Project analysis

Ⅰ. 프로젝트 분석

스케줄링(Scheduling)은 다중 프로그래밍을 가능하게 하는 운영체제(Operating System)의 동작 기법이다.
운영체제는 process들에게 CPU 등의 자원 배정을 적절히 함으로써 시스템의 성능을 개선할 수 있다.

우리가 이번에 설계해야 할 스케줄링 알고리즘은 다음과 같다.

1. FIFO Scheduling
2. RR Scheduling
3. MLFQ Scheduling
4. Stride Scheduling

1. FIFO Scheduling

FIFO(First In First Out) Scheduling은 가장 기본적인 스케줄링 알고리즘이다. FIFO는 말 그대로 먼저 들어온 것이 먼저 나가는 즉 먼저 들어온 것을 먼저 스케줄링 하는 알고리즘이다.
우리는 FIFO가 들어오는 그대로 실행한다 이 말에 초점을 맞추고 들어오는 순서에 맞춰 바로바로 Service만큼 실행되도록 구현해보았다.

2. RR Scheuling

RR(Round Robin) Scheduling 알고리즘은 기본적으로 FIFO에 time quantom이라는 개념을 추가하여 도착한 순서대로 time quantom만큼씩만 실행해주는 알고리즘이다. 여기서 time quantom을 모두 사용한 process는 queue에서 나와 queue의 맨 뒤로 들어가게 된다. 여기서 우리가 명심해야 할 부분은 새로운 process가 들어오면 그것을 알고 스케줄러가 실행하고 있는 process가 time quantom을 모두 사용하였는지 확인하여 다음 process로 CPU를 넘겨주는 과정을 제대로 구현해야 한다는 것이다. 이 부분이 RR Scheduling의 핵심이라고 볼 수 있다. 이러한 점을 preempt라고 한다.

우리가 RR 설계를 계획 할 때 넣어야 한다고 생각한 것은 기본적으로 FIFO 구조에 조건문을 넣어 time quantom 사용 여부를 체크하면서 새로운 process 가 queue 에 도착했는지 확인을 하는 것을 기본적인 알고리즘이라 가정하고 구현을 시도하기로 했다.

3. MLFQ Scheduling

MLFQ(Multi Level Feedback Queue) Sceduling은 커널 내의 준비 queue를 여러 개의 queue로 분리하여 queue 사이에도 우선순위를 부여하는 스케줄링 알고리즘이다. 최상위 queue는 우선순위가 가장 높고 밑의 queue로 내려갈수록 우선순위가 낮아지게 된다. 우선순위가 높을수록 먼저 실행되게 되고, 우선순위가 낮은 queue에 들어있는 process는 상위 queue에 process가 존재하면 CPU 점유가 불가능하다. 이름에 들어있는 Feedback 기능은 process의 행동을 토대로 우선순위를 낮출지 높일지 결정하는 것을 의미한다. 예를 들어 MLFQ는 기본적으로 RR의 방식과 유사하다고 볼 수 있다. 각각의 queue는 레벨에 따라서 time quantom을 가지고 있는데, 만약 최상위 queue의 time quantom이 1이고 다음이 2 그 다음 레벨이 4라고 했을 때, 최상위 queue에 들어있던 어떤 process가 service가 3이여서 time quantom 1만큼을 다 사용하고 나면, 다음 queue로 우선순위를 낮추는 과정이 일어나게 된다.
다음 MLFQ의 5가지 규칙이다

- **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) = Priority(B), A & B run in RR.
- **Rule 3:** When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 5:** After some time period *S*, move all the jobs in the system to the topmost queue.

규칙 1. A 의 우선순위가 B 보다 높으면 A 실행

규칙 2. A 와 B 의 우선순위가 같으면 RR 로 A 와 B 를 수행

규칙 3. 시스템에 process 가 처음 들어오면, 최상위 queue 에 넣어줌

규칙 4. process 가 주어진 level 에 주어진 time_slice 를 다 사용했으면 우선순위를 낮춤

규칙 5. S 라는 시간이 지나면 모든 프로세스를 최상위 queue 로 부스트함

기본적으로 RR과 많이 유사하지만, queue가 여러 단계로 나뉘어 있다는 것에 초점을 마춰서 각 queue에 주어진 time quantom을 다 사용하면 다음 queue로 보내는 방식으로 구현을 시도해 볼 것이다.

