[HW3]_20172129_박예솔

Process / Thread Synchronization

과목	운영체제
제출 일자	2020년 10월 21일
담당 교수	박호현 교수님
학 과	컴퓨터공학과
학 번	20172129
이 름	박예솔

1. 아래의 프로그램은 x86 프로세서에서 xchg 명령어를 이용하여 f1과 f2 threads 간의 mutual exclusion을 테스트하는 프로그램이다.

(1) f1과 f2가 우리은행에서 돈을 빌려서 사용하고 갚는 프로그램 시뮬레이션

① 코드 설명

메인함수는 기존의 프로그램의 코드와 같다. critical_region( ) 함수와 각각의 thread 함수를 수정했는데, 그 코드는 다음과 같다.

```
void noncritical_region(char *p) {
    int d = rand()%1000000;
    printf("%s is doing economic activity...\n", p);
    usleep(d);
}

static void* f1(void* p) {
    int borrow = 0;
    for(int i=0; i<10; i++) {
        puts("f1 wait for f2");
        // borrow money,
        enter_region();
        puts("");
        printf("========== f1 start banking.(critical section) ==========\n");
        borrow = critical_region(p, borrow);
        printf("========== f1 end banking.(critical section) ==========\n");
        puts("");
        leave_region();
        // do economy activity
        noncritical_region(p);
    }
    pthread_exit((void*)borrow);
}

static void* f2(void* p) {
    int borrow = 0;
    for(int i=0; i<10; i++) {
        puts("f2 wait for f1");
        // borrow money,
        enter_region();
        puts("");
        printf("========== f2 start banking.(critical section) ==========\n");
        borrow = critical_region(p, borrow);
        printf("========== f2 end banking.(critical section) ==========\n");
        puts("");
        leave_region();
        // do economy activity
        noncritical_region(p);
    }
    pthread_exit((void*)borrow);
}
```

우선 스레드 함수부터 설명하면 f1과 f2가 빌려간 금액을 저장하는 지역변수 borrow를 각각 선언한다. critical section에 들어가면 critical_region 함수를 부르게 되는데, 이때 입력 파라미터 p는 해당 스레드의 포인터, borrow는 해당 스레드에서 현재까지 빌린 금액이다. critical_region 함수의 출력 값으로는 은행업무가 수행된 후 갱신된 대출금액을 정수 값으로 return하도록 설계하였다. 따라서 borrow 변수에 함수의 출력 값을 저장해주면 된다. 반복문을 다 돌고 스레드가 종료될 때, f1과 f2 각각이 빌린 총 대출금을 메인 함수에 return할 수 있도록 pthread_exit를 사용해주었다.

```c
int critical_region(char *p, int borrow) {
    int d = rand()%1000000;
    int m = rand()%1000;
    int c = rand()%2;

    // error handling
    if (borrow == 0 && c == 1) c = 0;
    if (total_money < m && c == 0){
        printf("The bank went bankrupt....T_T");
        return 0;
    }

    // case 0: borrow money
    if (c == 0){
        printf("Before banking : %s borrowed [%d won].\n", p, borrow*1000);
        total_money = total_money - m;
        borrow = borrow + m;
        printf(">> %s borrow [%d won]. (Current Bank's balance = %d)\n", p, m*1000, total_money*1000);
        printf("After banking : %s borrowed [%d won].\n", p, borrow*1000);
        usleep(d);
        return borrow;
    }
    // case 1: repay money
    else if (c == 1){
        // error handling
        if (borrow < m) m = borrow;

        printf("Before banking : %s borrowed [%d won].\n", p, borrow*1000);
        total_money = total_money + m;
        borrow = borrow - m;
        printf(">> %s repay [%d won]. (Current Bank's balance = %d)\n", p, m*1000, total_money*1000);
        printf("After banking : %s borrowed [%d won].\n", p, borrow*1000);
        usleep(d);
        return borrow;
    }
}
```

