운영체제 1차 프로젝트 보고서

소프트웨어학과

2017 23272

안창희

<<목차>>

1. 동작실험 결과

2. 자가진단표

3. 토의 및 느낀점

운영체제 과제 2

1. Producer/Consumer 프로그램

동작시험 결과 (프로그램이 어느 수준까지 동작되었는지 설명하고 근거 자료를 제시)

동기화 기능을 구현하지 않은 채로 생산과 소비를 하는 스레드를 병행으로 동작시키는 프로그램을 만들었다.

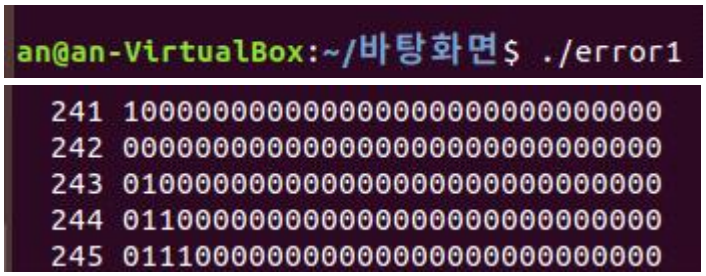
프로그램의 개요는 Readme의 내용과 동일하게 30개의 item을 처리할 수 있는 작업대에서 생산자와 소비자가 mutual exculsive하게 동작하도록 하는 것이며, 생산과 소비를 합한 총 작업량이 1000회라고 가정했을 때,

생산 1차시, 소비 1차시, 생산 2차시, 소비 2차시, ...

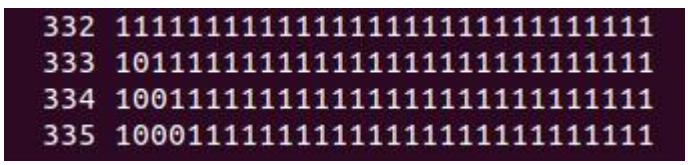
이런 식으로 번갈아서 실행하면 생산 16차시와 소비16차시가 끝난 직후 남은 작업량은 20회가 되므로 생산 17차시의 20회를 만큼을 수행하고 프로그램은 종료된다.

하지만 이 경우는 mutual exclusive한 성질이 완벽히 구현되었을 때의 가정이며 실제 프로세스의 가동에서는 오류가 발생하였다.

동기화가 구현되지 않는 error1.c 의 코드의 프로그램인 error1의 실행결과는 다음과 같다.

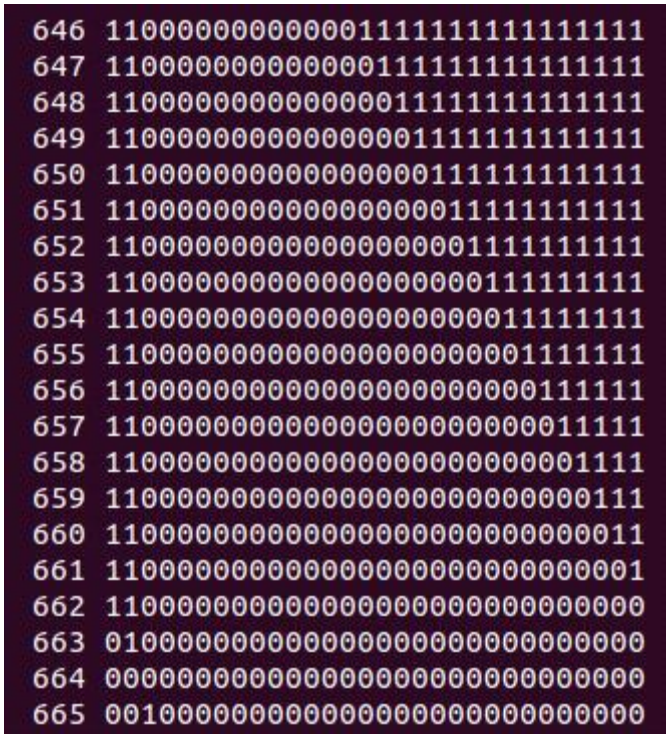


241, 242로 이어지는 경우에서 생산자 스레드만이 동작해야 하나 242에서 소비자 스레드가 동작하여 30칸의 작업대에서 첫 번째 칸이 비어버리게 되었다.

