오퍼레이팅 시스템 2차 과제

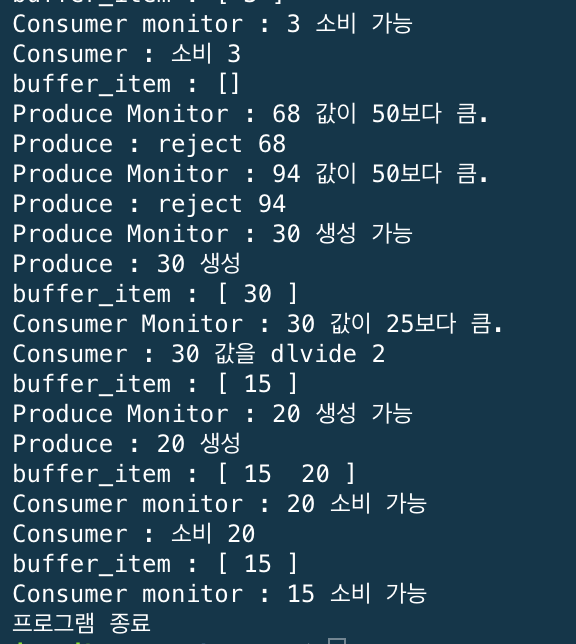
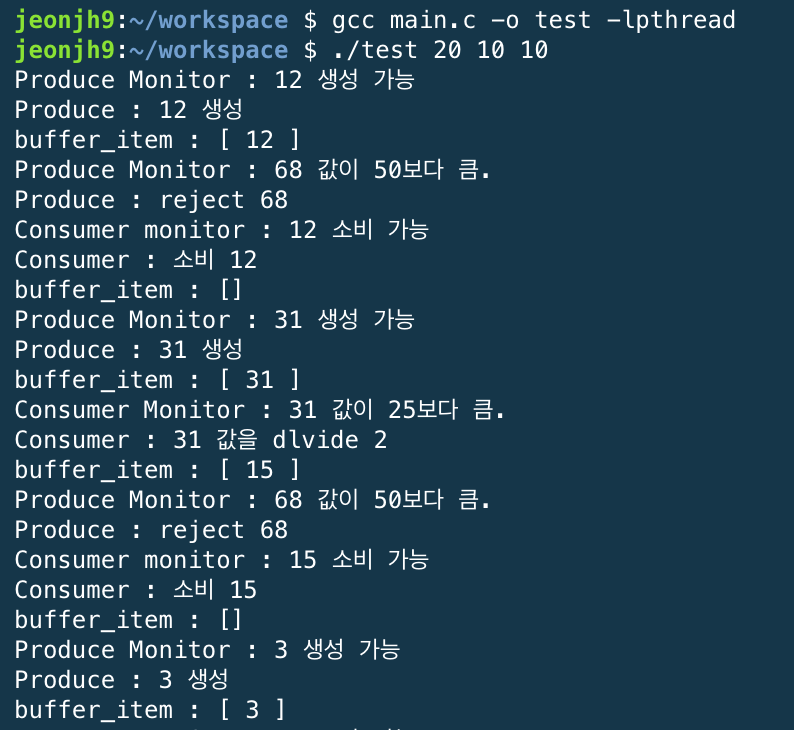
컴퓨터공학과 12131593

