# COSE341
# Operating System

Department: 컴퓨터학과

Student Number: 2022320039

Name: 정진욱

Submission date: 5/28

Freedays: 5

**Development environment**

- **IDE: vi**
- **Linux version: 4.20.11**
- **Ubuntu version: 18.04.2**

**Explanation of CPU scheduling and policies**

**First-Come, First-Served (FCFS)**

Processes are executed in the order they arrive in the ready queue. Once a process starts execution, it runs to completion without being preempted. This policy ensures predictable processing based on arrival time. But this policy can lead to the "convoy effect", where short processes wait for long processes to complete, resulting in poor average waiting time.

**Shortest Remaining Time First (SRTF)**

SRTF, a preemptive version of the Shortest Job Next (SJN) algorithm, selects the process with the shortest remaining execution time. If a new process arrives with a shorter remaining time than the current process, the current process is preempted. This algorithm minimizes average waiting time. But, in real world, it is difficult to predict the length of CPU burst time and cause starvation of longer processes if short processes keep arriving.

**Round Robin (RR)**

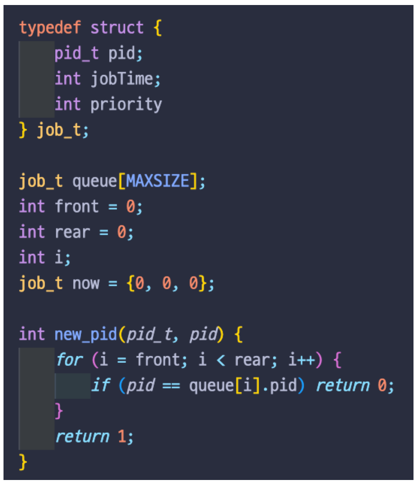
Round Robin scheduling is designed for time-sharing systems. Each process is assigned a fixed time slice (quantum). If a process does not finish within its time slice, it is preempted and moved to the end of the ready queue. It ensures all processes get a fair share of the CPU and short response time. But context switching overhead can be high.

**Priority Scheduling**

In Priority Scheduling, each process is assigned a priority. The CPU is allocated to the process with the highest priority. It ensures that important processes are handled first but can cause starvation of low priority process.

## Implementation

1. Data structure for ku_cpu



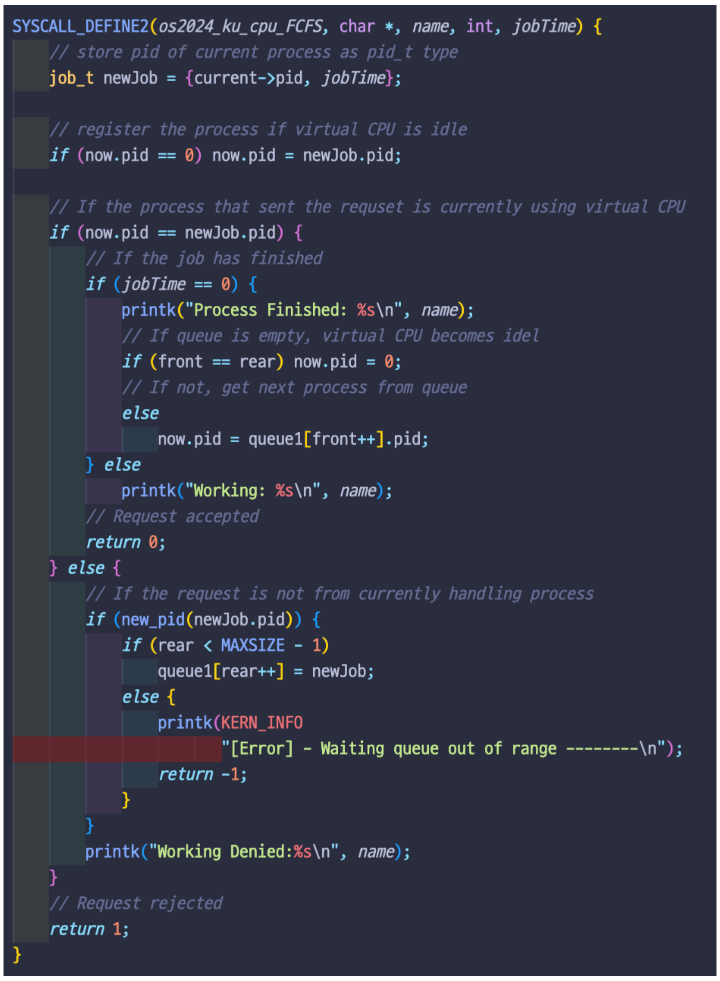
```c
typedef struct {
    pid_t pid;
    int jobTime;
    int priority
} job_t;

job_t queue[MAXSIZE];
int front = 0;
int rear = 0;
int i;
job_t now = {0, 0, 0};

int new_pid(pid_t, pid) {
    for (i = front; i < rear; i++) {
        if (pid == queue[i].pid) return 0;
    }
    return 1;
}
```

The system defines a 'job_t' structure to represent a job. It includes the PID, jobTime, and priority for CPU Scheduling.

Waiting queue is defined by job_t array. Front is the index of first contents of current waiting queue and rear is the index of the first empty field.

New_pid is function for checking called process is already in the waiting queue or not.

