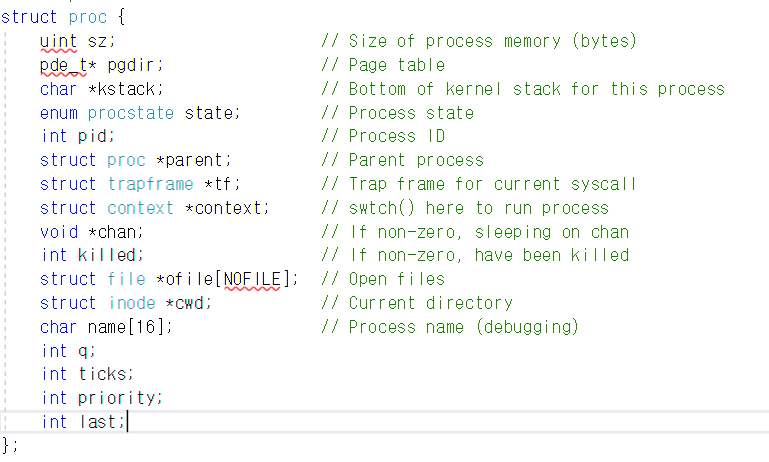
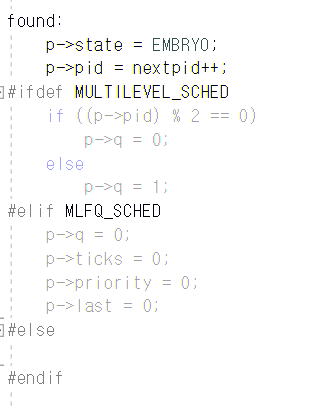
1. Multilevel queue
2. 디자인

Multilevel queue의 구현을 위해 struct proc에 q라는 인자를 추가해 pid가 짝수이면 q의 값을 0으로, 홀수이면 1로 설정해주어 구분해주었습니다. 스케줄러에서는 먼저 q가 0인 프로세스중 RUNNABLE한 프로세스의 개수를 세어줍니다. 만약 이런 프로세스가 하나라도 있다면 q가 0인 프로세스들을 대상으로 ROUND ROBIN을 이용해 스케줄링을 하고, 이런 프로세스가 하나도 없다면 q가 1인 프로세스들을 대상으로 FCFS를 이용해 스케줄링합니다.

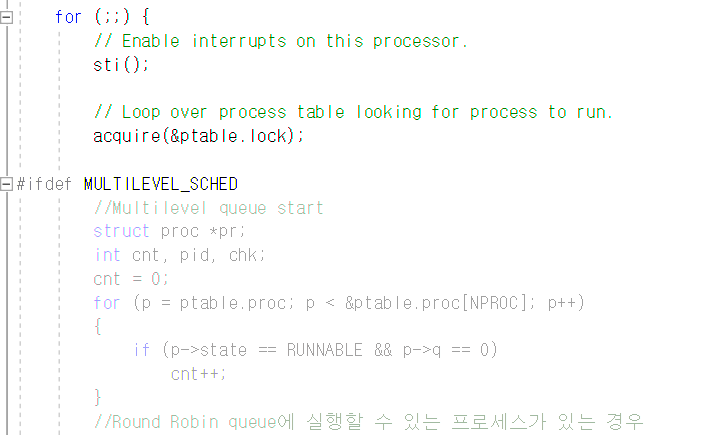
1. 구현



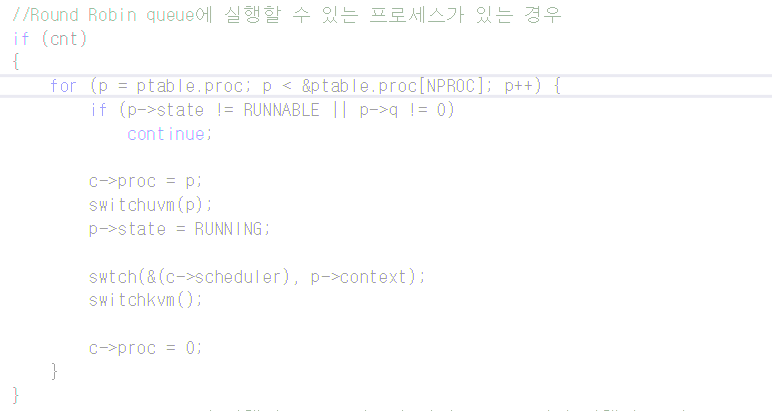
먼저 pid가 짝수인지, 홀수인지에 대해 다른 queue에 담아야 하는데 이를 구분해주기 위해 struct proc에 인자를 추가해주었습니다. Q는 multilevel queue, MLFQ 둘 다 사용하는 인자이고, ticks, priority, last 인자는 MLFQ에서 사용하는 인자입니다.



