2023 OS Term Project

권민선(2021310459)

1. Shell scripts (commands) to build and execute the source code.

```
mskwon0601@DESKTOP-K0KCSVT:/mnt/d/대학/3-2/운영체제/osTermProject$ g++ -o rr rr.cpp
mskwon0601@DESKTOP-K0KCSVT:/mnt/d/대학/3-2/운영체제/osTermProject$ ./rr >> schedule_dump.txt
```

2. 요약 테이블

Whether the code implementation is complete	O
The setting of time quantum (which executed without errors)	300ms
Whether IPC message queue is used	O
How the i/o operation is implemented (one of four options in section 3)	Option 2
Whether the multi-level queue was implemented (extra score in section 3)	X

3. 설정한 파라미터 정보

Time tick	300ms
Time quantum	5(변경)
IO 발생 확률	0.6
CPU Burst 범위	1~50
IO Burst 범위	1~9

4. 타임퀀텀 변화에 따른 average response time과 total execution time 변화

각 시도별 로그기록을 **rr_설정한타임퀀텀_시도번호.txt**에 남겨 제출물zip 파일 안 staticLog폴더 안에 넣었습니다.

A. 타임퀀텀 3일 때 average response time과 total execution time

타임퀀텀 : 3		
시도번호	average response time(Tick)	total execution time(Tick)
1	8.9	212
2	10.7	354
3	10	199
4	9.4	218
5	7.9	308
평균	9.38	258.2

B. 타임퀀텀 5일 때 average response time과 total execution time

타임퀀텀 : 5		
시도번호	average response time(Tick)	total execution time(Tick)
1	12.3	214
2	13.6	223
3	12	350
4	12	245
5	8.2	258
평균	11.62	258

C. 타임퀀텀 10일 때 average response time과 total execution time

타임퀀텀 : 10		
시도번호	average response time(Tick)	total execution time(Tick)
1	31.4	260
2	22.6	278
3	16.8	261
4	23.4	203
5	18.7	201
평균	22.58	240.6

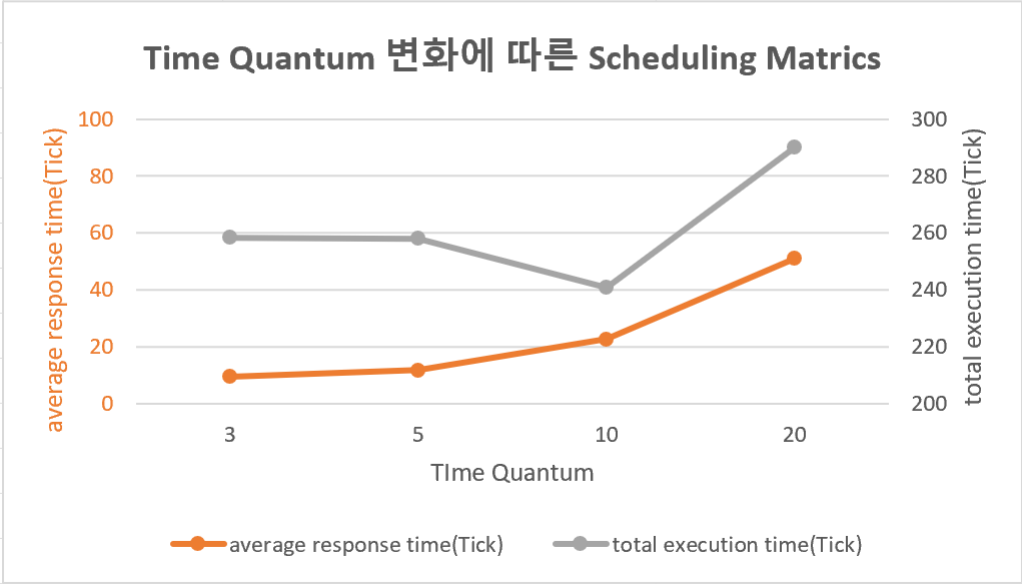
D. 타임퀀텀 20일 때 average response time과 total execution time

타임퀀텀 : 20

시도번호	average response time(Tick)	total execution time(Tick)
1	45	326
2	59.7	257
3	48.5	309
4	53.6	295
5	48.5	264
평균	51.06	290.2

E. 타임퀀텀 변화에 따른 average response time과 total execution time의 평균 비교

Time Quantum	average response time(Tick)	total execution time(Tick)
3	9.38	258.2
5	11.62	258
10	22.58	240.6
20	51.06	290.2



위 실험을 통해 **Time Quantum이 커질수록 average response time이 크다**는 것을 확인할 수 있다. 또한 total execution time은 Time Quantum 값과 상관관계가 없다는 것을 알 수 있었다.