2. FCFS



```c
SYSCALL_DEFINE2(os2024_ku_cpu_FCFS, char *, name, int, jobTime) {
    // store pid of current process as pid_t type
    job_t newJob = {current->pid, jobTime};

    // register the process if virtual CPU is idle
    if (now.pid == 0) now.pid = newJob.pid;

    // If the process that sent the requset is currently using virtual CPU
    if (now.pid == newJob.pid) {
        // If the job has finished
        if (jobTime == 0) {
            printk("Process Finished: %s\n", name);
            // If queue is empty, virtual CPU becomes idel
            if (front == rear) now.pid = 0;
            // If not, get next process from queue
            else
                now.pid = queue1[front++].pid;
        } else
            printk("Working: %s\n", name);
        // Request accepted
        return 0;
    } else {
        // If the request is not from currently handling process
        if (new_pid(newJob.pid)) {
            if (rear < MAXSIZE - 1)
                queue1[rear++] = newJob;
            else {
                printk(KERN_INFO
                       "[Error] - Waiting queue out of range --------\n");
                return -1;
            }
        }
        printk("Working Denied:%s\n", name);
    }
    // Request rejected
    return 1;
}
```

Ku_cpu keeps track of which process is currently running by now.

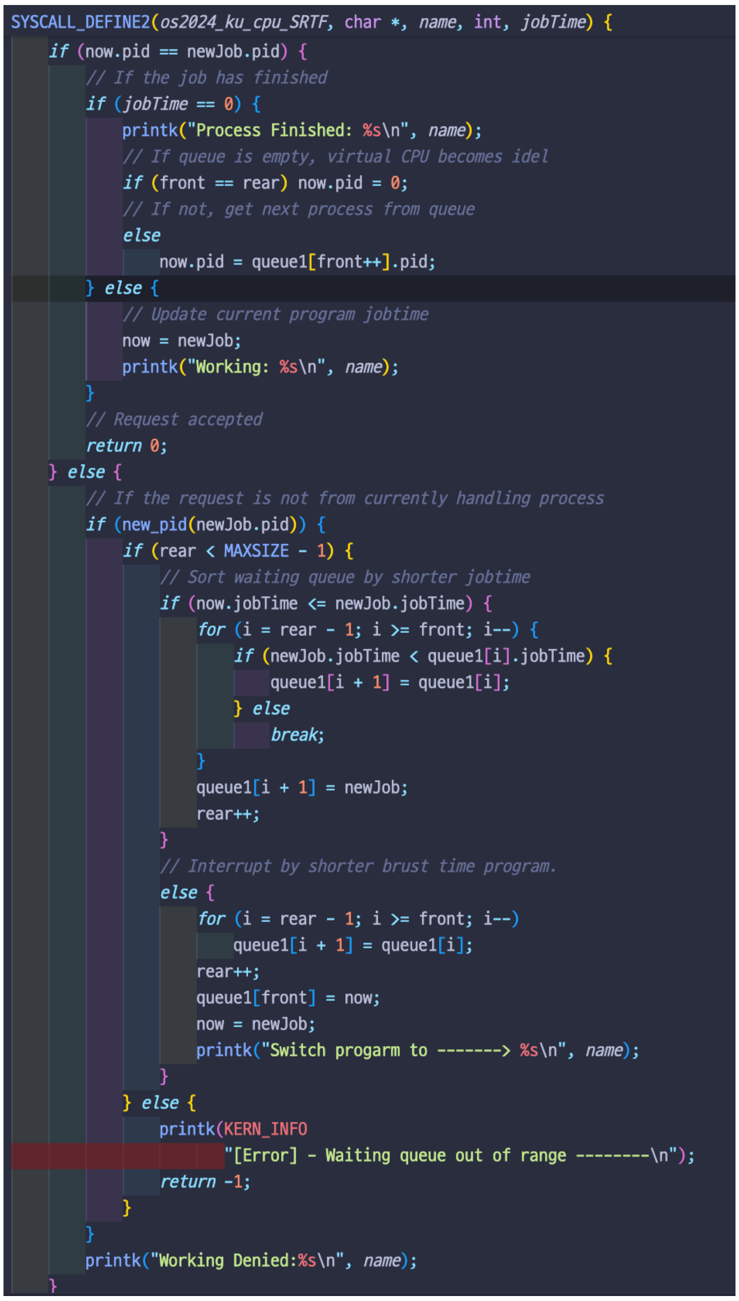
If the ku_cpu is idle (pid == 0), the first process that request the ku_cpu is allowed to occupy it. Ku_cpu sets this process as the running process and return 0.

If the process currently requesting the ku_cpu is the same one that is already occupying it, the system will return 0 to confirm the successful continuation of ku_cpu usage.

If the different process requests the ku_cpu while it is occupied, the different process is added to the waiting queue only if it never enqueued already. The system then returns 1 to notify the user process of the waiting status. Additionally, I implemented error handling to prevent exceeding the maximum size of the queue.

If the process currently using the ku_cpu finishes, the system checks if there are other processes in the waiting queue. If there are, the next process from the queue is assigned to the CPU. Otherwise, the CPU is set to idle.

