**2017029589 컴퓨터소프트웨어학부 류지범**

**운영 체제 HW#6**

**제출 일자 : 2021/04/29**

1. **과제 A**
2. **자료구조 설명**

2명의 reader와 5명의 writer는 thread를 사용해서 만든다. Critical section은 cur\_writer와 cur\_count이고, critical section에 접근한 reader의 수를 세기 위한 readcount가 있다. Critical section을 보호하기 위해 semaphore를 사용하고, S 와 wrt를 사용한다.

1. **함수 설명**

Void \*writer(void \*n);

Writer는 writer 함수를 통해 cur\_writer와 cur\_count를 바꾼다. For문을 통해서 COUNTING\_NUMBER만큼 loop를 돌고, 쓰레드의 로컬 변수 count를 선언해서 특정 writer가 cur\_writer에 접근한 횟수를 기록할 수 있도록 했다. Usleep(100000)은 writer thread들이 100000ms 간격으로 권한을 요청하도록 했다. Writer가 critical section에 접근하기 전에, sem\_wait(&wrt)를 써서 다른 writer나 reader가 critical section에 들어가 있으면 기다리도록 하고, 아니면 접근할 수 있도록 한다. Critical section에 접근해서, cur\_writer에 writer의 id를 넣고, cur\_count에 해당 writer가 critical section에 접근한 횟수인 로컬 변수 count 값을 넣는다. Writer들이 critical section의 변수에 기록을 할 때마다 print문을 통해서 출력을 할 수 있도록 해준다.(과정을 더 원활하게 확인하기 위해서 writer에도 print문을 넣었다.) 마지막으로 sem\_post(&wrt)를 써서 critical section에서 나왔음을 알린다.

Void reader();

Reader는 reader 함수를 통해 가장 최근에 critical section에 접근한 writer의 id와 count를 출력하도록 한다. 처음에 usleep(30000)을 사용해서 reader thread들이 30000ms 간격으로 권한을 요청하도록 했다. Critical section에 접근한 reader들의 수를 세기 위한 readcount은 reader thread들에 의해 동시에 접근되면 안되기 때문에 S semaphore로 보호한다. sem\_wait(&S)를 사용해서 reader thread가 이미 접근해 있는지 확인하고, 접근해 있으면 기다린다. 이후 readcount의 값을 증가시킨다. 만약 readcount가 1이라면, 해당 reader thread가 모든 reader thread들 중에 가장 최초로 접근을 시도한 reader이므로, critical section에 writer가 있는지 검사하고, 있으면 기다린다. 이후 sem\_post(&S)를 써서 readcount의 보호를 해제한다. 다음으로, 최근에 접속한 writer의 id와 count값을 출력한다. 마지막으로 다시 readcount의 값을 감소시키기 위해 우선 다른 reader가 readcount에 접근해 있는지 확인하고 있으면, 기다린 후, readcount의 값을 1 감소시킨다. 감소시킨 이후 readcount의 값이 0이라면, 해당 reader가 마지막으로 나가는 reader이기 때문에 writer에게 이제 critical section에 접근해도 된다고 신호를 보내야 한다. 그래서 sem\_post(&wrt)를 통해 신호를 보내주고, sem\_post(&S)를 통해 readcount의 보호를 해제해준다.

1. **프로그램 구조 설명**

프로그램엔 2개의 reader가 있고, 5개의 writer가 있는데 각각 thread로 돌아간다. Critical section을 보호하기 위한 S와 wrt를 init해주고, reader와 writer의 thread를 생성한다. Writer thread를 생성할 때는 writer의 id도 argument로 같이 넘겨준다. Writer thread는 writer 함수를 실행하고, reader thread는 reader함수를 실행한다. 마지막에 pthread\_join으로 thread들을 wait하고, semaphore S, wrt를 destroy하고 프로그램을 종료한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **프로그램이 어떻게 First Reader-Writers Problem을 해결하는지 설명**

이 프로그램은 reader와 writer가 critical section에 접근하는 문제를 semaphore를 사용하여 해결하는 것인데, reader가 접근하고 있을 땐 어떠한 writer도 접근하지 못하지만, 다른 reader는 접근할 수 있도록 해서 동기화 문제를 해결한다. Writer 함수에서 sem\_wait를 통해 다른 writer나 reader가 이미 접근해 있을 경우 wait하도록 했고, 이후 critical section에 접근해서 값을 변경한 이후, sem\_post로 signal을 보내서 critical section에서 나왔음을 알린다. Reader 함수에선 readcount를 사용해서 처음 들어온 reader thread, 마지막으로 나가는 reader thread를 확인할 수 있도록 했고, 처음 들어온 reader thread는 writer가 이미 있을 경우에만 sem\_wait를 통해 기다리도록 하고, 없을 경우엔(다른 reader가 있어도 상관 없다) critical section에 접근할 수 있도록 했고, 마지막으로 나가는 reader thread가 sem\_post로 이제 writer가 들어올 수 있다고 신호를 보내주도록 한다. 이 프로그램에선 Reader는 이미 reader가 한 개라도 들어가 있을 경우에는 계속해서 진입할 수 있기 때문에 writer보다 권한을 얻기 쉽다. 그래서 만약에 reader가 엄청난 수가 들어와버리면 writer가 starvation에 빠질 수도 있다. 하지만 이 프로그램에서는 작은 수의 COUNTING\_NUBMER 만큼만 프로그램을 실행하기 때문에 starvation에 빠질 일은 없다. 또한 reader와 writer가 critical section에 접근하는 타이밍을 일정하게 조절해서 확인하기 쉽게 하기 위해 각각 usleep 100000, 30000을 넣어주었는데, reader가 3.3개정도 접근할 동안 writer는 1개가 접근하도록 했다. 그래서 프로그램 결과를 보면 reader thread 2개가 각각 3번 접근하고 난 이후에, writer가 접근할 기회가 생기므로, 5개의 writer thread가 각각 1번씩 접근한다. 마지막엔 waiting 하고있던 나머지 writer들이, 모든 reader가 접근한 이후에 접근하는 모습을 보여준다.