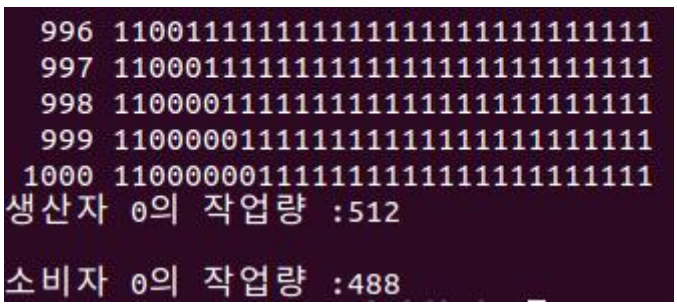


위의 경우와 마찬가지로 두 번째 칸이 비어버리게 되었다.

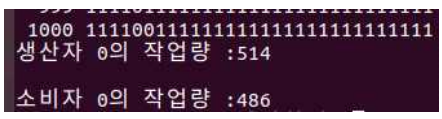
똑같은 경우가 393에서도 포착된다.

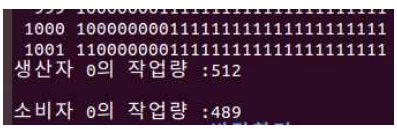


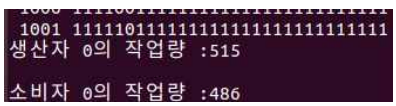
30회 작업 수행마다 번갈아가며 생산과 소비를 반복해야하는데, 생산과 소비의 과정이 뒤얽혀서 작업대를 완전히 비우거나 완전히 채운 상태로 context switch가 일어나지 않는 장면이다.

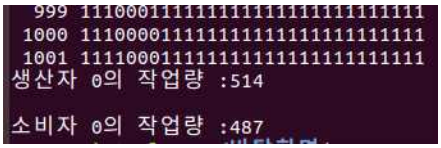


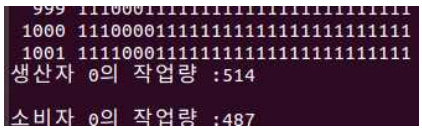
마지막까지도 문제발생이 계속되었으며 스레드의 동작에 대한 mutual exclusive 조건이 만족되지 않아 프로세스의 실행마다 생산자와 소비자 스레드의 작업량이 계속해서 변했다.





