운영체제(MS) - Project 1
Scheduling Simulation

목차

32162548 안정현
32191818 박주은

Left Freeday: 3

1.개요

사람들은 과제를 하거나 일을 할 때 어떤 일을 먼저 할지, 그리고 언제 할지 등에 대한 계획을 세운다. 컴퓨터도 사람처럼 여러 작업이 있을 때 어떤 작업을 먼저 할지, 어떤 작업에 시간을 얼마나 투자할지를 결정하기 위해 스케줄링이라는 기법을 사용합니다. 스케줄링 방법으로는 Round-Robin, FIFO(First In First Out), SJF(Shortest Job First) 등이 있습니다.

Project 1의 목표는 Round-Robin, FIFO, SJF 스케줄링의 원리와 장단점을 분석하고 스케줄링을 시뮬레이션하는 프로그램을 개발하여, run-queue, wait-queue를 출력하는 것입니다.

2. 개념

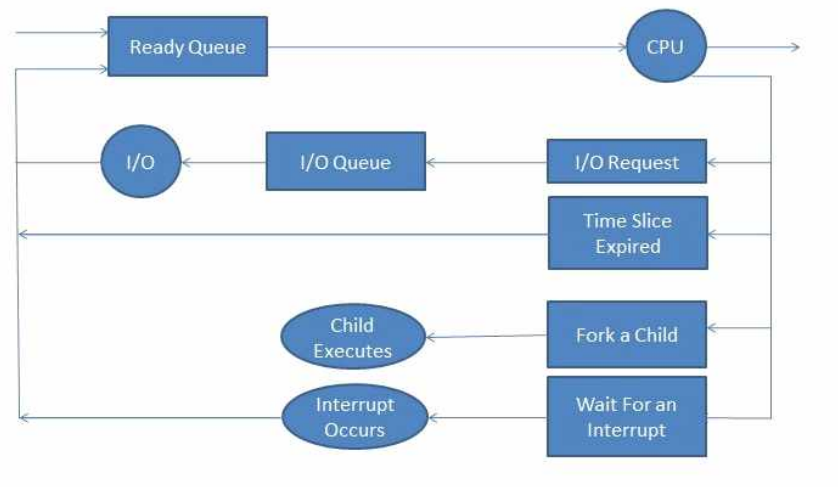
2.1 스케줄링

CPU 스케줄링은 프로세스가 작업을 수행할 때, 언제 어떤 프로세스에 CPU를 할당할지를 결정하는 작업입니다.

2.2 스케줄러 종류

- 장기 스케줄러(Long-term scheduler): ready-queue로 가져올 프로세스를 결정합니다.
- 단기 스케줄러(Short-term scheduler): 다음에 실행할 프로세스를 결정하며 CPU를 할당합니다. 시스템에 따라, 시스템 내의 유일한 스케줄러일 수도 있습니다.

이번 프로젝트에서 중요한 스케줄러는 단기 스케줄러(CPU 스케줄러)입니다.



프로세스 스케줄링

2.3 Round-Robin

- 각 프로세스는 특정 시간할당량(->quantum) 동안만 실행시간을 부여받습니다.
- quantum 안에 작업이 끝나지 않으면 다음 프로세스로 넘어갑니다.
- 평균대기시간이 FIFO와 SJF보다 짧습니다
- quantum의 길이에 따라 성능이 변하며, quantum이 너무 길면 FIFO가 되고, 너무 짧으면 Context Switching을 자주하여 효율성이 떨어집니다.

Process	Arrival Time	Execute Time
P0	0	5
P1	1	3
P2	2	8
P3	3	6

Process	Wait Time : Service Time - Arrival Time
P0	(0-0) + (12-3) = 9
P1	(3-1) = 2
P2	6-2) + (15-9) = 10
P3	(9-3) + (18-12) = 12

Round-Robin 예시

평균 대기 시간: (9+2+10+12) / 4 = 8.25

2.4 FIFO(First In First Out)

- ready에 들어온 순서대로 프로세스를 실행합니다.
- 구현이 편하고 이해하기 쉽습니다.
- 평균대기시간이 높기 때문에 성능이 좋지 않습니다.

