**Operating System HW3**

**학번:** 2016113832

**이름:** 윤동준

**담당교수:** 강순주

1. **개요**

과제3에서 thread의 수가 현저히 적어도 race condition이 일어날 수 있음을 확인하고 race condition을 mutex를 사용하여 해결하는 걸 보여준다.

swap함수를 사용하여 전역 변수를 계속 다른 값에 넣어주고 해서 원래는 create,threadadd, threadsub, join이 실행되고 있는데, swap이란 함수가 들어가게 되면서 우선순위가 바뀌어서이다. 계산이 선점이 되는 확률을 높이면서 쉽게 깨지게 preemption

1. **프로그램 구조 설명**

**2.1 함수에 대한 설명**

**-void add():** while문을 활용하여 전역 변수 first가 10번 더해진다.

**-void sub():** while문을 활용하여 전역 변수 first가 10번 빼진다.

**-void swap():** 이번 과제의 중점적으로 봐야하는 race condition을 보여줄 함수이다.

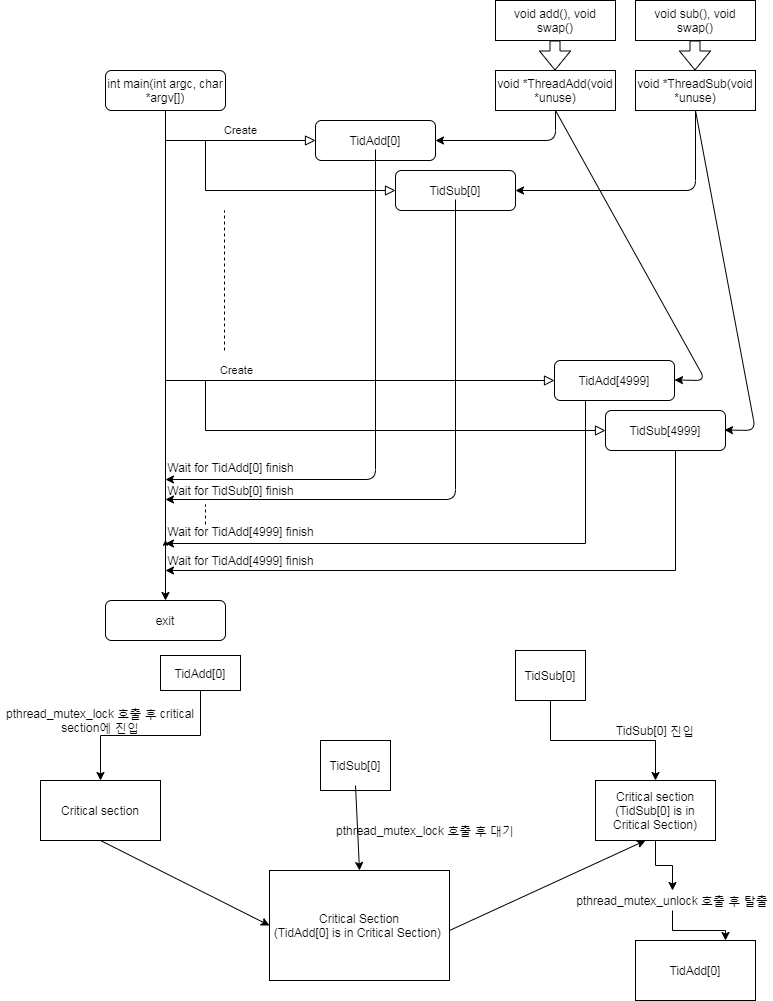
for문을 사용하여 전역 변수 값인 first를 temp를 사용하여 계속 switching해준다. 계속 switching함으로써 preemption이 자주 일어나게 하였다.

**-void \*ThreadAdd(void \*unuse):** thread함수를 만들어서 add함수와 swap함수를 넣어줬다. Pthread\_create의 세 번째 인자의 반환 값이 void \*타입이라 매개변수를 void\*로 선언하였다. 그리고 앞서 선언한 add함수와 swap함수를 넣어줬다. Mutex를 사용하지 않았을 때는 race condition이 일어나는 것을 확인할 수 있을 것이고, mutex를 사용하여 critical section을 만들어 주면

-**void \*ThreadSub(void \*unuse):** thread함수를 만들어서 sub함수와 swap함수를 넣어줬다. Pthread\_create의 세 번째 인자의 반환 값이 void \*타입이라 매개변수를 void\*로 선언하였다.

