**Operating System**

**[Scheduler programing with Thread synchronization]**

**12184102 양유진**

**1. 개요**

어떤 방식으로 스케쥴러를 만들지 고민을 하다, 예제에서 보여주신 방법으로 Multi-level queue기법을 구현하였습니다. 생각보다 thread간의 synchromization이 이론만큼 되는 느낌이 아니였습니다. 따라서 큐간의 스케줄링은, 가장 쉽게 구현 할 수 있다고 생각되는 Priority스케줄링을 사용해 주었습니다. (1번째 큐>2번째 큐 > 3번째 큐 > 4번째 큐) (큐는 4개까지 생성하였습니다.)

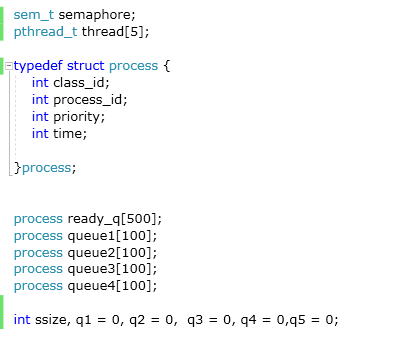
1번째 큐는 FCFS 스케줄링을 합니다.(입력시간은 모두 0이라고 동일한 것을 알지만, input한 순서대로 도착하였다고 생각했습니다.)

2번째 큐는 SJF 스케줄링을 합니다.

3번째 큐는 Round Robin 스케줄링을 합니다.(이것또한 마찬가지로, input순서대로 입력시간이 정렬되었다고 가정하였습니다.)

4번째 큐는 우선순위 스케줄링을 합니다.

**2. 구현 방법, 알고리즘 명세**

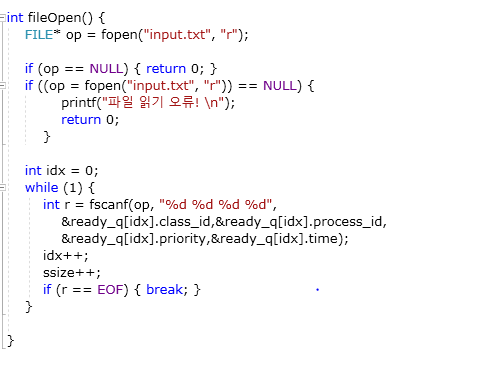
**(1) 선언해둔 전역변수, struct**

각각의 thread를 create할 때 사용할 thread변수와, thread들의 접속을 제어해줄 semaphor를 선언해줍니다.

또한 process에 관한 정보를 한번에 저장하고 처리할 process 구조체를 선언하여 주었습니다. 처음 input파일을 받을때에는 ready\_q에 받은 후, 큐간의 프로세싱을 할 때 각각의 class\_id와 같은 넘버를 가진 큐로 이동을 할 예정입니다.

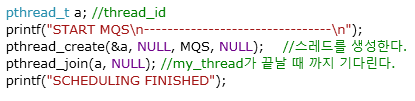
ssize변수는 ready\_q에 입력되는 process의 숫자를 세어줍니다.q1~q5는 각각의 큐에 들어가는 process의 숫자를 세어줍니다.

**(2)input file을 오픈하고, process에 대한 정보를 ready\_q에 저장하는 함수**

 File open을 하였을 때 입력 받을 정보가 없다면, 예외처리를 해줍니다.

입력 받는 숫자들의 순서대로, class\_id, process\_id, priority, time(burst time)을 입력해줍니다. 더 이상 입력 받을 텍스트가 없을 때 까지 진행해 줍니다.

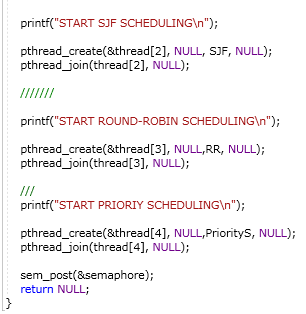
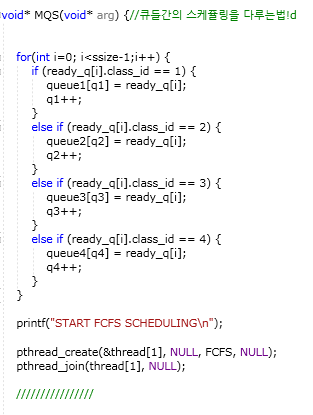
Ready\_q 에 process의 정보가 한번씩 입력될때마다, ssize라는 전역변수를 사용하여, ready\_q에 들어있는 process의 개수를 저장합니다.

**(3)- 큐간의 스케줄링을 하는 함수(MQS)**

main함수에서 MQS스케줄링을 쓰레드를 생성하여, 실행합니다->

pthread\_create() ->쓰레드를 생성합니다

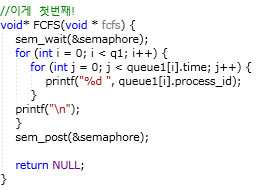
pthread\_join()->특정 쓰레드가 종료하기를 기다렸다가, 쓰레드가 종료 된 후 그 다음을 진행하는 함수입니다.