4. Stride Scheduling

Stride Scheduling은 보통 각각의 process의 service의 비율을 활용하여 각각의 service의 비율에 맞춰 공평하게 실행시키는 Scheduling 알고리즘이다. Stride Scheduling의 등장 배경은 Lottery Scheduling의 not deterministic에 의해서 이다. Lottery Scheduling은 매 순간순간의 비율을 다르지만 결국엔 비슷해지거나 같아진다. 하지만 Stride는 일정한 간격으로 쪼개 보아도 실행된 비율이 일정하기에 Lottery 대신에 선택됐다고 할 수 있다. 보통 Stride Scheduling은 각각 process에 service를 가지고 균등한 값을 부여하지만 우리는 ppt에 나와있는데로 각각의 process에 ticket을 부여하여 가지고 있는 ticket들의 공배수를 구하여 다시 그 공배수에서 티켓을 나눠준 값들로 코드를 구현해볼 생각이다.

Result

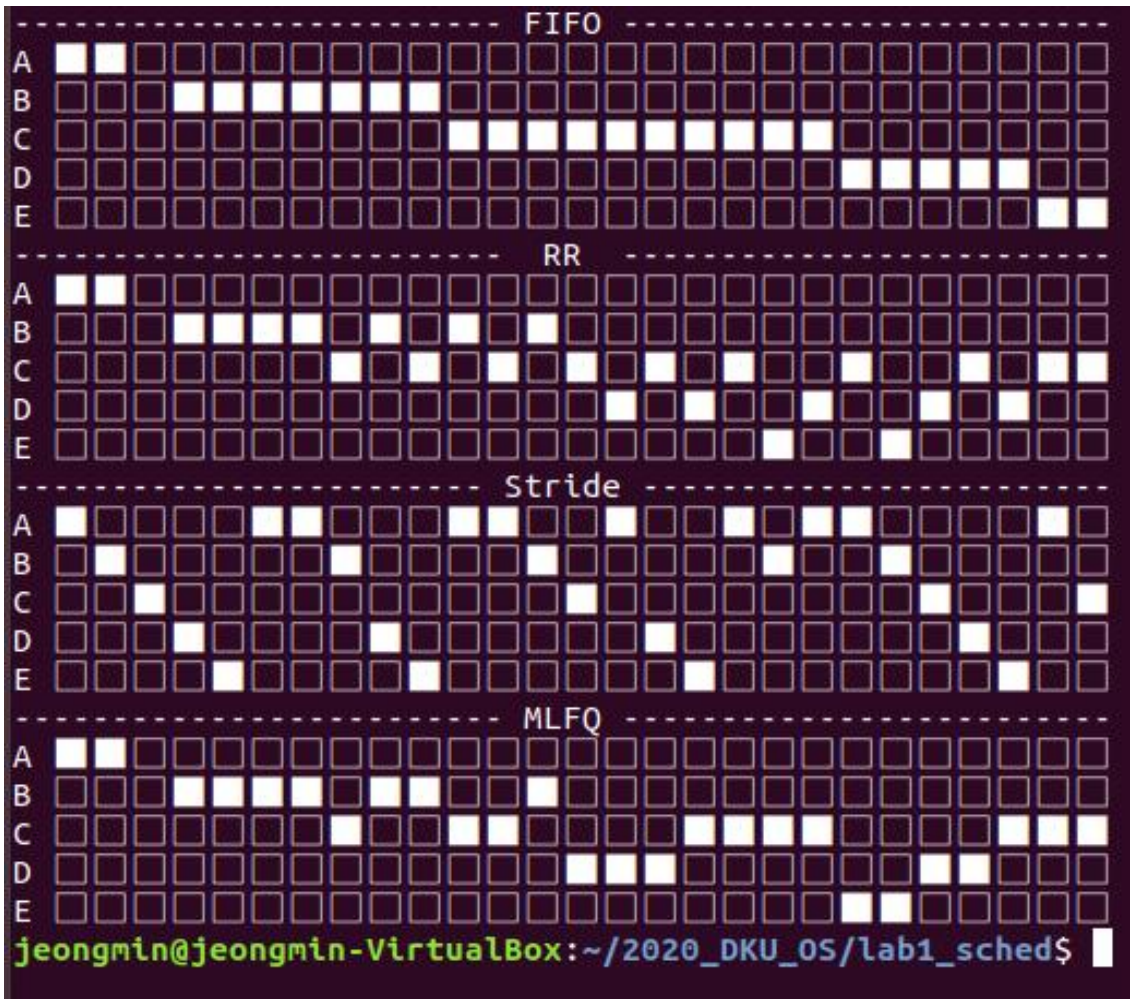
Ⅱ. 실행 결과 + Workload

Process 1.

Task	Arrival	Service	Ticket
A	0	3	60
B	2	6	20
C	4	4	10
D	6	5	5
E	8	2	5

Process 2.

Task	Arrival	Service	Ticket
A	0	2	40
B	3	7	20
C	7	10	10
D	13	5	15
E	17	2	15



※ Process 1은 ppt에 있는 값으로 실행해 보았고 Process 2는 다른 값들을 부여하여 실행하였다.

Discussion

Ⅲ. 개인 고찰

< 32163006 이건욱 >

Lab 1은 운영체제에서 핵심적인 기능을 담당하는 Scheduler를 여러 기법으로 구현해 보는 것이였다.