Process	Burst Time
Process 1	24
Process 2	3
Process 3	3

FIFO 예시

프로세스별 대기시간: P1 = 0, P2 = 24, P3 = 27

평균 대기시간: (0 + 24 + 27) / 3 = 17

2.5 SJF(Shortest Job First)

- 프로세스들의 실행시간을 비교하여 실행시간이 짧은 순으로 실행하는 알고리즘입니다.
- 대기시간 최소화 측면에서는 최고의 방식입니다.
- 실행시간이 긴 작업은 순서가 뒤로 밀려 기아가 발생합니다.
- 짧은 프로세스가 우선으로 실행되기 때문에 공정하지 않습니다.
- 선점, 비선점이 선택 가능합니다.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	3
P2	2	8	8
P3	3	6	16

Process	Wait Time : Service Time - Arrival Time
P0	3 - 0 = 3
P1	0 - 0 = 0
P2	16 - 2 = 14
P3	8 - 3 = 5

SJF 예시

평균대기시간: (3 + 0 + 14 + 5) / 4 = 5.5

3. 구현 및 코드

3.1 Round-Robin

tick이 지날 때마다 parent는 시그널(SIGALRM)을 받게 되며 시그널에 따라 시그널 handler가 호출됩니다. Handler가 실행되면 가장 빠른 순서를 부여받은 프로세스에 quantum 만큼 프로세스가 실행됩니다. quantum 동안에 실행이 모두 끝나면 종료되고, 끝나지 않으면, 큐의 맨 뒤로 들어가 다시 대기합니다. 모든 프로세스가 종료될때까지 이를 반복하여 child들이 모두 실행완료되면 parent 도 종료됩니다.