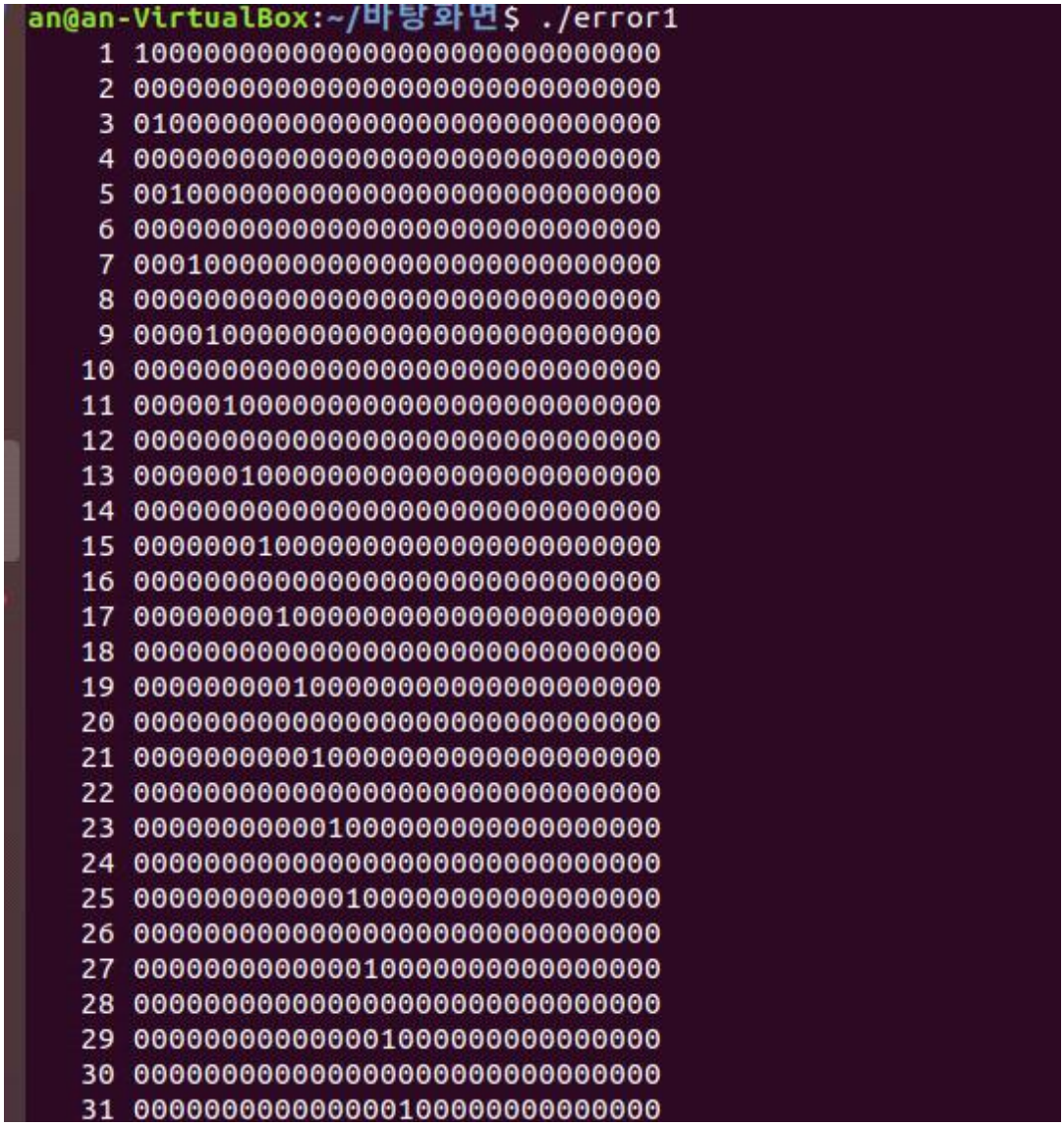






위와 같이 race condition의 발생 시각과 그에 따른 작업량은 일정하지 않게 계속 달라졌으며

프로세스 가동 중 두 스레드의 동기화 기능이 있어야 producer-consumer 문제를 해결할 수 있음을 알 수 있었다.