처음에 수업을 듣고 과제에 대한 설명을 들었을 때는 간단하게 구현할 수 있을거라고 생각했다. 실제 시간에 맞추어 프로세스를 만드는 것도 아니고 우리가 시간을 주면서 구현하는 것이기 때문에 어렵지 않게 수행 할 수 있을 거라고 생각했다. 특히나 FIFO, Stride는 구현하는데 어렵지 않았지만 RR과 MLFQ를 구현하는데 어려움을 많이 느꼈다. FIFO와 Stride에서는 필요성을 느끼지 못했던 Node와 Queue 개념을 RR과 MLFQ에서는 저 개념들을 안쓰고는 도저히 못 할거라 느끼고 저 개념들을 숙지하고 난 후에 RR과 MLFQ를 구현할 수 있었다.

과제를 진행하면서 느낀 각각의 Scheduler의 어려운 점들은 아래와 같다.

1. FIFO에서의 공백 나타내기

FIFO의 원리는 First In First Out 이였기 때문에 처음 구현하는데 큰 어려움은 없었다. 들어오는 대로 바로바로 출력하고 서비스 타임이 끝나면 다음 Task로 넘어가는 형식으로 작성하고 테스트를 하는 부분에서도 지장이 없었다.
하지만 문득 만약에 전의 Task가 모든 서비스시간을 다 쓰고 종료되었는데 그 다음 Task가 도착하지 않았으면 생기는 그 공백을 어떻게 할까라는 생각이 들었다. 그래서 생각해보던 와중 turnaround를 활용해보자라는 생각이 들었고 원래의 turnaround 개념과 똑같이 활용하지는 않았지만 이 전 Task의 도착시간과 서비스시간을 더해서 그 시간보다 크거나 같은 다음 Task의 도착시간이 없으면 turnaround를 1씩 증가시키며 다음 Task의 도착시간과 같아질때까지 공백을 출력해주는 식으로 이 문제를 해결하였다.

2. Stride에서 만약에 Ticket이 0이라면?

크게 어려움을 느꼈던 부분은 아니지만 또 실제로 Ticket이 0인 경우가 말이 안되긴 하지만 혹시나 하는 마음에 Ticket이 0인 경우를 추가하였다. 처음에는 단순히 Ticket이 0이면 그냥 Stride값을 0으로 하면 되지 라고 생각했지만 그것은 나의 오산이였고 제일 작은 값을 찾고 그 작은 값인 0에 계속 0을 더해주어 무한루프가 돌았다. 그래서 Stride 값이 0이면 continue로 이 문제를 해결하였다.