3. SRFT

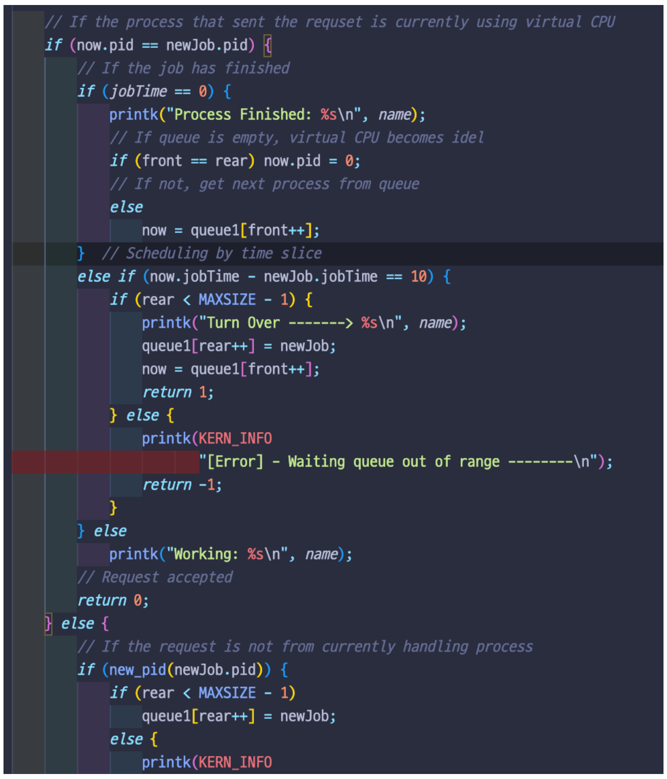


```c
SYSCALL_DEFINE2(os2024_ku_cpu_SRTF, char *, name, int, jobTime) {
    if (now.pid == newJob.pid) {
        // If the job has finished
        if (jobTime == 0) {
            printk("Process Finished: %s\n", name);
            // If queue is empty, virtual CPU becomes idel
            if (front == rear) now.pid = 0;
            // If not, get next process from queue
            else
                now.pid = queue1[front++].pid;
        } else {
            // Update current program jobtime
            now = newJob;
            printk("Working: %s\n", name);
        }
        // Request accepted
        return 0;
    } else {
        // If the request is not from currently handling process
        if (new_pid(newJob.pid)) {
            if (rear < MAXSIZE - 1) {
                // Sort waiting queue by shorter jobtime
                if (now.jobTime <= newJob.jobTime) {
                    for (i = rear - 1; i >= front; i--) {
                        if (newJob.jobTime < queue1[i].jobTime) {
                            queue1[i + 1] = queue1[i];
                        } else
                            break;
                    }
                    queue1[i + 1] = newJob;
                    rear++;
                }
                // Interrupt by shorter brust time program.
                else {
                    for (i = rear - 1; i >= front; i--)
                        queue1[i + 1] = queue1[i];
                    rear++;
                    queue1[front] = now;
                    now = newJob;
                    printk("Switch progarm to -------> %s\n", name);
                }
            } else {
                printk(KERN_INFO
                       "[Error] - Waiting queue out of range --------\n");
                return -1;
            }
        }
        printk("Working Denied:%s\n", name);
    }
```

The basic structure is almost identical to FCFS. However, for the SRTF algorithm, the remaining job time of the current process must be continuously tracked. Therefore, when a system call from the same process is received, the `now` variable is updated with the current job time.

If the requesting process is different from the currently running process, it first compares the job time with the currently running process. If the incoming process has a longer job time, it is inserted into the waiting queue at the appropriate position by comparing from the back to sort the queue by ascending job time. Otherwise, the current process is placed at the front of the queue, and `now` is replaced with the incoming process.

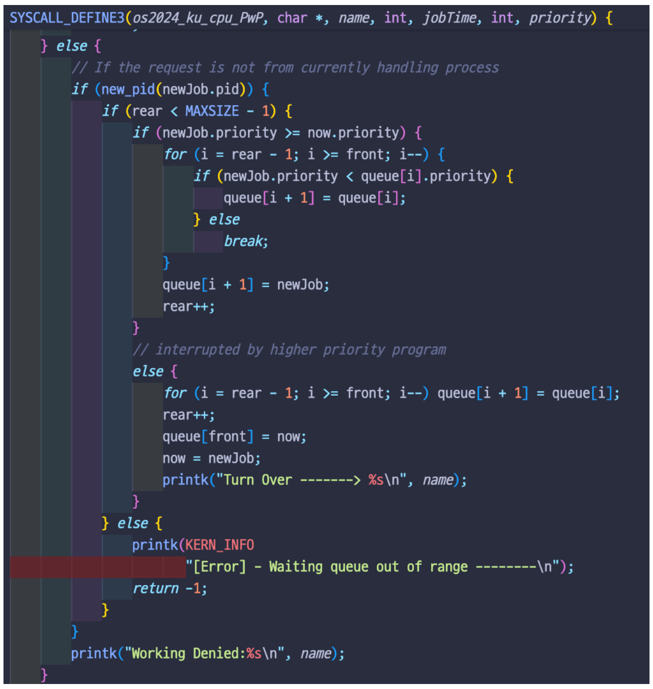
4. RR (Timeslice: 1sec)



```
    // If the process that sent the requset is currently using virtual CPU
    if (now.pid == newJob.pid) {
        // If the job has finished
        if (jobTime == 0) {
            printk("Process Finished: %s\n", name);
            // If queue is empty, virtual CPU becomes idel
            if (front == rear) now.pid = 0;
            // If not, get next process from queue
            else
                now = queue1[front++];
        }  // Scheduling by time slice
        else if (now.jobTime - newJob.jobTime == 10) {
            if (rear < MAXSIZE - 1) {
                printk("Turn Over -------> %s\n", name);
                queue1[rear++] = newJob;
                now = queue1[front++];
                return 1;
            } else {
                printk(KERN_INFO
                       "[Error] - Waiting queue out of range --------\n");
                return -1;
            }
        } else
            printk("Working: %s\n", name);
        // Request accepted
        return 0;
    } else {
        // If the request is not from currently handling process
        if (new_pid(newJob.pid)) {
            if (rear < MAXSIZE - 1)
                queue1[rear++] = newJob;
            else {
                printk(KERN_INFO
```

I implemented a Round Robin scheduler with a timeslice of 1 second. Similarly, the basic structure is like FCFS. When a system call from the same process is received, if there is a 1-second difference in the job time, the process is switched by moving the current process to the end of the queue and replacing it with the first process in the waiting queue. This way, context switching is implemented.