전 준 호

* **What to do?**

1. **Producer-Consumer Problem을 구현 할 것.**

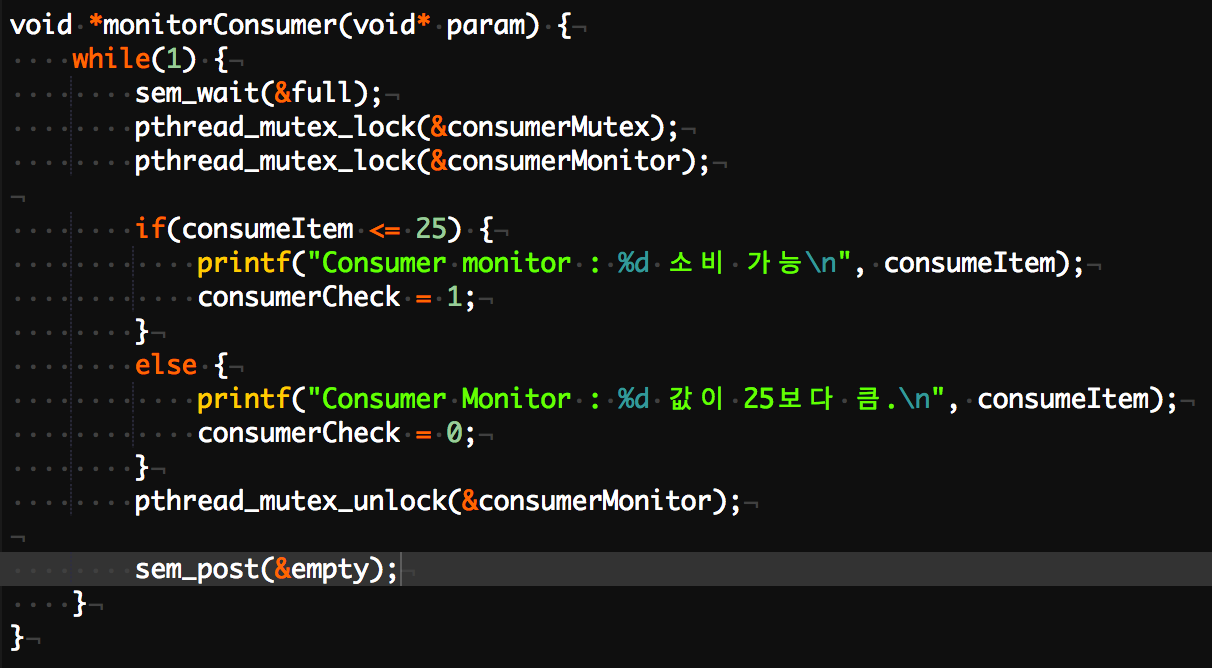
Producer-Consumer Problem은 버퍼 한정 문제로 생산자는 데이터를 만들며 공간에 저장하는데 저장할 때 공간이 없는 문제가 발생할 수 있다. 또 소비자는 데이터가 필요할 때 생산자가 저장한 공간에서 데이터를 하나 소비한다. 이 때는 소비할 데이터가 없는 문제가 발생할 수 있다는 것이다. 이 문제를 해결 하기 위해 버퍼공간을 동기화시켜 문제가 발생하지 않도록 하는 것이 필요하다. 이 동기화를 semaphore를 통해 해결 할 수 있다.



이 이미지들은 최종 출력물로 Produce쓰레드에서 데이터를 생성, Consumer쓰레드에서 소비를 하는 것을 볼 수 있다.

buffer\_item의 데이터들을 출력해줌으로써 명시적으로 나타내 쉽게 알 수 있다. 동기화 문제는 Semaphore와 Mutex를 사용했으며 자세한 내용은 3번에서 다룰 것이다.

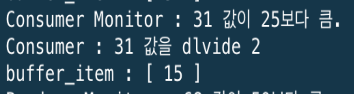
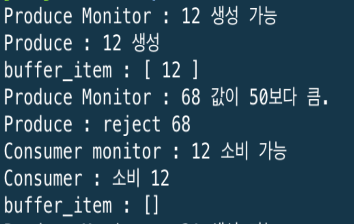
1. **어떻게 검증할 것인지를 기술할 것 - 2개의 Monitoring thread가 동작하는지 기술**



생산자 모니터 쓰레드 소비자 모니터 쓰레드

생산자 모니터 쓰레드에서 buffer에 item을 추가 할 때 값이 1 ~ 50 이어야 추가가 가능하도록 설계 하였으며 item값이 51이상일 경우는 reject를 한다.

