2016025469 컴퓨터공학과 서건식

운영 체제 HW#5

제출 일자 : 2020/04/23

A. 과제 A

1. 자료구조 설명

선언한 변수를 기준으로 자료구조를 설명하겠습니다.

1) Int critical_section_variable

각 쓰레드가 접근할 전역 변수입니다.

2) Atomic_int turn, flag0, flag1

Atomic_int 클래스의 변수이므로, 값은 atomic하게 변합니다.

2. 동기화 방법 설명

동기화 방법은, lock과 unlock 함수에서 thread_num을 인자로 받아 그 인자의 값(쓰레드의 넘버)에 따라 flag과 turn값을 변경하며 동기화 합니다.

```
int critical_section_variable = 0;

atomic_int turn(0);
atomic_int flag0(0), flag1(0);

void lock(int self)
{
        if (self == 0) // if the thread_num is 0, set turn variable to another thread number (1)
and flag0 variable to 1. because flag0 is the flag of thread_num 1
        {
                flag0 = 1;
                turn = 1;
                while(flag1 == 1 && turn == 1 );
        }
        if(self == 1) // same as if self == 0
        {
                flag1 = 1;
                turn = 0;
                while(flag0 == 1 && turn == 0);
        }

}
```

lock함수에서, 들어온 인자값에 따라 경우를 나눠주었으며 flag0과 flag1은 각각 쓰레드1과 쓰레드2의 플래그 값이므로 경우마다 접근하여 값을

변경합니다.

```
void unlock(int self)
{
        if(self == 0) //same as lock function. if the thread_num is 0 , set thread_num's flag to
0
        {
                flag0 = 0;
        }
        else if (self == 1)
        {
                flag1 = 0;
        }
}
```

Unlock 함수 또한 동일합니다. 각 쓰레드넘버에 해당하는 flag값을 false 로 바꿔줌으로써 자신의 critical section 작업이 끝났음을 알려줍니다.

3. 프로그램 구조 설명

```
void* func(void *s)
{
        int* thread_num = (int*)s;

        for(int i = 0; i<COUNTING_NUMBER ; i++)
        {
                lock(*thread_num);
                critical_section_variable++;
                unlock(*thread_num);
        }
        return 0;
}
```

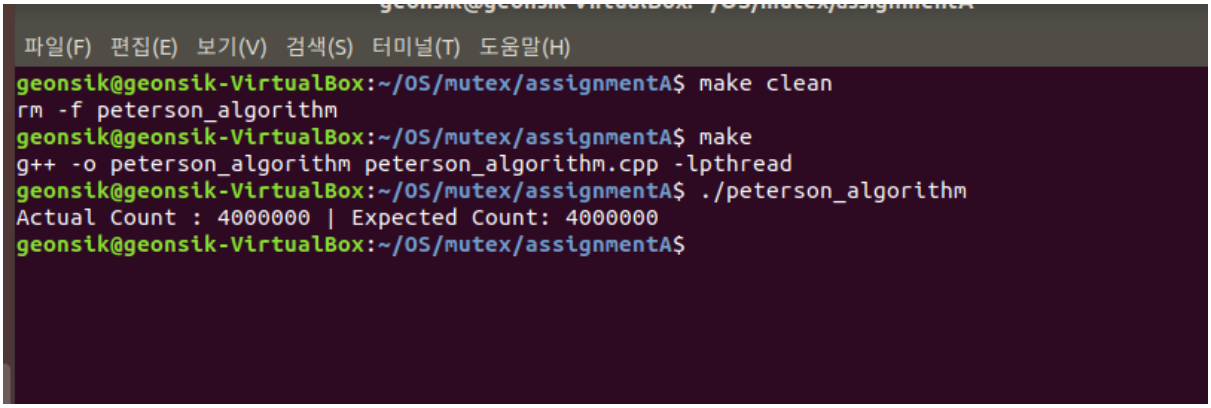
위에서 설명한 lock과 unlock 함수를 기반으로, critical_section_variable이라는 변수(이곳이 critical section 입니다.)에 접근하기 전, 후에 lock 과 unlock함수를 호출하여 동기화를 실행합니다.

또한 인자로 받는 s는 thread_num을 의미하며, main을 보시면

```c
int main(void)
{
        pthread_t p1, p2; // make 2 thread

        int parameter[2] = {0,1};

        pthread_create(&p1,NULL,func,(void*)&parameter[0]);
        pthread_create(&p2,NULL,func,(void*)&parameter[1]);

        pthread_join(p1,NULL);
        pthread_join(p2,NULL);

        printf("Actual Count : %d | Expected Count: %d\n", critical_section_variable,COUNTING_NU
BER*2);
        return 0;
}
```

각각 0과 1로 이루어져 있습니다.

main문의 끝에서 실제 계산된 값과 예상된 값을 출력합니다.



실행결과 예상된 값과 실제 계산된 값이 같으므로 동기화 문제를 해결함을 확인할 수 있습니다.

B. 과제 B

1. 자료구조 설명

1) Parent.c

```c
#define COUNTING_NUMBER 2000000

typedef struct smStruct{
        int processidassign;
        int turn;
        int flag[2];
        int critical_section_variable;
} smStruct;
```

(1) smStruct

해당 구조체는 제시해주신 것과 동일합니다.

(2) Shared memory 변수들

```
int main(void)
{
        int shmid;
        char * shmaddr;
        int ret;
        smStruct *smstruct;
```

Shared memory 할당을 위한 변수들입니다.