5. Priority

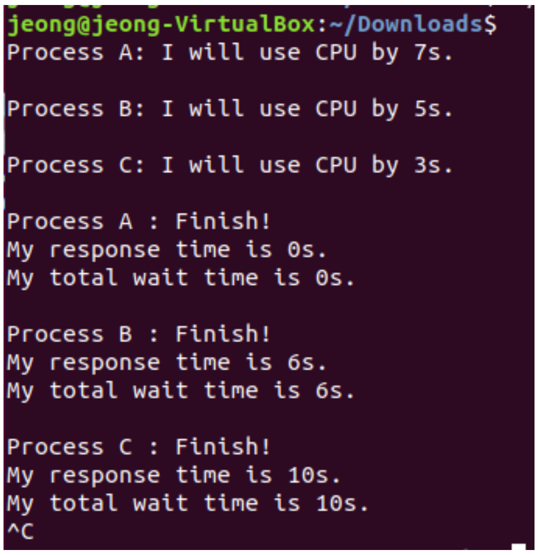


```
SYSCALL_DEFINE3(os2024_ku_cpu_PwP, char *, name, int, jobTime, int, priority) {
    } else {
        // If the request is not from currently handling process
        if (new_pid(newJob.pid)) {
            if (rear < MAXSIZE - 1) {
                if (newJob.priority >= now.priority) {
                    for (i = rear - 1; i >= front; i--) {
                        if (newJob.priority < queue[i].priority) {
                            queue[i + 1] = queue[i];
                        } else
                            break;
                    }
                    queue[i + 1] = newJob;
                    rear++;
                }
                // interrupted by higher priority program
                else {
                    for (i = rear - 1; i >= front; i--) queue[i + 1] = queue[i];
                    rear++;
                    queue[front] = now;
                    now = newJob;
                    printk("Turn Over -------> %s\n", name);
                }
            } else {
                printk(KERN_INFO
                       "[Error] - Waiting queue out of range --------\n");
                return -1;
            }
        }
        printk("Working Denied:%s\n", name);
    }
```

Priority scheduling implementation is like SRTF. However, when receiving a system call, the priority is also received to determine the priority of the process. The priorities of the current job and the incoming job are compared. If the incoming job has a lower priority (higher numerical value), it is inserted into the queue sorted by ascending priority. If it has a higher priority (lower numerical value), the current process is moved to the front of the queue, and the incoming process is assigned to the CPU.

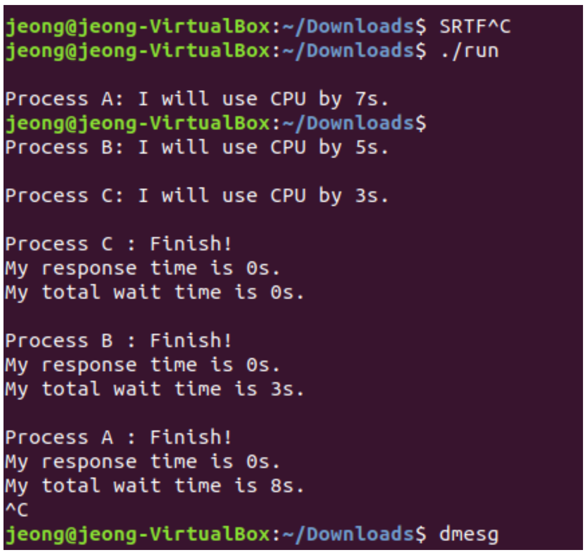
**Experiment results and your analysis with screenshots of execution results (logs).**

**FCFS**

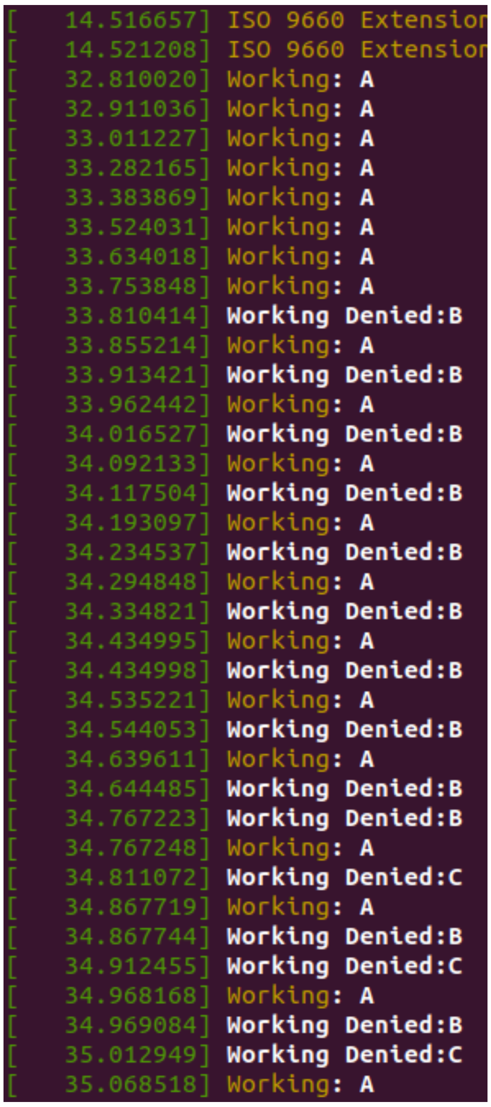


```
jeong@jeong-VirtualBox:~/Downloads$
Process A: I will use CPU by 7s.

Process B: I will use CPU by 5s.

Process C: I will use CPU by 3s.

Process A : Finish!
My response time is 0s.
My total wait time is 0s.

Process B : Finish!
My response time is 6s.
My total wait time is 6s.

Process C : Finish!
My response time is 10s.
My total wait time is 10s.
^C
```