3. RR(Round Robin)에서의 Queue 삽입 순서 정하기

RR를 구현하기 전부터는 Node와 Queue 개념을 완전히 숙지한 후 코딩을 시작했어야 했기 때문에 일단 이것부터 어려웠다...
일단 RR을 구현하는데 있어서는 우리가 수업시간에 배운 것을 토대로 작성하려고 했다. 처음 들어온 프로세스부터 수행하고 중간에 새로운 프로세스가 들어오면 Queue에 넣고 타음 슬라이스가 끝나면 넘어가도록 설계를 진행하였지만 계속해서 순서가 다르게 나오는 문제가 발생하였다. 여러 시도 끝에 Queue에 들어가는 순서와 도착하는 순서가 중요하다는 것을 깨닫게 되었고 도착시간과 Queue에 들어가는 순서를 맞춤으로 해결하였다.

4. MLFQ에서 각 Queue에 담긴 프로세스가 제대로 된 순서로 동작하게 하기

우선순위는 Queue를 이용해서 줬다. 따로 우선순위를 변수로 선언하지는 않고 높은 순서의 Queue가 비었을 경우에만 다음 Queue에 있는 프로세스가 수행 되도록 설계하였다.
하지만 MLFQ에서는 이러한 사항들을 고려하기 위해서는 다양한 조건을 검사해줘야 했다. 예를 들면 1번 규ㅐ서 어뜬 경우에는 다시 1번 Queue로 돌아가는지 또는 1번 Queue가 지나고 2번 Queue가 수행하는 사이에 새로운 프로세서가 들어오는 경우 등 많은 경우들이 있었고 여러 가지 테스트를 진행하면서 빠진 부분들을 채워주고 다시 알고리즘을 검사하는 등 오랜 시간을 들이며 수정을 하며 해결하였다.

사실 위에 언급한 점 이외에도 다른 어려운 점들이 많았다. 수업을 들었음에도 실제로 구현을 하려하니 각각의 세부사항을 쪼개는게 여간 어려운 일이 아니였다. 심지어 VM와 Ubuntu도 이번에 처음 사용해보아 익숙하지 않았다. 하지만 이번 과제를 수행하며 얻은 것이 정말 많아 과제를 모두 해결하고 나니 매우 뿌듯하였다.

< 32164420 조정민 >

개인적으로 이번 과제는 정말 어려웠다. 작년 시스템 프로그래밍 수업 때 했던 마지막 팀 과제는 아무것도 아니였구나라는 생각이 들었다. 우선 scheduling algorithm을 꼼꼼하게 공부했다. 과제에서 나온 알고리즘은 들어온 순서대로 처리를 해주는 FIFO, 이 FIFO에 preemptive가 추가된 Round Robin, 그리고 큐를 여러 개 사용해 우선순위를 주는 MLFQ, 마지막으로 Stride Scheduling이다.

개념을 이해하고 그것을 바탕으로 scheduling 그림을 그리는 것은 그다지 어렵지 않았는데, 막상 코딩을 하려고 하니 쉽지가 않았다. 리눅스는 C를 지원하기 때문에 C++에서처럼 class가 아닌 구조체를 사용해야 했다. 그리고 자료구조 수업 때 배운 연결리스트나 연결큐 같은 개념들을 사용했다. 정확한 코드가 기억이 나지 않아 작년에 사용했던 ppt를 참고한 것이 도움이 되었다. FIFO Scheduling과 Stride Scheduling은 RR(Round Robin)과 MLFQ(MultiLevel Feedback Queue)보다 구현이 비교적 쉬웠다.

FIFO는 들어오는 Task 먼저 출력하면 되는 것이라 어려움은 없었지만 Task와 Task 사이에 공백이 있을 경우 생기는 오류를 처리하지 못해 고민을 했다. 바로 이전 Task의 arrival time과 service time을 더해서 현재 Task의 arrival time보다 작으면 time_count를 하나씩 늘려주는 것이다. 더한 값이 현재 Task의 arrival time과 같을 때까지 빈 네모칸을 출력하면 되는 것이다. 이 문제를 해결하기 위해 어떤 방법을 사용해야 할지 고민을 오래 했지만 이건욱 학생의 도움으로 생각보다 빠르게 해결할 수 있었다.