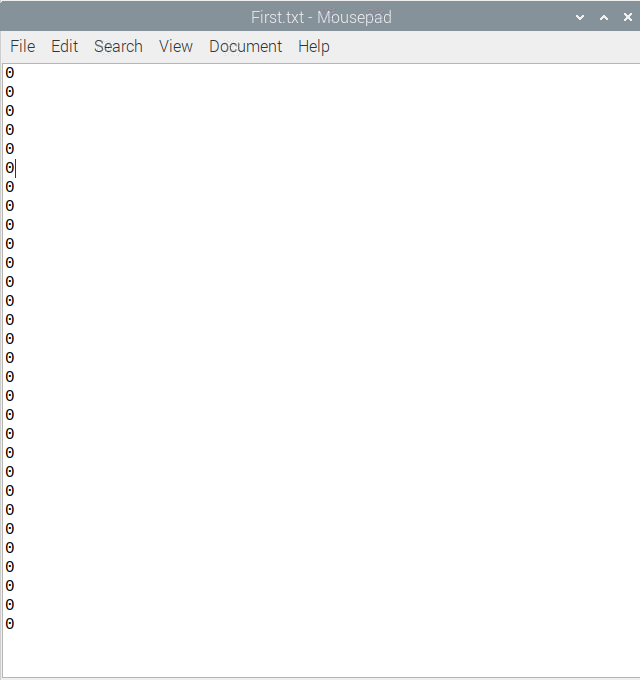
**-int main (int argc, char\* argv[]):** atoi를 활용하여 입력 받은 숫자를 integer형식으로 바꿔서 전역 변수에 저장을 하고, pthread\_t TidAdd[10], TidSub[10]라는 자료형을 선언 해주었다. pthread\_t는 pthread의 자료형이고 그 뒤는 thread의 이름이다. atoi를 사용하여 터미널에서 쓴 2번째 인자를 integer로 바꿔서 전역 변수 first에 저장했다. 그리고 for문을 사용하여 pthread\_create(&TidAdd[i], NULL, ThreadAdd, NULL), pthread\_create(&TidSub[i], NULL, ThreadSub, NULL) thread 총 10개를 만들어 주었다. Pthread\_create의 첫 번째 인자에는 pthread 식별자로 thread가 성공적으로 생성되면 thread 식별 값이 주어지고, 두 번째 인자에는 pthread 속성이 들어가는데, 기본적인 thread속성을 사용하기 위해 NULL값을 넣어줬다. 세 번째 pthread로 분기할 함수인 ThreadAdd와 ThreadSub를 넣어줬고, 마지막 인자에는 NULL값을 사용하여 함수에 넘겨줄 인자 값이 없다는 것을 알려주었다. 그리고 pthread\_join를 사용하여 pthread가 종료될 때까지 기다려주었다. 총 30번 돌려주기 위해 새로 만든 for문안에 pthread\_create, pthread\_join, 전역변수값을 출력해주는 것을 넣어줬다.

