Multilevel Feedback Queue

(운영체제 Assignment 1)

경영학과 4학년 김재균

2013002277

0. 개발환경

운영체제: Ubuntu

컴파일러: gcc compiler

언어: C

소스 코드 에디터: Vim

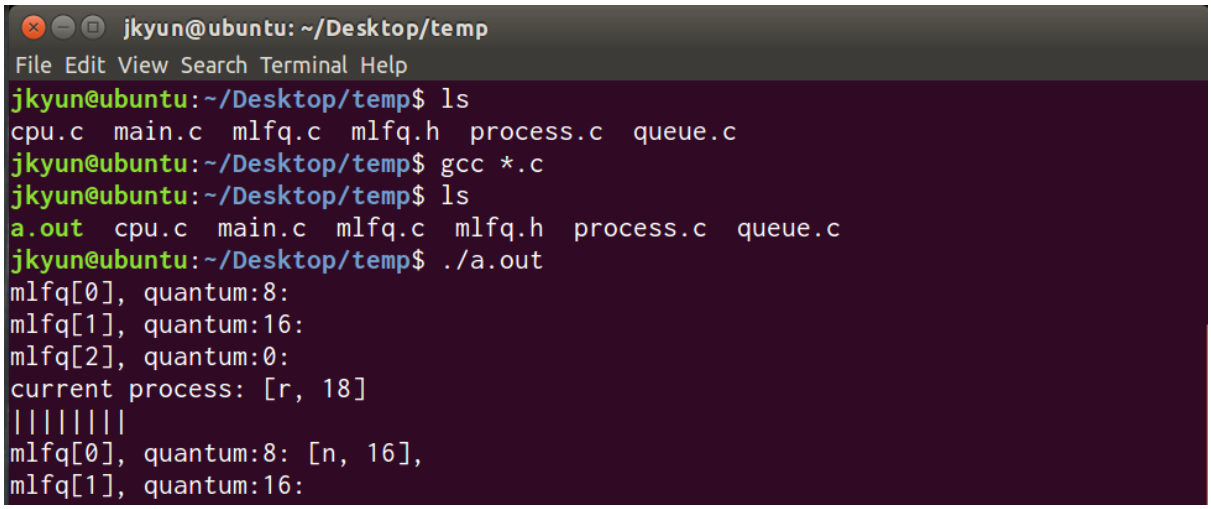
소스 코드 버전 관리: Git

1. 실행 방법

소스코드 파일: main.c / cpu.c / mlfq.c / process.c / queue.c

헤더파일: mlfq.h

Ubuntu 터미널에서 **1. gcc *.c** 입력 후, **2. /a.out** 입력해서 파일 실행



2. Input 파일 설명

cpu.c mlfq.c process.c queue.c는 소스 코드 파일 이름과 같이 각각의 함수에 대한 내용을 담고 있습니다. **main.c** 파일 하나로 할 경우 코드가 복잡해지는 경향을 피하기 위해 분리하였습니다. 각 4개의 소스코드 파일의 헤더파일은 **mlfq.h** 파일 하나로 통일 했습니다.

3. 함수 설명

main.c < buffers

```
 1 /* Multilevel feedback Queue
 2  * Jae Kyun Kim
 3  * 1801410
 4  */
 5
 6 #include <stdlib.h>
 7 #include <stdio.h>
 8 #include <unistd.h>
 9 #include <time.h>
10
11 #include "mlfq.h"
12
13 // MLFQ: 0-queue:RR(quantum=8), 1-queue:RR(quantum=16), 2-queue:FCFS
14 #define MLFQ_LEVEL 3
15 #define MLFQ_CAPACITY 10
16
17 // Make Multilevel Feedback Queue and CPU
18 struct mlfq_t* mlfq[MLFQ_LEVEL];
19 struct cpu_t cpu;
20
```

- 메인 함수에는 총 5개의 헤더파일을 include 했습니다. unistd.h 는 출력할 때 사용 했으며 time은 process를 생성할 때 rand() 함수를 사용하기 위해 사용 했습니다.

- 13번째 줄: MLFQ의 큐들은 총 3가지로 구분하였고, 0번째 큐는 Round Robin 알고리즘을 사용했으며 quantum은 8 입니다. 1번째 큐도 Round Robin 알고리즘을 사용 했지만 quantum은 16 입니다. 마지막 큐는 FCFS 알고리즘을 사용 했습니다.
모든 큐의 사이즈는 편의상 10으로 통일 했습니다.

```
47
48 int main(int argc, const char* argv[]){
49     srand(time(NULL));
50     // Make MLFQ and CPU
51     InitMLFQ();
52     InitCPU();
53
54     /* Assume that 1 loop is 1 burst */
55     while(1){
56         GenerateProcessRandomly();
57         // if cpu is not pointing any process
58         if(!cpu.current){
59             FindNextProcess();
60             usleep(1 * 1000000); // usleep function -> microsecond
61         } else {
62             HandleProcess();
63             // If process burst is done
64             if(Done()){
65                 DeleteProcess();
66             }// If process burst is done but because of CPU givien time
67             else if(TimeOut()){
68                 LowerQueue();
69             }
70         }
71         PrintStatistic();
72         usleep(1 * 1000000);
73     }
74     return 0;
75 }
```

1) 먼저 InitMLFQ() 함수를 통해 Multilevel Feedback Queue를 만들었고 초기화 시킵니다. InitMLFQ 함수는 queue.c의 queue와 mlfq.c의 mlfq 속성들을 이용하여 만들었습니다.