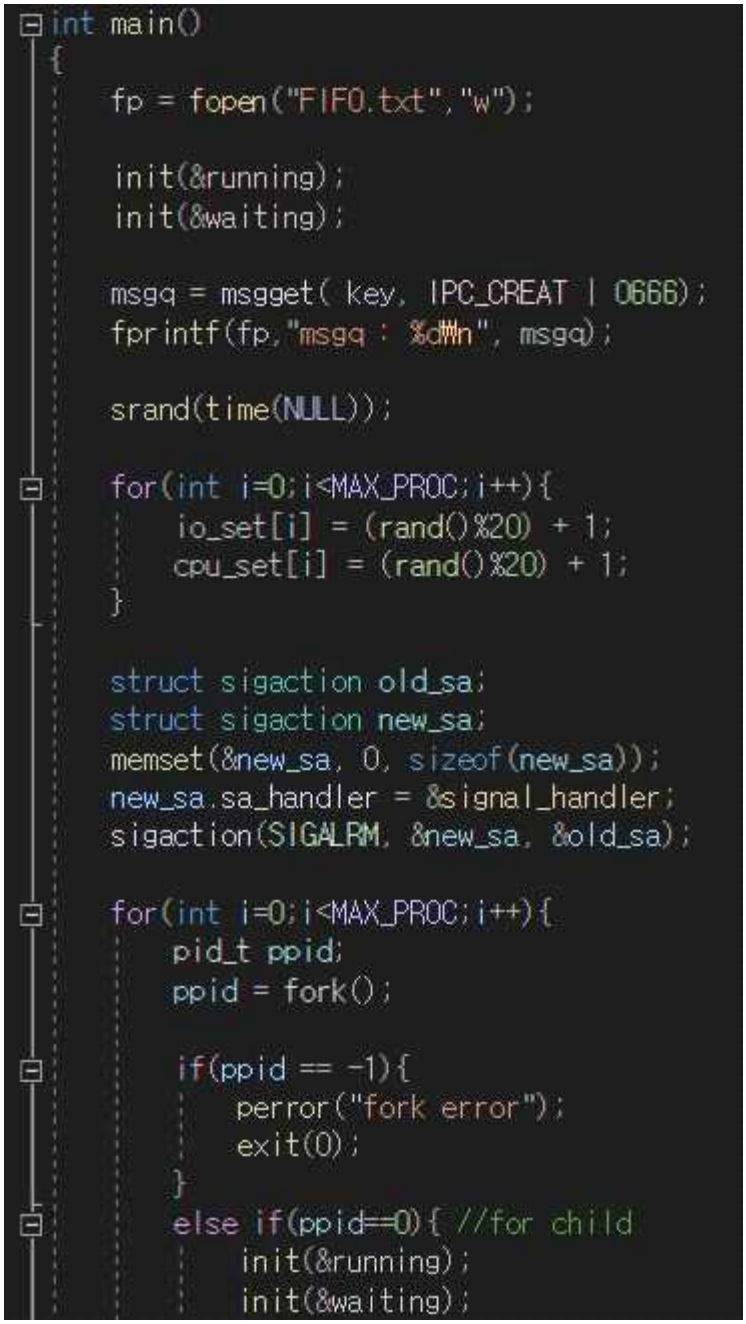
```
int main(){
    fp = fopen("RR.txt","w");

    init(&running);
    init(&waiting);

    srand(time(NULL));
    msgq = msgget( key, IPC_CREAT | 0666);
        fprintf(fp,"msgq : %d₩n", msgq);

    for(int i=0;i<MAX_PROC;i++){
        io_set[i] = (rand()%20) + 1;
        cpu_set[i] = (rand()%20) + 1;
    }
    struct sigaction old_sa;
    struct sigaction new_sa;
    memset(&new_sa, 0, sizeof(new_sa));
    new_sa.sa_handler = &signal_handler;
    sigaction(SIGALRM, &new_sa, &old_sa);

    for(int i=0;i<MAX_PROC;i++){
        pid_t ppid;
        ppid = fork();

        if(ppid == -1){
            perror("fork error");
            exit(0);
        }
        else if(ppid==0){ //for child
            init(&running);
            init(&waiting);

            user_proc[i].cpu_burst = cpu_set[i];
            user_proc[i].io_burst = io_set[i];
            user_proc[i].pid = getpid();
```

```
void do_child(int signo){
    int set_order = -1;
    for(int i=0; i < MAX_PROC; i++){
        if(user_proc[i].pid == getpid()){
            set_order=i;
            break;
        }
    }

    user_proc[set_order].cpu_burst--;

    if(user_proc[set_order].cpu_burst==0){
        user_proc[set_order].cpu_burst=cpu_set[set_order];
        msg.mtype = 1;
        msg.order = set_order;
        msg.pid = getpid();
        msg.io_time = user_proc[set_order].io_burst;

        int ret = msgsnd(msgq, &msg, sizeof(msgbuf), 0);

        if(ret==-1){
            perror("msgsnd error!₩n");
        }
    }
}
```

```
void signal_handler(int signo){
    count++;
    running_tq();
    if(count == 10000){
        for(int i =0; i< MAX_PROC ; i++){
            kill(pcb[i].pid, SIGINT);
        }
        exit(0);
    }
    return ;
}
```