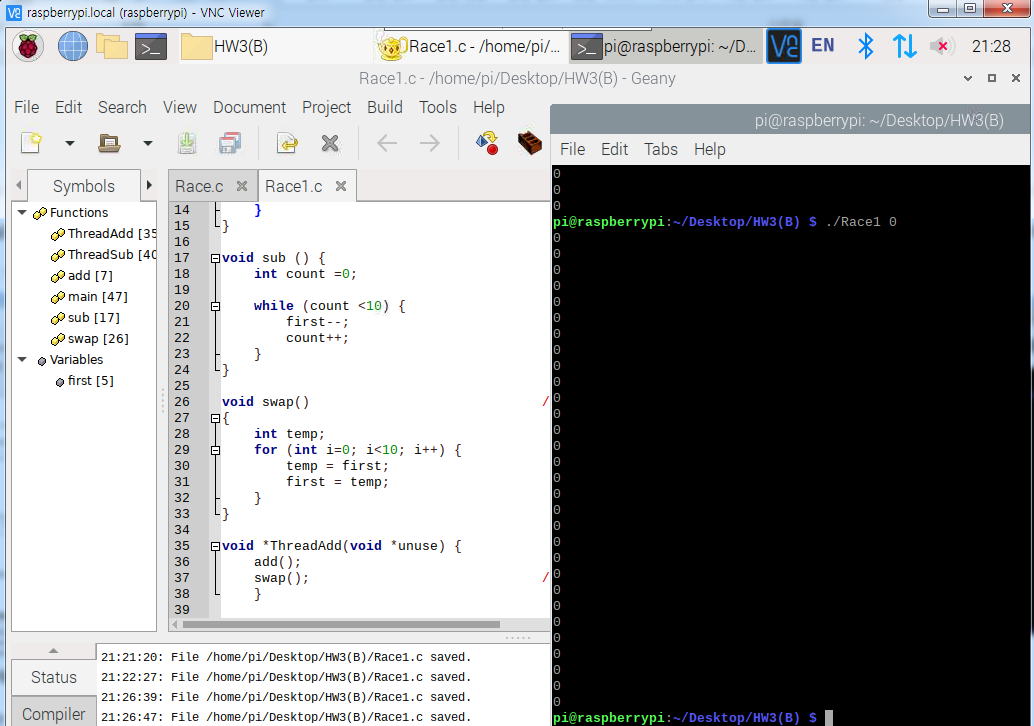
**2.2 다이어그램**



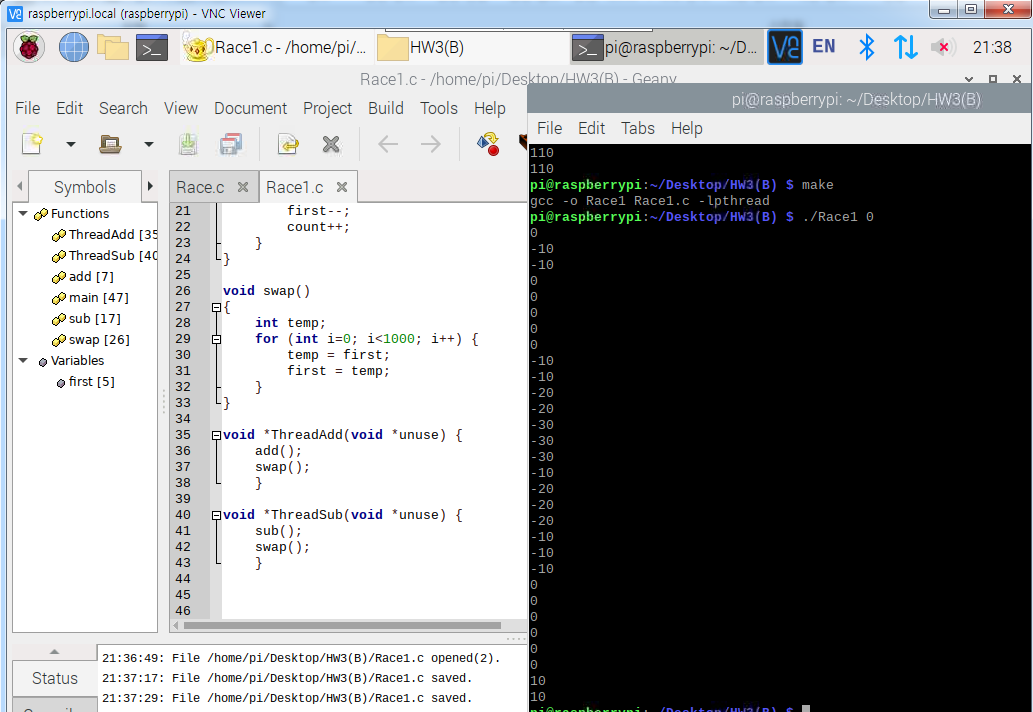
1. **실행결과**

1. 기존의 과제3에서 count 10번만 하는 프로그램에서 thread의 수를 5000개에서 5개로 줄였을 때, ./Race1 0으로 실행하였을 때 전역 변수 first의 값이 0이 나오는 것을 볼 수 있다. (first.txt, second.txt, third.txt 총 3번 실행파일 결과를 txt로 변환해서 저장하였다.)

