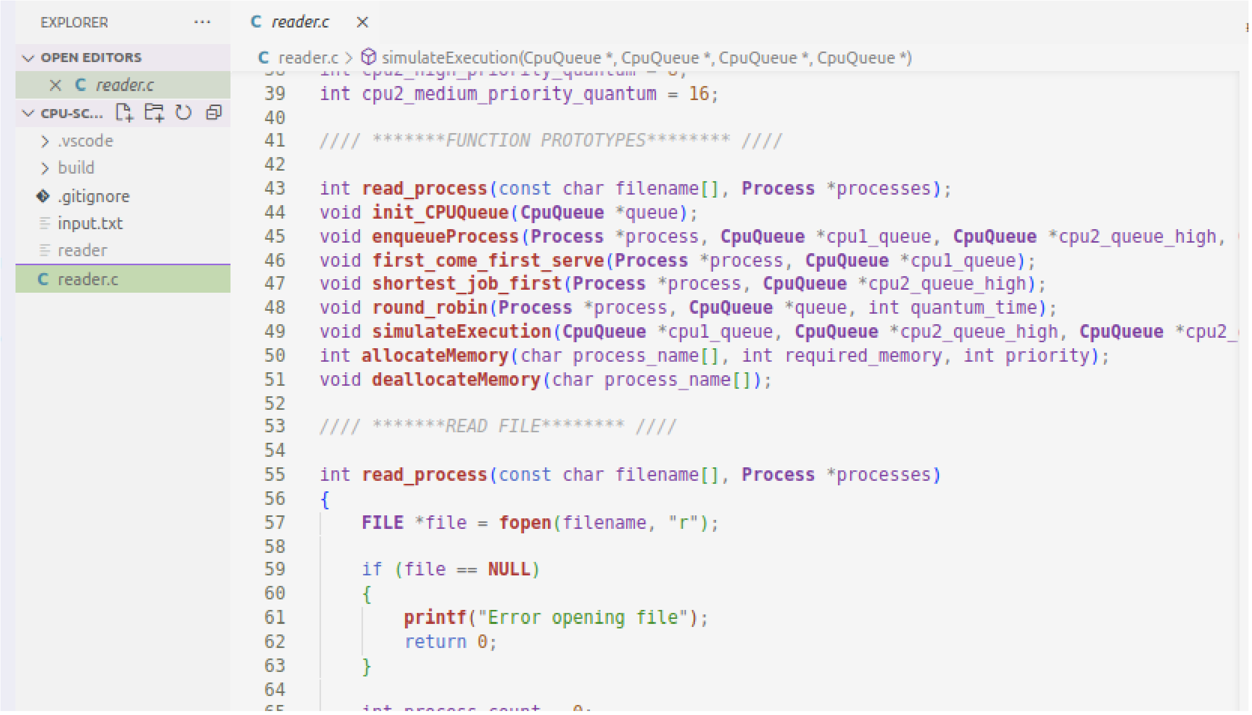
Main Function ⇛

First processes are read and initialized with read_process also making use of process data structure and allocateMemory. Then these processes are enqueue based on their priorities to different queues. These queues then simulated according to their burst times and the information is written to the output with write_to_output. After making sure that the memory is freed and the file is closed, the program exits with 0.



