**REPORT**

**| Operating Systems**

**| Scheduler simulation**

**| 컴퓨터정보공학과**

**| 12141540 박영창**

# Objective

Operating system 수업 시간에 여러가지 scheduling 방식에 대해 배웠습니다. FCFS, SJF, RR 등 여러가지 방식이 있었지만, 이번 과제에서는 Linux에서 사용되는 scheduling 방식인 multi-level queue scheduling의 RR 방식과 CFS 방식을 직접 구현해 보는 것이 목적이었습니다.

1차 과제 때와 마찬가지로 구름IDE를 통해 Linux 환경에서 컴파일, program 실행, 파일 입출력을 했습니다.

중간고사 기간에 나왔던 내용이라 공부를 많이 했지만, 막상 구현하려고 하니 생각해야할 부분이 아주 많았습니다. 이 과제 프로그램을 구현하고 나니 중간고사 이전에 공부했던 내용보다 더 Linux에서 사용되는 scheduling 방식에 대해 깊게 알게 된 것 같습니다.

# Program

**Linux scheduling – multi-level queue scheduling**

Linux scheduling의 multi-level queue scheduling을 구현한 프로그램입니다. Linux에는 3가지 class가 존재합니다. FIFO방식을 사용하는 real-time class, RR 방식을 사용하는 real-time class, 그리고 conventional class입니다. 기존의 multi-level queue와 같이 queue간의 scheduling도 존재하며 이는 priority scheduling을 사용하고 real-time class가 conventional class보다 우선적입니다.

프로그램에서는 real-time RR class, conventional class만을 구현했습니다. 0~139 사이의 우선순위(priority)를 갖는 각각의 task는, 우선순위가 0~99 에 존재한다면 real-time RR class, 100~139에 존재한다면 conventional class로 들어가게 됩니다. Real-time class가 우선적이기 때문에, real-time class에 있는 모든 task가 실행되고 나서야 conventional class의 task가 실행됩니다.

**Real-time RR class**

RR(Round-Robin) 방식은 각 task를 time quantum만큼 수행하고 종료되지 않으면 queue의 맨 뒤의 순서로 옮겨지고 이를 반복하는 방식입니다. Linux에서 사용되는 real-time RR class의 scheduling 방식은 모든 task를 한 queue로 보는 것이 아닌, 우선순위가 같은 task를 한 queue로 보는 RR scheduling 입니다. 따라서, 우선순위가 같은 task들이 여러 개 존재한다면 그 task들의 queue에서 RR 방식이 진행됩니다. 이 real-time RR class에 들어가는 task의 우선순위는 0~99이며, 문제에서 주어진 time quantum은 2이므로, 이에 기반하여 구현하였습니다.

**Conventional class**

Conventional class에는 우선순위가 100~139인 task들이 들어갑니다. Conventional class에서 사용되는 scheduling 방식은 CFS(complete fair scheduling) 입니다. CFS 방식은 각 task에 비중을 할당하여, 그 비중을 기준으로 모든 task들이 공평하게 수행할 수 있도록 하는 scheduling 방식입니다. 공평하게 수행된다는 것은 비중이 클수록 더 많이 실행되고, 비중이 작을수록 더 적게 실행되는 것을 말하는데, 이는 vruntime(task의 비중을 통해 알아내는 가상 runtime)을 통해 비교할 수 있습니다. CFS scheduling 방식을 구현하는 과정에서 이 문제에서는 조건이 3가지 주어졌습니다.

첫 번째로, 우선순위에 따른 비중은 각각 100~109 : 10, 110~119 : 8, 120~129 : 6, 130~139 : 4로 정해져 있습니다. 이는 conventional class에 들어갈 task의 구조체를 만들어 비중을 각각 우선순위에 맞게 저장하는 방식으로 적용했습니다.