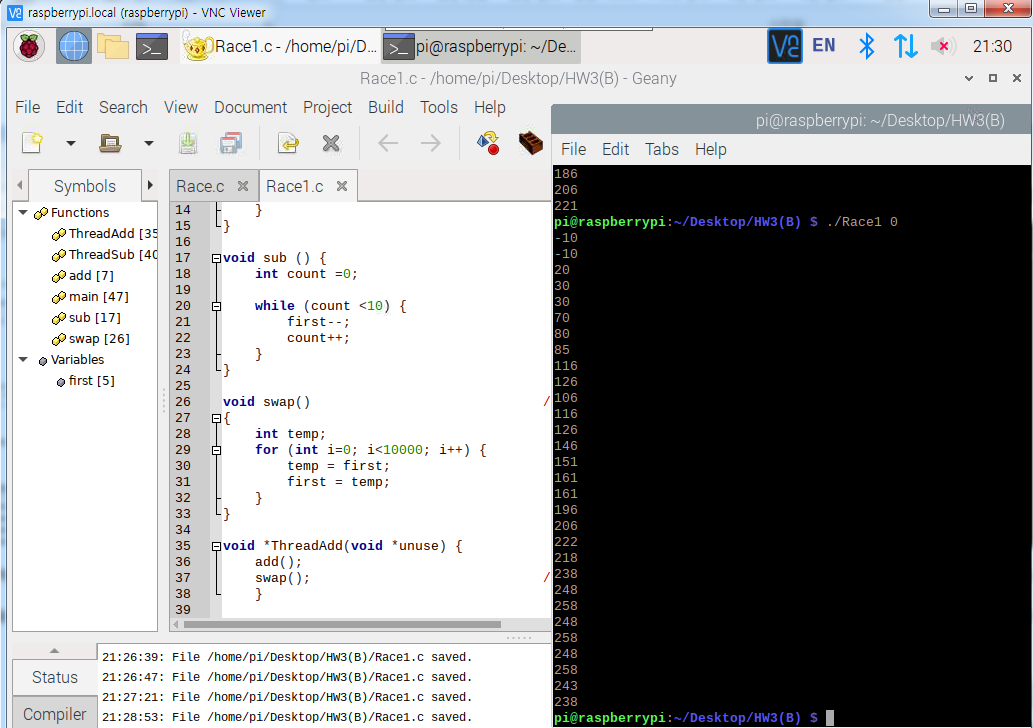


2. swap함수를 만들어서 void \*ThreadAdd(void \*unused)와 void \*ThreadSub(void \*unused)에 add(),sub()밑에 넣어줬다. Swap함수안에 for문의 i값을 10으로 설정하고, 실행을 10번해본 결과 충돌이 일어나지 않았다. 

그래서 i값을 1000으로 설정하였을 때는 충돌이 10번 실행했을 때, 9번 일어났고, race condition이 발생하는 것을 확인 할 수 있었다.