->전역변수로 저장해둔 ssize변수를 통하여, ready\_q에 들어가있는 class\_id의 번호에 따라서 각각의 번호에 맞는 큐로 재배치 해줍니다. 각각의 큐에 들어가있는 process의 개수는 q1~q4변수에, process가 입력될때마다 업데이트 해줍니다.

->각각의 큐들에 재배치가 끝났다면, **priority스케줄링**:각각의 큐 번호 순서대로 스레드를 생성해주고, 해당 순서의 쓰레드가 종료하기를 기다린 후, 종료 되면 다음 순서의 쓰레드를 실행해주는 과정을 반복하여 줍니다. MQS함수는 큐간 스케줄링을 위하여 semaphore를 통한 스레드 동기화를 진행합니다. 큐들간의 스케줄링이 끝난 후 , MQS스레드를 sem\_post를 통하여 세마포어 값을 1 증가시켜줍니다.

<- 처음 MQS를 시작할 때 wait를 시켜주지 않는 이유는, main함수에 semaphore를 0으로 초기화 하였기 때문입니다. 첫번째 큐를 진행할 때 wait를 진행하기 때문입니다. 세마포어 값을 1감소시킨 후 또1 을 감소시킬 수는 없기 때문에, MQS를 시작할 때는 sem\_wait를 해주지 않습니다.

**(4)-각각의 큐에서 진행되는 스케줄링 함수**

**<FCFS scheduling>**

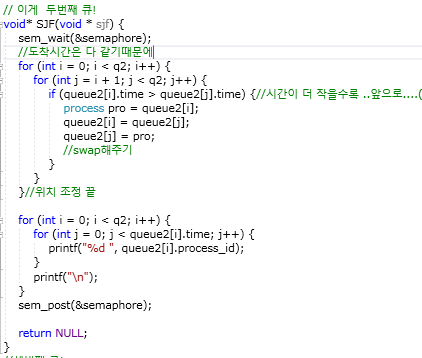
sem\_wait를 사용하여, 세마포어를 1 감소시킨 후에 스레드를 실행하여 준다.(대여의 개념)

입력순으로 출력을 하는 프로그램이여서, 큐에 저장된 time만큼 process\_id를 단순 출력하면 된다.

sem\_post를 사용하여 세마포어의 값을 1 증가시켜준다.(반납의 개념으로 생각해주면 된다.)

**<SJF scheduling>**

이 이하로 sem\_wait, sem\_post의설명은 생략 하겠습니다.(위와 같기 때문)

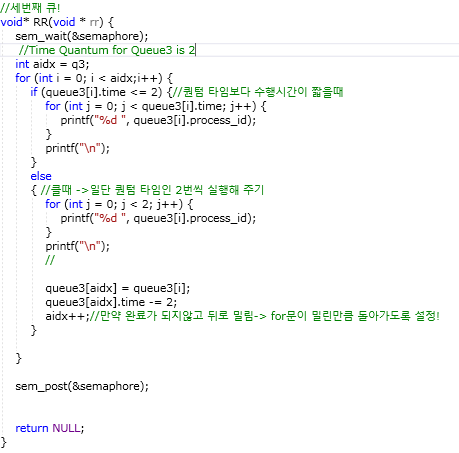


두번째 큐에서 진행할 스케줄링은 SJF스케줄링 입니다. SJF스케쥴링은, 준비 큐에 있는 프로세스 중 작업시간이 가장 짧은 프로세스부터 cpu를 할당하는 스케줄링입니다.

이 때문에 출력하기전, queue2에 입력순으로 저장되어 있는 process들을 작업시간이 짧은 순으로 정렬시켜준다.

정렬이 끝난 후, 두번째 큐에 있는 process들을 정렬된 순서대로(짧은시간 순서), time만큼 process id를 출력해주며 마무리합니다.

**<Round-Robin scheduling>**

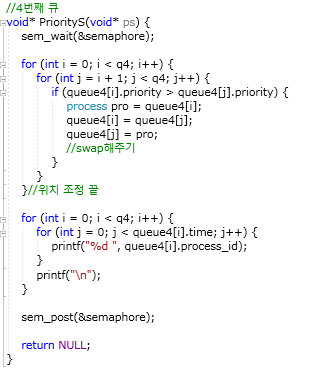
****

세번째 큐에서 진행할 스케줄링은 round-robin스케줄링 입니다.Round robin 큐의 time quantum은 2로 설정했습니다. 위와 마찬가지로, 입력된 순서대로(input) 진행하였습니다.

queue3에 저장 되어있는 순서대로, 각각의 process의 수행시간이 퀀텀 타임인 2보다 큰지, 같거나 작은지 검사를 해줍니다. 만약 process의 수행시간이 퀀텀타임보다 작거나 같다면, 그대로 출력시켜줍니다.