소비자 모니터 쓰레드에서는 buffer에 저장되어 있는 item을 소비 할 때 값이 1 ~ 25일 때 소비, 즉 제거가 가능하다. 26이상일 경우는 item의 값을 2로 나눠주어 다시 저장한다.



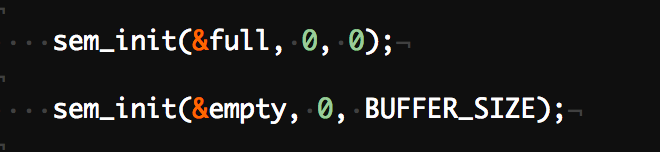
위 이미지들은 출력 이미지에 일부분이며 monitor쓰레드들이 계속해서 monitoring해주며

생성할 때 item값이 1 ~ 50 이면 생성 buffer\_item에 추가, 51 이상일 경우 reject.

소비할 때 item값이 1 ~ 25 이면 소비 buffer\_item에서 제거, 26 이상일 경우 2로 나눠주는 것을 볼 수 있다.

1. **2개의 Monitoring thread 기술 - semaphor, Mutex 동기화 기술**

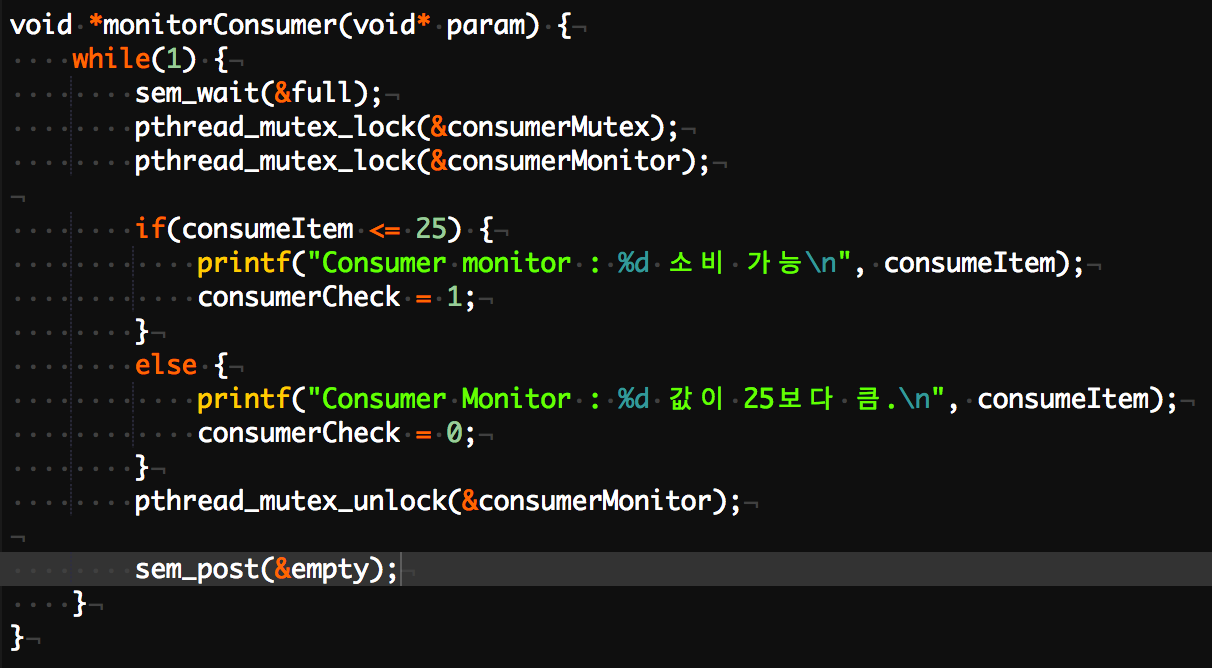
* **Semaphor**



Semaphor객체인 full과 empty를 생성하여 sem\_init으로 초기화 하며

full객체는 버퍼공간에 존재하는 item을 의미하며 0으로 초기화 해준다.

empty객체는 버퍼공간에 비어있는 공간을 의미하며 BUFFER\_SIZE로 초기화해준다.