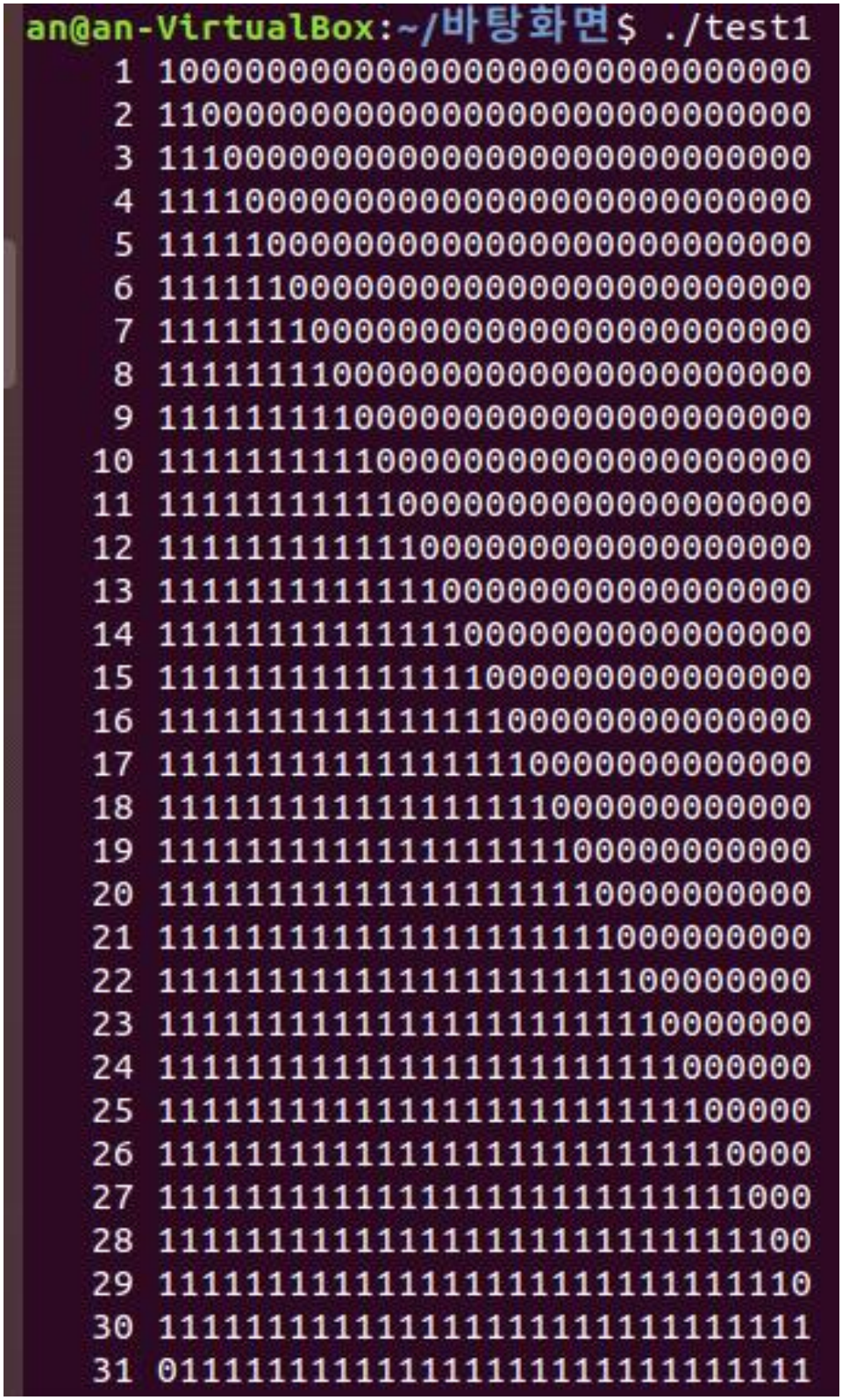


위는 sleep() 함수를 통해 mutual exclusive 한 context switching을 구현하려고 수정후 가동시킨 error1 프로그램이다.