다음으로 critical_region 함수의 수정 내용을 설명하면, m값은 랜덤으로 부여된 대출금 or 상환금의 가격이고, c는 case의 약자로 0과 1을 랜덤으로 부여하게 설정하여 0이면 대출, 1이면 상환으로 지정하였다. 이때 발생할 수 있는 에러의 경우의 수를 3개 정도로 분류하였는데 첫째는 빌린 대출금이 없는데 상환을 하는 경우이고, 둘째는 은행이 가지고 있는 현재 잔액보다 대출금이 클 경우이다. 이 두가지 경우를 case에 들어가기 전에 if 문을 사용해서 첫번째의 경우는 상환이 아니라 대출이 될 수 있도록 c값을 변경시켜주었고, 둘째의 경우는 '은행이 파산했다'는 메시지를 출력하고 종료하도록 하였다. 마지막 세번째 에러의 경우는 빌린 대출금을 초과하는 금액을 상환하는 경우인데, 이것은 c가 1(상환)일 때 m(상환금)값을 borrow값으로 바꿔주어 대출금을 0으로 만들어 모두 갚는 시나리오로 수정하였다.

critical section에서 잘 작동하는지 확인을 용이하게 하기위해, 은행거래 이전의 대출금액을 먼저 출력해주고, 은행에서 대출한 금액이나 상환한 금액을 출력하면서 현재 은행의 잔고도 같이 보여주었다. 마지막으로는 은행 거래 후, 갱신된 총 대출금을 확인할 수 있도록 하였다. 여기서 대출 금액/상환 금액을 보여주기 바로 직전에 전역 변수로 선언된 은행의 잔고(shared data)와 빌린 대출금을 갱신해주었다. 여기서 잔고의 단위를 천원 단위로 설정하였기 때문에 1000을 곱하여 출력하였다.

마지막으로 main 함수에서 최종 f1의 대출금, f2의 대출금, 은행의 잔고를 출력하게 하여 프로그램의 sync가 잘 맞는지 확인할 수 있도록 하였다.

② 실행 화면

위에서 왼쪽을 시작으로 1,2/3,4 순서이다. 처음 은행의 잔고는 10,000,000원 임을 알 수 있다.

```
yesol@yesol-virtual-machine:~/문서/OS-homework/hw3$ ./xchg1
********* Initial Bank's Balance = 10000000 won *********
f2 wait for f1

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [0 won].
>> f2 borrow [886000 won]. (Current Bank's balance = 9114000)
After banking : f2 borrowed [886000 won].
f1 wait for f2
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [0 won].
>> f1 borrow [335000 won]. (Current Bank's balance = 8779000)
After banking : f1 borrowed [335000 won].
f2 wait for f1
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [886000 won].
>> f2 borrow [421000 won]. (Current Bank's balance = 8358000)
After banking : f2 borrowed [1307000 won].
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...
f1 wait for f2

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [335000 won].
>> f1 repay [59000 won]. (Current Bank's balance = 8417000)
After banking : f1 borrowed [276000 won].
f2 wait for f1
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...
```

```
========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [1307000 won].
>> f2 borrow [426000 won]. (Current Bank's balance = 7991000)
After banking : f2 borrowed [1733000 won].
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...
f1 wait for f2

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [276000 won].
>> f1 repay [276000 won]. (Current Bank's balance = 8267000)
After banking : f1 borrowed [0 won].
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...
f2 wait for f1

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [1733000 won].
>> f2 borrow [530000 won]. (Current Bank's balance = 7737000)
After banking : f2 borrowed [2263000 won].
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...
f1 wait for f2

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [0 won].
>> f1 borrow [135000 won]. (Current Bank's balance = 7602000)
After banking : f1 borrowed [135000 won].
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...
f2 wait for f1

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [2263000 won].
>> f2 repay [58000 won]. (Current Bank's balance = 7660000)
After banking : f2 borrowed [2205000 won].
========== f2 end banking.(critical section) ==========
```

```
f2 is doing economic activity...
f1 wait for f2

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [135000 won].
>> f1 repay [135000 won]. (Current Bank's balance = 7795000)
After banking : f1 borrowed [0 won].
f2 wait for f1
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [2205000 won].
>> f2 repay [373000 won]. (Current Bank's balance = 8168000)
After banking : f2 borrowed [1832000 won].
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...
f1 wait for f2

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [0 won].
>> f1 borrow [537000 won]. (Current Bank's balance = 7631000)
After banking : f1 borrowed [537000 won].
f2 wait for f1
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [1832000 won].
>> f2 repay [370000 won]. (Current Bank's balance = 8001000)
After banking : f2 borrowed [1462000 won].
f1 wait for f2
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [537000 won].
>> f1 borrow [980000 won]. (Current Bank's balance = 7021000)
After banking : f1 borrowed [1517000 won].
========== f1 end banking.(critical section) ==========
```

