운영체제 과제3

201921166 정의철

1. 주요 코드 설명

가. 구조체 설명

```
typedef struct _Process
{
    size_t remaining_time;
    size_t burst_time;
    size_t arrival_time;
    size_t exit_time;
    size_t pid;
} Process;
```

Process 구조체는 input.txt에서 입력 받은 프로세스의 정보를 담고 있습니다. remaining_time은 남은 burst time을 의미합니다.

나. 주요 함수 설명

1) process.h

가) int is_null_process(Process* process)

process가 null인지 아닌지를 판별합니다. process의 address가 0이거나, remaining_time이 0 일 때 true를, 그 반대일 때 false를 반환합니다.

나) int make_process_null(Process* process)

process의 remaining_time을 0으로 초기화합니다.

2) queue.h

가) void init_queue()

큐의 내용물을 비웁니다.

나) void offer(Process* value)

value프로세스를 큐에 삽입합니다.

다) Process* poll()

프로세스를 큐에서 빼오고 삭제합니다.

라) Process* dispatch_process_by_pid(size_t pid)

해당 pid를 가진 프로세스를 큐에서 빼오고, 삭제합니다.

마) Process** queue_as_array()

큐에 저장되어 있는 프로세스들을 배열의 형태로 변환한 뒤, 주소 값을 반환합니다.

바) size_t get_queue_size()

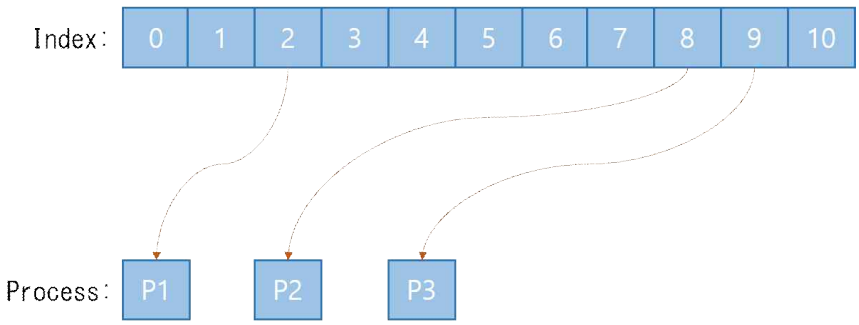
큐에 저장되어 있는 요소의 개수를 반환합니다.

3) timeline.h

가) void init_timeline(Process* process_array, size_t length)

process들이 저장되어 있는 배열과 그 크기를 받고, 이를 map과 유사한 형태로 변환합니다.

Arrival time	pid
2	P1
8	P2
9	p3



나) Process* get_process_in_time(size_t time)

time 밀리 초에 arrival 하게 될 프로세스를 반환합니다. 만약 time 밀리 초에 arrival 하는 프로세스가 없을 경우 NULL을 반환합니다.

다) int is_done(size_t time)

time 밀리 초 이후에 arrival 할 프로세스가 있는지 없는지 판별합니다. 만약 time 밀리 초 이후에 도착할 프로세스가 없다면 true를 반환합니다.

다. Vector Queue

c++ STL의 기능 중 하나인 vector와 queue를 사용하려고 했으나, 배운 적 없는 c++을 쓰는 것은 과제의 범위와 벗어나는 것 같아서 가변길이 배열과 원형 큐를 구현하였습니다.

```c
/* EOL에 도달할 때 까지 계속해서 입력을 받음 */
while ((tmp_result = fscanf(fp, "%d %d", &temp1, &temp2)) != -1) {
    process_list[process_input].remaining_time = temp1;
    process_list[process_input].burst_time = temp1;
    process_list[process_input].pid = process_input;
    process_list[process_input++].arrival_time = temp2;
    // 담을 공간이 부족할 경우 배열 길이 증량
    if (process_input == process_size - 1) {
        process_size += 5;
        process_list = realloc(process_list, sizeof(Process) * process_size);
    }
}
```

input.txt 파일을 open하고, EOL(=-1)에 도달할 때 까지 while문을 반복하여 입력을 받아들이는 로직입니다. 이 때, process input(:읽어 들인 프로세스의 개수)이 process_size(:초기 설정되어 있는 프로세스 배열의 길이)를 넘어설 경우 배열의 크기를 5씩 증가해 나가도록 하였습니다. 따라서 OS가 허용하는 한도 이내로 무한개의 input을 받을 수 있게 하였습니다.

라. CPU 클럭

CPU의 클럭을 표현하기 위해서 for문을 사용하였습니다.

```c
for (size_t current_time = 0; current_time < TIMEMAX; current_time++)
{

}
```

여기서 current_time 변수가 현재의 시간(ms)를 의미할 수 있도록 하였습니다.