3.2 FIFO

tick이 지날때마다 parent가 시그널(SIGALRM)을 받게 되며 시그널에 따라 시그널 handler가 호출됩니다. handler가 실행되면 가장 먼저 들어온 프로세스가 실행됩니다. child가 시간을 계속하여 확인하며, 실행이 완료되였으면 종료시킵니다. 마지막에 들어온 프로세스가 실행완료 될 때까지 이를 반복하며, 그 이후 parent도 종료됩니다.



```
int main()
{
    fp = fopen("FIFO.txt","w");

    init(&running);
    init(&waiting);

    msgq = msgget( key, IPC_CREAT | 0666);
    fprintf(fp,"msgq : %d₩n", msgq);

    srand(time(NULL));

    for(int i=0;i<MAX_PROC;i++){
        io_set[i] = (rand()%20) + 1;
        cpu_set[i] = (rand()%20) + 1;
    }

    struct sigaction old_sa;
    struct sigaction new_sa;
    memset(&new_sa, 0, sizeof(new_sa));
    new_sa.sa_handler = &signal_handler;
    sigaction(SIGALRM, &new_sa, &old_sa);

    for(int i=0;i<MAX_PROC;i++){
        pid_t ppid;
        ppid = fork();

        if(ppid == -1){
            perror("fork error");
            exit(0);
        }
        else if(ppid==0){ //for child
            init(&running);
            init(&waiting);
```

```c
void do_child(int signo){
    int set_order=-1;

    for(int i=0; i<MAX_PROC; i++){
        if(user_proc[i].pid == getpid()){
            set_order=i;
            break;
        }
    }

    user_proc[set_order].cpu_burst--;

    if(user_proc[set_order].cpu_burst==0){
        user_proc[set_order].cpu_burst=cpu_set[set_order];
        msg.mtype = 1;
        msg.order = set_order;
        msg.pid = getpid();
        msg.io_time = user_proc[set_order].io_burst;

        int key = 32161620;
        msgq = msgget( key, IPC_CREAT | 0666);
        int ret = msgsnd(msgq, &msg, sizeof(msgbuf), 0);

        if(ret==-1){
            perror("msgsnd error ");
        }
    }
}
```

```c
void signal_handler(int signo){
    count++;
    running_fifo();

    if(count==10000){
        for(int i =0; i<MAX_PROC ; i++){
            kill(pcb[i].pid, SIGINT);
        }
        exit(0);
    }

    return ;
}
```

3.3 SJF

tick이 지날 때 마다 parent는 시그널(SIGALRM)을 받고 시그널에 따라 시그널 handler가 호출됩니다. handler가 호출되면 프로세스의 실행시간을 확인하여 가장 짧은 프로세스를 실행합니다. 해당 프로세스의 실행시간만큼 실행을 완료하면 ready에서 나갑니다. 실행하는 동안에 child가 시간을 확인하며, 실행이 끝나면 종료시킵니다. 실행시간이 가장 긴 프로세스가 실행이 완료될 때까지 이를 반복하여, 이 이후 parent도 종료됩니다.

```
int main()
{
    fp = fopen("SJF.txt","w");

    init(&running);

    srand((unsigned)time(NULL));
    for(int k=0; k<MAX_PROC; k++)
    {
        set[k] = (rand() % 10) + 1; // get the set randomly
    }

    struct sigaction old_sa;
    struct sigaction new_sa;
    memset(&new_sa, 0, sizeof(new_sa));
    new_sa.sa_handler = &signal_handler;
    sigaction(SIGALRM, &new_sa, &old_sa);

    //  srand((unsigned)time(NULL));
    srand(time(NULL));
    sort_t sort_arr[MAX_PROC];


    for(int i=0;i<MAX_PROC;i++){
        pid_t ppid;
        ppid = fork();

        if (ppid == -1) {
            perror("fork error");
            exit(0);
        }
        else if(ppid==0){
            user_proc[i].cpu_burst=set[i];
            user_proc[i].pid = getpid();
            fprintf(fp,"₩ncpu_burst[%d]₩tset[%d]₩n", user_proc[i].cpu_burst,set[i]);
            fprintf(fp,"child proc %d ₩n₩n", getpid());

            new_sa.sa_handler = &do_child;
            sigaction(SIGUSR1, &new_sa, &old_sa);
            while(1);
        }
        else{
```