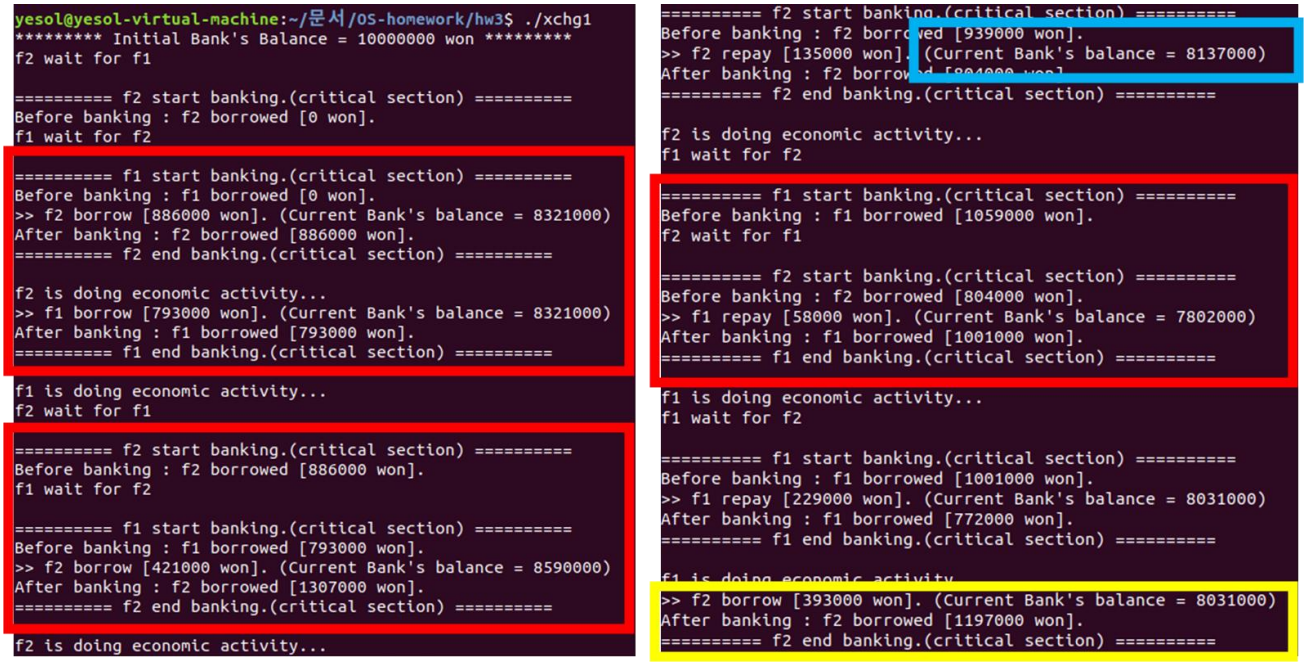
```
f1 is doing economic activity...
f2 wait for f1

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [1462000 won].
>> f2 borrow [170000 won]. (Current Bank's balance = 6851000)
After banking : f2 borrowed [1632000 won].
f1 wait for f2
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [1517000 won].
>> f1 borrow [925000 won]. (Current Bank's balance = 5926000)
After banking : f1 borrowed [2442000 won].
f2 wait for f1
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [1632000 won].
>> f2 borrow [505000 won]. (Current Bank's balance = 5421000)
After banking : f2 borrowed [2137000 won].
f1 wait for f2
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [2442000 won].
>> f1 borrow [857000 won]. (Current Bank's balance = 4564000)
After banking : f1 borrowed [3299000 won].
f2 wait for f1
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...

========== f2 start banking.(critical section) ==========
Before banking : f2 borrowed [2137000 won].
>> f2 borrow [545000 won]. (Current Bank's balance = 4019000)
After banking : f2 borrowed [2682000 won].
f1 wait for f2
```

```
f1 wait for f2
========== f2 end banking.(critical section) ==========

f2 is doing economic activity...

========== f1 start banking.(critical section) ==========
Before banking : f1 borrowed [3299000 won].
>> f1 repay [364000 won]. (Current Bank's balance = 4383000)
After banking : f1 borrowed [2935000 won].
========== f1 end banking.(critical section) ==========

f1 is doing economic activity...
All threads finished.
f1 borrow [2935000], f2 borrow [2682000], Final Bank's balance [4383000]
yesol@yesol-virtual-machine:~/문서/OS-homework/hw3$
```

f1, f2 각각 10번씩 critical section을 잘 수행하는 것을 전체 결과를 통해 알 수 있었다. 최종적으로 빌린 금액 역시, f1이 2,935,000원 f2가 2,682,000원으로 전체에서 5,617,000원을 빌린 것 이므로 처음 잔고인 천만원에서 빌린 금액을 빼면 현재 은행에 남은 잔고는 4,383,000원으로 일치하는 것을 알 수 있었다.