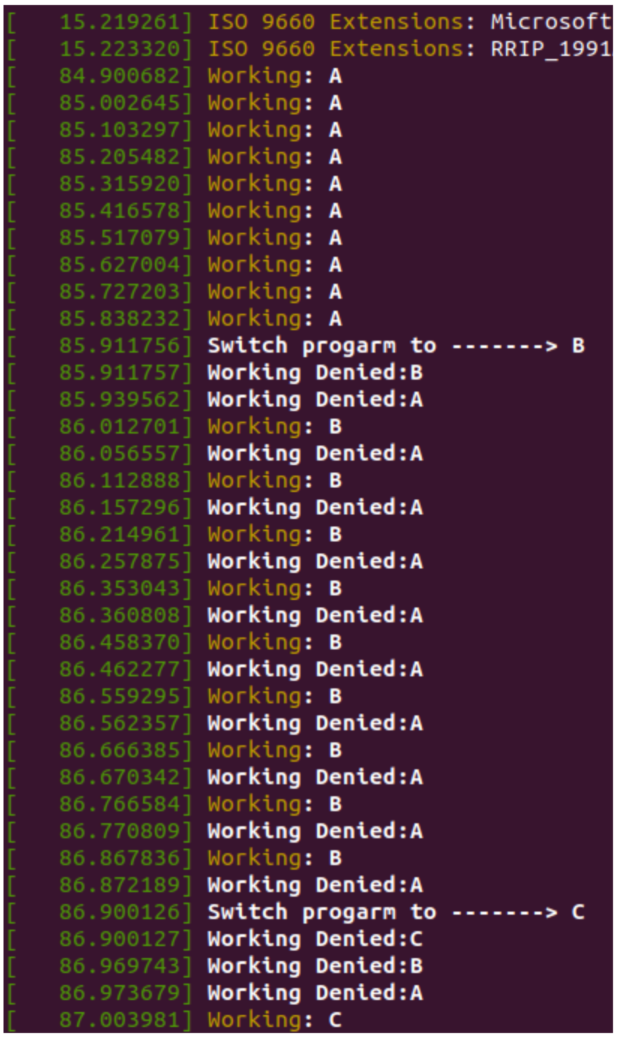
**SRTF**



```
jeong@jeong-VirtualBox:~/Downloads$ SRTF^C
jeong@jeong-VirtualBox:~/Downloads$ ./run

Process A: I will use CPU by 7s.
jeong@jeong-VirtualBox:~/Downloads$
Process B: I will use CPU by 5s.

Process C: I will use CPU by 3s.

Process C : Finish!
My response time is 0s.
My total wait time is 0s.

Process B : Finish!
My response time is 0s.
My total wait time is 3s.

Process A : Finish!
My response time is 0s.
My total wait time is 8s.
^C
jeong@jeong-VirtualBox:~/Downloads$ dmesg
```

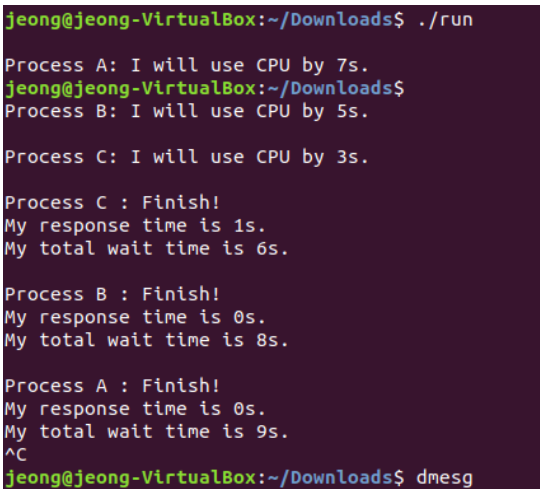


```
[   14.516657] ISO 9660 Extension
[   14.521208] ISO 9660 Extension
[   32.810020] Working: A
[   32.911036] Working: A
[   33.011227] Working: A
[   33.282165] Working: A
[   33.383869] Working: A
[   33.524031] Working: A
[   33.634018] Working: A
[   33.753848] Working: A
[   33.810414] Working Denied:B
[   33.855214] Working: A
[   33.913421] Working Denied:B
[   33.962442] Working: A
[   34.016527] Working Denied:B
[   34.092133] Working: A
[   34.117504] Working Denied:B
[   34.193097] Working: A
[   34.234537] Working Denied:B
[   34.294848] Working: A
[   34.334821] Working Denied:B
[   34.434995] Working: A
[   34.434998] Working Denied:B
[   34.535221] Working: A
[   34.544053] Working Denied:B
[   34.639611] Working: A
[   34.644485] Working Denied:B
[   34.767223] Working Denied:B
[   34.767248] Working: A
[   34.811072] Working Denied:C
[   34.867719] Working: A
[   34.867744] Working Denied:B
[   34.912455] Working Denied:C
[   34.968168] Working: A
[   34.969084] Working Denied:B
[   35.012949] Working Denied:C
[   35.068518] Working: A
```

