**2018008304 컴퓨터소프트웨어학부 박경하**

**운영 체제 HW#6**

**제출 일자 : 2021/05/04**

1. **과제 A**
2. **자료구조 설명**

****

문제해결을 위한 자료구조는 semaphore을 이용하였습닏니다. Sem\_t은 초기화와 동작에 필요한 두 가지 함수(sem\_wait(), sem\_post())가 존재하며, semaphore읆 모두 사용한 후 파괴하는 sem\_destroy() 함수가 존재합니다. 이번 과제에서는 이진세마포어를 이용하였는데, 먼저 sem\_t의 초기값을 1로 주고, wait을 이용해 lock, post을 이용해 unlock 기능을 수행하였습니다.

rw\_mutex은 reader와 writer의 상호배타를 위한 것이고, mutex은 atomic하게 일어날 수 있게 write의 접근을 차단하기 위한 semaphore 입니다.

1. **함수 설명**

****

먼저 writer함수를 보면, 전체적인 구조는

usleep();

lock;

critical section:

unlock;

와 같습니다. Pthread\_create에서 writer함수를 불러, 해당 스레드의 id를 인자로 받으면, sem\_wait()함수를 이용하여 현재 critical section 안에 다른 스레드가 존재하는지 확인한 후, 만일 스레드가 존재한다면 계속 기다리게 됩니다. Critical section 안에 들어가게 되면, writer은 현재 스레드의 tid를 저장한 뒤, critical section에 들어온 횟수를 증가시킨 뒤, sem\_post() 함수를 통해 unlock 합니다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

Reader()함수는 reader끼리 동시에 critical section에 들어가는 것을 허용하기 때문에 writer의 접근을 막아주면 됩니다. 그를 위해 sem\_wait에서는 wirter의 접근을 차단하기 위한 lock이 진행되고, critical section에 들어가면 가장 최근의 writer tid와 critical section에 들어간 횟수를 출력합니다.

1. **프로그램 구조 설명**

텍스트이(가) 표시된 사진

자동 생성된 설명

전체적인 프로그램 구조를 설명하면, 먼저 rw\_mutex와 mutex를 1로 초기화 해줍니다. 이 때, lock을 획득할 수 있는 스레드의 개수가 최대 1개로 정하였기 때문에 1로 초기화 한 것입니다. 그 뒤, 문제에서 원하는 만큼 writer와 reader 스레드를 만든 뒤, 각각 writer()함수와 reader()함수로 넘어가 2번과 같은 과정을 거치게 됩니다. 각각 define 해둔 COUNTING\_NUMBER 만큼 횟수를 거치게 되면 세마포어를 파괴함으로써 프로그램을 종료하게 됩니다.

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

First Reader-Writers Problem에서 발생할 수 있는 문제는 다음과 같습니다.

1. Write 하는 도중에 다른 Write가 critical secion에 들어가는 경우

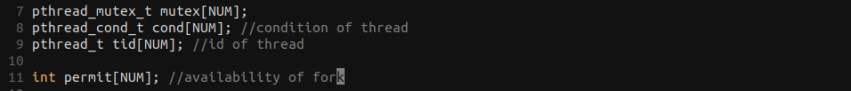
2. Write 하는 도중에 Read 되는 경우

이를 해결하기 위해서는, 먼저 write를 하는 동안에 Read, write 모두 접근하지 못하도록 해야하며 반대로 read하는 도중에 write가 접근하지 못하도록 해야합니다. Read에서는 다른 read 요청이 들어와 동시에 데이터를 read하는 경우에는 데이터 값에 영향을 주지 않으므로 배제합니다.

이를 구현하기 위해 2번에서와 같이 세마포어를 이용한 lock과 unlock으로 접근제한을 두었습니다.

**과제 B**

1. **자료구조 설명**

****

과제 해결을 위한 자료구조 사용은 위와 같습니다. Pthread\_mutex\_t는 공유자원에 대해 접근제어를 이용하여 동기화를 달성하기 위한 변수이고, pthread\_cond\_t은 스레드의 상태값을 나타내는 변수입니다. Permit에는 fork를 사용할 수 있는지의 가능여부를 나타내는 변수로, 만약 사용가능하다면 1이 됩니다.

1. **함수 설명**

텍스트이(가) 표시된 사진

자동 생성된 설명**텍스트이(가) 표시된 사진

자동 생성된 설명**

위의 두 함수의 전체적인 구조는 비슷합니다. Pthread\_create에서 각각의 함수를 불러오게 되면, 홀수 tid을 가진 스레드는 오른쪽에 있는 fork을 먼저 집고, 짝수 tid을 가진 스레드는 왼쪽에 있는 fork을 먼저 잡습니다. 그 뒤, fork 함수에서 성공적으로 fork을 잡게 되면 반대쪽의 fork을 잡고 식사를 한 뒤, 반대의 순서로 fork을 내려놓습니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Philospher 함수에서 get\_fork을 불러오게 되면, 먼저 해당 철학자 번호의 mutex을 잠근 뒤, 포크가 사용 중이면 while문을 통해 계속 기다립니다. 그 뒤, 포크를 들게 되면 최종적으로 해당 포크를 사용불가로 만들기 위해 다시 permit을 0으로 만들고, mutex의 unlock을 풀어줍니다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이 함수에서는 마찬가지로 인자로 들어온 해당 번호의 mutex을 lock 한뒤, critical section에 접근합니다. 여기서는 식사를 다 마친 상태이기 때문에 permit을 1로 만들어 다시 사용가능하게 만들어준 뒤, 현재 대기중인 cond 스레드 하나를 깨웁니다. 그 뒤 mutex의 unlock을 풀게 됩니다.

1. **프로그램 구조 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

전체적인 프로그램 구조는 먼저 mutex와 con을 초기화 시켜준 뒤, permit을 1로 만들어줌으로써 모든 포크를 사용가능한 상태로 만듭니다. 그 뒤, odd 번호를 가진 스레드와 even 번호를 가진 스레드를 각각 나누어 각각의 함수를 호출하게끔 합니다. 마찬가지로 2번의 과정을 거친 뒤에 최종적으로 뮤텍스를 파괴함으로써 종료하게 됩니다. 이 프로그램에서는 while문을 사용하였기 때문에 무한반복하도록 만들었습니다.

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

기존의 Dining-Philosophers 프로그램에서는 교착상태(Deadlock)이 일어납니다. 교착상태의 4가지 조건을 모두 만족하기 때문인데, 이 과제에서는 그 4가지 조건 중 cicular wait 을 깸으로써 교착상태에서 벗어나게 합니다. 여기서 사용한 알고리즘은, 기존의 철학자들은 왼쪽 젓가락부터 집어드는 프로시저를 갖고 있었기 때문에 이 철학자들의 id가 정수인 것을 이용하여 id가 even인 철학자는 왼쪽, id가 odd인 철학자들은 오른쪽부터 fork를 집게하여 해결하였습니다. 이 알고리즘을 구현하기 위해 odd와 even을 나눠 philosophers을 실행한 것이 2번의 내용입니다.