****

컴퓨터 정보 공학과 3학년

12131579 이진아

**오퍼레이팅 시스템**

**2차 과제**

담당 교수: 송민석 교수님

**1. 개요**

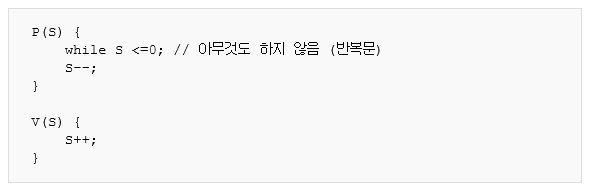
구현 목표 : Critical Section problem 중 하나인 Producer-Consumer problem를 semaphore와 mutex를 이용하여 해결하고 Producer와 Consumer를 감독하는 monitor thread를 생성하여 procuder와 consumer가 실행 된 다음 실행하도록 semaphore를 이용하여 구현한다. 이를 통해 semaphore와 mutex의 동작 과정을 이해 할 수 있다.

**2. 개념 설명**

**1)Semaphore**

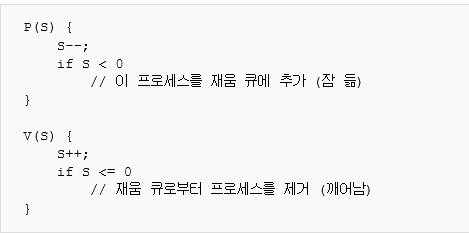
운영 체계 또는 프로그램 작성 내에서 공유 자원에 대한 접속을 제어하기 위해 사용되는 신호이다. 병형 내지 병렬로 동작되는 둘 이상의 프로세서 사이에서 마이크로프로세서 시간이나 입출력 접속구와 같은 공유 자원을 동시에 사용할 수 없기 때문에 한 프로세서가 사용하고 있는 동안에 세마포어를 세워서 다른 프로세서를 대기 시키고 사용이 끝나면 해제시키는 방법으로 사용한다.

-방법1(busy waiting)



P함수는 wait함수, V는 signal함수일 경우 P함수를 수행 하였을 때 emaphore S의 값이 0보다 작거나 같을 경우 반복문을 수행하고 0보다 큰 경우 반복문을 빠져 나와 S의 값을 하나 줄여준다. V함수가 수행이 되면 s를 하나 증가 시켜 준다. 이는 단일 처리기 다중프로세스 환경에서 처리기 효율이 떨어진다는 단점이 있다.

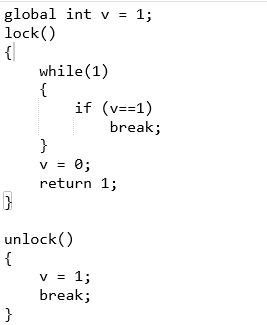
-방법2(재움 queue활용)



P함수에서 S를 감소 시켜 0보다 작은 경우 재움 큐에 넣어 준다. V함수에서는 S를 증가 시키고 S가 0보다 작거나 같은 경우 재움 큐에 재워두었던 프로세스를 깨워준다.

**2) Mutex**

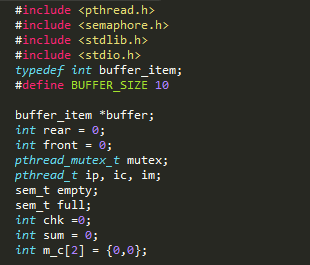
Semaphore는 크게 counting semaphore(계수 세마포어)와 binary semaphore(이진 세마포어)로 나눠지는데 Mutex는 binary semaphore와 같다. Mutex는 0또는 1을 값으로 가진다.

****

V를 mutex라고 생각하면 lock함수 일 경우 v가 1인 경우에 break로 반복문을 빠져 나와 v를 0으로 바꿔준다. 반면 v가 1이 아닌 경우 즉 0인 경우에는 계속해서 반복문을 수행하게 된다. unlock함수가 수행이 되어야 v가 1로 바뀌며 잠금을 풀어주게 되는 것이다. Process p1과p2가 있을 때 p1이 먼저 수행이 되어 lock함수를 수행하면 반복문을 빠져나오면서 v가 0으로 변하게 된다. 따라서 그 후 p2가 그 후 lock함수를 수행 하면 p1이 unlock함수를 실행하여 v를 1로 바꿔줄 때 까지 반복문을 수행하게 된다. 따라서 critical section 전에 lock함수를 넣고 후에 unlock함수를 넣어주면 mutual exclusion을 만족시킬 수 있다.