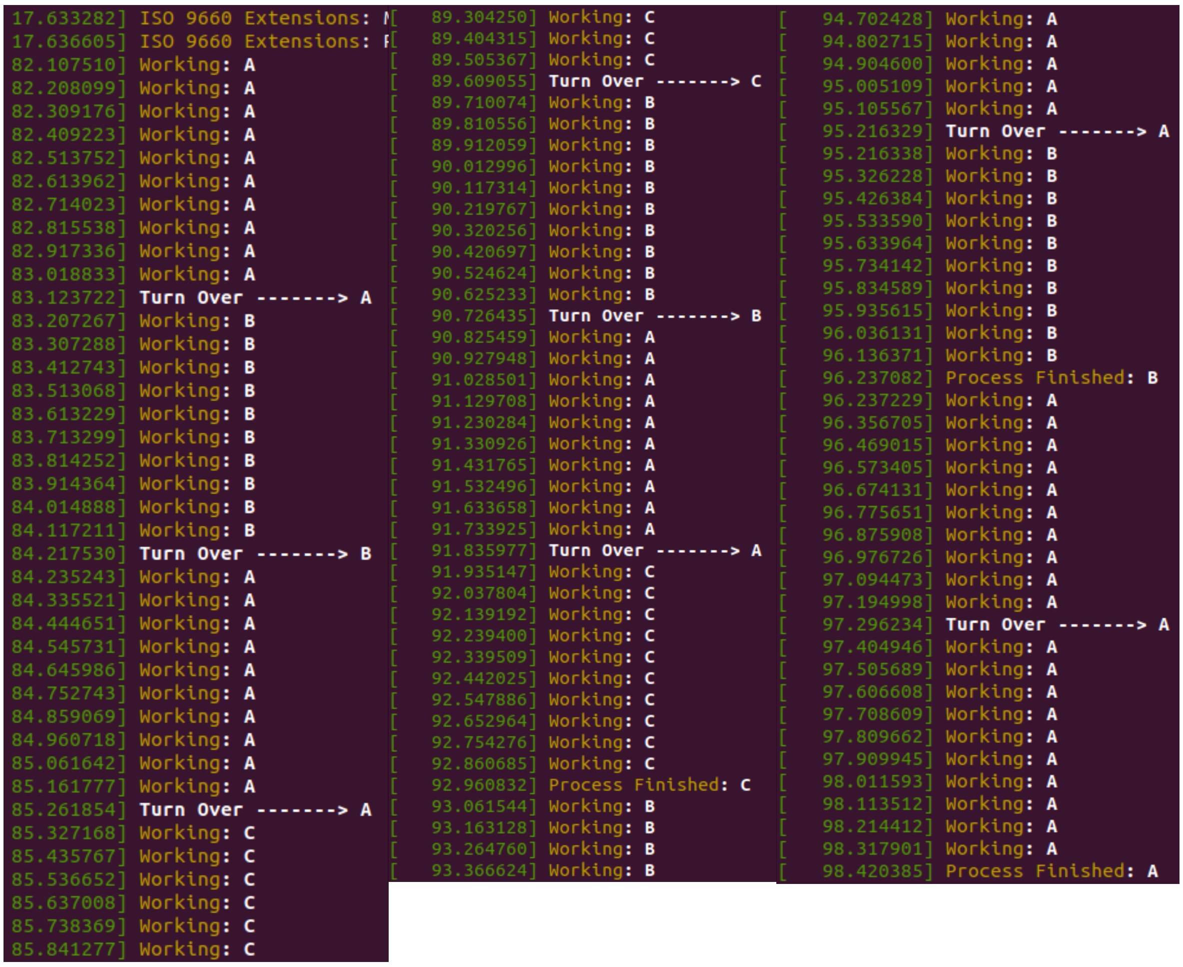


```
[   15.219261] ISO 9660 Extensions: Microsoft
[   15.223320] ISO 9660 Extensions: RRIP_1991
[   84.900682] Working: A
[   85.002645] Working: A
[   85.103297] Working: A
[   85.205482] Working: A
[   85.315920] Working: A
[   85.416578] Working: A
[   85.517079] Working: A
[   85.627004] Working: A
[   85.727203] Working: A
[   85.838232] Working: A
[   85.911756] Switch progarm to -------> B
[   85.911757] Working Denied:B
[   85.939562] Working Denied:A
[   86.012701] Working: B
[   86.056557] Working Denied:A
[   86.112888] Working: B
[   86.157296] Working Denied:A
[   86.214961] Working: B
[   86.257875] Working Denied:A
[   86.353043] Working: B
[   86.360808] Working Denied:A
[   86.458370] Working: B
[   86.462277] Working Denied:A
[   86.559295] Working: B
[   86.562357] Working Denied:A
[   86.666385] Working: B
[   86.670342] Working Denied:A
[   86.766584] Working: B
[   86.770809] Working Denied:A
[   86.867836] Working: B
[   86.872189] Working Denied:A
[   86.900126] Switch progarm to -------> C
[   86.900127] Working Denied:C
[   86.969743] Working Denied:B
[   86.973679] Working Denied:A
[   87.003981] Working: C
```

**RR**



```
jeong@jeong-VirtualBox:~/Downloads$ ./run

Process A: I will use CPU by 7s.
jeong@jeong-VirtualBox:~/Downloads$
Process B: I will use CPU by 5s.

Process C: I will use CPU by 3s.

Process C : Finish!
My response time is 1s.
My total wait time is 6s.

Process B : Finish!
My response time is 0s.
My total wait time is 8s.

Process A : Finish!
My response time is 0s.
My total wait time is 9s.
^C
jeong@jeong-VirtualBox:~/Downloads$ dmesg
```