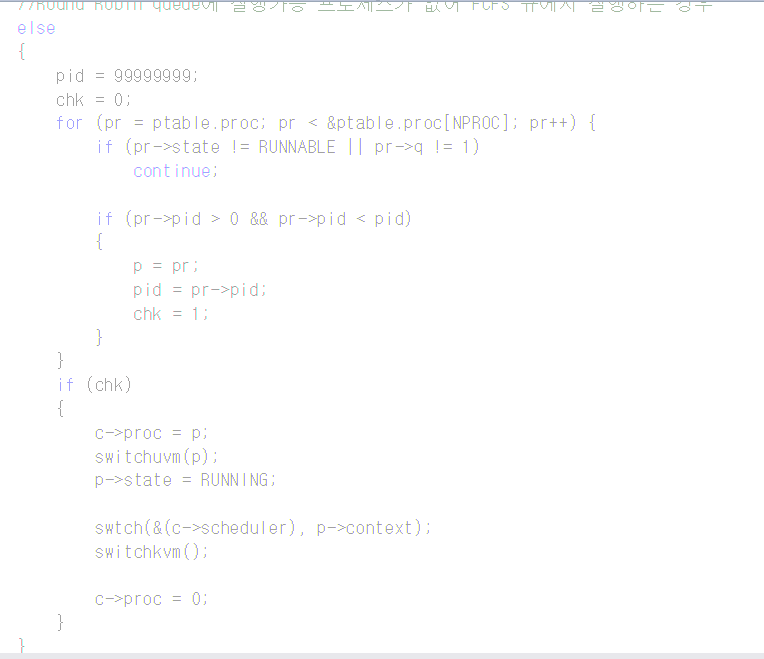
그리고 allocproc 함수 밑에 다음과 같은 조건을 추가해줍니다. Multilevel queue에서는 pid가 짝수이면 q에 0을, 홀수이면 q에 1을 넣는 작업을 해줍니다.



원래 scheduler 함수의 for(;;)무한루프 안의 스케줄링 부분을 다음과 같이 교체해 주었습니다. 먼저 처음 for문을 돌면서 ptable안의 모든 프로세스를 확인하면서 RUNNABLE한 pid가 짝수인 프로세스의 개수를 세서 cnt에 저장해줍니다.



만약 cnt가 0이 아니라면, 즉 pid가 짝수인 프로세스중 RUNNABLE한 프로세스가 있다면 이것을 가지고 원래 코드처럼 ROUND ROBIN을 이용해 스케줄링합니다.



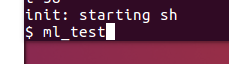
그렇지 않다면, pid가 홀수인 큐 안에서 FCFS를 이용해 스케줄링합니다. P에 pid가 가장 작은 프로세스를 넣고, pid에 이 프로세스의 pid를 넣으며 pid가 가장 작은 프로세스를 찾아줍니다.

시스템 콜 및 그 외의 함수 구현은 MLFQ에서 설명하도록 하겠습니다.