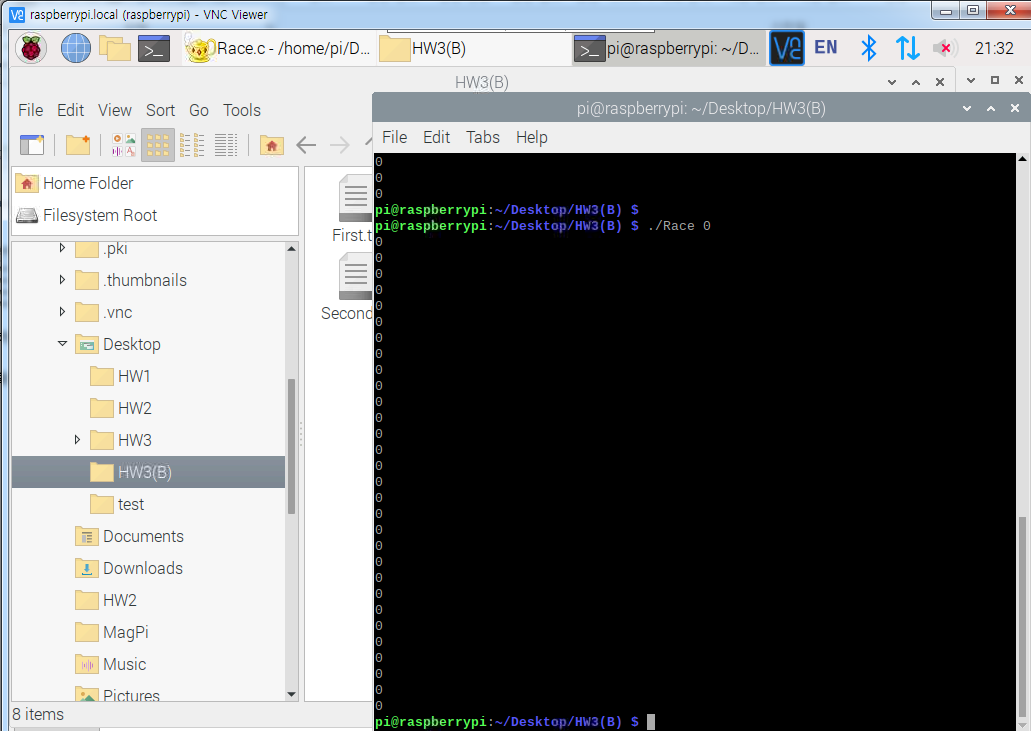


i값을 10000으로 설정하였을 때는 실행 할 때 마다 충돌이 무조건 일어나는 것을 볼 수 있었고, for문을 1000번 돌렸을 때보다 확연히 race condition이 자주 일어나는 것을 볼 수 있었다.

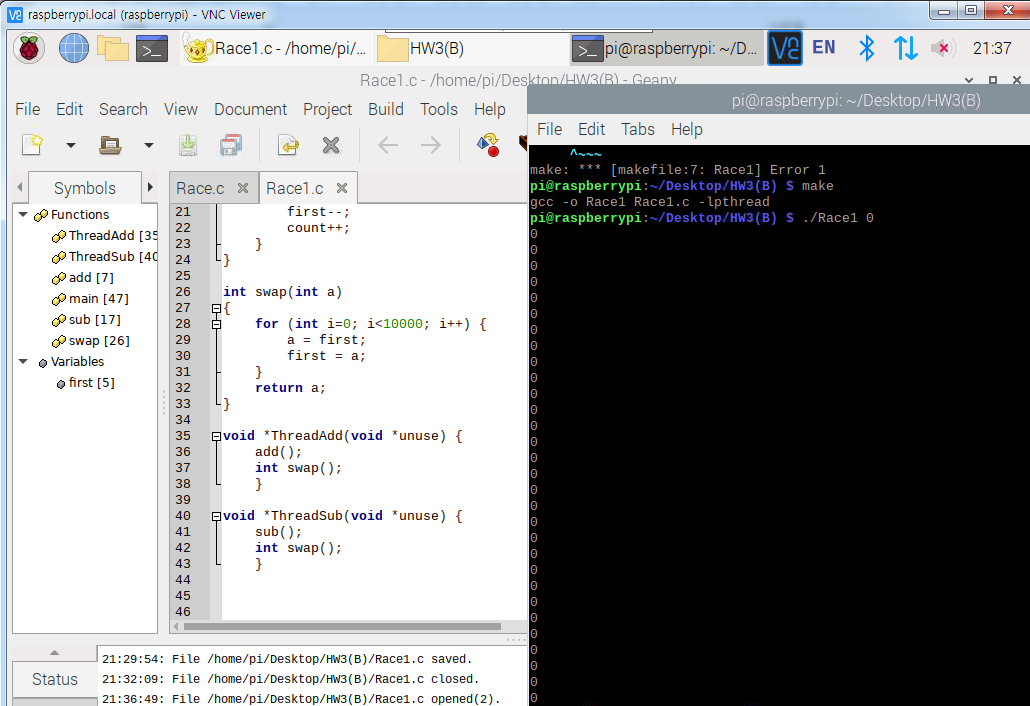


3.

3. 수업시간에 배운 mutex를 이용하니 race condition이 일어나지 않았다.