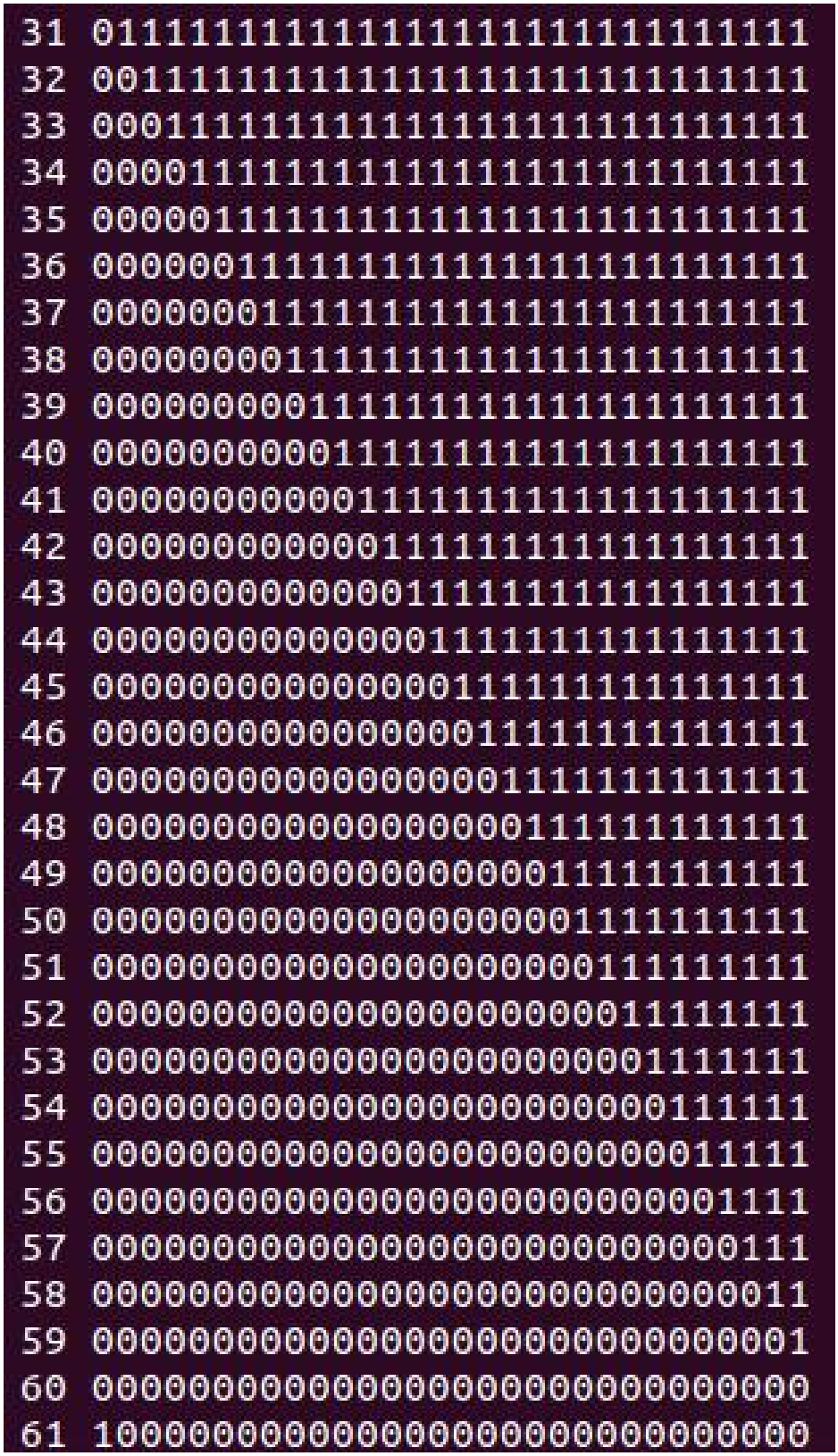
선형 큐와 같이 자료가 한칸씩 뒤로 밀리며 본래 30개 item의 단위로 context switch가 일어나야 하는데 1초에 1번씩만 이루어져 시간이 너무 오래 걸리고 작업대를 전부 활용할 수 없는 문제가 발생했다.

mutual exclusive한 성질은 일부 구현할 수 있었으나 의도한 대로 프로그램을 동작시킬수 없었다.

따라서 sleep을 지우고 mutex로 mutual한 개념을 구현후 test1 프로그램으로 동작시켰다.



생산완료 및 소비의 시작
1~30은 생산 스레드의 동작부이며
31 부터는 소비 스레드가 동작한다.



소비완료 및 생산의 시작

31~60은 소비 스레드의 동작부이며 61부터는 다시 생산 스레드가 동작한다.

30개단위로 생산과 소비가 완벽히 mutual exclusive 하게 동작하였으며 작업량 또한 이상적으로 예측한 바와 일치하였다.

자가진단표

구현 단계 (단계에 체크)	내용	주요 bug 및 프로그램의 문제점
10□	동기화 기능 구현하여 문제점(race condition) 해결 확인	프로그램 설계 시 생산과 소비 작업의 횟수가 item의 개수에 따라 유동적으로 입력되었으면 더 이해시키기 쉬운 프로그램일 것 같다.
9□	동기화 기능 동작 확인	
8□	동기화 기능 구현	
7□	동기화 제외한 프로그램의 문제점(race condition) 확인	
6□	동기화를 제외한 프로그램 동작 확인	
5□	동기화를 제외한 프로그램 일부 동작	
4□	동기화를 제외한 프로그램 완성	

2. Reader/Writer 프로그램

동작시험 결과 (프로그램이 어느 수준까지 동작되었는지 설명하고 근거 자료를 제시)

동기화 기능을 구현하지 않고 read와 write 스레드를 생성하여 동작시키는 프로그램을 만들었다.