RR(Round Robin)은 Node와 Queue를 사용해서 그런지, FIFO 스케줄링보다 알고리즘이 더욱 복잡해 MLFQ 다음으로 시간 투자를 많이 했다. Time slice가 끝나면 실행되고 있는 프로세스를 Queue의 맨 뒤에 넣고 삭제를 해야 하는데 FIFO와 마찬가지로 프로세스들의 도착 시간 사이에 빈 공간이 생겼을 경우 오류가 생겨서 무한루프에 빠지는 문제가 생겼었다. 변수 하나를 미리 생성한 후 빈 큐가 되는 상황에 도착 시간 검사를 다시 실행해서 큐에 새로운 프로세스를 넣는 방식을 쓰고, 반복문 자체를 빈 큐일 때까지 돌리지 않고 프로세스들의 service time이 0이 될 때까지 반복하는 방법으로 해결을 시도했다.

MLFQ는 4개의 스케줄링 알고리즘 중에 제일 구현하기 힘든 알고리즘이었다. 기본적으로 5가지 규칙 중 부스트를 제외한 4가지 규칙을 사용했다. RR 코드에 큐를 단계별로 만들어서 Time slice가 끝나면 다음 큐로 내려주고 새로운 프로세스가 들어오면 최상위 큐에 넣는 방식으로 구현하였다. MLFQ는 큐 여러 개에 우선순위가 있어서 다음 레벨 큐로 내려가면 도착시간 사이에 차이가 있어 최상위 큐가 빈 큐인 채로 종료된다는 부분을 고려하지 못한 채로 코드를 짜고 실행을 하니 오류가 있었다. 변수를 하나 생성해서 최상위 큐에 프로세스가 들어있지 않고 시간 안에 도착한 프로세스가 없으면 다음 도착시간을 검사하는 방식으로 해결을 시도했다. 추가적으로 2번 큐에서도 프로세스가 들어오는지 확인해서 새로운 프로세스가 들어오면 break를 해주는 것으로 오류 없이 실행할 수 있었다.

Stride는 RR과 MLFQ보다는 비교적 알고리즘 구현을 하는 데에 힘들지 않았다. 각 Task의 티켓 값의 최소공배수를 구해서 작은 티켓 값에 스트라이드를 더해주면 되는 문제라 오랜 시간을 투자하지 않아도 구현을 할 수 있었다.

이번 과제는 시스템프로그래밍 수업 때 했던 과제들과는 수준이 너무나도 달랐다. 이번 스케줄링 알고리즘을 구현하면서 다른 과목에 비해 시간 투자를 굉장히 많이 한 것도 사실이고

어려운 부분이 많아서 팀원과 함께 고민만 몇 시간동안 하기도 했었다. 우리가 배웠던 이론적인 내용을 코드에 직접 적용하는 것은 그냥 이론을 이해하는 것보다 몇 십배, 몇 백배는 어렵다는 사실을 이번 과제를 통해 뼈저리게 느꼈다.

이번 기회에 Virtual machine을 통해 윈도우 환경인 내 노트북에서 리눅스 환경을 적용하여 사용한 것은 정말 좋은 경험이었다. 또한 터미널에서 Make를 통해 컴파일을 할 수 있다는 새로운 사실도 알게 되어 새롭다. 앞으로 팀과제를 하면서 이번 과제처럼 새롭게 알아갈 수 있는 부분들이 많을 거라는 점에서 굉장히 영광스럽다고 생각한다. 우리를 힘들게 하는 과제일지라도 우리가 직접 얻어갈 수 있는 것이 많다고 생각하면 이 정도의 시간 투자는 정말 보람찰 것이라 생각이 든다.

우리 둘다 직접 A4 용지에 직접 그림을 그려가고 손으로 써가면서 열심히 한 과제인만큼 보람과 뿌듯함을 더 크게 느낄 수 있었다.