5. 코드설명 및 실행 캡쳐

A. Pcb 구조체 및 message 구조체

```
//프로세스 구조체
struct Pcb {
    long pid;
    int remainCpuBurst;
    bool doIo=false;
    int ioStartTick=-1;
    int remainIoBurst=-1;
    bool firstRun=true;

}
;
```

- 프로세스 번호인 pid
- 남은 cpu burst time
- Io실행 여부인 doIo
- Io 시작 틱
- 남은 Io burst time
- 프로세스가 한 번도 실행되지 않았다는 정보 담는 firstRun

```
// ipc메시지 구조체
struct Msg {
    //누구한테 보내는 메시지인지 구분할 수 있는 변수
    long type;
    //내용물
    int content;
}
;
```

B. 구조체 ProcessTimeInfo 및 맵 객체 ProcessTimeInfo

```
struct ProcessTimeInfo{

  //프로세스가 처음으로 ready큐에 들어온 시간(틱)
  int arrivalTick=0;
  //프로세스가 처음으로 run된 시간(틱)
  int firstRunTick;
  //프로세스가 완료될때까지(cpu burst 0될때까지) 걸린 시간(틱)
  int CompleteTick;
};

//스케쥴링 성능(scheduling metric) 확인할 정보 담는 map객체
map<int, ProcessTimeInfo> schedulingMetricMap;
```

- ProcessTimeInfo: Scheduling matric 구하기 위한 정보 담는 구조체

- schedulingMetricMap: key값이 프로세스의 pid이고 value가 ProcessTimeInfo 정보인 맵 객체

C. main함수 흐름

i. msgget()함수를 통해 IPC 메시지 큐 생성한다.

ii. memset()함수를 통해 타임 틱 설정하고 signal(SIGALRM, perTimeTick); 을 통해 설정한 타임 틱마다 사용자 정의 함수인 perTimeTIck함수 호출한다.

iii. 자식 프로세스 10개 생성 반복문

1. 부모 프로세스에서는 pcb 포인터 객체를 생성하고 자식 프로세스의 pid정보와 랜덤으로 생성한 remainCpuBurst시간을 객체에 넣은 다음 pcb포인터 객체를 readyQueue에 삽입

2. 자식 프로세스에서는 부모 프로세스에서 프로세스 상태 정보를 담은 IPC 메시지가 올 때까지 대기하다가 전달받은 내 프로세스 상태가 START면 io 작업 여부를 결정하고 io작업 여부가 결정되면 부모프로세스에게 content에 1을 담은 IPC메시지를 보낸다. 이후 ioStartTick과 remainIoBurst을 랜덤으로 생성해 다시 부모프로세스에게 보낸다. 이때 메시지 content의 자료형을 int로 설정했기 때문에 ioStartTick*10+remainIoBurst를 메시지의 content로 설정한다. 이렇게 되면 부모 프로세스에서는 전달받은 메시지의 content를 10으로 나눴을 때 몫은 ioStartTick 정보이고 나머지는 remainIoBurst 정보이다.

   만약 전달받은 내 프로세스 상태가 TERMINATED면 exit(0)으로 프로세스를 종료한다.

iv. wait(NULL)을 반복문으로 돌며 자식프로세스 10개가 모두 종료될 때까지 기다린다.