**-오류**



-처음 swap함수를 만들어서 전역 변수 first값을 a에 넣고 다시 first에 a값을 넣어주려고 했지만, swap함수 for문의 i값을 많이 올려도 최종 first값이 변하지 않았다. 왜 그런지 확인해보니 swap함수의 매개변수를 int a로 선언해주었는데, 이렇게 하게 되면 매개변수값이 고정이 되게 되면서 first값을 매 번 받아오지 않아서 race condition이 일어나지 않는 것 같다. 그래서

void swap()

{

int temp;

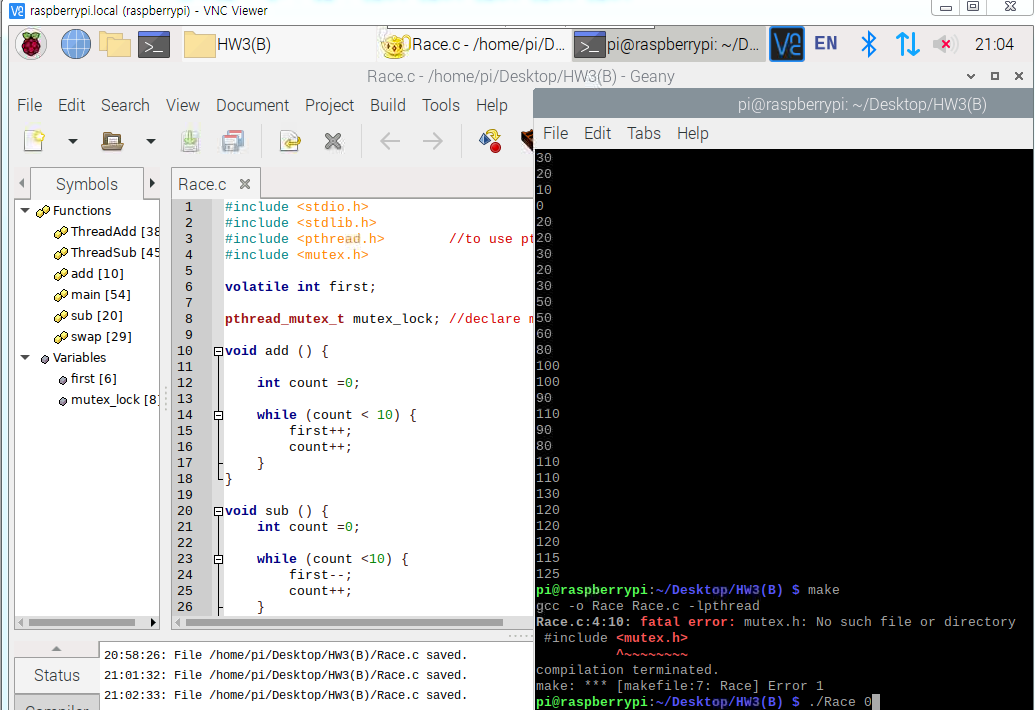
for (int i=0; i<10000; i++) {

temp = first;

first = temp;

}

로 전역 변수 first값을 매 번 받아올 수 있게 수정하니 race condition을 확인할 수 있었다.



-HW3(B) 과제 안내문에 보면 mutex.h헤더를 사용해야한다고 해서 선언을 해주니 에러가 떴다. 그래서 인터넷에서 mutex를 활용하기 위해 어떤 헤더 파일을 추가해야 되는지에 대해 검색을 해보니, mutex를 사용하기 위해 필요한 헤더에 mutex.h이 포함이 되어 있지 않아서 헤더를 지우고 실행해보니 해결되었다.