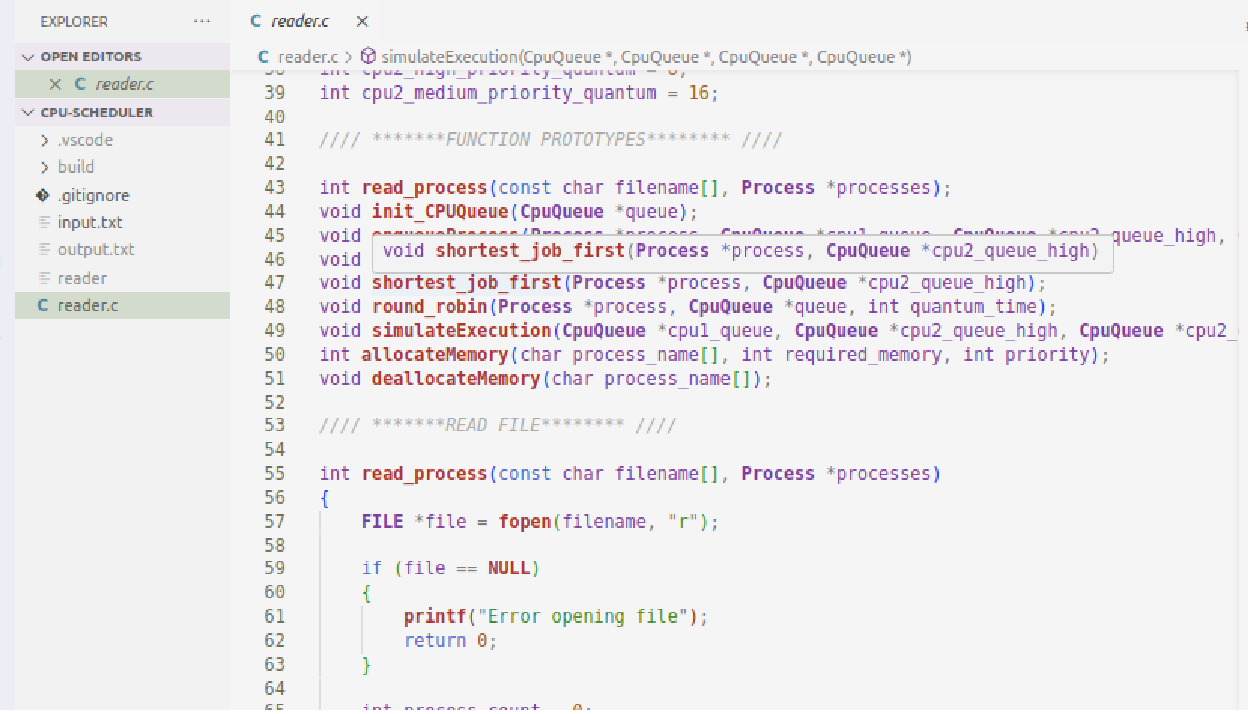
```
P1,0, 1, 2, 574, 4
P2,1, 0, 1, 50, 6
P3,1, 3, 2, 515, 8
P4,1, 0, 3, 65, 2
P5,1, 2, 2, 625, 3
P6,2, 2, 3, 765, 18
P7,2, 0, 4, 55, 12
P8,2, 0, 4, 7, 8
P9,3, 0, 2, 28, 20
P10,4, 2, 4, 832, 4
P11,5, 0, 3, 28, 2
P12,5, 3, 2, 853, 2
P13,6, 3, 2, 158, 4
P14,6, 1, 2, 1074, 6
P15,8, 1, 4, 78, 8
P16,9, 3, 4, 1017, 24
P17,11, 0, 4, 62, 20
P18,12, 0, 4, 33, 6
P19,14, 2, 2, 759,36
P20,15, 0, 4, 40, 12
P21,16, 3, 3, 100, 4
P22,18, 3, 2, 379,16
P23,22, 2, 3, 457, 8
P24,23, 3, 2, 417, 4
P25,24, 1, 2, 351, 2
```

