

Operating Systems Project 01

- Implement simple schedulers on xv6-

Test program

기본 구조

- 다음의 코드들을 xv6 디렉토리와 Makefile에 추가하여 컴파일하고 실행합니다.
- 테스트시 실행할 프로그램 소스
 - ml_test.c
 - mlfq_test.c

유의사항

1. 출력은 꼬일 수 있습니다.
2. 숫자가 정확하게 같을 필요는 없습니다.
 1. 논리적으로 올바른 결과가 나와야 합니다.
 2. ppt의 예시와 유사한 결과가 나와야 합니다.
3. PC의 성능에 따라 테스트 프로그램의 상수값들을 조정해야 올바른 결과를 확인 가능할 수도 있습니다.
4. 출력의 결과가 다르지만 명세를 만족시키는 선에서 논리적으로 옳다면 왜, 어떻게 다른지를 위키에 적어주시면 됩니다.

ml_test.c – Test 1

- Sleep이나 yield를 하지 않고 4개의 프로세스가 자신의 출력을 합니다.
- pid가 짝수인 프로세스 2개가 우선순위가 더 높으므로 먼저 출력을 하게 됩니다.
- 짝수인 프로세스끼리는 round robin이므로 타이머 인터럽트가 발생했을 때 실행 흐름이 바뀔 수 있습니다.
- 홀수인 프로세스끼리는 FCFS이므로 번호가 작은 쪽이 먼저 모두 출력된 후에 번호가 큰 쪽이 출력을 시작하게 됩니다.

```
$ ml test  
Multilevel test start  
[Test 1] without yield / sleep  
Process 4  
Process 4  
Process 4  
Process 4  
Process 4  
Process 4  
Process 4  
Process 4  
Process 4  
Process 4  
Process Process 6  
Process 6  
Process 6  
Process 6  
Process 6  
Pr4  
Process 4  
Process 4  
Process 4  
Process 4  
Pocess 6
```

[illegible]

ml_test.c – Test 2

- 프로세스들이 서로 계속 yield를 합니다.
- pid가 짝수인 프로세스들이 먼저 모두 실행됩니다.
- pid가 짝수인 프로세스들끼리는 round robin이므로 yield가 일어날 때마다 번갈아 실행되고, 따라서 거의 동시에 프로세스가 끝나게 됩니다.
- pid가 홀수인 프로세스들끼리는 FCFS이므로 pid가 더 작은 프로세스에서 yield가 발생해도 다시 그 프로세스가 선택되어야 합니다. 결과적으로, pid가 작은 프로세스가 모든 작업을 끝내고 종료된 뒤, pid가 큰 프로세스가 작업을 시작하여 종료하기까지 시간이 걸려야 합니다.

```
[Test 2] with yield  
Process 8 finished  
Process 10 finished  
Process 9 finished  
Process 11 finished
```

[Test 3] with sleep

- [illegible]

한양대학교
HANYANG UNIVERSITY

mlfq_test.c – Test 1

k=2

k=5

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0 ~ k-1)을 각각 카운트합니다.
- 모든 프로세스가 자신에게 주어진 quantum을 전부 쓰고 비슷한 시기에 낮은 레벨로 내려가기 때문에 서로 비슷한 시기에 끝나게 됩니다.
 - pid가 큰 프로세스가 먼저 끝날 수도 있습니다.

```
$ mlfq_test
MLFQ test start
[Test 1] default
Process 4
L0: 41141
L1: 58859
L2: 0
L3: 0
L4: 0
Process 7
L0: 40741
L1: 59259
L2: 0
L3: 0
L4: 0
Process 5
L0: 40331
L1: 59669
L2: 0
L3: 0
L4: 0
Process 6
L0: 40991
L1: 59009
L2: 0
L3: 0
L4: 0
[Test 1] finished
```

```
$ mlfq_test
MLFQ test start
[Test 1] default
Process 4
L0: 16387
L1: 22225
L2: 27365
L3: 34023
L4: 0
Process 5
L0: 16282
L1: 23951
L2: 28062
L3: 31705
L4: 0
Process 6
L0: 16433
L1: 24716
L2: 29147
L3: 29704
L4: 0
Process 7
L0: 18360
L1: 28917
L2: 37250
L3: 7332
L4: 8141
[Test 1] finished
```