(2) Mutual exclusion을 하는 경우 vs 안 하는 경우 비교

앞의 코드에서 enter_region 과 leave_region 함수를 주석 처리하고 코드를 실행시켜보았다. Mutual exclusion의 차이를 뚜렷하게 느끼기 위해 돈이 들어오고 나갈 때, 시간 차이를 반영하였다.



중간 과정을 생략하였음에도 시작부터 f2의 banking이 끝나기 전에 f1이 끼어드는 것을 확인할 수 있다. 왼쪽 첫번째 빨간 박스를 보면 f2가 1000만원에서 886,000원을 빌리고 남은 값이 은행의 잔고에 출력되어야 하는데, f2가 끝나기 전에 f1이 banking을 시작해버려서 f1이 빌리는 793,000원 역시 은행의 잔고에서 같이 빠져나가고 f2에 출력되게 되었다. 오른쪽 빨간 박스 역시 바로 직전의 거래 후 은행 잔고는 8,137,000원이고 f1이 58,000원을 상환하는 것이므로 은행의 현재 잔고는 증가해야 된다. 그러나 f1의 거래가 끝나기 전에 f2가 끼어들어서 393,000원을 대출해갔다. 따라서 f1의 거래 결과는 오히려 잔고가 감소하는 것으로 출력되어버렸다.

2. 앞에서 구현한 프로그램을 xchg 명령어 대신에 Linux semaphore을 이용하여 재구현하고 테스트하시오.

① 코드 설명

```
int total_money = 10000;
pthread_mutex_t mutex;
```

1번의 xchg 명령어를 사용한 프로그램에서 enter_region 과 leave_region 함수를 사용한 부분을 지우고 mutex (binary semaphore)을 넣어주었다. 우선 전역변수로 semaphore id인 mutex를 만들어주었다.

```
static void* f1(void* p) {
    int borrow = 0;
    for(int i=0; i<10; i++) {
        puts("f1 wait for f2");
        // borrow money,
        //enter_region();
        pthread_mutex_lock(&mutex);
        puts("");
        printf("========== f1 start banking.(critical section) ==========\n");
        borrow = critical_region(p, borrow);
        printf("========== f1 end banking.(critical section) ==========\n");
        puts("");
        //leave_region();
        pthread_mutex_unlock(&mutex);
        // do economy activity
        noncritical_region(p);
    }
    pthread_exit((void*)borrow);
}
```

enter_region 대신 pthread_mutex_lock을 통해 critical section에 두 스레드가 동시에 들어가지 못하도록 해주었다.

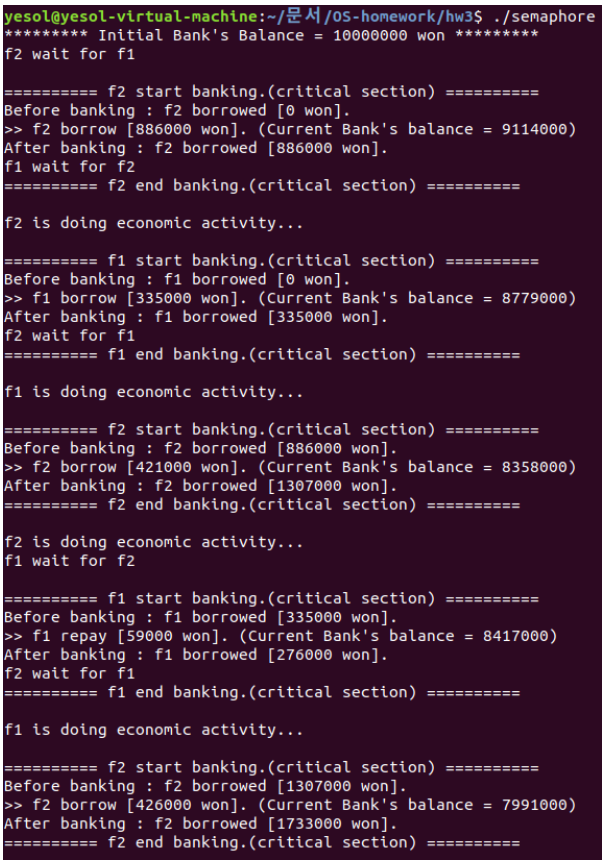
```
static void* f2(void* p) {
    int borrow = 0;
    for(int i=0; i<10; i++) {
        puts("f2 wait for f1");
        //enter_region();
        pthread_mutex_lock(&mutex);
        puts("");
        printf("========== f2 start banking.(critical section) ==========\n");
        borrow = critical_region(p, borrow);
        printf("========== f2 end banking.(critical section) ==========\n");
        puts("");
        //leave_region();
        pthread_mutex_unlock(&mutex);
        // do economy activity
        noncritical_region(p);
    }
    pthread_exit((void*)borrow);
}
```