Input text file



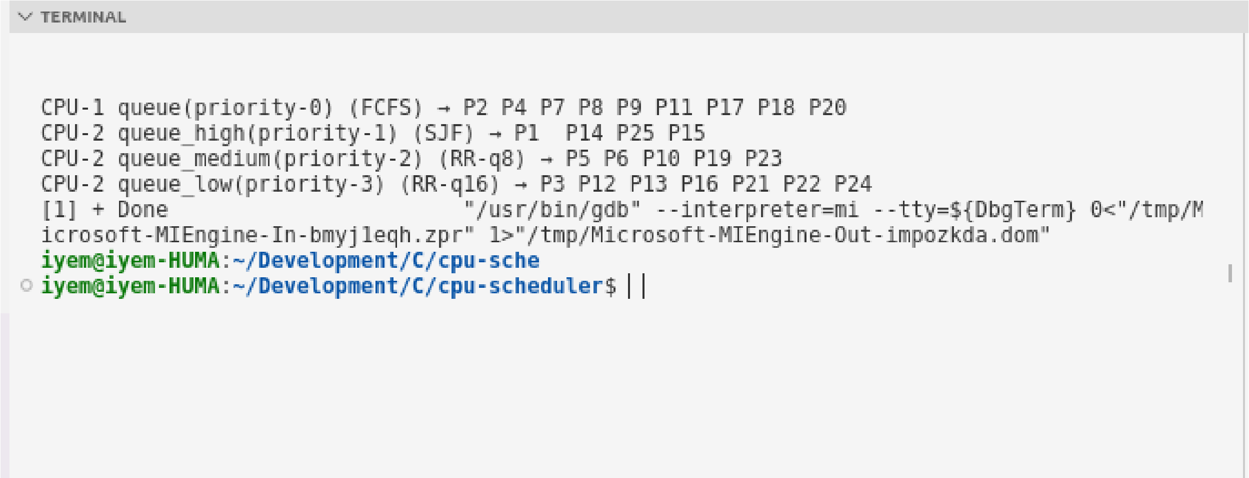
```c
int cpu2_high_priority_quantum = 8;
int cpu2_medium_priority_quantum = 16;

//// *******FUNCTION PROTOTYPES******** ////

int read_process(const char filename[], Process *processes);
void init_CPUQueue(CpuQueue *queue);
void enqueueProcess(Process *process, CpuQueue *cpu1_queue, CpuQueue *cpu2_queue_high,
void first_come_first_serve(Process *process, CpuQueue *cpu1_queue);
void shortest_job_first(Process *process, CpuQueue *cpu2_queue_high);
void round_robin(Process *process, CpuQueue *queue, int quantum_time);
void simulateExecution(CpuQueue *cpu1_queue, CpuQueue *cpu2_queue_high, CpuQueue *cpu2_
int allocateMemory(char process_name[], int required_memory, int priority);
void deallocateMemory(char process_name[]);

//// *******READ FILE******** ////

int read_process(const char filename[], Process *processes)
{
    FILE *file = fopen(filename, "r");

    if (file == NULL)
    {
        printf("Error opening file");
        return 0;
    }

    int process_count = 0;
```

Function prototypes from the source code with also showing the file structure



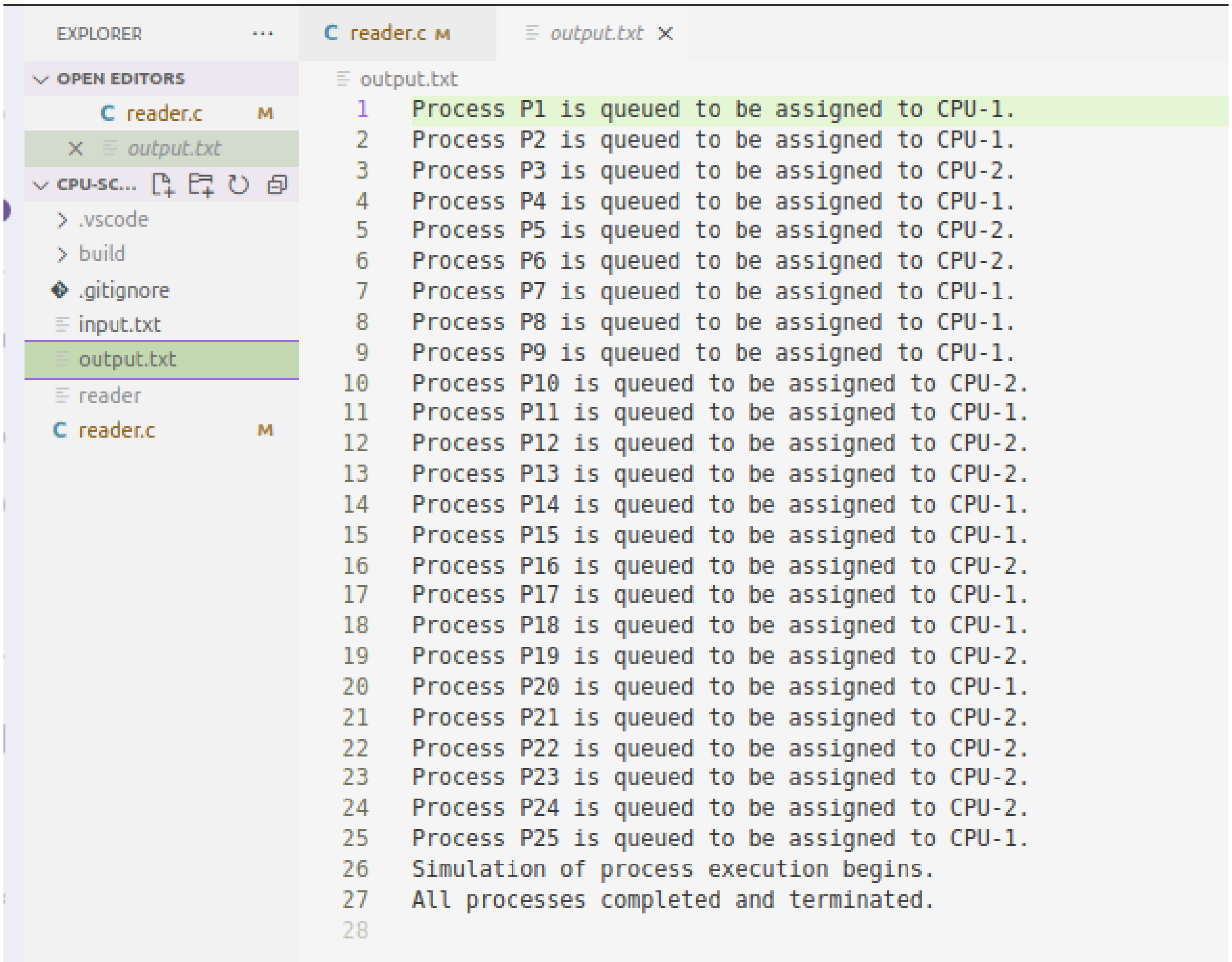
```c
int cpu2_high_priority_quantum = 8;
int cpu2_medium_priority_quantum = 16;

//// *******FUNCTION PROTOTYPES******** ////

int read_process(const char filename[], Process *processes);
void init_CPUQueue(CpuQueue *queue);
void enqueueProcess(Process *process, CpuQueue *cpu1_queue, CpuQueue *cpu2_queue_high,
void  void shortest_job_first(Process *process, CpuQueue *cpu2_queue_high)
void shortest_job_first(Process *process, CpuQueue *cpu2_queue_high);
void round_robin(Process *process, CpuQueue *queue, int quantum_time);
void simulateExecution(CpuQueue *cpu1_queue, CpuQueue *cpu2_queue_high, CpuQueue *cpu2_
int allocateMemory(char process_name[], int required_memory, int priority);
void deallocateMemory(char process_name[]);

//// *******READ FILE******** ////

int read_process(const char filename[], Process *processes)
{
    FILE *file = fopen(filename, "r");

    if (file == NULL)
    {
        printf("Error opening file");
        return 0;
    }

    int process_count = 0;
```