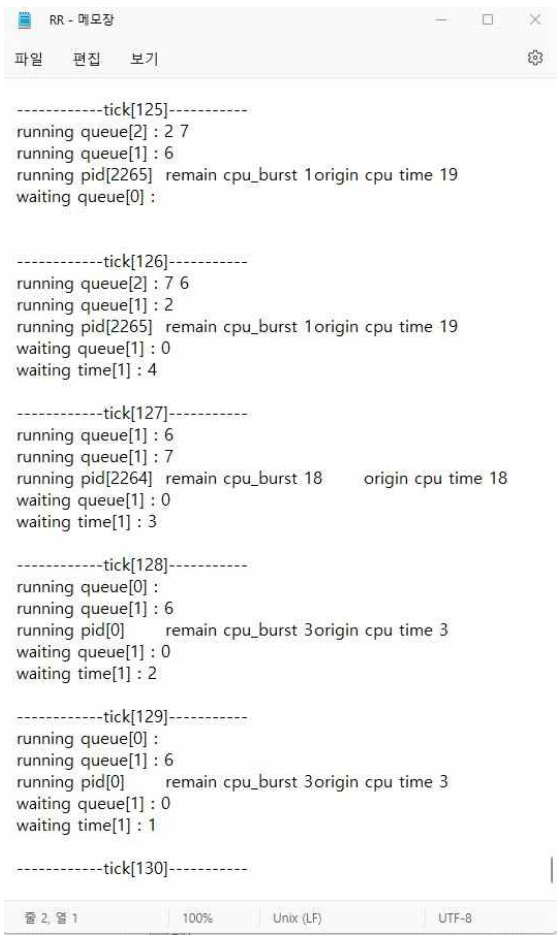
```c
void do_child(int signo){
    int set_order=-1;

    for(int i=0; i<MAX_PROC; i++){
        if(user_proc[i].pid == getpid()){
            set_order=i;
            break;
        }
    }

    user_proc[set_order].cpu_burst--;
    fprintf(fp,"proc[%d] remain_cpu : %d ₩n",set_order,user_proc[set_order].cpu_burst);

    if(user_proc[set_order].cpu_burst==0){
        exit(0);
    }
}
```

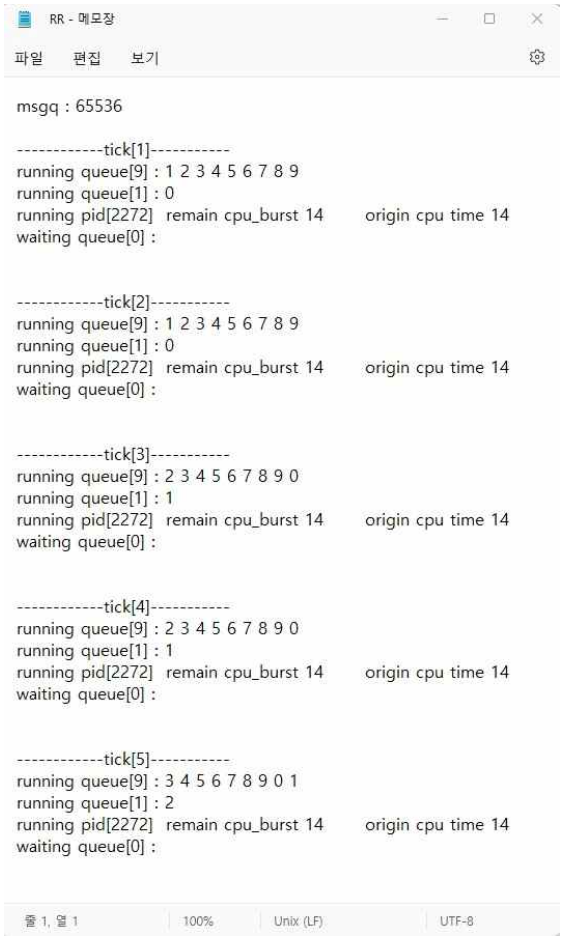
```c
void signal_handler(int signo){
    count++;
    running_fifo();
    return ;
}
```

```c
void running_fifo(){
    if(next==1){

        order = dequeue(&running);
        if(order==-1){
            float total=0;
            fprintf(fp,"₩n----------waiting time----------₩n");
            for(int i=0;i<MAX_PROC;i++){
                fprintf(fp,"prod[%d] : %d ₩n",i,waiting_time[i]);
                total = waiting_time[i] + total;

            }
            fprintf(fp,"₩n");

            fprintf(fp,"₩nAverage Waiting Time : %f ₩n",total/MAX_PROC);

            exit(0);
        }
        next=0;
    }
    fprintf(fp,"₩n----------tick[%d]----------₩n",count);
    int num_running = counting(&running);
    int run_arr[num_running];

    fprintf(fp,"ready queue[%d] : ",num_running);
    sorting(&running, run_arr);
    for(int k=0; k<num_running; k++){
        fprintf(fp,"%d ",run_arr[k]);
    }
    fprintf(fp,"₩n");


    fprintf(fp,"running queue[1] : %d ₩n", order);
```