마. Waiting time 계산

한 프로세스의 Waiting time은 Turnaround time - Burst time과 같습니다. Turnaround time은 Exit time - Arrival time과 같으므로, Waiting time은 Exit time - Arrival time - Burst time으로 정리할 수 있습니다.

$$WT = TT - BT$$
$$TT = ET - AT$$
$$\therefore WT = ET - AT - BT$$

따라서 Exit time만 구할 수 있으면, Average Waiting time까지 구할 수 있으므로, 시뮬레이션을 통해서 각각의 프로세스 마다 Exit time을 구할 수 있도록 하였습니다.

바. 디버깅 코드

CPU 클럭 마다 프로세스 스케줄링이 어떻게 이루어지는지 직접 볼 수 있으면 재밌

을 것 같다는 생각에 개발하였습니다.

```
buffer_cur += sprintf(debug_buffer+buffer_cur, "STRING");
```

모든 수행 과정을 printf로 출력하면 콘솔창이 난잡해지므로, debug_buffer 배열을 선언하여 원할 때에만 수행과정을 볼 수 있게 개발하였습니다.

만약에 FCFS의 수행과정만 보고 싶다면 DEBUG_MODE 매크로의 값을 0b100 으로 설정하면 되고, RR의 수행과정과 HRN의 수행과정을 보고 싶으면 각각 0b010 0b001로 설정하면 됩니다. 모든 수행과정이 보고 싶다면 0b111로 설정할 수 있습니다.

2. 스케줄링 기법

가. FCFS

First Come First Serviced라는 뜻을 가진 FCFS 스케줄링 알고리즘은 먼저 도착한 프로세스 순서대로 CPU 자원을 이용할 수 있습니다. 이를 구현하기 위해서는 FIFO 형태의 자료구조인 큐가 필요합니다.

Arrival하는 모든 프로세스에 대해서 큐에 집어넣는 연산(push)을 수행하고, running state에 있는 프로세스가 없으면 큐에서 프로세스를 빼오는 연산(pop)을 수행할 수 있어야 합니다. 이를 의사코드로 표현하면 아래와 같습니다.

```
while
{
    if (is_there_any_arrival_process(current_time))
    {
        arrival_process = get_process(current_time)
        push_to_queue(arrival_process)
    }
    if (is_there_any_running_process())
    {
        service_running_process()
    }
    else
    {
        running_process = pop_from_queue()
    }
}
```

나. RR

Round Robin은 한 프로세스가 Time Quantum을 초과하여 실행할 수 없습니다. running process가 몇 밀리 초만큼이나 서비스 되었는가를 파악하기 위해서 timer가 필요합니다. 만약 timer가 Time Quantum에 도달하였으면, running process를 waiting queue에 push하고, 새로운 프로세스를 waiting queue에서 pop하는 연산을 수행하게 됩니다. RR의 기본적인 알고리즘은 FCFS와 같기 때문에 FCFS의 코드에서 timer 변수의 선언과, timer가 Time Quantum에 도달하였는지를 감지하는 조건문만 추가해 주면 됩니다.

다. HRN

HRN 또는 HRRN 스케줄링은 Response Ratio가 가장 높은 프로세스를 먼저 실행하는 방식입니다. 따라서 Highest Response Ratio Next라는 이름이 붙게 되었습니다. Response Ratio = (Waiting time + Burst time) / Burst time 으로 구할 수 있습니다. Response Ratio는 Waiting time과 비례관계에 있기 때문에, Waiting time이 긴 프로세스 일수록 aging되어 기아현상을 막을 수 있습니다.

마찬가지로, 기본적인 베이스는 FCFS와 유사하므로 FCFS의 코드에서 response ratio를 계산하고, 다음 프로세스를 결정하여 waiting queue에서 빼오는 연산을 수행하는 로직만 구현하면 됩니다.

3. 개발 history 및 소감

이번과제에서는 포인터 연산을 상당이 많이 수행해야 했기 때문에 조금 까다로웠던 것 같습니다. 포인터 주소 값에 대해서 잘못된 접근을 수행하면 Segmentation fault가 발생하는데, 몇 번째 라인에서 에러가 발생하였는지를 알 수 없어서 이를 gdb를 이용하여 디버깅하는데 많은 시간이 걸렸습니다. 또한 vector와 queue 기능을 직접 구현해야했기에 운영체제 과제를 통틀어 제일 공을 들였던 것 같습니다. Java로 이번 과제를 수행할 수 있었으면 쉽게 끝낼 수 있겠다는 생각이 들었으나, 다양한 에러를 경험해보고 직접 자료구조를 구현해볼 수 있었다는 점에서 학습에 많은 도움도 되었습니다.

또한 각 스케줄링 알고리즘이 어떻게 세세하게 동작하는지 출력하는 코드를 작성하고 실행해보아, FCFS RR HRN 스케줄링에 관한 문제가 정보처리기사 등의 시험에 나오더라도 꼭 맞출 수 있는 자신이 생겼습니다.