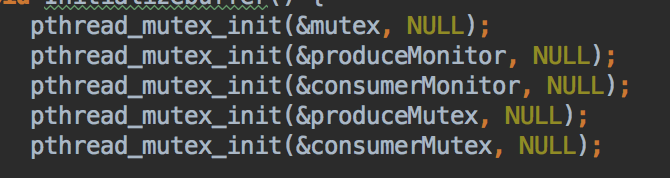
 

생산자 소비자

sem\_wait()는 semaphore객체를 원자적으로 1을 감소시키는 역활을 한다. 해당 semaphore객체가 0보다 큰 숫자를 가지고 있을 때는 1을 감소시키며 0과 같을 때는 객체가 증가될 때까지 기다린다(차단). 그렇기 때문에 생산자에서는 sem\_wait(empty)를 함으로써 빈 공간이 있을 경우에는 임계 구역에 접근하며 빈 공간이 없을 경우에는 공간이 생길 때 까지 기다린다. 소비자에서는 sem\_wait(full)로 공간 중 하나의 공간이라도 있다면 접근, 채워진 공간이 없을 경우 기다린다.

sem\_post()는 semaphor객체를 1 증가시키는 역활을 하며 semaphor가 실행될 수 있는 조건을 만들어준다. 그렇기 때문에 생산자에서는 sem\_post(full)로 생산을 하였기 때문에 full객체를 증가시킴으로써 하나의 공간을 사용하고 있음을 알려준다. 소비자에서는 sem\_post(empty)로 생산자와는 반대로 소비를 하였기 때문에 empty객체를 증가시킴으로써 하나의 빈 공간을 생겼다는 것을 알려준다.

* **Mutex**

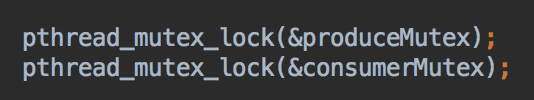


mutex는 생산자, 소비자 쓰레드에서 임계 구역에 같이 접근하는 것을 방지.

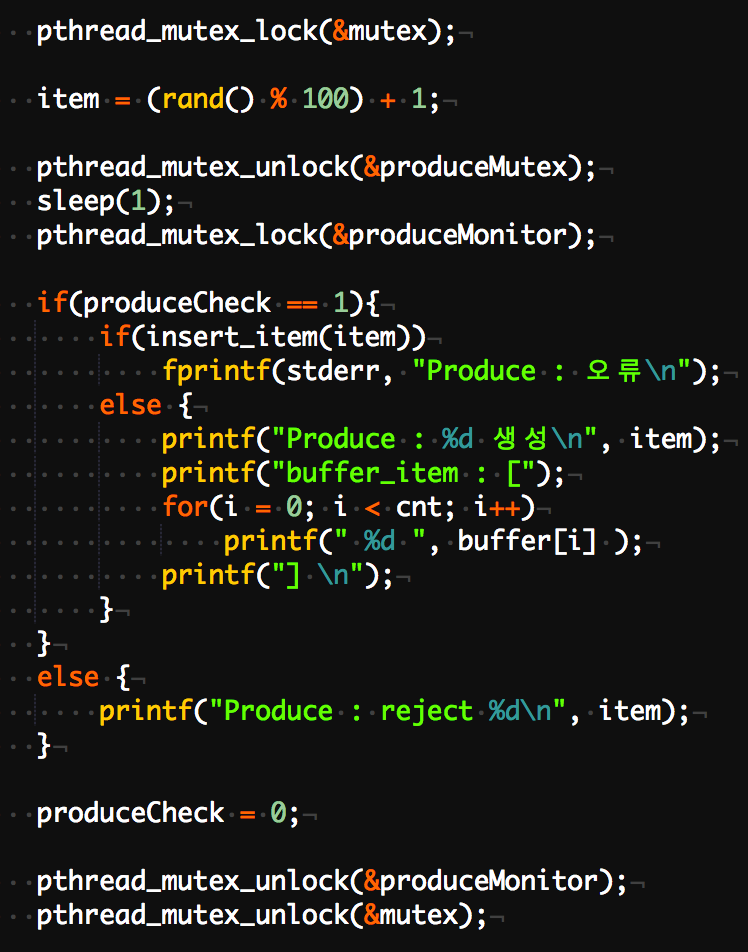
produceMonitor는 생산자 쓰레드와, 생산자 모니터링 쓰레드가 같이 접근 하는 것을 방지하여 모니터링 되지 않는 값을 생산자에서 미리 생산하는 것을 방지한다.

