이번 과제는 운영체제의 'Scheduler'를 구현하는 과제였습니다. 크게 총 6개의 Scheduler를 구현해야 했습니다. 과제를 하기 위해 구조체들을 이해해야 했고, list 함수들을 이해하는 것이 먼저였습니다. 스켈레톤 코드를 읽으면서 어떤 식으로 실행되는지를 이해하려고 노력했습니다.

먼저 첫 번째는 'SJF' Scheduler입니다. SJF Scheduler는 readyqueue에 들어와 수행되기를 기다리는 process들 중에, 가장 timespan이 낮은 process부터 실행시킵니다. 먼저 readyqueue에 존재하는 프로세스들 중에 제일 작은 것을 뽑기 위해 list_for_each_entry함수를 사용하였습니다. List를 돌면서 smallest 프로세스를 갱신하여 가장 짧은 값을 찾아내어 리턴했습니다. 그 후 current가 존재하면, 그 프로세스의 age와 lifespan을 비교해서 계속 수행되야 하는지, 아니면 수행을 마쳤는지를 확인하고 current 또는 NULL을 반환하도록 했습니다.

두 번째는 'SRTF' Scheduler입니다. SRTF Scheduler는 SJF와 마찬가지로 timespan이 제일 작은 프로세스부터 수행시킵니다. 그러나 수행되는 도중에 자신보다 timespan이 작은 프로세스가 들어온다면, 그 프로세스를 먼저 실행시켜야 합니다.

먼저 current가 존재하지 않는 경우 list_for_each_entry함수를 통해 smallest 프로세스를 반환합니다. 그 이후 current가 존재하면, current의 실행필요 여부를 확인하고, readyqueue 리스트에 다시 add했습니다. 이렇게 하면 그 이후 list_for_each_entry함수로 가장 작은 값을 또 찾아내고, 아직 다 돌지 않았으면 자기 자신을 리턴합니다. 또한 더 이상 readyqueue에 프로세스가 없는 경우는 자기가 마지막 프로세스인 것이므로, lifetime 이 끝날 때까지 실행시켜줍니다.

두 번째는 'RR' Scheduler입니다. RR Scheduler는 특정한 시간을 정해서 그 시간만큼 모든 프로세스들이 골고루 수행되도록 합니다. 먼저 제일 처음에 들어오는 프로세스를 수행하고, 그 이후 자기가 끝나지 않았으면 nrrProc변수에 저장한 후 list에 다시 추가해줬습니다. 이렇게 하면 끝날 때까지 돌 수 있습니다. RR에서도 역시 자기가 마지막이면 자기 자신을 끝날때까지 리턴하도록 했습니다.

네 번째는 'Priority' Scheduler 입니다. 'Priority' Scheduler는 우선순위를 지정하고,

우선순위에 따라 가장 높은 프로세스부터 수행하도록 합니다. 먼저 currnet가 없거나, process가 리소스를 사용해야 해서 상태가 WAIT라면, 가장 높은 우선순위를 가진 프로세스를 찾아 수행시킵니다. 그러나 중간에 자기보다 우선순위가 높은 프로세스가 들어오면 바꿔줘야 합니다. 그래서 여기서도 SRTF처럼 다시 리스트에 넣는 방법을 썼습니다. 만약 age < timespan이라면 아직 다 돌지 않은 것이므로, 다시 리스트에 추가해 주고, each_entry 함수로 가장 높은 프로세스를 찾습니다. 아직 수행이 끝나지 않았으면 자기 자신을 반환하여 수행하고, 자기 자신이 마지막 프로세스라면 끝날 때까지 자신을 반환합니다.

다섯 번째는 'PCP Scheduler'입니다. Prio 기반으로 동작을 하다 보면 리소스로 인한 문제가 생길 수 있어서, 우선순위를 조정하여 수행이 되도록 하는 것입니다. 먼저 스케줄러는 prio스케줄러를 이용했고, fcfs acquire와 fcfs realse함수를 수정하여 구현했습니다. Acquire 함수에서는 리소스를 사용할 수 있으면 바로 우선순위를 max로 올려, 가장 먼저 돌아가도록 했습니다. release에서는 waitqueue에 있는 프로세스들 중 가장 우선순위가 높은 프로세스를 waiter 프로세스에 넣어 readyqueue로 들어가게 했습니다.

여섯번째는 'PIP Scheduler'입니다. 이 역시 우선순위 기반으로 돌아갑니다. 다른 점은 리소스를 사용하고 있다고 해서 바로 올리는 것이 아니라, 현재 사용하고자 하는 current의 prio가 owner의 prio보다 높을 때 owner의 prio를 current의 prio로 바꾸어, 가장 먼저 수행하도록 했습니다. Release 부분은 위와 같습니다.

이렇게 6개의 scheduler를 구현하면서, 프로세스들이 어떤 방식으로 수행되고 있는지를 각 스케줄러마다 이해하는데 정말 큰 도움이 되었습니다. 그러나 prio구현에서 큰 어려움을 겪어 많은 시간을 소비한 점이 아쉽습니다. 이번 과제는 list함수들도 많이 알게 되어서 정말 좋았고, 교수님께서 만들어두신 process변수들을 많이 이용했는데, 실제 프로세스가 어떻게 구성되어 있을 것이고 어떻게 값을 변화해 가며 동작할 것이다 같은 걸 직접 생각해보며 과제를 할 수 있었기에, 이론을 이해하는데 굉장히 도움이 많이 되었습니다.