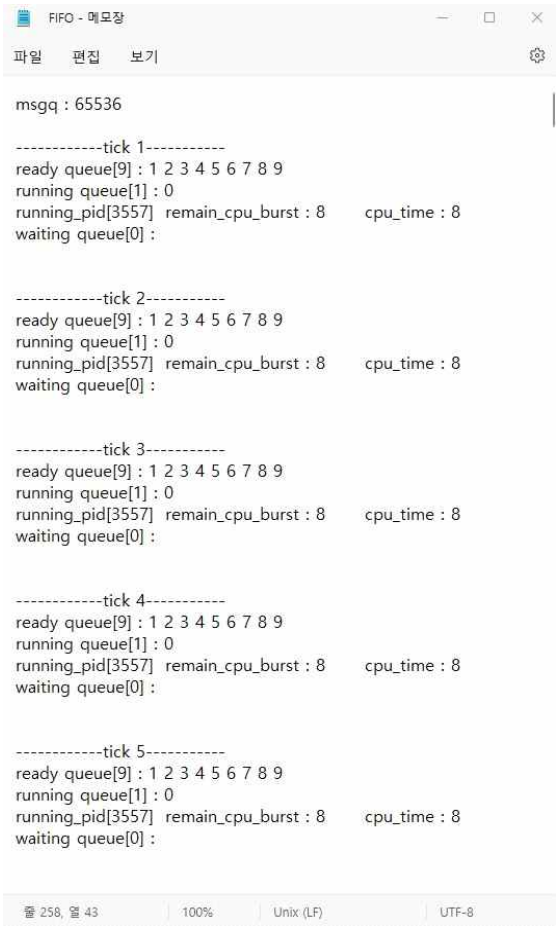
4. 결과

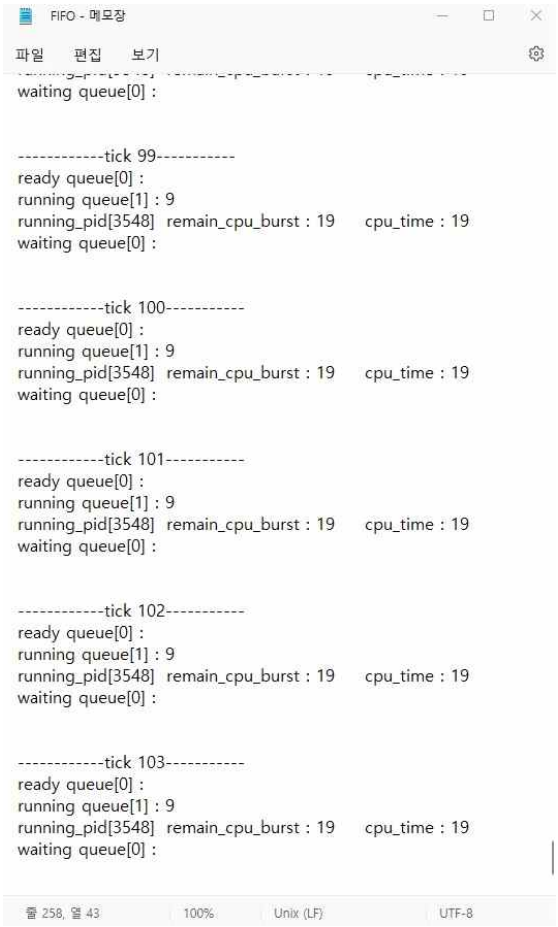
RR.txt


RR - 메모장
파일 편집 보기
----------tick[125]----------
running queue[2] : 2 7
running queue[1] : 6
running pid[2265] remain cpu_burst 1origin cpu time 19
waiting queue[0] :
----------tick[126]----------
running queue[2] : 7 6
running queue[1] : 2
running pid[2265] remain cpu_burst 1origin cpu time 19
waiting queue[1] : 0
waiting time[1] : 4
----------tick[127]----------
running queue[1] : 6
running queue[1] : 7
running pid[2264] remain cpu_burst 18 origin cpu time 18
waiting queue[1] : 0
waiting time[1] : 3
----------tick[128]----------
running queue[0] :
running queue[1] : 6
running pid[0] remain cpu_burst 3origin cpu time 3
waiting queue[1] : 0
waiting time[1] : 2
----------tick[129]----------
running queue[0] :
running queue[1] : 6
running pid[0] remain cpu_burst 3origin cpu time 3
waiting queue[1] : 0
waiting time[1] : 1
----------tick[130]----------
줄 2, 열 1
100%
Unix (LF)
UTF-8