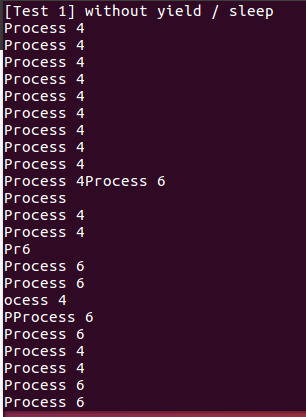
1. 실행결과



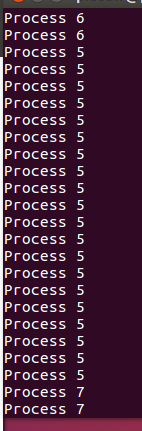




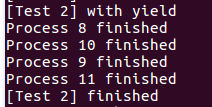
먼저 다음과 같은 순서로 make한 뒤 실행해줍니다



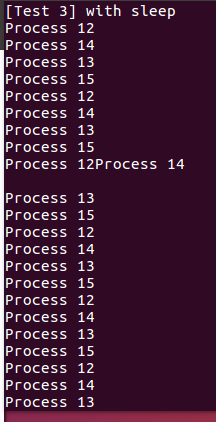
Test 1에서는, pid가 짝수인 queue에서 ROUND ROBIN을 이용하므로, 다음과 같은 결과가 나오게 됩니다.



그리고 짝수 프로세스들의 작업이 모두 끝난 후, pid가 작은 프로세스부터 FCFS를 이용해 pid가 작은 프로세스의 작업이 모두 끝난 후 다음 프로세스가 실행됩니다.



Test 2에서는 짝수인 프로세스는 서로 번갈아가면서 yield되므로 비슷하게 끝나고, 짝수 프로세스의 작업이 모두 끝난 후 홀수 프로세스는 FCFS를 이용해 하나의 프로세스가 끝난 후 다음 프로세스가 실행되므로 하나의 프로세스가 끝나고 어느정도 시간이 지난 후 다른 프로세스가 끝나게 됩니다.



Test 3에서는 프로세스가 sleep하며 실행되므로 짝수인 프로세스들이 실행되다가 sleep하면 홀수인 프로세스가 실행이 되고, 홀수 프로세스가 실행되다가 sleep하면, 이보다 pid가 큰 프로세스도 실행이 되게 됩니다.

1. 트러블슈팅

처음 p->q값을 초기화시켜줄 때 allocproc 함수가 아니라 userinit 함수에서 초기화시켜줬었는데, 나중에 allocproc 함수로 초기화시켜주는 문장을 옮겨주었더니, 정상적으로 작동했습니다.

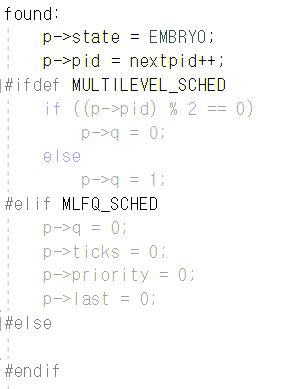
1. MLFQ
2. 디자인

Struct proc에 q, priority, last, ticks 인자를 추가해주었습니다. q에는 큐의 레벨(0~4)를 저장해주고, ticks는 현재 큐에서의 time quantum, priority에는 프로세스의 priority를 저장해주고, 만약 이 프로세스가 실행되었다면, last 인자를 1로 바꾸어 이 큐 안에서 최근에 실행되었던 프로세스라는 표시를 해 주고, time quantum이 넘어가서 다음 레벨의 큐에 담기게 되면 last 인자를 0으로 초기화해주었습니다. K는 MLFQ\_K를 통해 값을 전달받았고, priority를 설정해주는 setpriority함수를 구현해주었습니다. 그리고 priority\_boosting 함수를 만들어서 100 ticks가 지날 때마다, 그리고 스케줄링할 프로세스가 존재하지 않을 때마다 실행시켜주었습니다. 그리고 sys\_yield, sys\_sleep 함수가 실행될 때 q, ticks, last 값을 0으로 초기화시켜주는 과정을 추가해주었습니다. 그리고 큐의 레벨을 리턴해주는 getlev 시스템 콜 함수도 구현해주었습니다.

