2018008059 컴퓨터 소프트웨어학부 김은수

운영 체제 HW#5

제출 일자 : 2020/04/23

A. 과제 A

1. 자료구조 설명

컴파일러는 CPU가 효율적으로 활용할 수 있도록 명령어를 재배치하거나 최적화한다. 이때 다른 쓰레드들을 고려하지 않기 때문에 멀티 쓰레드 환경에서는 예상치 못한 결과가 나올 수 있다. 그 이유는 원자적인 연산이 아닌 경우에는 모든 쓰레드에서 같은 수정 순서를 관찰할 수 있음이 보장되지 않기 때문이다. 원자적 연산이란 CPU가 명령어 1개로 처리하는 명령으로 중간에 다른 쓰레드가 끼어들 여지가 없는 연산이다. C++에서는 몇몇 타입들에 원자적인 연산을 할 수 있도록 atomic을 제공한다. 이 과제에서는 atomic int를 통해 turn과 flag를 원자적 변수로 정의하여 critical_section_variable이 정확한 값이 나오도록 했다.

2. 동기화 방법 설명

피터슨 알고리즘은 flag과 turn이라는 변수로 임계 영역에 들어갈 프로세스나 스레드를 결정하는 방식이다. Flag 값은 스레드 중 누가 임계 영역에 진입할 것인지 나타내는 변수이고 turn변수는 누가 임계 영역에 들어갈 차례인지 나타내는 변수이다. Flag0이 1이라면 0번 스레드가 들어가고 싶다고 알리는 것이고 그러면 turn을 상대방 1으로 바꾸면서 차례를 상대방한테 양보한다. 상대방의 flag가 1고 turn까지 상대방이라면 계속 while문에서 기다린다. 그 다음 상대방이 임계 영역을 다 실행하고 나서 flag를 0으로 바꾸면 0번 스레드가 while문에서 나와 임계 영역에 들어갈 수 있다 1번 스레드의 경우에도 위와 마찬가지이다.

3. 프로그램 구조 설명

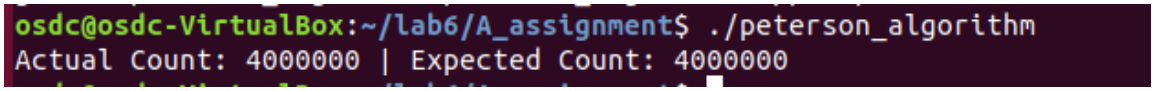
Atomic_int로 turn과 flag0, flag1을 선언해주고 초기값은 0으로 설정해 주었다. **lock함수는** 메인문에서 여러 스레드가 critical_section_variable에 접근하는 것을 막아준다. lock함수는 인자로 self를 받는다. 이는 스레드가 넘긴 인자이다. 이 값이 0이면 flag0을 1로 바꿔주어 들어가고 싶다는 것을 알린다. 그 다음 turn에 1을 배정해 1번 스레드의 차례로 준다. 만약 flag1이 1이고 turn도 1이라면 1번 스레드가 임계 영역에 있는 것이므로 while문에 빠진다. 반대로 self가 0이아니라면 여기서는 1번스레드가 critical_section_variable에 접근

하고 싶다는 것이다. 그러면 flag1을 1로 바꿔주고 turn을 0으로 둬 상대방 스레드에 양보한다. 그 다음 위와 마찬가지로 flag0이 0이고 turn도 0이라면 0번 스레드가 임계 영역에 있는 것이므로 while문에 빠진다. **Unlock함수는** 스레드가 임계영역을 나올 때 사용하는 함수로 인자 self로 어떤 스레드가 임계 영역을 나왔는 지 않다. 그 다음 그에 해당하는 flag를 0으로 설정하여 다른 스레드가 들어올 수 있도록 한다.

func함수는 critical_section_variable을 COUNTING_NUMBER(#define으로 선언함)만큼 증가시켜주는 함수이다. 이 함수는 인자 s를 통해 이 계산을 하는 스레드가 몇 번인지 받는다. 그 다음 critical_section_variable을 증가시키기 전에 lock함수의 인자로 넣어줘 critical_section_variable++에 여러 스레드가 한번에 접근하지 못하도록 한다. 그다음 critical_sesction_variable계산을 다 마치면 unlock을 통해 lock을 풀어 다른 스레드가 들어올 수 있도록 한다.