두 번째 조건은 conventional class에 들어갈 task중에 priority가 같은 task는 단 하나 뿐이라는 것입니다. 이를 확인해서 예외 처리하는 부분은 따로 구현하지 않았지만, 같은 우선순위에 여러 개의 task가 있다면 이를 처리하는 방법을 생각해 구현하면 할 수 있다고 생각합니다.

세 번째 조건은 vruntime이 동일한 두 task가 있다면, 더 높은 우선순위를 갖는 task가 먼저 실행됩니다. 이 경우는 같은 비중과 다른 우선순위를 가지는 두 개의 task, 그리고 비중이 4인 task와 비중이 8인 task 간의 알맞은 수행순서를 구현하는 것이라고 생각했고, conventional class에 있는 task들의 vruntime을 비교해 이를 구현했습니다.

**변수**

* Struct task | task의 정보가 저장되는 구조체
  + int priority | 우선순위
  + int taskNum | task 번호
  + int runtime | task의 수행 시간
* struct ctask | conventional class에 들어갈 task의 정보가 저장되는 구조체
  + int priority, taskNum, runtime | task와 동일
  + int weight | task의 우선순위에 따른 비중
  + int vruntime | task의 가상 수행 시간
* task tempT, ctask tempCT | task, ctask의 연산에 사용될 임시 변수
* task taskList[100] | 모든 task가 저장되어 있는 배열
* task realTime[101] | real-time RR class
* ctask conventional[101] | conventional class
* int tasks | 전체 task의 개수
* int timeQuantum | real-time RR class에서 쓸 time quantum
* int pIndex | real-time, conventional에 접근할 인덱스
* int rIndex, cIndex | real-time, conventional 각각의 의 task 개수
* int data, FILE \*input, \*output | 파일 입출력에 쓰일 변수
* int totalRuntime(realtime) | 우선순위가 같은 task들의 총 수행시간
* int sameIndex | 우선순위가 같은 task들의 개수
* int totalRuntime(conventional) | conventional class의 모든 task의 총 수행시간
* int totalCurrentRuntime | 현재까지의 총 수행시간
* double totalWeight | conventional class의 모든 task의 비중의 합

**코드 실행**

1. input.txt라는 텍스트 파일로부터 task의 정보를 taskList 배열에 최대 100개까지 저장합니다.
2. 각 task의 우선순위를 확인해 0~99 사이면 real-time RR class인 realtime 배열에, 100~139 사이면 conventional class인 conventional 배열에 저장합니다. Conventional 배열로 저장할 때는 task의 정보를 ctask 구조체에 옮겨 저장한 뒤 weight를 같이 저장합니다.
3. Realtime 배열과 conventional 배열에 있는 구조체들을 우선순위를 기준으로 정렬합니다.
4. Real-time RR class의 우선순위가 conventional class의 우선순위보다 높으므로 pIndex에 0을 저장한 뒤 realtime 배열에 먼저 접근합니다.
5. pIndex의 인덱스에 해당하는 realtime 배열 내의 task와 우선순위가 같은 task들의 개수와 그 task들의 수행시간의 총합을 구해줍니다.
6. 총 수행시간동안 task들이 모두 수행될 때 까지, 우선순위가 같은 task들을 RR 방식으로 수행합니다. 이 때 time quantum이 2이므로 각 task를 2번씩 수행한 뒤에 다음 task를 수행합니다. Task가 수행될 때마다 해당 task의 남은 수행시간을 1 감소시키고, 동시에 총 수행시간을 감소시켜줍니다.
7. 같은 우선순위의 task의 출력이 완료되고 나면, 같은 우선순위 내의 모든 task가 실행되었음을 의미하므로 개행하고, pIndex에 위에서 구했던 같은 우선순위를 갖는 task의 개수를 더해 다음으로 우선적인 우선순위를 갖는 인덱스로써 배열에 접근합니다. 만약 이 연산으로 인해 pIndex가 realtime 배열에 있는 task의 수 (rIndex) 와 같아졌다면, realtime 배열의 모든 task를 처리한 것이므로 반복문을 종료합니다.
8. Real-time RR class의 모든 task를 처리했으므로, conventional class에 접근해야 합니다. Conventional 배열에 task가 존재한다면, 모든 task의 총 수행시간과 비중의 합을 구합니다.
9. vruntime이 가장 적은 task를 찾아 pIndex에 해당 task의 인덱스를 저장합니다. 동일한 vruntime을 갖고 있어도, conventional 배열이 우선순위에 따라 정렬되어 있고, 우선순위가 같은 task는 conventional 배열에 존재하지 않으므로, 배열을 차례대로 탐색한다면 vruntime이 가장 적으면서 우선순위가 vruntime이 같은 task중에서 가장 높은 task의 인덱스가 저장됩니다.
10. pIndex에 해당하는 task의 수행시간이 남아 있다면, 이전에 수행된 task의 정보가 저장되어 있는 tempCT와 현재 수행할 task를 확인해 같은 task가 아니라면 개행합니다. 그 다음 현재 수행할 task의 정보를 출력한 뒤, tempCT에 수행한 task를 저장함과 동시에 현재 총 수행시간을 1 증가시키고, 수행한 task의 남은 수행시간을 1 감소시킵니다. 그리고 수행한 task의 vruntime을 갱신해줍니다.

