Project 02 - Test Program

Implementing simple schedulers on xv6

Due date

2024. 04. 21. 11:59 pm

기본 구조

- 다음의 코드는 xv6 디렉토리와 Makefile에 추가하여 컴파일하고 실행합니다.

- 테스트시 실행할 프로그램 소스
  - mlfq_test.c

유의사항

- 출력은 꼬일 수 있습니다.
- 숫자가 정확하게 같을 필요는 없습니다.
  - 논리적으로 올바른 결과가 나와야 합니다.
  - ppt의 예시와 유사한 결과가 나와야 합니다.
- PC의 성능에 따라 테스트 프로그램의 상수값을 조정해야 올바른 결과를 확인 가능할 수도 있습니다.
- 출력의 결과가 다르지만 명세를 만족시키는 선에서 논리적으로 옳다면 왜, 어떻게 다른지를 위키에 적어주시면 됩니다.
- Test 4 의 경우 구현에 따라 결과값이 나오지 않을 수 있습니다.
  이 경우 본인이 실행한 테스트 케이스 및 설명을 함께 위키에 첨부해주시길 바랍니다.

mlfq_test.c - Test 1

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0~3 및 MoQ)을 각각 카운트합니다.

- 모든 프로세스가 자신에게 주어진 quantum을 전부 쓰고 비슷한 시기에 낮은 레벨로 내려가기 때문에 서로 비슷한 시기에 끝나게 됩니다.
  - L1 큐가 L2 큐보다 우선 순위가 높으므로, pid가 홀수인 프로세스가 대체로 먼저 끝나게 됩니다.

  - pid가 큰 프로세스가 먼저 끝날 수도 있습니다.

```
$ mlfq_test
MLFQ test start
[Test 1] default
Process 5
L0: 23706
L1: 26801
L2: 0
L3: 49493
MoQ: 0
Process 7
L0: 18083
L1: 43300
L2: 0
L3: 38617
MoQ: 0
Process 9
L0: 20926
L1: 31538
L2: 0
L3: 47536
MoQ: 0
Process 11
L0: 25339
L1: 40946
L2: 0
L3: 33715
MoQ: 0
```

```
Process 4
L0: 28248
L1: 0
L2: 61753
L3: 9999
MoQ: 0
Process 6
L0: 27993
L1: 0
L2: 63645
L3: 8362
MoQ: 0
Process 8
L0: 24468
L1: 0
L2: 63949
L3: 11583
MoQ: 0
Process 10
L0: 22432
L1: 0
L2: 62642
L3: 14926
MoQ: 0
[Test 1] finished
```

mlfq_test.c - Test 2

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0~3 및 MoQ)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 19
L0: 24725
L1: 22795
L2: 0
L3: 52480
MoQ: 0
Process 18
L0: 24193
L1: 0
L2: 41049
L3: 34758
MoQ: 0
Process 16
L0: 37118
L1: 0
L2: 44670
L3: 18212
MoQ: 0
Process 17
L0: 30648
L1: 27277
L2: 0
L3: 42075
MoQ: 0
```

```
Process 14
L0: 38522
L1: 0
L2: 61478
L3: 0
MoQ: 0
Process 15
L0: 36119
L1: 43132
L2: 0
L3: 20749
MoQ: 0
Process 13
L0: 37916
L1: 46695
L2: 0
L3: 15389
MoQ: 0
Process 12
L0: 13168
L1: 0
L2: 21588
L3: 65244
MoQ: 0
[Test 2] finished
```

mlfq_test.c - Test 3

- 각 프로세스가 자신이 속해 있는 큐의 레벨(0~3 및 MoQ)을 각각 카운트합니다.
- pid가 큰 프로세스에게 더 높은 우선순위(priority)를 부여합니다.
- 각 프로세스는 루프를 돌 때마다 바로 sleep 시스템 콜을 호출합니다. sleep 상태에 있는 동안에는 스케줄링 되지 않고 다른 프로세스가 실행될 수 있기 때문에 거의 동시에 작업을 마무리하게 됩니다.
- 대부분 L0에 머무르기 때문에, pid가 작은 프로세스가 먼저 끝날 확률이 높습니다.