v. 이후 schedulingMetricMap을 반복문으로 돌며 자식 프로세스의 정보(pid, arrivalTick, firstRunTIck, CompleteTick)을 출력하며 Response Time(firstRunTick-arrivalTick)과 TurnaroundTIme(completeTick-arrivalTick)을 계산해 출력한다

```
cout<<"자식프로세스 모두 종료"<<endl<<endl;
cout<<"========================================"<<endl;
cout<<"Scheduling Metrics Info"<<endl;

for(auto info:schedulingMetricMap){
    cout<<"----------------------------------------"<<endl;
    cout<<"#PID "<<info.first<<"의 Scheduling Metrics Info"<<endl;
    cout<<"arrivalTick:"<<info.second.arrivalTick<<endl;
    cout<<"firstRunTick:"<<info.second.firstRunTick<<endl;
    cout<<"CompleteTick:"<<info.second.CompleteTick<<endl;
    responseTime=info.second.firstRunTick-info.second.arrivalTick;
    cout<<">> Response Time: "<<info.second.firstRunTick-info.second.arrivalTick<<endl;
    avgResponseTime+=responseTime;
    cout<<">> Turnaround Time:"<<info.second.CompleteTick-info.second.arrivalTick<<endl;

}
cout<<"----------------------------------------"<<endl;
cout<<endl<<">> totalTimeTick: "<<timeTickPassed-1<<endl;
cout<<">> avgResponseTime: "<<avgResponseTime/CHILDPROCESSNUM<<endl;
return 0;
```

```
자식프로세스 모두 종료

========================================
Scheduling Metrics Info
----------------------------------------
#PID 18172의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:0
CompleteTick:142
>> Response Time: 0
>> Turnaround Time:142
----------------------------------------
#PID 18173의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:3
CompleteTick:210
>> Response Time: 3
>> Turnaround Time:210
----------------------------------------
#PID 18174의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:8
CompleteTick:163
>> Response Time: 8
>> Turnaround Time:163
----------------------------------------
#PID 18175의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:10
CompleteTick:125
>> Response Time: 10
>> Turnaround Time:125
```

```
----------------------------------------
#PID 18176의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:11
CompleteTick:223
>> Response Time: 11
>> Turnaround Time:223
----------------------------------------
#PID 18177의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:14
CompleteTick:133
>> Response Time: 14
>> Turnaround Time:133
----------------------------------------
#PID 18178의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:16
CompleteTick:132
>> Response Time: 16
>> Turnaround Time:132
----------------------------------------
#PID 18179의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:21
CompleteTick:206
>> Response Time: 21
>> Turnaround Time:206
----------------------------------------
#PID 18180의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:26
CompleteTick:90
>> Response Time: 26
>> Turnaround Time:90
----------------------------------------
```

```
----------------------------------------
#PID 18181의 Scheduling Metrics Info
arrivalTick:0
firstRunTick:27
CompleteTick:126
>> Response Time: 27
>> Turnaround Time:126
----------------------------------------

>> totalTimeTick: 223
>> avgResponseTime: 13.6
```

D. 타임 틱마다 실행되는 함수인 perTImeTIck()함수

i. io큐에 있는 프로세스들의 remainIoBurst를 하나씩 감소시킨다

```
queue<Pcb*> tempIoQueue;

while (!ioQueue.empty()) {
    Pcb* currentIoPcb = ioQueue.front();
    currentIoPcb->remainIoBurst--;
    //remainIoBurst0이면 ready큐로 들어간다
    if(currentIoPcb->remainIoBurst<=0){
        currentIoPcb->remainIoBurst = -1;
        currentIoPcb->ioStartTick = -1;
        readyQueue.push(currentIoPcb);
    }else{
        tempIoQueue.push(currentIoPcb);
    }
    ioQueue.pop();
}
ioQueue = tempIoQueue;
```

ii. 현재 실행중인 pcb가 없으면 dispatch()함수로 새로운 pcb를 가져온다

iii. 현재 실행중인 pcb가 있다면 현재 pcb의 cpu burst time을 감소시킨다. 만약 cpu burst가 0이 되면 해당 pcb의 pid를 pid로 가지는 자식 프로세스에게 TERMINATED라는 내용을 담은 IPC 메시지를 전달해 이를 받은 자식 프로세스가 exit()되도록 하고 schedulingMetricMap에 해당 프로세스의 CompleteTick을 현재 timeTIckPassed로 설정해 종료 틱 시간을 기록한다.

만약 현재 running중인 프로세스가 타임슬라이스 만큼 running을 했다면 contextSwitch()를 통해 다음 프로세스로 contextSwitch가 일어나도록 한다.