After program is executed, the output file is created



```
CPU-1 queue(priority-0) (FCFS) → P2 P4 P7 P8 P9 P11 P17 P18 P20
CPU-2 queue_high(priority-1) (SJF) → P1  P14 P25 P15
CPU-2 queue_medium(priority-2) (RR-q8) → P5 P6 P10 P19 P23
CPU-2 queue_low(priority-3) (RR-q16) → P3 P12 P13 P16 P21 P22 P24
[1] + Done                       "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/M
icrosoft-MIEngine-In-bmyjleqh.zpr" 1>"/tmp/Microsoft-MIEngine-Out-impozkda.dom"
iyem@iyem-HUMA:~/Development/C/cpu-sche
iyem@iyem-HUMA:~/Development/C/cpu-scheduler$ ||
```

Terminal output after execution



```
Process P1 is queued to be assigned to CPU-1.
Process P2 is queued to be assigned to CPU-1.
Process P3 is queued to be assigned to CPU-2.
Process P4 is queued to be assigned to CPU-1.
Process P5 is queued to be assigned to CPU-2.
Process P6 is queued to be assigned to CPU-2.
Process P7 is queued to be assigned to CPU-1.
Process P8 is queued to be assigned to CPU-1.
Process P9 is queued to be assigned to CPU-1.
Process P10 is queued to be assigned to CPU-2.
Process P11 is queued to be assigned to CPU-1.
Process P12 is queued to be assigned to CPU-2.
Process P13 is queued to be assigned to CPU-2.
Process P14 is queued to be assigned to CPU-1.
Process P15 is queued to be assigned to CPU-1.
Process P16 is queued to be assigned to CPU-2.
Process P17 is queued to be assigned to CPU-1.
Process P18 is queued to be assigned to CPU-1.
Process P19 is queued to be assigned to CPU-2.
Process P20 is queued to be assigned to CPU-1.
Process P21 is queued to be assigned to CPU-2.
Process P22 is queued to be assigned to CPU-2.
Process P23 is queued to be assigned to CPU-2.
Process P24 is queued to be assigned to CPU-2.
Process P25 is queued to be assigned to CPU-1.
Simulation of process execution begins.
All processes completed and terminated.
```

Output file that has been created