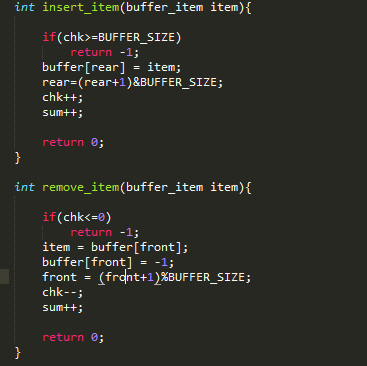
**3.코드 설명**

**1)구조체 및 변수 생성**



Producer와 Consumer가 사용할 buffer를 생성하고 consumer가 쓰게 될 data위치를 저장할 변수는 front, producer가 넣게 될 data위치를 저장 할 변수를 rear라고 두었다. 이진 semaphore를 구현하기 위한 mutex를 생성하였고 producer thread인 ip, consumer thread인 ic, monitor thread인 im을 선언하였다. Semaphore를 수행하기 위해 buffer안에 data가 얼마나 들어가고 얼마나 비어있는지 여부를 확인하는 full과 empty변수를 선언하였다. Chk는 buffer안에 들어있는 data의 수를 확인하는 변수이고 sum은 총 producer와 consumer가 실행된 횟수를 저장하는 변수이다. m\_c배열은 monitor를 실행시키기 위해 사용한 배열이다.

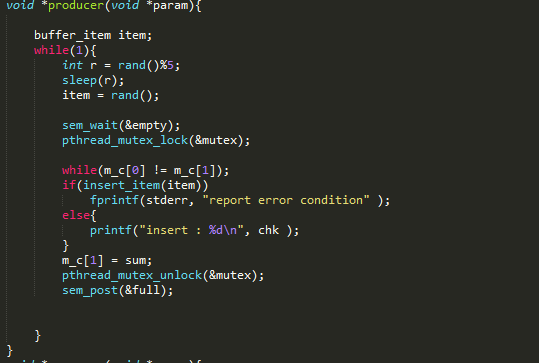
**2)Insert/Remove 함수**



insert 함수에서 버퍼 안에 들어있는 데이터 수가 버퍼의 사이즈와 같거나 크면 더 이상 data를 insert할 수 없으므로 -1을 retrun해준다. 버퍼 안에 들어있는 데이터 수가 버퍼의 사이즈 보다 작으면 버퍼의 뒷 부분에 data를 넣고 넣게 될 위치를 1증가 해준다. 이 때 버퍼 사이즈의 나머지를 rear에 저장해 주는데 이는 환형큐로 buffer를 사용하기 때문이다. Data를 넣어 준 후 버퍼 안에 들어있는 데이터 수를 저장하는 chk를 하나 늘려주고 총 연산한 수인 sum를 하나 증가시켜 준다.

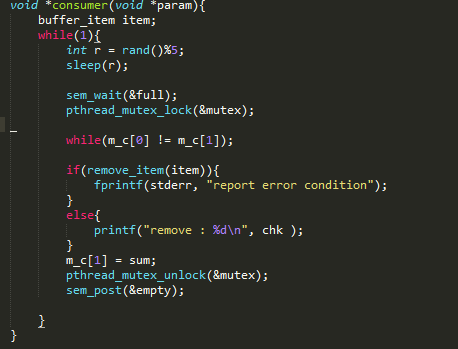
remove 함수에서는 버퍼 안에 들어있는 수가 0보다 작거나 같으면 -1을 출력해준다. 버퍼 안에 들어있는 수가 0보다 크면 즉 버퍼 안에 data가 있는 경우이면 버퍼의 가장 앞부분에 있는 data를 받아오고 data가 없다는 의미인 -1를 넣어준다. 그 후 앞부분을 1증가 시켜주고 버퍼 사이즈의 나머지로 front에 저장해준다. 그 후 data가 하나 빠졌으므로 chk를 하나 감소 시켜주고 총 연산인 sum을 하나 증가시켜준다.