main에서는 pthread p1과 p2를 선언해준다. 다음에 parameter[2]={0,1}을 선언해준다. Pthread_create함수로 p1 스레드를 만들어주고 func함수를 실행시켜준다. 그 함수에 인자로 parameter[0]을 넣어줌으로써 func함수에서 동기화 과정에 사용하는 thread_num을 제공해준다. 마찬가리고 p1또한 pthread_create함수로 만들어주고 func함수를 실행시키는데 이때는 parameter[1]을 func함수의 인자에 넣어줌으로써 func에서 p1과 구분되게 해준다. pthread_join으로 스레드의 종료를 기다린 후 스레드가 모두 종료하면, printf문을 통해 스레드들이 증가시킨 critical_section_variable과 실제 예상되는 값을 출력시켜준다.

<실행모습>



B. 과제 B

1. 자료구조 설명

두 프로세스가 공유하는 메모리에 들어갈 정보들 turn, flag, critical_section_variable를 가지는 smStruct라는 구조체를 만들어준다. Turn은 차례를 나타내는 변수이고, flag[2]는 누가 임계영역에 진입할 것인지 알려주는 변수이다. 자식 프로세스가 2개이기 때문에, 2의 크기를 가지는 배열로 선언해주었다. Critical_section_variable은 프로세스가 증가시키는데 사용하는 변수이다. 메인문에서는 이 smStruct라는 구조체로 정의된 자료형을 통해 새 변수를 생성할 것이다. 프로세스가 공유하는 Shared memory의 주소를 이 구조체에 할당하여 두 프로세스가 이 구조체 안에 있는 변수들을 공유할 수 있도록 하고, 특히 critical_section_variable를 두 프로세스 모두 증가시킬 것이다. 이 변수에 접근하여 증가

시킬 때 lock을 걸어 두 프로세스가 동시에 접근하지 못하도록 할 것인데 이는 프로세스의 id를 getpid로 얻어서 사용할 것이다.

2. 동기화 방법 설명

Peterson's algorithm을 사용해서 동기화 하였다**. lock함수**에서는 인자로 공유메모리인 smstruct과 프로세스의 order(혹은 **몇 번 프로세스인지 구분하는**)인 order를 받는다. Order에 해당하는 프로세스가 임계영역에 진입하고 싶다는 뜻이므로 smstruct의 order에 해당하는 flag[order]를 1로 바꿔준다. 그 다음 turn을 상대로 바꿔준다. Order가 1이라면 상대는 0이고, order가 0이라면 상대는 1이므로 1-order식을 사용해서 turn을 바꿔준다. 그 다음 상대의 flag가 1이고 turn도 1이라면 상대프로세스가 임계영역에서 나올 때까지 while문에서 대기한다**. Unlock 함수**에서는 자신이 임계지역에서 나오면 lock을 풀어주는 것으로 해당하는 프로세스의 flag를 0으로 바꿔줘야 한다. 따라서 flag가 있는 smstruct과 해당 프로세스를 알려주는 order를 인자로 받고 smstruct->flag[order]를 0으로 바꿔준다. 이렇게 함수 설정을 했다면, critical_section_variable을 lock과 unlock 사이에 두어 두 프로세스가 동시에 접근할 수 없도록 한다.

