**SFJ**

말 그대로 lifespan이 짧은 프로세스부터 실행한다.

Non-Preemptive 라서 별 다른 설명이 필요 없다.

**STCF**

Lifespan으로 priority을 정한다. SFJ랑 다르게 preemptive하다. Current가 없을 때는(시작) list\_for\_each\_entry를 사용하면서 가장 짧은 프로세스를 고른다. Current가 있으면 두 가지 경우로 나뉜다. List\_empty인 경우, list\_empty가 아닌 경우. List empty인 경우, current가 끝나지 않았으면 current을 리턴하고, 끝났으면 NULL을 반환한다. List empty가 아닌 경우, current.lifespan – current.age로 가장 짧은 프로세스를 선택한다. 자세한 건 코드에 있고, 대략적인 개요는 이걸로 끝이다.

**RR**

기본적으로 RR은 FIFO 형태로 스케줄링을 해준다. 또한 preemptive 하기 때문에 current가 존재하는 경우, 이를 readyque에 넣고 새로 프로세스를 선택해야 한다.

**Priority Scheduler(Basic)**

여기서부터는 acquire, release 함수도 짜줘야 한다. 우선 Basic에서 사용하는 acquire release 함수는 전에 사용하던 것과 똑같다. 이것도 preemptive라서 current가 존재하는 경우, 이를 readyque에 넣고 새로 프로세스를 선택해야 한다. 별로 새로운 내용은 없다

**Priority Scheduler(Aging)**

Priority schedule basic이랑 근본적으로는 같은 원리이나, 프로세스마다 priority 바꿔주면 된다. 특별한 원리가 있는게 아니기 때문에 설명은 생략.

**PCP**

Acquire이랑 release을 새로 만들어줘야 한다. 근본적으로 위에서 사용한 것들과 거의 차이가 없다. 하지만 acquire 과정에서 owner의 priority을 MAX로 설정해줘야 하고, release 과정에서 prio\_orig로 다시 바꿔줘야 한다. 또한 process blocked 되면 그 다음에 선택될 프로세스의 priority을 MAX로 설정해야 한다.

**PIP**

Acquire에서 BLOCKED당하면 owner의 priority을 blocked 당한 프로세스의 priority로 바꿔준다(blocked priority > owner priority)일 때. Release할 때는 prio을 prio\_orig로 바꿔준다. 그 외 나머지 코드는 정말 다른 것들과 비슷 혹은 똑같다.

**PIP adv2 설명**

우선 tick이 0일 때는 프로세스1 밖에 없다. 프로세스 1이 실행되고, 프로세스 1은 리소스(1, 2, 3, 4)에 접근해서 사용한다. 이제 다른 프로세스들은 해당 리소스가 release 될 때까지 사용 할 수 없다. Tick이 1일 때 프로세스 2가 fork 된다. 프로세스2 가 리소스 1을 접근하지만 실패한다. 이때 프로세스1의 priority는 5가 된다. Tick이 2일 때는 프로세스 3,4 가 fork 된다. 프로세스 4는 프로세스 1에 접근하지만 실패한다. 이때 프로세스 1의 priority는 30이 된다. Tick이 3이 되면, readyque에 프로세스 1,3밖에 없다. 프로세스 2,4는 readyque에 있기 때문이다. 그럼 당연히 프로세스 1의 priority가 30이니까 실행된다. 리소스 1이 release된다. 이제 프로세스 2,4가 readyque로 이동한다. 또한 프로세스 1의 priority가 기존의 priority로 돌아간다. 이제 readyque에 프로세스 1,2,3,4가 모두 다 있다. 이런 상황에서는 프로세스 4가 priority가 가장 높으니까 실행된다. Tick 5에 프로세스 4는 끝난다. 이제 readyque에는 프로세스 1, 2, 3 만 있다. 이런 상황에서 프로세스 3이 가장 우선 순위가 높다. 프로세스 3은 tick이 6까지 그냥 실행하고, 7초에 리소스 2를 접근하다가 실패한다. Waitque로 들어간다. 리소스 2는 프로세스 1이 가지고 있으니 프로세스 1의 우선순위는 10이 된다. Readyque에 프로세스 1, 2가 있다. 허나 프로세스 1의 우선수위가 더 높으므로 프로세스 1이 실행된다. Tick이 8이 될 때 리소스 2가 release 되고, 프로세스 1의 우선 순위가 원래대로 돌아오고, 프로세스 3도 다시 readyque에 들어온다. 당연히 이제 레디큐에 3개의 프로세스가 있고, 프로세스 3이 가장 우선순위가 높다. 프로세스 3이 실행되고 tick이 11일 때 프로세스 3은 종료된다. 이제 레디큐에 프로세스 1,2 가 있는데, 프로세스 2가 우선순위가 더 높다. 그러므로 실행되고, 종료 될 때까지 실행한다. 이제 프로세스 1만 남았으므로, 프로세스 1을 실행하면 된다.

**Lesson learned**

솔직히 새로 배운 것은 없는데, pip adv2 에서 프로세스 1이 0 tick에 fork 되었는데도 가장 마지막으로 종료 되고, 리소스 3,4에 제일 나중에 접근하는 것을 보고 pip도 항상 최선의 결과, 효율적인 결과를 가져오진 않다 라는 것을 느꼈다.