**3)Producer함수**



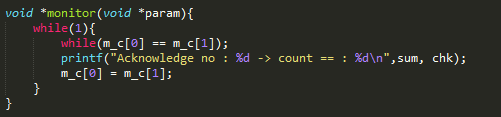
r변수에 랜덤으로 시간을 받아 r시간만큼 sleep하도록 한다. 그 후 랜덤하게 버퍼에 저장 할 data 값을 받고 sem\_wait()함수를 수행한다. 이 함수에 파라미터로 empty값을 넣어주는데 empty값이 0보다 크면 버퍼에 data를 넣을 공간이 있다는 뜻이므로 empty값을 하나 감소 시킨 후 함수를 빠져 나오게 된다. empty값이 0 즉 버퍼에 빈 공간이 없으면 빈 공간이 생길 때 까지 함수를 빠져 나오지 못한다. Sem\_wait함수를 빠져나오면 mutual exclusion을 만족시키기 위해pthread\_mutex\_lock()함수를 사용한다. 이 함수에는 파라미터로 mutex를 주는데 mutex는 이진 semaphore로 0과1을 갖는다. Mutex가 1인 경우 mutex를 0로 바꾼 후 함수를 빠져 나온다. 이를 mutex를 잠궜다고 한다. 1인 경우는 이미 잠근 다른 thread가 mutex의 잠금을 해제할때까지 block되게 된다. 그 후 while문이 하나 더 걸려 있는데 이는 monitor를 하기 위해 사용된 반복문이다. m\_c[0]과 m\_c[1]이 다르면 monitor thread가 수행 중이라는 뜻이므로 while문을 빠져 나오지 못한다. m\_c[0]에는 이전 sum의 값이 들어 있고 m\_c[1]에는 producer나 consumer가 수행 되면 바뀌는 sum의 값을 넣는다. 따라서 m\_c[0]과 m\_c[1]이 다르다는 의미는 producer나 consumer 둘 중 하나가 수행이 되었다는 뜻이므로 monitor thread가 수행 되고 있다는 의미이다. Monitor thread의 수행이 끝나면 m\_c[0]과 m\_c[1]의 값이 같아져 반복문을 빠져나오게 된다. Sem\_wait, pthread\_mutex\_lock, while문을 거치고 난 후에야 insert함수를 수행할 수 있다. 그 후 m\_c[1]값에 sum값을 대입하고 pthread\_mutex\_unlock()함수를 통해 mutex를 1으로 바꿔준다. 즉 잠금을 해제해 준다. 그리고 sem\_post()함수를 통해 full을 하나 증가 시켜 준다.

**4)Consumer 함수**

****

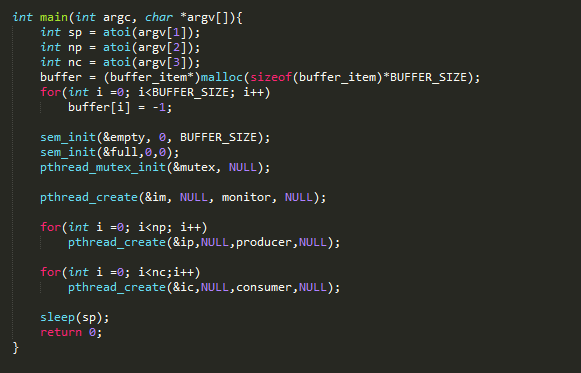
producer함수와 비슷하게 수행이 된다. 다만 sem\_wait함수의 파라미터로 full을 줘서 full값을 하나 감소 시켜 주고 sem\_post함수의 파라미터로 empty를 줘서 empty값을 하나 증가시켜 주면 된다. 중간에 insert함수 대신 remove함수를 사용해 주어 producer함수와 반대로 data를 하나 remove하도록 작성하였다.

**5)Monitor 함수**



m\_c[0]과 m\_c[1]이 같으면 아직 producer와 consumer둘 중 아무 것도 수행이 되지 않았다는 의미이므로 반복문을 수행한다. 둘 중 하나가 수행이 되면 sum값이 하나 증가가 되고 m\_c[1]에 sum값을 넣어주므로 반복문을 빠져나오게 된다. 반복문을 빠져 나오면 지금 까지 수행한 총 연산 수와 버퍼에 들어있는 값을 출력해 주고 m\_c[0]에 m\_c[1]값을 넣어 다시 반복문을 수행하도록 해준다.

**6)main함수**

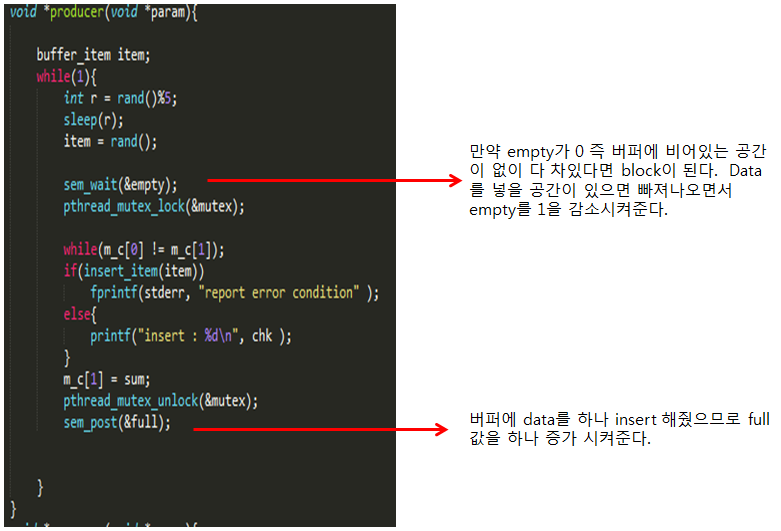
****

종료하기 전 얼마나 sleep을 할건지에 대한 값을 argv[1]에 받아 sp에 저장하고 producer thread의 수를 argv[2]에 받아 np에 저장, consumer thread의 수를 argv[3]에 받아 nc에 저장하였다. Buffer를 buffer사이즈 만큼 동적할당 해주고 버퍼에 -1로 초기화 해준다. monitor함수를 create해주고 np, nc의 값 만큼 producer thread와 consumer thread를 create해준다.