만약 process의 수행시간이 퀀텀타임보다 크다면, 일단 퀀텀타임만큼만 출력 해준 후, process를 기존time을 ,기존에서 퀀텀 타임을 뺀 값을 업데이트 해줍니다. 그 후 queue3 어레이의 가장마지막에 새롭게 저장하고, aidx값을 늘려주어 밀린만큼 반복문이 진행되도록 설정해 줍니다.

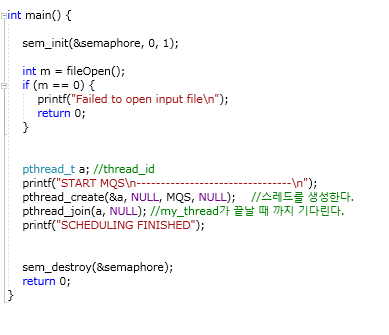
**<우선순위 기반 스케줄링>**

SJF알고리즘과 달라진 것이 거의 없다고 생각하면 됩니다.

SJF알고리즘은 수행시간이 짧은 순으로 정렬한 것이라면, Priority 기반 스케줄링은 process에 저장된 priority가 높은 순서대로(숫자는 작은 순)

정렬을 해주고, 출력을 해주었습니다.

모든 함수의 마지막에 return NULL을 써주었습니다. 주어진 기능을 모두 수행하였다면 함수를 종료시켜주어서 thread를 종료시켜 주었습니다.

**(4)-main함수 설명**

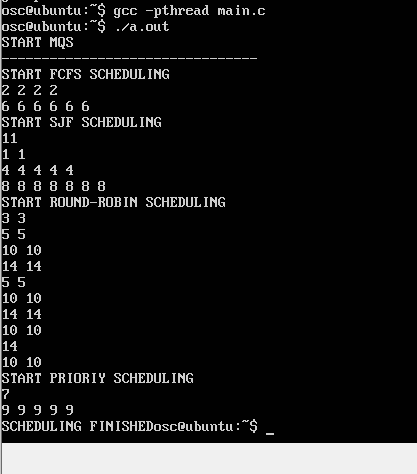
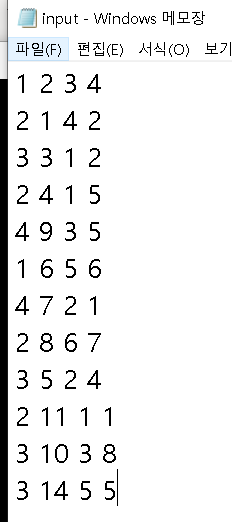
세마포어를 초기화 해준 후에, fileopen함수를 이용하여 input file을 열고 ready\_q 배열에 값들을 입력시켜 줍니다.(실패했을 경우 예외처리 )

MQS를 실행 해준 후, 쓰레드가 끝나게 되면 scheduling이 완료되었다고 출력해줍니다.

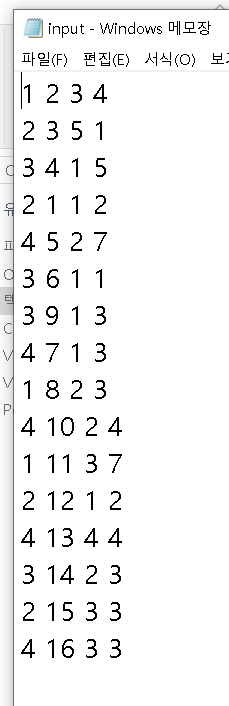
그 후 세마포어와 관련된 리소스들을 소멸시켜줍니다ㅏ.

**3. 예시 입출력**

각 큐마다 2~4개씩 들어가도록 구성 한 후, 충분히 각 큐의 기능이 나타나도록 예시를 구성했습니다.

**1번예시)**

**2번예시)**



**4. 과제에 대한 총 평**

큐들간의 스케줄링을 진행하는 방법을 너무 쉽게 구현한건 아닌가 싶어서 아쉬웠습니다. 다음에 스케줄러를 만들게 된다면 각 큐들마다 제일 첫번째 들어온 process의 time을 비교하여, 큐들간의 순서들을 정하든지 하는 스케줄러를 많은 시간을 들여 만들어보려 합니다. 배열에 대한 포인터를 직접 함수의 parameter에 넘겨주어 프로그램을 구성하려 하였는데, 시간 관계상 전역변수를 사용하였습니다. 또한 직접 process 들의 arrival time을 설정하여, preemtive가 가능한 스케줄러도 만들어보고 싶다는 생각을 하였습니다.

여러모로 아쉬움이 많이 남지만, thread를 어떻게 호출하는 것인지 , semaphore가 어떤 식으로 프로세스들을 대기시키는지 직접 경험하고, 실행 해보면서 semaphore의 개념에 대해 조금 더 제대로 이해 한 것 같아 의미 있었습니다.