consumerMonitor 또한 소비자 쓰레드와, 소비자 모니터링 쓰레드가 같이 접근 하는 것을 방지한다.

produceMutex와 consumerMutext는 모니터링 쓰레드는 계속해서 돌아가기 때문에 처음부터 미리 쓰레드가 도는 것을 방지한다.



main쓰레드에서 produceMutex와 consumerMutex를 lock함으로써 모니터링 쓰레드를 잠가버리고 생산자, 소비자 쓰레드에서 접근할 때 unlock함으로써 모니터링 쓰레드 작동이 시작된다.



생산자 쓰레드 생산자 모니터링 쓰레드

시작하기전에 produceMutex를 unlock함으로써 모니터링쓰레드를 동작시킨다.

그 후 produceMonitor를 이용해 생성자 쓰레드와 모니터링 쓰레드가 동시 접근하지 못하도록 방지한다.



소비자 쓰레드 소비자 모니터링 쓰레드

생산자 쓰레드와 마찬가지로 consumerMutex를 unlock하여 모니터링 쓰레드를 동작시킨다.

그 후 consumerMonitor를 이용해 소비자 쓰레드와 모니터링 쓰레드가 동시 접근하지 못하도록 방지한다.

그리고 Mutex를 이용해서 소비자 쓰레드와 생산자 쓰레드가 동시에 접근하지 못하도록 방지하며 동기화를 유지한다.

* 개발 환경

온라인 통합 개발환경인 Cloud9에서 c를 이용 개발 및 gcc 컴파일.

* 결과

Produce-Consumer Problem의 가장 중요한 점은 Produce와 Consumer 쓰레드를 Semaphore와 Mutex를 사용해서 동기화 시키는 것이다. 동기화 시키기 위해서는 semaphore와 mutex가 어떻게 작동되는지를 이해하는 것이 가장 중요했던 것 같다. 그냥 강의자료에서만 봤을 때는 쉽게 이해할 수 있었지만 실제로 프로그램을 작성하려하니 생각대로 움직이지 않아서 굉장히 힘들었었는데 세마포어와 뮤텍스가 어떻게 작동되는지를 이해하니 쉽게 문제를 해결 할 수 있었다.

* 개선해야 할 점

모니터링 쓰레드를 추가함으로 써 더욱 정교한 Mutex사용이 필요했었다. 처음에는 적은 Mutex만을 이용하여 문제를 해결 하려 했지만 생각대로 잘 안되어 개수를 늘렸고 문제를 해결하기는 했지만 Mutex를 많이 적게 사용해서 해결하지 못 한 점이 아쉽다.

또 생산자나 소비자 쓰레드에서 모니터링 쓰레드를 Unlock해주고 sleep(1)을 통해서 모니터링 쓰레드가 먼저 작동되도록 설계를 하였는데 이 sleep을 사용하지 않으면 생산자, 소비자 쓰레드가 모니터링 쓰레드보다 먼저 작동하는 경우가 있어 이 것을 해결하기 위해 사용하였지만 이 방법이 적절한 방법인지 의문이 들었고 다른 방법으로 해결하려 했지만 해결할 수 없었다.