3. 프로그램 구조 설명

Parent.c

공유할 변수들을 담은 구조체 smStruct를 만든다. 이 구조체에는 각 프로세스가 증가시킬 critical_section_variable과 이 변수에 프로세스가 동시에 접근하지 못하도록 하는 lock에 사용되는 turn과 flag변수가 있다. main문에서 자식프로세스 pid1과 pid2를 선언한다. 그 다음 공유메모리로 사용할 구조체 smstruct를 선언해준다. 그다음에 shmget((key_t)1234,1024,IPC_CREAT|0666))를 통해 shared memory의 id를 얻어온다. 식별할 수 있는 고유번호는 1234로, 크기는 1024로 설정하고 해당하는 shared memory가 없다면 새로 생성할 수 있도록 IPC_CREAT를 3번째 인자로 넣어준다. 이 함수가 실패했다면 -1을 리턴하기 때문에 리턴값이 -1이라면 "shmget failed"를 출력하도록 한다. 그다음 shmat함수를 통해서 아직 메모리 공간에 붙여지지 않은 shared memory를 프로세스 메모리 공간에 attach할 수 있도록 한다. Shmat을 통해 반환되는 공유메모리 주소를 shmaddr에 저장해준다. 이 값이 -1라면 실패한 것이기 때문에 shmaddr값이 -1이면 "failed attact address"를 출력하도록 한다. Smstruct = shmaddr를 통해서 shared memory 공간에 smstruct가 그대로 들어갈 수 있도록 해준다. Smstruct의 critical_section_variable은 0으로 설정해주고 flag[0]과 flag[1]도 0으로 설정해준다. 그다음 pid1=fork()를 통해 자식 프로세스를 만든다. 이 프로세스는 execl함수를 통해 child프로그램을 실행시킨다.

Pid2=fork()를 통해 두번째 자식 프로세스를 만들고 이 프로세스 또한 execl함수를 통해 child 프로그램을 실행시킨다. 그 다음 waitpid를 통해 두 자식이 모두 종료될 때 까지 기다린다. 자식 프로세스가 모두 종료되었다면, 공유메모리의 critical_section_variable과 실제로 나올 것이라고 예상되는 값을 출력해준다. 그다음 shmat을 통해 프로세스 메모리 공간에서 공유 메모리를 뗀다. 실패하면 -1을 반환하므로 리턴 값이 -1일 경우 "detach failed"를 출력한다. 그다음 shmctl함수에 IPC_RMID를인자로 넣어줌으로써 shared memory를 제거한다. 실패했다면 -1을 반환하므로 리턴값이 -1일 경우 "remove failed"를 출력해준다.

Child.c

Parent 코드와 마찬가지로 공유할 메모리의 모습인 smStruct 구조체를 만들어준다. 그 다음 임계영역에 동시에 접근하지 못하도록 하는 lock함수와 unlock함수를 만든다. (2. 동기화 방법에서 자세하게 설명) critical_section_variable의 값을 실제로 count하기 위해 사용되는 변수 localcount를 선언해준다. smStruct *smstruct를 선언해준다. shmget함수를 통해 공유메모리의 id를 얻어오거나 없으면 새로 생성한다. 그 다음 shmat을 통해서 공유메모리 주소를 얻고 이를 shmaddr 변수에 할당한다. Smstruct = shmaddr을 통해 shared memory에 smstruct가 그대로 들어갈 수 있도록 해준다. (공유메모리 부분에 대한 자세한 설명은 parent.c 코드설명에 있다.) 그 다음 이 프로세스에 대한 정보를 얻기 위해 Myorder라는 변수를 선언해준다. 자식프로세스는 연달아 2개 생긴다. Getpid는 프로세스의 id를 얻는 함수이다. getpid%2를 하면 한 프로세스는 0, 다른 프로세스는 1의 값을 가질 것이다. 이 0과 1이 lock과 unlock에서 flag와 turn을 결정할 때 사용된다. for문을 통해서 COUNTING_NUMBER만큼 critical_section_variable과 local count를 증가시켜 줄 것이다. 이 critical_section_variable은 공유하는 메모리의 들어있는 변수이기 때문에 이 변수 사이에 lock과 unlock 함수를 사용해서 동시에 두 프로세스가 접근할 수 없도록 해준다. 그 다음 order와(프로세스의 lock과 unlock의 사용되는 프로세스를 구분하는 변수) 와 실제 process pid를 출력한다. 또한, "child finish! Local count"를 통해 critical_section_variable의 값을 증가시키는 행위를 실제로 센 localcount 변수를 출력시킨다. 마지막으로 shmdt함수를 통해 프로세스의 메모리 공간에서 공유메모리를 뗀다.

<실행모습>