OS - Assignment 3

201720715 박주현

1. 무엇을 했고, 어디까지 진행했는지

- spinlock: spinlock은 busywaiting 으로 구현하였다. Spinlock을 구현하기 위해 lock.c에 있는 함수를 채웠다. 수업시간에 배운대로 (강의노트를 참고함) init은 lock_num을 초기화 해주었고, acquire과release에선 compare_and_swap함수를 통해 값을 atomic하게 바꿔주었다. 또한 spinlock의 enque deque를 구현하기 위해, 먼저 queue를 만들어주었다. Queue는 전역 변수로 head와 tail buffer를 가지고 있다. Circular queue를 구현하기 위해 구글링을 참고하였다. 이때 goto문을 사용하였다.

-mutex: 교수님의 라이브가 많이 도움이 됐다.

먼저 acquire에선 TAILQ로 쓰레드를 관리하기 위해, 먼저 acquire해주려면 spinlock으로 잡아 크리티컬 섹션을 보호하였다. 그후 mutex 구조체에 av라는 변수를 두었다. Av가 1일 때, 사용 가능하다고 판단해 쓰레드를 TAILQ의 TAIL에 연결해주었고 시그널을 보내고 release로 풀어주었다. 이 때, flag를 이용해 깨어났는지 안 깨어났는지 확인해주었다. 만약 av가 사용가지 않으면 av를 1로 바꾸고, 잡아 놓은 spinlock을 release해주었다. release에선 마찬가지로 acquire로 spinlock을 잡고 TAILQ가 empty가 아니면 존재하는 것이므로 remove해주었고, again goto문으로 깨어났는지 확인하였다. 이때 signal을 잃어버리지 않도록 sleep을 사용하였다. Enqueue dequeue는 spinlock과 마찬가지로 구현하였다.

-semaphore: semaphore는 counting 뮤텍스이다. 3가지 세마포어 full,empty,mutex을 선언하여 사용하였으며, empty는 n, full은 0 mutex는 1로 S값을 초기화하였다. Queue의 전체적인 구조는 뮤텍스와 같았다. 뮤텍스 세개를 이용하여 wait signal함수를 사용하여 enqueue dequeue를 구현하였다. wait함수에서는 sem->value--; 를 통해 count하였고, 이 값이 0보다 작으면 TAILQ에 추가해주었다. Signal은 반대로 sem->value++한 값이 0보다 작을 때 TAILQ_REMOVE를 해주었고 pthread_kill을 사용하였다. 이 모든 과정에서 spinlock으로 뮤텍스와 마찬가지로 크리티컬 섹션을 보호해주었다.

2. 이번 과제를 진행하면서 배우게 된 것

: 수업 시간에 배웠던 개념들을 직접 구현함으로써 어떻게 되는지 어떤 문제가 생길 수 있는지 확인할 수 있는 시간이었던 것 같다.

3. 이번 과제에 대한 피드백

: office hour에 갔더니 특정 몇몇 학생들이 자리를 비켜주지 않고 자리를 몇 시간동안 차지하였고, 20분을 기다리다 결국 돌아갔다. 또한 다른 학생들이 어떻게 해요 이해가 안가요 fail 떠요 등등의 질문에서 그냥 알려주시고, 심지어 어떤 학생에게는 코드를 메일로 보내보라고 하셨다고 들었다. 또 오피스 시간이 원래 7시까지였는데 10시까지 연장하셨으면서 그에 대한 재공지도 없었고, 따라서 office hour가 끝난 줄 알아서 찾아 뵙지 못하였다. 또 그렇게 연장한 시간에 아까 말한 특정한 학생들과 피자를 먹었다고 들었는데, 그렇게 되면 office hour의 취지에 어긋나지 않나는 생각이 들었다.