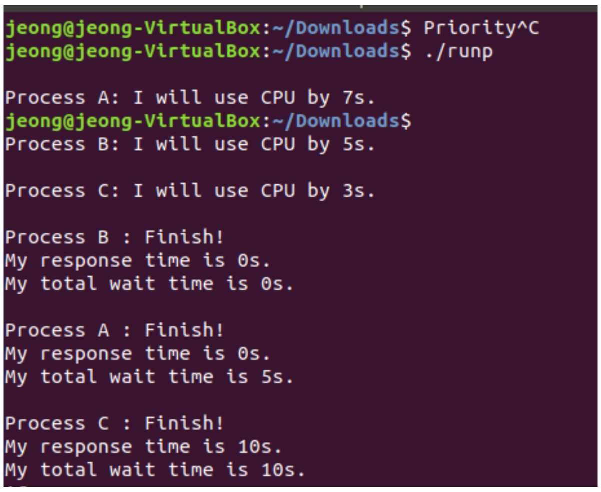


```
17.633282] ISO 9660 Extensions: M
17.636605] ISO 9660 Extensions: R
82.107510] Working: A
82.208099] Working: A
82.309176] Working: A
82.409223] Working: A
82.513752] Working: A
82.613962] Working: A
82.714023] Working: A
82.815538] Working: A
82.917336] Working: A
83.018833] Working: A
83.123722] Turn Over -------> A
83.207267] Working: B
83.307288] Working: B
83.412743] Working: B
83.513068] Working: B
83.613229] Working: B
83.713299] Working: B
83.814252] Working: B
83.914364] Working: B
84.014888] Working: B
84.117211] Working: B
84.217530] Turn Over -------> B
84.235243] Working: A
84.335521] Working: A
84.444651] Working: A
84.545731] Working: A
84.645986] Working: A
84.752743] Working: A
84.859069] Working: A
84.960718] Working: A
85.061642] Working: A
85.161777] Working: A
85.261854] Turn Over -------> A
85.327168] Working: C
85.435767] Working: C
85.536652] Working: C
85.637008] Working: C
85.738369] Working: C
85.841277] Working: C
```

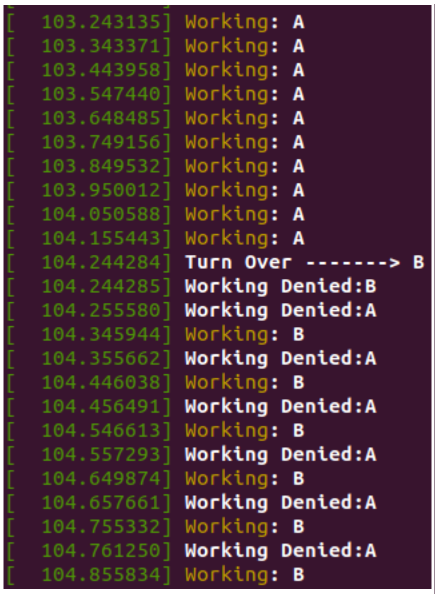
```
[   89.304250] Working: C
[   89.404315] Working: C
[   89.505367] Working: C
[   89.609055] Turn Over -------> C
[   89.710074] Working: B
[   89.810556] Working: B
[   89.912059] Working: B
[   90.012996] Working: B
[   90.117314] Working: B
[   90.219767] Working: B
[   90.320256] Working: B
[   90.420697] Working: B
[   90.524624] Working: B
[   90.625233] Working: B
[   90.726435] Turn Over -------> B
[   90.825459] Working: A
[   90.927948] Working: A
[   91.028501] Working: A
[   91.129708] Working: A
[   91.230284] Working: A
[   91.330926] Working: A
[   91.431765] Working: A
[   91.532496] Working: A
[   91.633658] Working: A
[   91.733925] Working: A
[   91.835977] Turn Over -------> A
[   91.935147] Working: C
[   92.037804] Working: C
[   92.139192] Working: C
[   92.239400] Working: C
[   92.339509] Working: C
[   92.442025] Working: C
[   92.547886] Working: C
[   92.652964] Working: C
[   92.754276] Working: C
[   92.860685] Working: C
[   92.960832] Process Finished: C
[   93.061544] Working: B
[   93.163128] Working: B
[   93.264760] Working: B
[   93.366624] Working: B
```

```
[   94.702428] Working: A
[   94.802715] Working: A
[   94.904600] Working: A
[   95.005109] Working: A
[   95.105567] Working: A
[   95.216329] Turn Over -------> A
[   95.216338] Working: B
[   95.326228] Working: B
[   95.426384] Working: B
[   95.533590] Working: B
[   95.633964] Working: B
[   95.734142] Working: B
[   95.834589] Working: B
[   95.935615] Working: B
[   96.036131] Working: B
[   96.136371] Working: B
[   96.237082] Process Finished: B
[   96.237229] Working: A
[   96.356705] Working: A
[   96.469015] Working: A
[   96.573405] Working: A
[   96.674131] Working: A
[   96.775651] Working: A
[   96.875908] Working: A
[   96.976726] Working: A
[   97.094473] Working: A
[   97.194998] Working: A
[   97.296234] Turn Over -------> A
[   97.404946] Working: A
[   97.505689] Working: A
[   97.606608] Working: A
[   97.708609] Working: A
[   97.809662] Working: A
[   97.909945] Working: A
[   98.011593] Working: A
[   98.113512] Working: A
[   98.214412] Working: A
[   98.317901] Working: A
[   98.420385] Process Finished: A
```

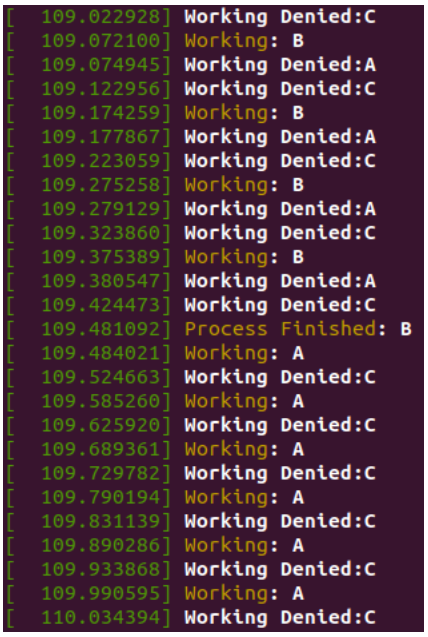
## Priority



```
jeong@jeong-VirtualBox:~/Downloads$ Priority^C
jeong@jeong-VirtualBox:~/Downloads$ ./runp

Process A: I will use CPU by 7s.
jeong@jeong-VirtualBox:~/Downloads$
Process B: I will use CPU by 5s.

Process C: I will use CPU by 3s.

Process B : Finish!
My response time is 0s.
My total wait time is 0s.

Process A : Finish!
My response time is 0s.
My total wait time is 5s.

Process C : Finish!
My response time is 10s.
My total wait time is 10s.
```