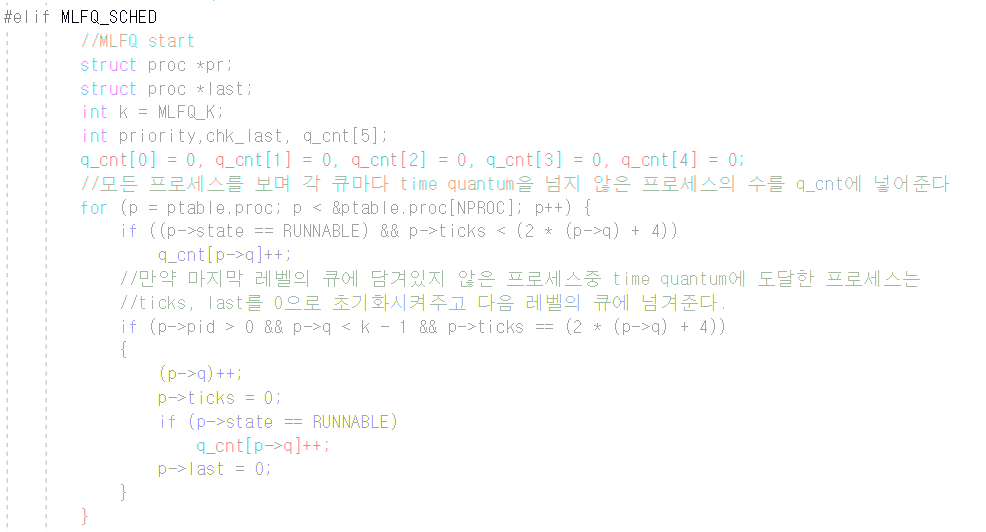
스케줄러 함수의 구현은 먼저 각 레벨의 큐에 RUNNABLE한 프로세스의 개수를 세 주고, time quantum이 지난 프로세스들은 다음 레벨의 큐에 담아주었습니다. 그리고 RUNNABLE한 프로세스가 있는 큐들 중 가장 priority가 높은 큐를 선택해서 그 큐 안에서 이전에 실행한 적이 있던 프로세스가 있다면 그 프로세스를 실행시켜주고, 그렇지 않다면 priority가 가장 높은 프로세스를 실행시켜주었습니다.

1. 구현

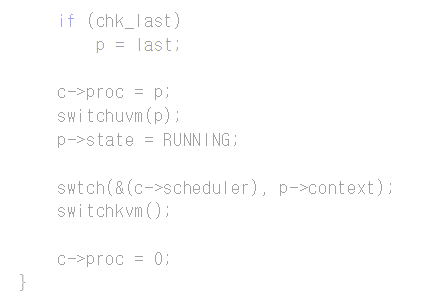
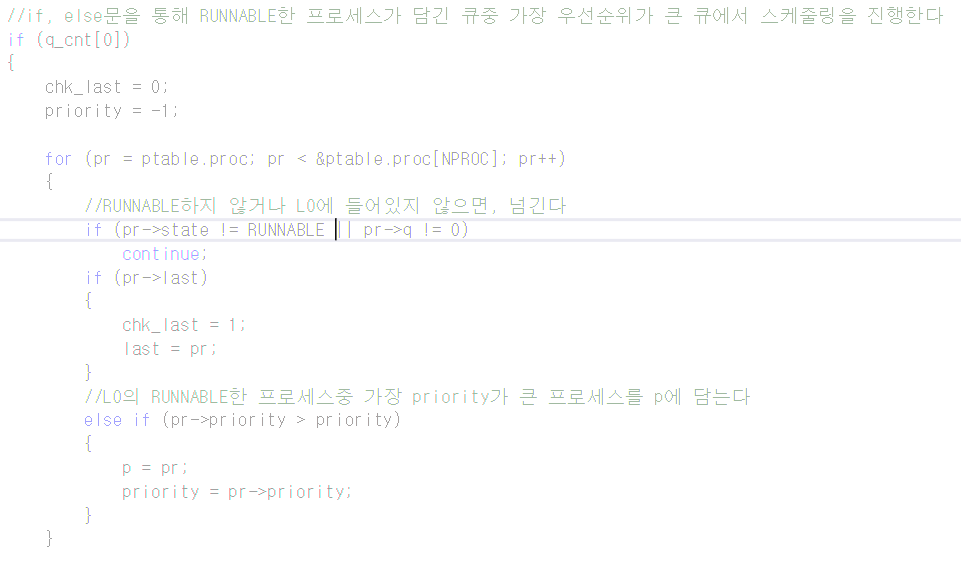
앞에서 말했듯이, struct proc에 q, ticks, priority, last 인자를 추가해주었습니다.