RR - 메모장
파일 편집 보기
msgq : 65536
----------tick[1]----------
running queue[9] : 1 2 3 4 5 6 7 8 9
running queue[1] : 0
running pid[2272] remain cpu_burst 14 origin cpu time 14
waiting queue[0] :
----------tick[2]----------
running queue[9] : 1 2 3 4 5 6 7 8 9
running queue[1] : 0
running pid[2272] remain cpu_burst 14 origin cpu time 14
waiting queue[0] :
----------tick[3]----------
running queue[9] : 2 3 4 5 6 7 8 9 0
running queue[1] : 1
running pid[2272] remain cpu_burst 14 origin cpu time 14
waiting queue[0] :
----------tick[4]----------
running queue[9] : 2 3 4 5 6 7 8 9 0
running queue[1] : 1
running pid[2272] remain cpu_burst 14 origin cpu time 14
waiting queue[0] :
----------tick[5]----------
running queue[9] : 3 4 5 6 7 8 9 0 1
running queue[1] : 2
running pid[2272] remain cpu_burst 14 origin cpu time 14
waiting queue[0] :
줄 1, 열 1
100%
Unix (LF)
UTF-8

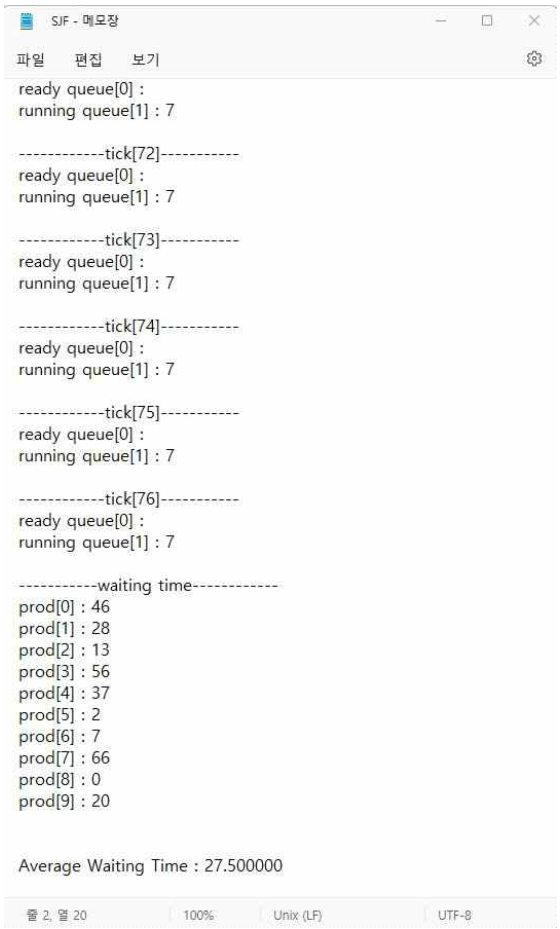
FIFO.txt


FIFO - 메모장
파일 편집 보기
msgq : 65536
----------tick 1----------
ready queue[9] : 1 2 3 4 5 6 7 8 9
running queue[1] : 0
running_pid[3557] remain_cpu_burst : 8 cpu_time : 8
waiting queue[0] :
----------tick 2----------
ready queue[9] : 1 2 3 4 5 6 7 8 9
running queue[1] : 0
running_pid[3557] remain_cpu_burst : 8 cpu_time : 8
waiting queue[0] :
----------tick 3----------
ready queue[9] : 1 2 3 4 5 6 7 8 9
running queue[1] : 0
running_pid[3557] remain_cpu_burst : 8 cpu_time : 8
waiting queue[0] :
----------tick 4----------
ready queue[9] : 1 2 3 4 5 6 7 8 9
running queue[1] : 0
running_pid[3557] remain_cpu_burst : 8 cpu_time : 8
waiting queue[0] :
----------tick 5----------
ready queue[9] : 1 2 3 4 5 6 7 8 9
running queue[1] : 0
running_pid[3557] remain_cpu_burst : 8 cpu_time : 8
waiting queue[0] :
줄 258, 열 43
100%
Unix (LF)
UTF-8