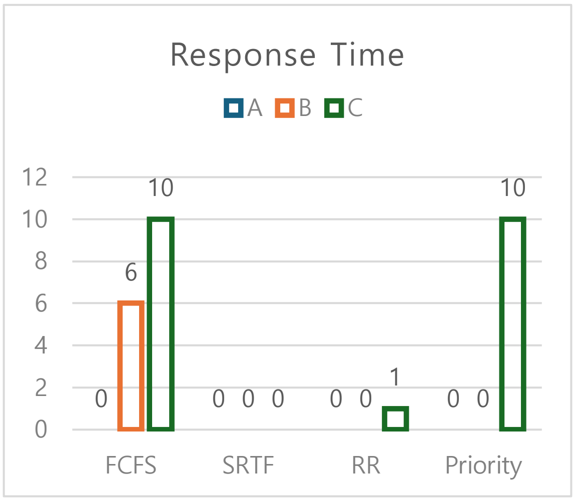


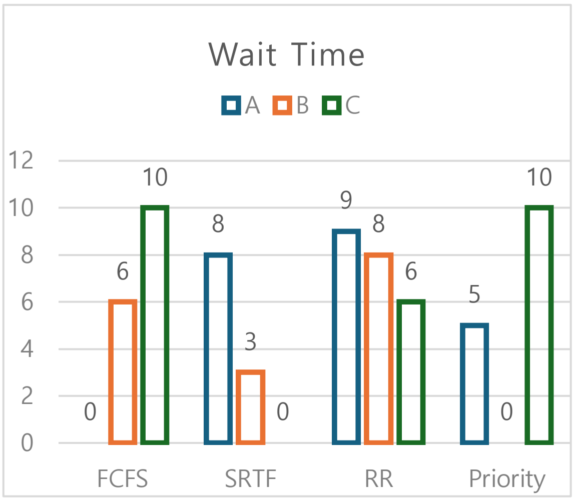
```
[  103.243135] Working: A
[  103.343371] Working: A
[  103.443958] Working: A
[  103.547440] Working: A
[  103.648485] Working: A
[  103.749156] Working: A
[  103.849532] Working: A
[  103.950012] Working: A
[  104.050588] Working: A
[  104.155443] Working: A
[  104.244284] Turn Over -------> B
[  104.244285] Working Denied:B
[  104.255580] Working Denied:A
[  104.345944] Working: B
[  104.355662] Working Denied:A
[  104.446038] Working: B
[  104.456491] Working Denied:A
[  104.546613] Working: B
[  104.557293] Working Denied:A
[  104.649874] Working: B
[  104.657661] Working Denied:A
[  104.755332] Working: B
[  104.761250] Working Denied:A
[  104.855834] Working: B
```



```
[  109.022928] Working Denied:C
[  109.072100] Working: B
[  109.074945] Working Denied:A
[  109.122956] Working Denied:C
[  109.174259] Working: B
[  109.177867] Working Denied:A
[  109.223059] Working Denied:C
[  109.275258] Working: B
[  109.279129] Working Denied:A
[  109.323860] Working Denied:C
[  109.375389] Working: B
[  109.380547] Working Denied:A
[  109.424473] Working Denied:C
[  109.481092] Process Finished: B
[  109.484021] Working: A
[  109.524663] Working Denied:C
[  109.585260] Working: A
[  109.625920] Working Denied:C
[  109.689361] Working: A
[  109.729782] Working Denied:C
[  109.790194] Working: A
[  109.831139] Working Denied:C
[  109.890286] Working: A
[  109.933868] Working Denied:C
[  109.990595] Working: A
[  110.034394] Working Denied:C
```

## Result





**First-Come, First-Served (FCFS)**

- **Advantages**: Easy to implement and understand. Ensures that processes are handled in the order they arrive.
- **Disadvantages**: Can lead to the "convoy effect," where short processes wait for long processes to complete, resulting in poor average turnaround time.

FCFS ensures that all jobs are processed without the overhead of complex scheduling decisions, so it is proper to use if the focus is on the completion of all processes.

**Shortest Remaining Time First (SRTF)**

- **Advantages**: Minimizes average turnaround time and waiting time.
- **Disadvantages**: Difficult to predict the length of the next CPU burst. Can cause starvation of longer processes if short processes keep arriving.

As the results show, If the goal is to minimize the average wait time, SRTF is deal. It shows very low response time. But it varies on processes burst time and order.

**Round Robin (RR)**

- **Advantages**: Ensures all processes get a fair share of the CPU. Reduces starvation and improves response time for interactive users.
- **Disadvantages**: Context switching overhead can be high. Performance depends on the length of the time quantum.

As the results show, the Round Robin algorithm demonstrates low response times. Therefore, it is suitable for situations where user responsiveness is important.

**Priority Scheduling**

- **Advantages**: Ensures that important processes are handled first. Can be used for real-time scheduling.
- **Disadvantages**: Can cause starvation if low-priority processes never get executed. Aging can be used to gradually increase the priority of waiting processes to prevent starvation.

It is suitable for situations where the priorities between processes are clearly defined, and tasks need to be completed in order of their priority rather than ensuring fair execution among processes.

## Problems encountered during the project and your solutions to them

After completing the code for the assignment and compiling the kernel to register the system call, I encountered an error indicating that variable names such as queue, front, rear, and i were conflicting. Consequently, the compilation failed. Although it is not reflected in the attached code, I was able to compile successfully by renaming all conflicting variables to queue1, front1, and so on.