mlfq_test.c – Test 2

k=2

k=5

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0 ~ k-1)을 각각 카운트합니다.
- pid가 큰 프로세스에게 더 높은 우선순위 (priority)를 부여합니다.
- 전체적인 시간 사용량은 결국 비슷해지기 때문에 끝나는 시간도 비슷하지만, pid가 큰 프로세스가 조금 더 먼저 끝날 확률이 높습니다.

```
[Test 2] priorities
Process 10
L0: 41854
L1: 58146
L2: 0
L3: 0
L4: 0
Process 9
L0: 41581
L1: 58419
L2: 0
L3: 0
Process 8
Process 11
L0: 40796
L1: 59204
L2: 0
L3: 0
L4: 0
L4: 0
L0: 40546
L1: 59454
L2: 0
L3: 0
L4: 0
[Test 2] finished
[Test 2] yield
```

```
[Test 2] priorities
Process 11
L0: 16246
L1: 22403
L2: 27458
L3: 33893
L4: 0
Process 10
L0: 15596
L1: 23367
L2: 30185
L3: 30852
L4: 0
Process 9
L0: 16485
L1: 24420
L2: 29601
L3: 29494
L4: 0
Process 8
L0: 19008
L1: 28003
L2: 37144
L3: 7654
L4: 8191
[Test 2] finished
```


mlfq_test.c – Test 3

k=2

k=5

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0 ~ k-1)을 각각 카운트합니다.
- pid가 큰 프로세스에게 더 높은 우선순위 (priority)를 부여합니다.
- 각 프로세스는 루프를 돌 때마다 바로 yield 시스템 콜을 호출합니다. 그 때마다 level과 사용 시간이 초기화되므로 계속 L0에 머무르게 되고, 그들 중에서는 pid가 가장 큰 프로세스가 우선순위가 높으므로 pid가 큰 프로세스부터 순차적으로 작업을 완료하게 됩니다.

```
[Test 3] yield
Process 15
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
Process 14
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
Process 13
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
Process 12
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
[Test 3] finished
```

```
[Test 3] yield
Process 15
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
Process 14
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
Process 13
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
Process 12
L0: 20000
L1: 0
L2: 0
L3: 0
L4: 0
[Test 3] finished
```

mlfq_test.c – Test 4

k=2

k=5

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0 ~ k-1)을 각각 카운트합니다.
- pid가 큰 프로세스에게 더 높은 우선순위 (priority)를 부여합니다.
- 각 프로세스는 루프를 돌 때마다 바로 sleep 시스템 콜을 호출합니다. Yield와 마찬가지로 계속 L0에 머무르게 되지만, sleep 상태에 있는 동안에는 스케줄링 되지 않고 다른 프로세스가 실행될 수 있기 때문에 거의 동시에 작업을 마무리하게 됩니다.

```
[Test 4] sleep
Process 19
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
Process 18
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
Process 17
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
Process 16
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
[Test 4] finished
```

```
[Test 4] sleep
Process 19
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
Process 18
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
Process 17
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
Process 16
L0: 500
L1: 0
L2: 0
L3: 0
L4: 0
[Test 4] finished
```

mlfq_test.c – Test 5

k=2

k=5

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0 ~ k-1)을 각각 카운트합니다.
- pid가 작은 프로세스부터 자신이 있기를 원하는 가장 낮은 레벨 (수가 큰 레벨)을 오름차순으로 지정해두어, 그 레벨을 넘어가게 된 순간 yield를 합니다.
 - pid가 가장 작은 프로세스는 L0에만 있으려 하고, 그 다음으로 작은 프로세스는 L1까지, ... k번째로 pid가 작은 프로세스는 Lk-1까지만 있으려고 합니다.
- 더 높은 레벨에만 있으려고 하는 프로세스 (pid가 작은 프로세스)들은 더 우선적으로 스케줄링되므로 먼저 작업을 마무리하게 됩니다.

```
[Test 5] max level
Process 20
L0: 99953
L1: 47
L2: 0
L3: 0
L4: 0
Process 21
L0: 43558
L1: 56442
L2: 0
L3: 0
L4: 0
Process 22
L0: 42738
L1: 57262
L2: 0
L3: 0
L4: 0
Process 23
L0: 42676
L1: 57324
L2: 0
L3: 0
L4: 0
[Test 5] finished
```

```
[Test 5] max level
Process 20
L0: 99953
L1: 47
L2: 0
L3: 0
L4: 0
Process 21
L0: 37989
L1: 61997
L2: 14
L3: 0
L4: 0
Process 22
L0: 26106
L1: 39469
L2: 34420
L3: 5
L4: 0
Process 23
L0: 23423
L1: 30280
L2: 26327
L3: 19967
L4: 3
[Test 5] finished
```

mlfq_test.c – Test 6

- setpriority를 올바르게 호출했는지 잘 판단하는가 여부를 테스트합니다.
- 과제 명세에 적힌 대로 올바른 호출이면 0, 그렇지 않으면 오류 종류에 따라 -1 또는 -2가 반환되어야 합니다.
- wrong으로 시작하는 메시지가 나오지 않아야 합니다.

오류 예시

```
[Test 6] setpriority return value
wrong: setpriority of other: expected -1, got -2
wrong: setpriority of parent: expected -1, got -2
wrong: setpriority of grandson: expected -1, got -2
done
[Test 6] finished
```