만약 현재 running중인 프로세스가 IO를 시작할 시간이 되었다면 현재 running중인 프로세스의 pcb포인터를 IO큐에 넣고 dispatch()함수를 실행해 ready큐에 있는 새로운 프로세스 pcb를 running 상태로 바꾼다.

```
// 현재 실행 중인 pcb가 없으면 새로운 pcb를 가져온다
if (runningPcb == NULL) {
    dispatch();
} else {
    // 현재 pcb의 CPU 버스트 시간 감소
    runningPcb->remainCpuBurst--;
    runningPcbRunTick++;

    // CPU 버스트가 완료되면, 프로세스 종료 처리
    if (runningPcb!=NULL && runningPcb->remainCpuBurst <= 0) {
        Msg msg;
        msg.type = runningPcb->pid;
        msg.content = TERMINATED;
        msgsnd(msgQueId, &msg, sizeof(Msg), IPC_NOWAIT);
        schedulingMetricMap[runningPcb->pid].CompleteTick=timeTickPassed;
        contextSwitch();
    }
    // 타임 슬라이스 초과 시, 다음 프로세스로 전환
    else if (runningPcb!=NULL && runningPcbRunTick >= TIMESLICE) {
        contextSwitch();
    }
    if(runningPcb!=NULL&&runningPcb->ioStartTick>0){
        runningPcb->ioStartTick--;
        if(runningPcb->ioStartTick==0){//io시작할 시간 되면
            ioQueue.push(runningPcb);//io큐에 들어가라
            runningPcb->doIo=false;
            dispatch();//새로운 프로세스 꺼내라
        }
    }
}
```

iv. 현재 타임 틱의 정보를 log로 출력 후 timeTickPassed를 증가시킨다

```
================10 번째 타임틱 ================
현재 cpu 차지 pid: 18175
remain cpu burst: 11

이 프로세스(18175)는 IO작업이 있는 프로세스
IO작업(IoBurst: 6)이 1틱 뒤에 실행

[ ready Queue ]
pid             CPU burst
-----------------------------
18176           29
18177           12
18178           26
18179           41
18180           10
18181           10
18172           11
18173           31


[ IO Queue ]
pid             CPU burst       IO burst
-----------------------------------------
18174           19              2
```

E. Ready 큐에서 실행할 프로세스를 가져오는 dispatch()함수

```
void dispatch() {
    if(!readyQueue.empty()) {
        //맨 앞의 pcb를 꺼내 현재 실행중인 pcb로 설정
        runningPcb=readyQueue.front();
        readyQueue.pop();
        runningPcbRunTick=0;
        Msg msg;
        msg.type=runningPcb->pid;
        msg.content=START;
        //꺼낸 pcb의 pid와 동일한 자식프로세스에게 너 실행해 라는 메시지 전송
        msgsnd(msgQueId, &msg, sizeof(Msg), 0);

        //만약 새로 꺼낸 pcb가 처음으로 running상태가 된거라면 firstruntime기록
        if(runningPcb->firstRun == true){
            schedulingMetricMap[runningPcb->pid].firstRunTick=timeTickPassed;
            runningPcb->firstRun=false;
        }

        Msg msgFromChild;


        if(msgrcv(msgQueId, &msgFromChild, 50,parentPid,0)>0){

            //자식에게서 온 메시지가 1(io작업 할거야)이면
            if(msgFromChild.content==1){
                runningPcb->doIo=1;
                //cout<<"getMsg!"<<endl;

                Msg ioInfoFromChild;

                msgrcv(msgQueId, &ioInfoFromChild, 50,parentPid,0);

                //자식이 보낸 ioStartTick정보(십으로 나눈 몫) 가져와 pcb에 저장
                runningPcb->ioStartTick=ioInfoFromChild.content/10;
                runningPcb->remainIoBurst=ioInfoFromChild.content%10;

            }else{
                return;
            }
        }
    }else{
    runningPcb=NULL;
    }
}
```

F. contextSwitch()함수

```
void contextSwitch() {
    //현재 실행중인 pcb의 remainCpuBurst가 남아있다면 ready큐의 맨 뒤에 넣어야함
    if(runningPcb!=NULL && runningPcb->remainCpuBurst >0) {
        readyQueue.push(runningPcb);
    }
    dispatch();
}
```