그리고 allocproc 함수 안에서 q, ticks, priority, last 값을 0으로 초기화해주었습니다. 그리고 scheduler 함수의 무한루프 for(;;)안의 내용을 다음과 같이 바꾸어 주었습니다.



먼저 q\_cnt 배열을 만들어서 각 레벨의 큐에 time quantum이 지나지 않고 RUNNABLE한 프로세스의 개수를 저장합니다. 그리고 만약 각 레벨의 큐에 time quantum이 지난 프로세스가 있다면, ticks와 last를 0으로 초기화해주고 다음 레벨의 큐로 옮겨줍니다.

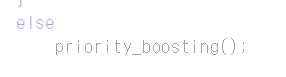


If, elseif, else를 사용해 RUNNABLE한 프로세스가 존재하는 큐들 중 우선순위가 가장 큰 큐에서 스케줄링을 진행합니다. 먼저 chk\_last를 0, priority를 -1로 설정해줍니다. 그리고 모든 프로세스를 돌면서 L0에 들어있는 큐들 중 만약 이전에 실행했던 프로세스가 있다면, trap.c에서 last의 값을 1로 설정하도록 구현했습니다. 그래서 last가 1인 RUNNABLE한 프로세스가 있다면 이 프로세스로 스케줄링하고, 그렇지 않다면 priority가 가장 큰 프로세스를 선택해 스케줄링합니다.

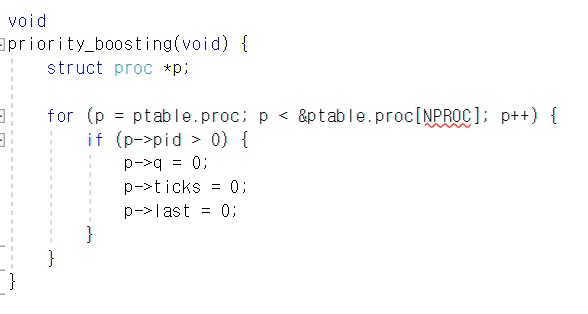




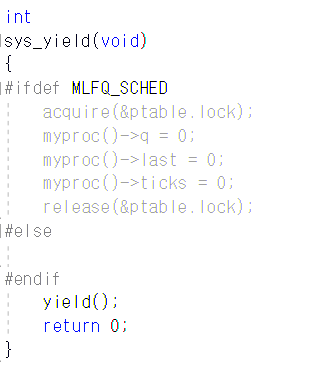
L1,L2,L3,L4도 이와 유사한 방법으로 q\_cnt의 값을 체크하는 부분과 pr->q의 레벨을 체크하는 부분만 수정해줍니다. L2부터는 K의 크기도 체크해주어서 이 level의 큐를 사용하는지도 확인해줍니다.



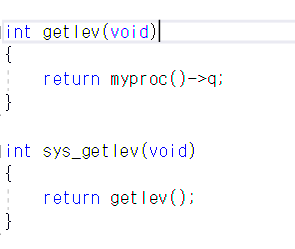
어떤 큐에서도 스케줄링이 일어나지 않는다면 priority\_boosting을 통해 모든 프로세스를 L0 큐로 올려주고, ticks, last 값을 0으로 초기화 시켜줍니다.



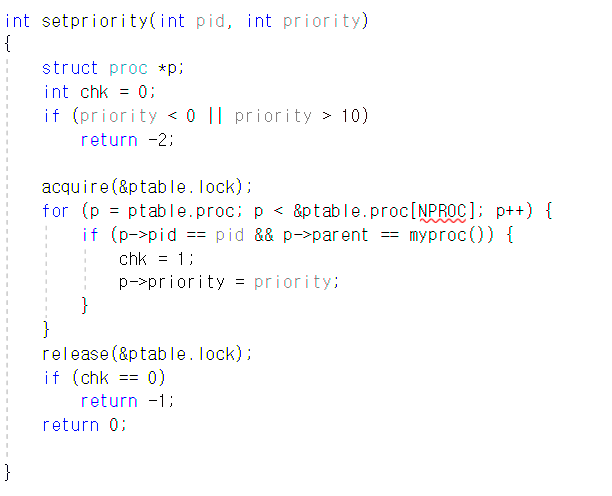
Priority boosting 함수가 호출되면 모든 프로세스를 L0로 올리고, ticks와 last 값을 0으로 초기화시켜줍니다.