```
[Test 3] sleep
Process 27
L0: 34
L1: 62
L2: 0
L3: 404
MoQ: 0
Process 20
L0: 500
L1: 0
L2: 0
L3: 0
MoQ: 0
Process 21
L0: 500
L1: 0
L2: 0
L3: 0
MoQ: 0
Process 22
L0: 500
L1: 0
L2: 0
L3: 0
MoQ: 0
```

```
Process 23
L0: 500
L1: 0
L2: 0
L3: 0
MoQ: 0
Process 24
L0: 500
L1: 0
L2: 0
L3: 0
MoQ: 0
Process 25
L0: 500
L1: 0
L2: 0
L3: 0
MoQ: 0
Process 26
L0: 500
L1: 0
L2: 0
L3: 0
MoQ: 0
[Test 3] finished
```

mlfq_test.c - Test 4

- 코드 fork_children3() 내부 본인의 학번을 입력한 후 실행 하시길 바랍니다.

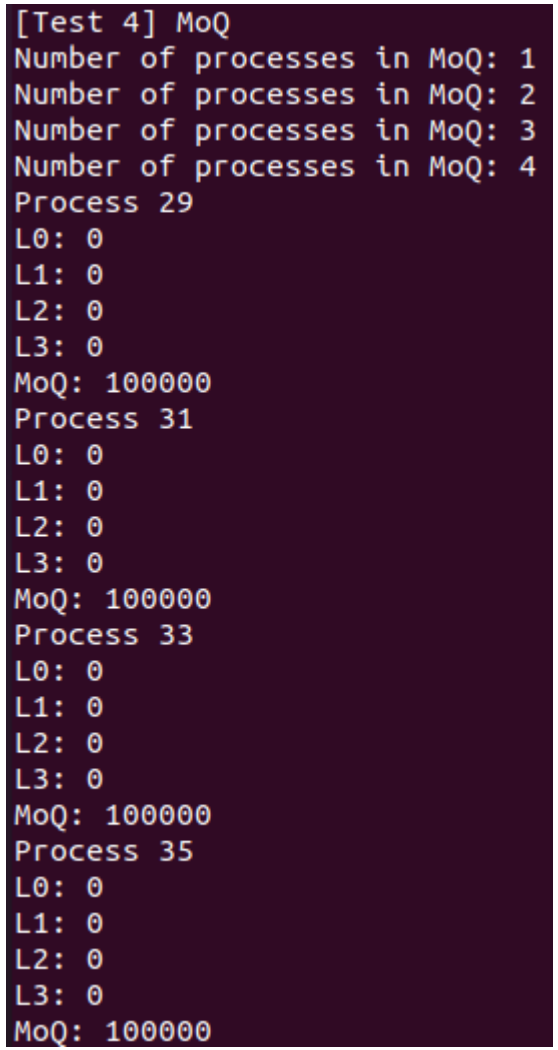
- 각 프로세스가 자신이 속해 있는 큐의 레벨(0~3 및 MoQ)을 각각 카운트합니다.

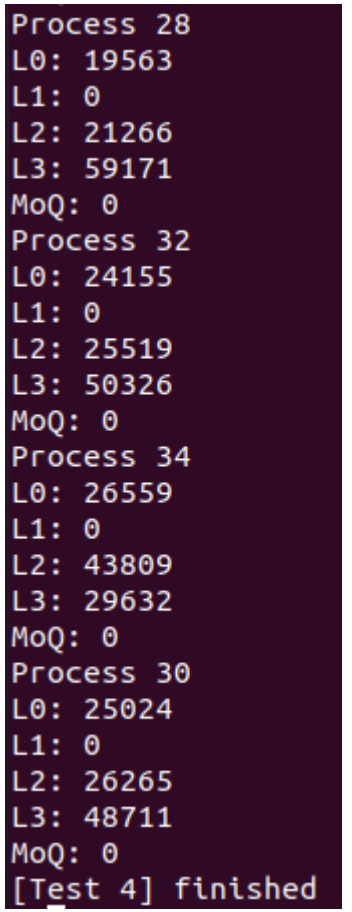
- pid가 홀수인 프로세스를 MoQ에 저장합니다.
- 새로운 프로세스가 추가될 때마다
  MoQ에 존재하는 프로세스의 개수를 출력합니다.

- 마지막 자식 프로세스(pid 36)는 monopolize를 호출하고 exit()를 호출합니다.

- MoQ에 존재하는 프로세스들은 FCFS 방식으로 스케줄링 되므로 pid가 작은 프로세스가 먼저 종료됩니다.

- 이후 기존 MLFQ 스케줄링으로 돌아와 나머지 프로세스들을 실행합니다.





Thank you