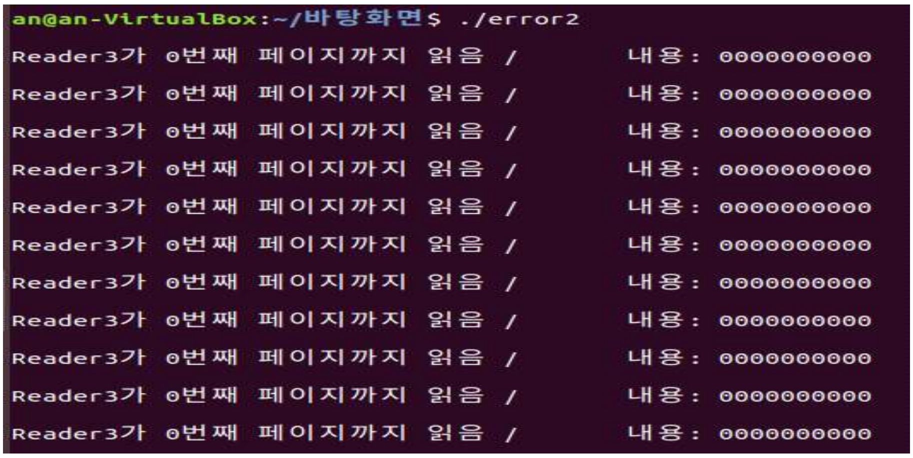
목표로하는 프로그램의 동작은 다음과 같다.

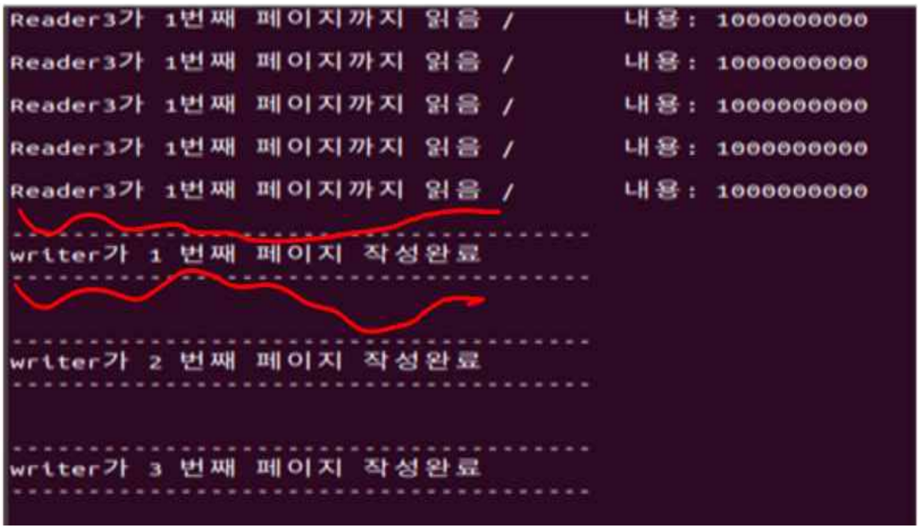
----

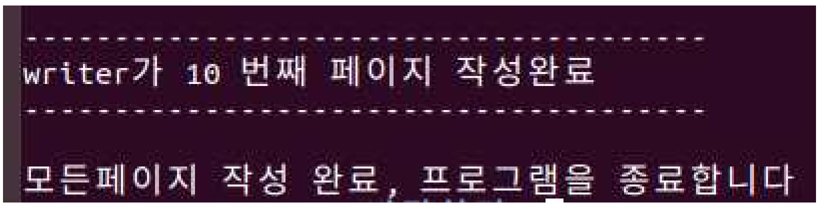
writer가 1페이지 작성
reader0 , reader1, reader2가 읽음

writer가 2페이지 작성
reader0 , reader1, reader2가 읽음

....반복







하지만 동기화 기능이 없이 동작한 프로그램은 위와 같은 문제를 일으켰으며 race condition 또한 포착되었다.