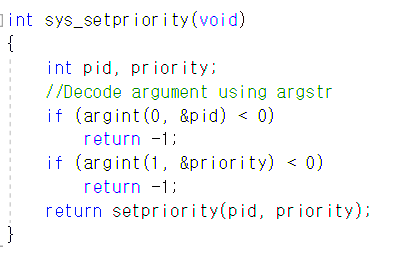
sys\_yield 함수를 만들어서 시스템 콜 yield가 호출되면 프로세스가 L0로 올라가고, last, ticks 값을 0으로 초기화시켜주도록 했습니다.



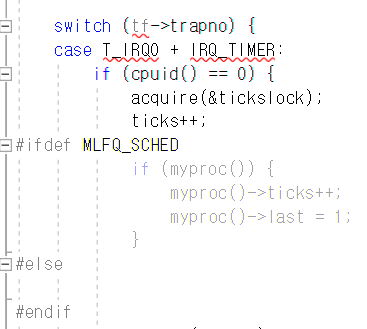
현재 프로세스가 담긴 큐의 레벨을 return해주는 getlev함수와 sys\_getlev 함수를 만들어서 시스템 콜 함수를 만들었습니다.



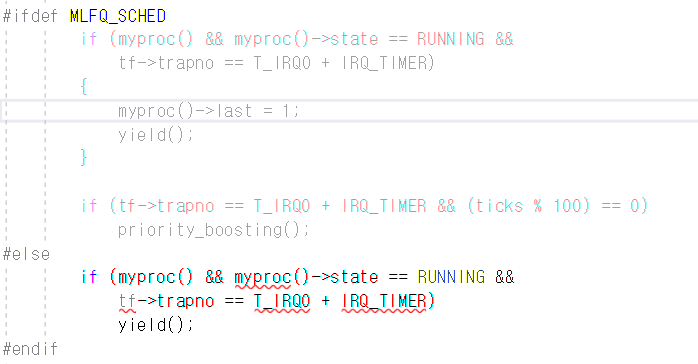
Setpriority 함수를 구현했습니다. 만약 priority 값이 0이상 10이하의 값이 아니라면 -2를 return 해주었습니다. 그리고 만약 priority를 변경해 줄 프로세스가 자신의 자식프로세스가 아니라면 -1을 return해주고, 자식 프로세스라면 priority값을 변경해주고 0을 return 해주었습니다.



그리고 sys\_setpriority 함수를 구현해 시스템 콜 함수를 만들어주었습니다.



Trap.c의 trap 함수에서 프로세스가 실행되고 있을 때 타이머 인터럽트가 발생했을 때, 내 프로세스의 ticks를 1 증가시켜주고, last 값에 1을 넣어주어서 현재 레벨의 큐에서 가장 최근에 실행되었다는 표시를 해 줍니다.



그리고 ticks가 100번 일어날 때마다 priority\_boosting을 실행시켜줍니다.

Sys\_sleep 함수에도 sys\_yield 함수처럼 q, last, ticks를 0으로 초기화시켜서 L0로 올려주는 코드를 추가해주었습니다.

1. 실행 결과

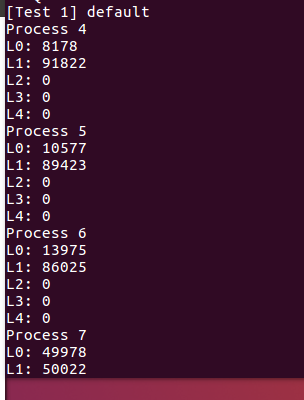




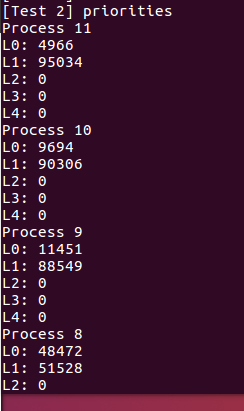




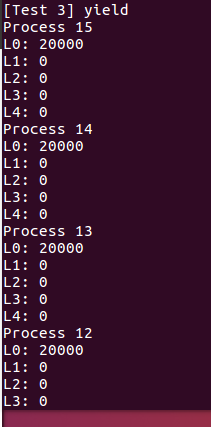
다음과 같은 순서로 먼저 k가 2일 때 프로그램을 실행시켜보았습니다.