1. **과제 B**
2. **자료구조 설명**

6명의 철학자는 각각 thread로 구현된다. 각 thread는 philosopher라는 함수를 실행하는데 이 함수는 식사를 하고, 생각을 하는 함수이다. 6개의 젓가락은 동기화와 deadlock 문제를 해결하기 위해 semaphore로 선언한다.

1. **함수 설명**

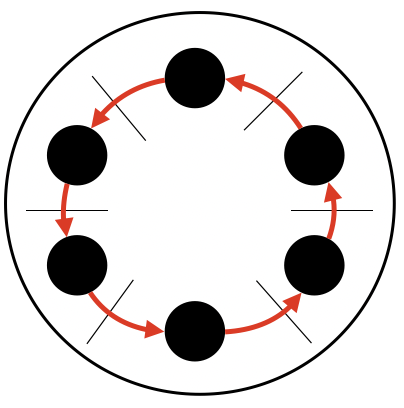
Void \*philosopher(void \*n);

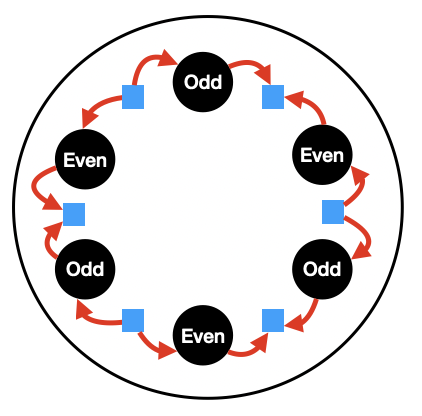
철학자가 식사를 하고, 생각을 하는 함수이다. Argument로는 철학자의 id를 받는다. 만약 철학자의 id가 짝수이면 왼쪽 젓가락을 먼저 집고, 그 후에 오른쪽 젓가락을 집는다. 젓가락을 집는 것은 sem\_wait(chopstick)으로 하는데, 다수의 인원이 동시에 한 젓가락을 집을 순 없기 때문에, 권한을 획득하기 위해 wait하는 것이다. 만약 철학자의 id가 홀수이면 오른쪽 젓가락을 먼저 집고, 그 후에 왼쪽 젓가락을 집는다. 이것도 마찬가지로 sem\_wait(chopstick)을 사용한다. 양쪽의 젓가락을 모두 집었으면, 해당 철학자의 id와 함께 식사중이라는 메시지를 출력한다. 식사가 끝났다면 젓가락을 내려놓아야 하기 때문에, sem\_post를 써서 오른쪽 젓가락을 내려놓고, 왼쪽 젓가락을 내려놓는다. 다 내려놓고 난 후에는 철학자의 id와 함께 생각중이라는 메시지를 출력한다. 철학자들을 서로 구분되게 출력하기 위해 생각중 메시지를 출력할 땐 줄바꿈을 두 번 했다.

1. **프로그램 구조 설명**

우선 각각의 철학자들을 위한 thread 변수 philosopher[6]를 선언해준다. 철학자의 id를 위한 argument[6]도 선언해준다. 젓가락은 동시에 사용할 수 없기 때문에 semaphore로 6개의 젓가락 chopstick[6]을 선언해주고, sem\_init으로 초기화해준다. 그 후 각각 철학자에 대해 thread를 생성하고 id에 해당하는 값을 argument로 넘겨준다. 철학자들은 void \*philosopher(void \*n) 함수를 COUNTING\_NUMBER 만큼 실행한다. 다 실행할 동안 pthread\_join을 통해서 thread의 종료를 기다린다. 모든 thread가 종료되고 나면, init해주었던 semaphore를 sem\_destroy로 파괴한다. 프로그램을 실행하면, 철학자들이 식사하고, 생각하는것이 출력되고, deadlock에 빠지지 않고, 진행된다. 텍스트이(가) 표시된 사진

자동 생성된 설명COUNTING\_NUMBER는 3으로 설정했다. 아래는 프로그램 실행 결과이다.

1. **프로그램이 어떻게 Dining-Philosophers Problem을 해결하는지 설명**

만약 모든 철학자들이 왼쪽 젓가락을 먼저 들고, 오른쪽을 다음에 들면, 위의 그림과 같이 cycle이 형성된다. 이때 리소스의 instance를 하나밖에 없기 때문에 Cycle이 형성되면 circular wait가 발생한다. 이는 deadlock 발생의 필요조건이고, cycle만 깰 수 있다면 deadlock 문제를 해결할 수 있다. LR solution(asymmetry solution)을 적용하면 이 cycle을 깰 수 있다. 짝수 id의 철학자는 왼쪽을 먼저 들고 난 후에 오른쪽을 들고, 홀수 id의 철학자는 오른쪽을 먼저 들고 왼쪽을 들면 된다. 아래의 그림은 LR solution을 graph로 표현한 것이며, 파란색 사각형은 젓가락(리소스)이고 검은색 원은 철학자이다. 그림에서 볼 수 있듯이 cycle이 형성되지 않는다. Deadlock의 필요조건인 Cycle을 깼기 때문에 deadlock은 발생하지 않게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명두 프로그램 모두 Makefile로 한 번에 컴파일했으며, Makefile은 다음과 같다.