빨간색으로 줄쳐진 부분을 보면 writer가 쓰기를 하기도 전에 reader가 내용을 읽는 부분이 나온다.

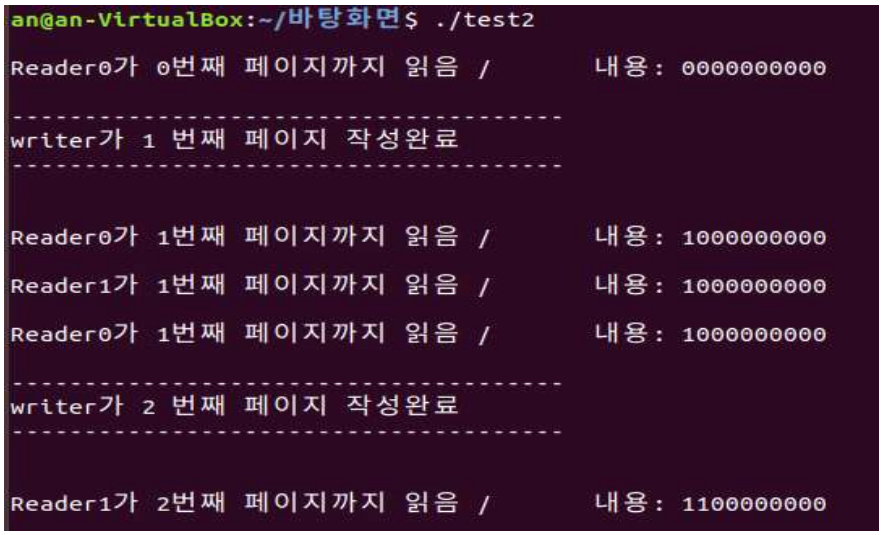
하지만 내용에 첫 번째 페이지에 해당하는 부분이 1로 작성이 완료됨을 표시하는 것으로 보아 병행으로 동작하는 스레드 끼리 순서가 얽히면서 위와 같은 상황이 발생했다고 볼 수 있다.

결과를 통해 추정되는 스레드의 동작순서는 다음과 같다.

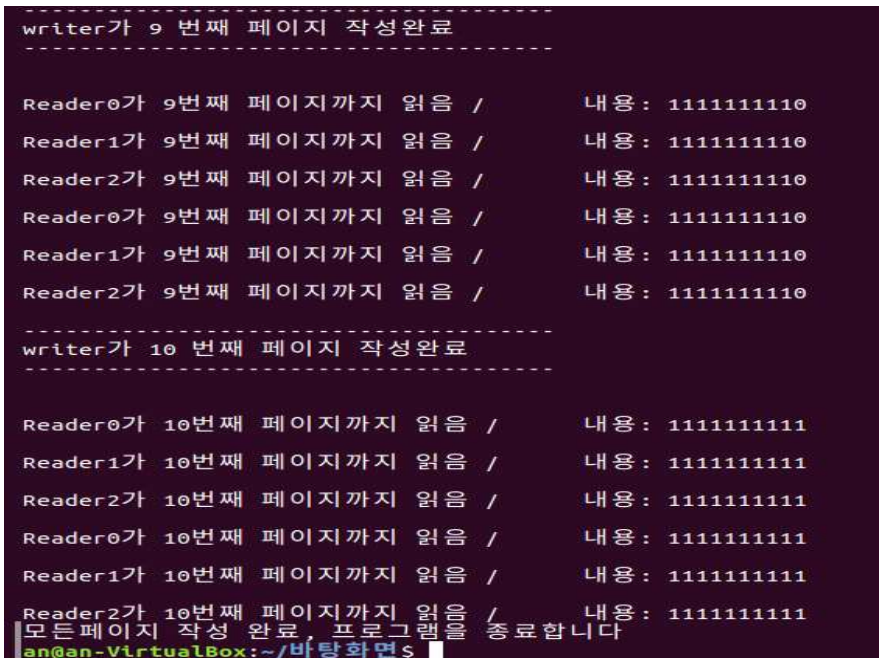
1. writer 스레드의 note 배열의 1번째 원소에 1 입력 (실질적인 쓰기 동작)
2. reader 스레드의 note 배열의 1번째 원소에 대한 출력 (실질적인 읽기 동작 + 동작 1회 완료)
3. writer 스레드의 쓰기동작 완료 문장 출력 (실질적인 쓰기는 미리 끝냈으나 완료 문장은 지금 출력)