2) CPU 또한 InitCPU() 함수를 통해 하나의 객체를 만들었습니다. Current 포인터는 NULL, burst_left = 0으로 초기화 했습니다.

3) while 반복문을 통해 한번 돌 때마다 초당 1 burst를 수행한다고 가정 했습니다. CPU 특성상 무한 루프라고 가정 했습니다.

4) GenerateProcessRandomly() 함수를 통해 process를 random하게 생성하여 queue로 push 했습니다. Process 이름은 a-z로 랜덤하게 생성하였고, 버스트도 1-50까지 랜덤하게 생성 했습니다. 50%의 확률로 process가 생성되게 설정 했고 편의상 10개의 프로세스만 생성하게 설정했습니다.

5) if 구문을 통해 cpu.current가 NULL을 가리키고 있지 않다면, 즉, 놀고 있다면 FindNextProcess() 함수를 통해 각 큐에 대기하고 있는 프로세스가 있는지 확인 합니다. Usleep() 함수를 이용하여 1초라는 간격을 주었습니다.

6) 만약에 cpu.current가 이미 프로세스를 가리키고 있다면, 즉, 프로세스를 수행 중이었다면 Handle Process()를 통해 각 큐의 조건에 맞게 계속해서 수행 합니다.

7) 만약에 Process가 주어진 CPU 시간 안에 끝이 났다면 그 Process는 DeleteProcess() 함수를 통해 제거 해주고 만약에 CPU 시간 안에도 burst가 끝이 나지 않았다면 LowerQueue() 함수를

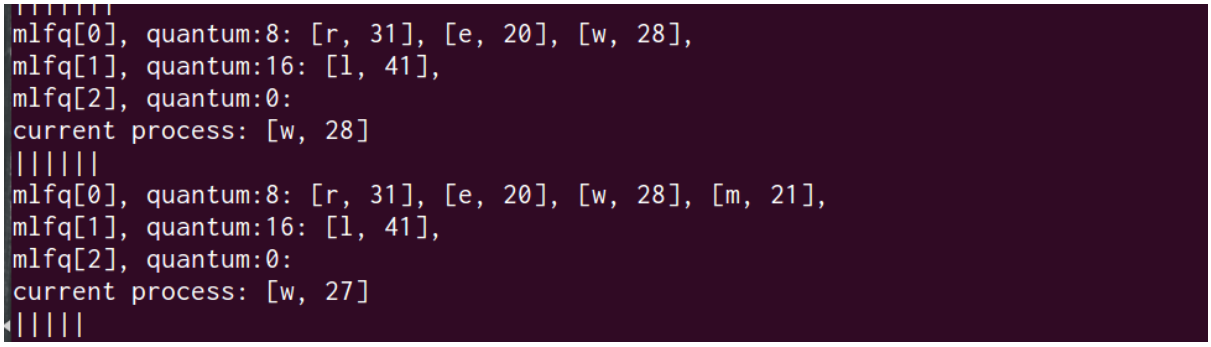
통해 다음 큐로 내려줍니다.

8) 마지막으로 현상황을 PrintStatistic() 함수를 통해 출력 합니다.

9) (1) ~ (8)까지 MLFQ에 아무 Process가 없을 때까지 반복 합니다.

* 세부적인 함수 설명은 각각의 해당 소스 코드 파일에 주석 달아났습니다.

4. 실행 결과



-mlfq[0]와 mlfq[1]은 Round Robin 알고리즘의 조건에 맞게 quantum을 8, 16으로 지정하였고 mlfq[2]는 FCFS 알고리즘이기 때문에 quantum을 편의상 0으로 설정 했습니다.

- 각 3줄의 큐들은 대기 프로세스들을 보여주고 있고, current process는 현재 CPU가 수행하고 있는 프로세스를 보여줍니다.

- 막대기는 CPU가 얼마만큼의 시간을 프로세스에게 할당하고 있는지를 시각적으로 보여주고 있습니다.

- Process가 랜덤하게 생성되므로 처음에 아무것도 들어오지 않거나 CPU가 한 프로세스가 끝날 때마다 current process: empty 가 출력되게 했습니다.

5. 에필로그

- 조교님이 설명할 때는 간단하게 짤 수 있을 것 같았으나 생각보다 많은 시간이 투자 되었습니다. 18.04.10에 시작해서 18.04.15에 마무리 했습니다. 약 20시간 정도 투자한 것 같습니다. 버전 관리는 https://github.com/jjkyun 에 했습니다.

- MLFQ 특성상 Process가 random하게 생성되지 않으면 CPU가 0-queue에서만 놀게 되는 현상이 생기므로 이를 막기 위해 Process를 Random하게 생성하여 큐에 넣는 부분을 구현했는데, 처음에 어떻게 구현할지 어려웠습니다.

- CPU가 1초마다 어떻게 burst를 수행할지 반복문으로 구현하는 것 또한 상당히 어려웠습니다.