Vruntime += (NICE\_O\_LOAD / weight) \* delta\_exec = (1024/비중)\*수행시간

여기서 어떤 task를 수행할지 확인과 수행하는 과정은 1번씩 진행되므로 한번 실행될 때마다 (1024/비중 )이 vruntime에 더해집니다.

1. pIndex에 해당하는 task의 수행시간이 더 이상 남아 있지 않다면, 해당 task를 conventional 배열의 맨 뒤로 밀어내고 conventional class의 총 task 개수(cIndex)를 감소시켜줍니다. 이는 conventional class에서 수행이 완료된 task를 삭제하는 과정입니다.
2. 10, 11 과정을 총 수행해야할 시간과 현재 수행한 시간이 같아질 때까지 반복합니다. 이 과정을 통해 conventional class에 존재하는 모든 task를 처리하게 됩니다.

**결과 및 분석**

출력 이미지를 보면, Linux 환경에서 이 프로그램을 정상적으로 컴파일 및 실행했습니다. Input 텍스트 파일에 맞는 출력을 가진 output 텍스트 파일도 알맞게 생성되었습니다.

Input 1의 이미지를 보면 우선 모든 task가 real-time RR class인 것을 알 수 있습니다. 40이라는 우선순위를 가진 두 task와 50이라는 우선순위를 가진 하나의 task가 있습니다. 우선순위가 40으로 같은 두 task가 time quantum(2)에 따라 수행되고 다음 우선순위로 넘어가 똑같이 수행하는 결과는 output 1 이미지처럼 나타납니다.

Input 2 의 이미지를 보면 real-time RR class에 들어갈 task가 하나, conventional class에 들어갈 task가 두 개인 것을 알 수 있습니다. Real-time RR class에 들어갈 task를 먼저 처리하고, 우선순위가 102, 113인 conventional class에 들어갈 두 task를 처리합니다. 처음에는 두 task의 vruntime이 같으므로 우선순위가 더 높은 task가 실행되고, 그 다음에는 vruntime이 더 적은(같다면 우선순위가 높은) task가 실행됩니다. 이런 식으로 반복해 나타나는 결과는 output 2 이미지처럼 나타납니다.

taskList 배열에 있는 task들을 우선순위에 따라 realtime, conventional 배열로 분배하고, realtime 배열을 먼저 처리함으로써 real-time RR class와 conventional class 간의 우선순위를 처리했습니다. task가 queue에 들어가는 시간이 주어지지 않고 맨 처음에 모든 task의 리스트가 주어졌기 때문에 이런 방식으로 구현했는데, task가 들어오는 시간이 주어진다면 구현하기 어려울 것 같습니다.