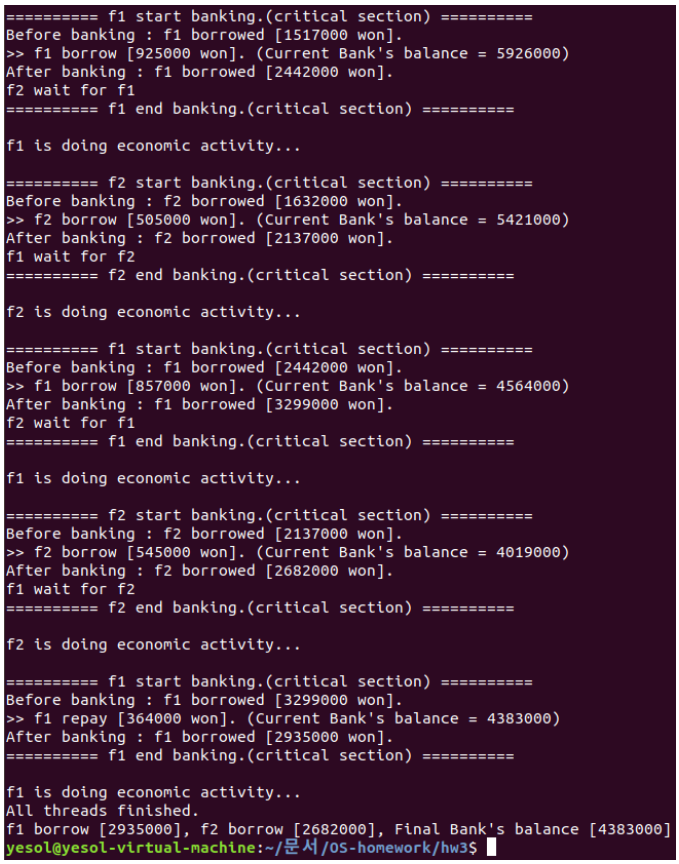
leave_region 대신 pthread_mutex_unlock을 통해 critical section을 나올 때 다른 스레드가 들어갈 수 있게 잠금을 풀어주도록 하였다.

```
int main() {
    printf("********* Initial Bank's Balance = %d won *********\n", total_money*1000);
    int rc;
    int f1_borrow;
    int f2_borrow;
    pthread_t t1, t2;
    pthread_mutex_init(&mutex, NULL);

    rc = pthread_create(&t1, NULL, f1, "f1");
    if(rc != 0) {
        fprintf(stderr, "pthread f1 failed\n");
        return EXIT_FAILURE;
    }

    rc = pthread_create(&t2, NULL, f2, "f2");
    if(rc != 0) {
        fprintf(stderr, "pthread f2 failed\n");
        return EXIT_FAILURE;
    }

    pthread_join(t1, (void**)&f1_borrow);
    pthread_join(t2, (void**)&f2_borrow);

    puts("All threads finished.");
    printf("f1 borrow [%d], f2 borrow [%d], Final Bank's balance [%d]\n", f1_borrow*1000,
f2_borrow*1000, total_money*1000);

    pthread_mutex_destroy(&mutex);
    return 0;

}
```

메인 함수에서는 전역변수로 선언했던 mutex id 값을 가져와서 mutex를 초기화해주고, 모든 스레드가 종료되면 mutex역시 destroy하도록 해주었다.

② 실행 화면





1-1의 결과와 마찬가지로 critical section에서 충돌없이 순차적으로 거래가 잘 진행되는 것을 알 수 있었다.