FIFO - 메모장
파일 편집 보기
waiting queue[0] :
----------tick 99----------
ready queue[0] :
running queue[1] : 9
running_pid[3548] remain_cpu_burst : 19 cpu_time : 19
waiting queue[0] :
----------tick 100----------
ready queue[0] :
running queue[1] : 9
running_pid[3548] remain_cpu_burst : 19 cpu_time : 19
waiting queue[0] :
----------tick 101----------
ready queue[0] :
running queue[1] : 9
running_pid[3548] remain_cpu_burst : 19 cpu_time : 19
waiting queue[0] :
----------tick 102----------
ready queue[0] :
running queue[1] : 9
running_pid[3548] remain_cpu_burst : 19 cpu_time : 19
waiting queue[0] :
----------tick 103----------
ready queue[0] :
running queue[1] : 9
running_pid[3548] remain_cpu_burst : 19 cpu_time : 19
waiting queue[0] :
줄 258, 열 43
100%
Unix (LF)
UTF-8

SJF.txt


SJF - 메모장
파일 편집 보기
cpu_burst[2] set[2]
child proc 3362
proc[8] remain_cpu : 1
proc[8] remain_cpu : 0
----------tick[1]----------
ready queue[9] : 5 6 2 9 1 4 0 3 7
running queue[1] : 8
----------tick[2]----------
ready queue[9] : 5 6 2 9 1 4 0 3 7
running queue[1] : 8
----------tick[3]----------
ready queue[8] : 6 2 9 1 4 0 3 7
running queue[1] : 5
----------tick[4]----------
ready queue[8] : 6 2 9 1 4 0 3 7
running queue[1] : 5
----------tick[5]----------
ready queue[8] : 6 2 9 1 4 0 3 7
running queue[1] : 5
----------tick[6]----------
ready queue[8] : 6 2 9 1 4 0 3 7
running queue[1] : 5
----------tick[7]----------
ready queue[8] : 6 2 9 1 4 0 3 7
running queue[1] : 5
----------tick[8]----------
ready queue[7] : 2 9 1 4 0 3 7
줄 2, 열 20
100%
Unix (LF)
UTF-8


SJF - 메모장
파일 편집 보기
ready queue[0] :
running queue[1] : 7
----------tick[72]----------
ready queue[0] :
running queue[1] : 7
----------tick[73]----------
ready queue[0] :
running queue[1] : 7
----------tick[74]----------
ready queue[0] :
running queue[1] : 7
----------tick[75]----------
ready queue[0] :
running queue[1] : 7
----------tick[76]----------
ready queue[0] :
running queue[1] : 7
----------waiting time----------
prod[0] : 46
prod[1] : 28
prod[2] : 13
prod[3] : 56
prod[4] : 37
prod[5] : 2
prod[6] : 7
prod[7] : 66
prod[8] : 0
prod[9] : 20
Average Waiting Time : 27.500000
줄 2, 열 20
100%
Unix (LF)
UTF-8

5. 느낀 점

이번 과제, 프로젝트1에서는 다중 프로세스를 이용한 스케줄링 시뮬레이션을 해보았습니다. 프로젝트의 세부사항들이 특정되어있지 않고 원하는 대로 자유롭게 개발하는 과제였는데, 이 점이 구현방법에 대해 많이 고민하게 만든 것 같습니다.
그리고, 이번이 그룹 프로젝트가 처음이었는데, 다음 프로젝트를 진행할 때는 역할분담을 더 잘하여, 더 편하고 서로 이해하기 쉽게 개발해보면 좋을 것 같습니다.