(3) 자식 process 생성을 위한 변수들

```
//make pid_t variables.
pid_t pid1;
pid_t pid2;
int status1;
int status2;
```

Process 2개를 생성하기 위한 변수들입니다.

2) Child.c

(1) 구조체, shared memory 변수는 parent.c 와 같습니다.

(2) Localcount, Myorder

```
int localcount = 0;
smStruct *smstruct;
int Myorder;
```

Localcount 변수는 실제로 critical_section_variable 값을 증가하는 횟수를 세기 위한 변수이며, 정상적으로 작동하면 2000000입니다.

Myorder는 smstruct의 processidassign 멤버변수를 저장합니다.

2. 동기화 방법 설명

동기화 방법은 다음과 같습니다.

Smstruct에 있는 processidassign 멤버변수를 통해 각 프로세스 마다 다른 processid를 배정합니다. 이를 통해서 프로세스 id가 구별되고, Peterson's algorithm을 구현함으로써 동기화 하였습니다.

Myorder라는 변수를 child.c에 선언하여, processidassign 의 값을 저장해 주었습니다.

```
//if processidassign is 0, set Myorder variable to that, and +1
if(smstruct->processidassign == 0)
{
        Myorder = smstruct->processidassign;
        smstruct->processidassign++;
}
//So the other child process can set Myorder variable to 1.
else
{
        Myorder = smstruct->processidassign;
}
```

0번 id가 다른 프로세스에서 지정되었다면 값을 올려줍니다. 따라서 다른 프로세스는 if문에 의해서 1을 저장하게 되어 각 프로세스를 구분하는 것이 가능합니다.

```
//lock. Myorder is processid.
void lock(smStruct *smstruct ,int Myorder)
{
        smstruct->flag[Myorder] = 1;
        smstruct->turn = 1-Myorder;
        while(smstruct->flag[1-Myorder] == 1 && smstruct->turn == 1-Myorder);
}
//unlock. Myorder is processid.
void unlock(smStruct *smstruct,int Myorder)
{
        smstruct->flag[Myorder] = 0;
}
```

이렇게 구분된 processid를 기반으로, Peterson's algorithm을 구현합니다.

해당 함수는 child.c에만 구현하였습니다.

3. 프로그램 구조 설명

1) Parent.c

부모 프로세스는 fork함수를 통해 2개의 자식프로세스를 생성합니다. execl함수를 사용하여 child 실행파일을 실행할 수 있도록 하였고,

2개의 자식 프로세스를 모두 생성하고 난 뒤에 waitpid 함수를 통해서 자식프로세스가 모두 종료될 때 까지 기다립니다.

만약 모두 종료되었다면, 실제값과 예상된 값을 출력합니다. 그 이후 shared memory를 detach 하고 delete합니다. Delete는 자식프로세스에서 진행하지 않고 오직 부모프로세스에서 진행합니다.

2) Child.c

Shard memory id를 얻고 attach한 후에 shmaddr를 smstruct에 저장하여 멤버변수 값에 접근합니다.

이후 동기화 과정을 거친 후에

```
pid = getpid();
printf("Myorder = %d,process pid = %d\n",Myorder,pid);
for(i = 0 ; i<COUNTING_NUMBER ; i++)
{
        localcount++;
        lock(smstruct, Myorder);
        smstruct -> critical_section_variable++;
        unlock(smstruct,Myorder);
}
printf("child finish! local count = %d\n",localcount);
```

자신이 부모로부터 부여받은 Myorder의 값과 pid를 출력합니다.

그 이후 critical_section_variable 값을 증가시키고(자식 프로세스에서만 증가시킵니다.), localcount값을 출력하면서 실제로 얼마나 수행되었는지를 출력합니다.

이 후 shared memory를 detach 합니다.

실행결과는 다음페이지에 있습니다.

```
geonsik@geonsik-VirtualBox:~/OS/mutex/assignmentB$ make clean
rm -f parent child
geonsik@geonsik-VirtualBox:~/OS/mutex/assignmentB$ make
gcc -o parent parent.c
parent.c: In function ‘main’:
parent.c:40:11: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
  smstruct = shmaddr;
           ^
parent.c:59:17: warning: null argument where non-null required (argument 2) [-Wnonnull]
                execl("child",NULL);
                ^~~~~
parent.c:59:17: warning: not enough variable arguments to fit a sentinel [-Wformat=]
parent.c:73:25: warning: null argument where non-null required (argument 2) [-Wnonnull]
                        execl("child",NULL);
                        ^~~~~
parent.c:73:25: warning: not enough variable arguments to fit a sentinel [-Wformat=]
parent.c:77:32: warning: implicit declaration of function ‘waitpid’; did you mean ‘getpid’? [-Wim
plicit-function-declaration]
                        pid1 = waitpid(pid1,&status1,0);
                               ^~~~~~~
                               getpid
gcc -o child child.c
child.c: In function ‘main’:
child.c:59:11: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
  smstruct = shmaddr;
           ^
geonsik@geonsik-VirtualBox:~/OS/mutex/assignmentB$ ./parent
Myorder = 0,process pid = 17090
Myorder = 1,process pid = 17091
child finish! local count = 2000000
child finish! local count = 2000000
Actual Count : 3999858 | Expected Count : 4000000
geonsik@geonsik-VirtualBox:~/OS/mutex/assignmentB$
```

동기화가 이루어져 실제 계산된 값이 예상된 값과 거의 비슷하게 계산된 것을 확인할 수 있습니다.