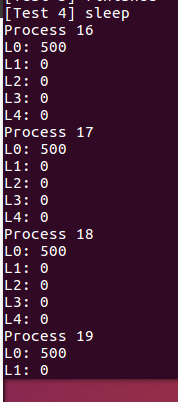
먼저 Test1의 경우 모든 프로세스가 자신에게 주어진 quantum을 모두 쓰고 내려가기 때문에 비슷한 시기에 끝나게됩니다. Pid가 낮은 프로세스가 먼저 끝나는 경향이 있습니다.



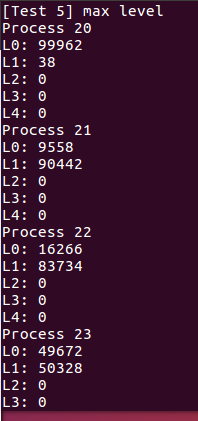
Test2의 경우 비슷한 시간에 끝나지만 pid가 큰 프로세스가 먼저 끝나는 경향이 있습니다.



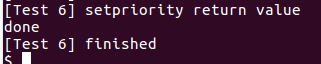
Test3의 경우 yield 시스템 콜을 사용하므로, L0에 프로세스가 머무르게 되고, pid가 큰 프로세스가 먼저 끝나게 됩니다.



Test4의 경우 sleep을 사용해서 L0에 머무르게 됩니다.



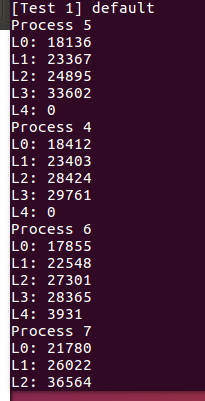
Test5의 결과 pid가 더 작은 프로세스가 먼저 끝나게 됩니다.



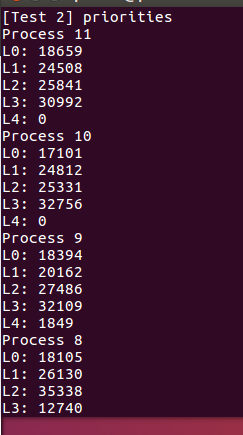
Test6 결과 오류메세지 없이 정상적으로 프로그램이 종료됩니다.



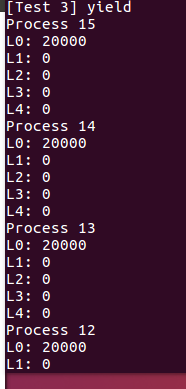
K가 5일 때도 같은 방법으로 실행시킵니다.



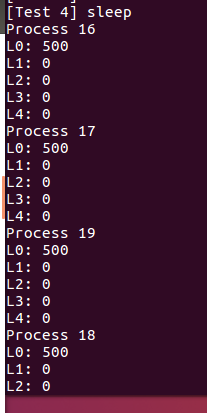
Test1의 결과입니다.



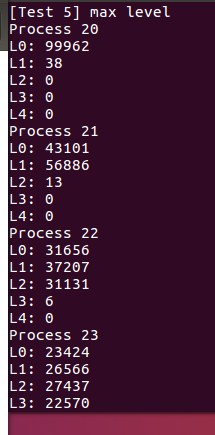
Test2의 결과입니다.



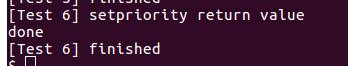
Test3의 결과입니다.



Test4의 결과입니다



Test5의 결과입니다.



Test6의 결과입니다.

1. 트러블슈팅

MLFQ를 짤 때에는 별다른 문제 없이 구현을 완료했습니다.