운영체제 Mylock 과제 보고서

201720720 조민재

이번 과제는 spinlock, mutex, semaphore를 통해 circular queue system에서 발생하는 문제를 해결하는 것이었다.

먼저 스핀락을 구현하기 위해, spinlock 구조체를 만들었다. 구조체에는 락을 잡았는지를 확인하는 값만 넣어주었다. 잡았을 때 1로 바꾸어 주고, 아무도 락을 잡지 않았으면 0인 상태로 설정하였다. 다음으로 acquire_spinlock 함수에서 compare_and_swap함수를 사용하여 atomic하게 락을 잡게 하였다. 락을 풀어줄 때는 락의 점유 상태만 변경하면 되므로 스핀락 값을 0으로 바꾸는 코드를 넣어주었다.

다음으로 링버퍼를 구현하였는데, 링버퍼 구조체를 선언하고 거기에 데이터를 넣을 큐, 반환할 데이터를 넣을 변수, 그리고 꽉 찼음을 확인하기 위한 in과 out변수를 만들어주었다. 그리고 init_ringbuffer함수에 이 변수들을 초기화해주었다. 큐는 nr_slots+1만큼 할당해주었는데, 이것은 링버퍼의 한 칸을 비우기 위한 것이다. 엔트리개수+1만큼 할당을 해주고, 실제 들어가는 값은 엔트리만큼 들어가게 하였다.

다음으로 스핀락을 이용하여 enqueue와 dequeue하는 함수를 구현했는데, 스핀락 구조체를 하나 생성해주고, 초기화 하는 함수에 init_spinlock을 해주어서 락을 초기화해주었다.
Enqueue_using_spinlock에서는 goto문을 사용하였는데, 락을 얻었는데 버퍼가 다 차있을 때, 락을 해제하고 버퍼가 빌 때까지 반복하기 위한 것이다. 버퍼가 빈 상태에서 락을 얻었으면, value값을 버퍼에 넣어주고, 다음 넣을 인덱스를 하나 더해준다. 그 후에, 스핀락을 해제한다.
Dequeue_using_spinlock도 비슷한 원리로, 버퍼에 값이 있을때까지 락 잡고 놓기를 반복한 후에, 락을 얻고 버퍼에 값이 있으면 그 값을 빼서 반환한다. 다음에 값을 뺄 인덱스를 하나 더해주고 락을 놓게 된다. 마지막으로 fini_using_spinlock에서 할당한 링버퍼의 큐를 할당해제해주었다.

그 다음, 뮤텍스를 구현하기 위해 뮤텍스 구조체 안에 락을 잡았는지 확인하는 변수, 뮤텍스의 스핀락, 쓰레드를 대기 큐에 넣기 위한 테일큐를 만들어주었다. 초기화 함수에는 락 변수를 0으로 해주었고, acquire_mutex 함수에서는 일단 critical section에 진입하기 위해 스핀락을 잡게 했다. 그 후에 뮤텍스값이 1이면 누군가 락을 잡았다는 뜻이므로, 해당 쓰레드는 자기 자신을 테일큐에 넣고, 스핀락을 해제한다. 그 후에 누군가 자신을 깨울 때까지 pause()상태로 바뀐다. 그리고 pause된 상태를 알리기 위해 flags값을 1로 바꾸어 주었다. 누군가 깨워줄 때 시그널을 전달해야 하므로, 사용자 지정 시그널을 받으면 시그널 핸들러를 실행하는 코드를 pause()바로 위에 넣어주었다. 그리고 만약 뮤텍스 값이 0이라서 현재 가져간 쓰레드가 없다면, 해당 쓰레드가 뮤텍스를 가져가고, 뮤텍스값을 1로 바꾼 후 락을 해제한다.

Release_mutex 함수에서는 역시 처음에 스핀락을 잡고, 만약 테일큐가 비어있지 않으면, 즉 어떤 쓰레드가 대기하고 있다면, 테일큐의 제일 앞에 있는 쓰레드를 꺼내서 pthread_kill로 사용자 지정

시그널을 보내서 깨운다. 시그널 핸들러는 함수 바깥에 만들어 주었다. 그런데 여기서 아까 쓰레드가 대기큐에 들어갈 때, 락을 놓고 pause()를 바로 실행하면 문제가 되지 않지만, 락을 놓자마자 다른 쓰레드가 잡고 시그널을 보낸다면, 아직 pause된 쓰레드가 없기 때문에 실패하게 된다. 그래서 여기서 usleep으로 잠깐의 시간이 지난 후에, pause상태를 확인하는 flags가 0이면, 아직 깨어나지 않았다는 뜻이므로 깰 때까지 pthread_kill을 반복하는 goto문을 사용했다. 그 후에 쓰레드가 깨어났으면, 스핀락을 놓는다. 만약 대기큐에 아무것도 없다면, 그냥 뮤텍스 값을 0으로 바꾸고 스핀락을 놓는다.

이제 이것을 링버퍼에 적용하는 것은 스핀락과 구조를 동일하게 하였다. goto문으로 버퍼의 상태를 확인한 후에 뮤텍스를 얻고 값을 링버퍼에 집어넣거나 반환해주고 마지막에 뮤텍스를 해제해주었다.

마지막으로 세마포어는 뮤텍스를 확장한 것으로 구현했다. 뮤텍스의 경우엔 락을 잡고 놓는다의 상태 0과 1, 즉 카운트가 binary 값이었던 반면에 세마포어는 접근할 수 있는 쓰레드의 개수를 초기값으로 설정해놓고, 그 이상의 쓰레드가 접근하려 하면 대기큐에 집어넣는 방식이다. 따라서 wait_semaphore 함수에는 맨 처음에 들어갈 수 있는 개수인 value값을 -해주고, 만약 더 이상 들어갈 수 없으면 뮤텍스와 똑같이 대기큐에 집어넣어 주었다. 반대로 signal_semaphore 함수에는 맨 처음에 value값을 ++ 해주고, 대기 큐에 쓰레드가 있으면 깨워주는 방식으로 구현했다.

세마포어를 링버퍼에 적용하는 것 역시 스핀락, 뮤텍스와 마찬가지로 같은 구조로 구현했다. 초기화 할때는 접근할 수 있는 개수, 즉 value값을 임의로 설정해주었다.

이번 과제에서 배운 점은, 멀티 쓰레드 프로그램에서는 디버깅이 매우 어렵다는 것이다. 대체 어디서 에러가 나는지 모르겠어서 어려움이 많았다. 처음부터 견고하게 잘 짜야 한다는 것을 배웠다. 또, 이렇게 동기화를 해주지 않으면 나중에 프로그램에 치명적인 오류가 생기므로 synchronization이 중요하다는 것을 몸으로 깨닫게 되었다. TAILQ라는 편리한 매크로도 알게 되었다.

이번 과제는 교수님이 쉽다고 하셨는데 전혀 쉽지가 않았다. 오히려 저번 과제보다 어려웠던 것 같다. 특히 어려웠던 점은 왜 돌아가고 왜 안돌아가는지 그때그때 상황 파악이 어려웠던 것이었다. 다음에는 디버깅이 좀 수월한 과제였으면 좋겠다. 테스트를 통과하는지 확인하는 방식은 간단하고 편해서 좋았다. 다음 과제도 이렇게 테스트가 수월했으면 좋겠다.