珍

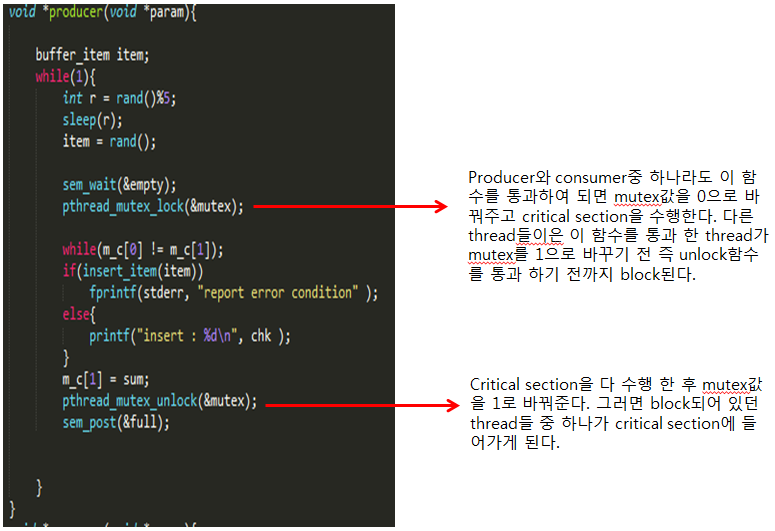
**5. Semaphore, mutex 사용 기술**

**1)Semaphore**

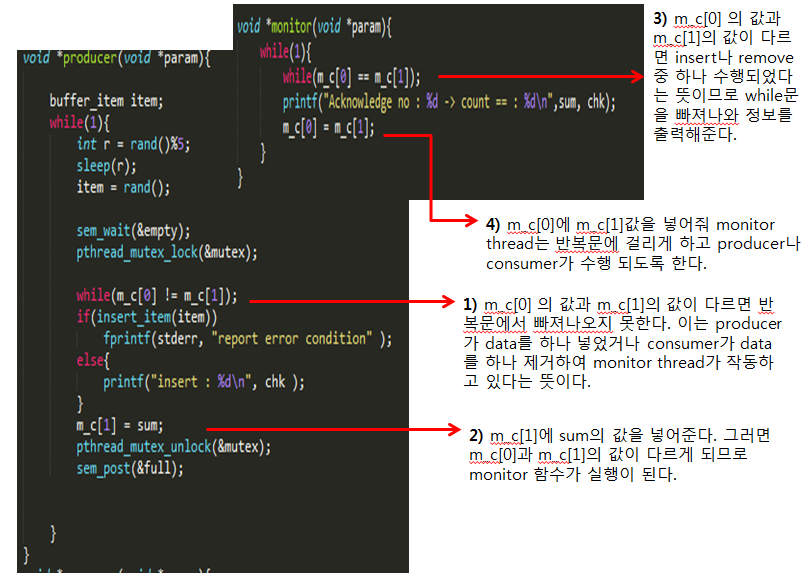
****

Consumer는 sem\_wait(&full), sem\_post(&empty)를 넣어 수행한다.

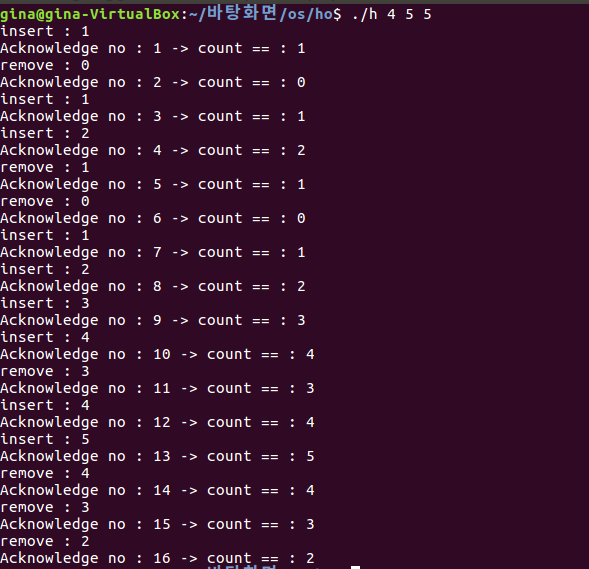
**2)Mutex**

****

**3)Monitor**

****

**4. 실행결과**

****

Sleep할 시간, Producer thread의 수, Consumer thread의 수를 입력 받고 수행한다.