따라서 reader 스레드와 writer 스레드가 mutual exclusive 하고, 각각의 reader 스레드끼리도 wrt세마포를 사용하는 영역에서 부분적으로 mutual exclusive 한 기능이 추가되어야 함을 알 수 있었다.

다음 결과화면은 sleep() 함수와 mutex, sempore를 이용하여 언급한 동기화 기능을 충족시켜 수정한 프로그램의 동작결과이다.



...중략



동기화에 의해 목표했던 스레드의 작동순서가 맞아떨어지게 동작한 모습이다.

자가진단표

구현 단계 (단계에 체크)	내용	주요 bug 및 프로그램의 문제점
10□	동기화 기능 구현하여 문제점(race condition) 해결 확인	예제의 틀을 실제 코드로 구현하면서 여러 가지 한계점을 느꼈다. 이렇다 할 만한 bug는 발생하지 않았으나 sleep 함수를 사용하지 않고 각 스레드의 생성과 동작을 순차적으로 구현하는 과정에서 어려움을 겪었고 결국 sleep을 사용하는 방법을 택했다. producer consumer처럼 빠른 속도로 동작하며 동기화가 이루어지도록 구현하지 못한 점이 아쉽다.
9□	동기화 기능 동작 확인	
8□	동기화 기능 구현	
7□	동기화 제외한 프로그램의 문제점(race condition) 확인	
6□	동기화를 제외한 프로그램 동작 확인	
5□	동기화를 제외한 프로그램 일부 동작	
4□	동기화를 제외한 프로그램 완성	

토의 및 느낀점

처음에 과제의 요구사항을 읽고 나서 이론적으로는 동기화 기능이 필요하지만 sleep이나 wait 같은 기능으로도 충분히 상호배타적이고 race condition을 발생시키지 않는 프로그램을 작성할 수 있지 않을까 그리고 효율도 별로 큰 차이가 없지 않을까 하는 의문이 들었다.

그래서 프로그램의 분석/설계를 동기화 기능 없이도 race condition이 생기지 않을 것이라는 가정 하에 진행했고 프로그램을 테스트해 보았으나, 코드의 순서배치나 sleep()함수만의 사용으로는 본인의 설계 조건을 완전히 만족시킬 수 없음을 깨달았다.

또한 매번 동작마다 강제로 한 스레드를 무한루프를 돌게 하거나 일시정지하게 하여 코드를 짜는 것보다 mutex라는 API를 이용하는 것이 훨씬 코드 간결성이나 리소스 효율성 면에서도 낫다는 것을 깨달았다.

이후 mutex에서 더 확장된 범위인 semapore의 사용방법까지 터득하면서 고전적인 방법일 수 도 있지만 동기화라는 개념의 적용이 스레드 단위의 프로그래밍에서는 상당히 중요하단 걸 깨달았다.

강의시간에 이론적으로 배웠던 동기화 문제를 과제 프로그램을 통해 직접 설계하는 과정에서 겪었고 왜 동기화 기능이 적용 되었는지 실질적으로 이해할 수 있는 경험이었다.

또한 이론적으로 배운 사항들을 이렇게 과제물로 남김으로써 차후에 겪을 프로그래밍에서 활용할 방안이 무엇일지 고민해보게 되었다.

다음번에는 현재의 동기화 기능에서의 장단점은 무엇이고 여기서 더 나아가 발생할 수 있는 문제와 개선점은 무엇인지 탐구하고 싶은 호기심이 생겼고, 이렇게 간단한 프로그램이 아닌 협력 프로젝트 단위의 프로그램을 이론에서 배운 지식을 통해 구현해보고 싶다는 열망이 생겼다.