1. **고찰**

HW3에서는 thread를 생성하여 join을 하면 우리가 원하는 값이 아닌 엉뚱한 값이 나오게 되었는데 그것은 race condition때문에 일어나는 것을 알게 됐었다. Race condition의 쉬운 현실적인 예제로는 계좌에 30만원이 들어있는, 같은 카드를 가지고 있는 3명의 사람이 동시에 현금 인출 5만원을 같이 뽑게 되면 25만원이 계좌에 남는 것과 유사하다. 이전 과제는 thread를 총 10,000개를 만들어서 race condition이 일어나는 것을 봤다면 이번 HW3(B)에서는 thread수가 매우 적어도 충분히 race condition이 일어 날 수 있다는 걸 볼 수 있고 이 충돌을 해결하기 위해서는 thread를 동기화해줘야 한다는 걸 알게 되었다. Multi-thread같은 경우에는 여러 자원을 thread끼리 공유하는데, HW3(B)에서는 전역 변수 first값을 공유한다. Race condition을 해결하는 현실적인 예제로는 화장실이 공유 자원이고 여러 명이 사용한다고 생각하면, 누군가 화장실을 사용하고 있다면, 다른 사람이 화장실을 사용하지 못하게 사용하고 있는 사람이 문을 걸어 잠그면 된다. 이 기능을 해주는 것이 pthread의 mutex이다. Mutex\_lock를 사용하여 thread하나가 first에 접근하면 mutex\_lock를 실행하여 thread가 끝날 때 까지 다른 thread는 대기하고 끝나면 다른 thread가 공유 자원인 first에 접근할 수 있게 된다. 현재 나의 코드에선 add(), sub(), swap()부분에서 같이 first를 접근하려고 하니 mutex\_lock과 mutex\_unlock를 사용하여 TidAdd가 mutex\_lock을 먼저 가지면 TidSub는 TidAdd가 mutex\_unlock될때까지 기다린다. 그래서 마지막 first값을 30번 출력하였을 때 처음 입력해준 first값과 똑같이 나오는 것이다.

교수님 수업을 듣기 전에는 thread의 수가 적으면 연산 속도가 빨라서 충돌이 일어나지 않을 것 같았지만 그게 아니라는 것을 알게 되었고, race condition을 mutex를 통해 해결할 수 있다는 것을 알게 되었다.

1. **프로그램 소스 파일**

**Race1.c**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h> //to use pthread function

volatile int first;

void add () {

int count =0;

while (count < 10) {

first++;

count++;

}

}

void sub () {

int count =0;

while (count <10) {

first--;

count++;

}

}

void swap() //(Important HW3(B) Question1)to see race condition eventhough number of threads are not that big

{

int temp;

for (int i=0; i<10000; i++) {

temp = first;

first = temp;

}

}

void \*ThreadAdd(void \*unuse) {

add();

swap(); //add swap function to make race condition

}

void \*ThreadSub(void \*unuse) {

sub();

swap(); //add swap function to make race condition

}

int main (int argc, char\* argv[]) {

int i;

first = atoi(argv[1]);

pthread\_t TidAdd[10];

pthread\_t TidSub[10];

for (int j=0; j<30; j++) {

for (i=0; i<5; i++) { //by using for(), I made 10 threads.

pthread\_create(&TidAdd[i], NULL, ThreadAdd, NULL);

pthread\_create(&TidSub[i], NULL, ThreadSub, NULL);

}

for (i=0; i<5; i++) {

pthread\_join(TidAdd[i], NULL);

pthread\_join(TidSub[i], NULL);

}

printf("%d\n", first);

}

return 0;

}

**Race.c**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h> //to use pthread function

volatile int first;

pthread\_mutex\_t mutex\_lock; //declare mutex object

void add () {

int count =0;

while (count < 10) {

first++;

count++;

}

}

void sub () {

int count =0;

while (count <10) {

first--;

count++;

}

}

void swap() //(Important HW3(B) Question1)to see race condition eventhough number of threads are not that big

{

int temp;

for (int i=0; i<10000; i++) {

temp = first;

first = temp;

}

}

void \*ThreadAdd(void \*unuse) {

pthread\_mutex\_lock(&mutex\_lock); //critical section start

add();

swap(); //add swap function to make race condition

pthread\_mutex\_unlock(&mutex\_lock); //critical section end

}

void \*ThreadSub(void \*unuse) {

pthread\_mutex\_lock(&mutex\_lock); //critical section start

sub();

swap(); //add swap function to make race condition

pthread\_mutex\_unlock(&mutex\_lock); //critical section end

}

int main (int argc, char\* argv[]) {

int i;

first = atoi(argv[1]);

pthread\_t TidAdd[10];

pthread\_t TidSub[10];

for (int j=0; j<30; j++) {

for (i=0; i<5; i++) { //by using for(), I made 10 threads.

pthread\_create(&TidAdd[i], NULL, ThreadAdd, NULL);

pthread\_create(&TidSub[i], NULL, ThreadSub, NULL);

}

for (i=0; i<5; i++) {

pthread\_join(TidAdd[i], NULL);

pthread\_join(TidSub[i], NULL);

}

printf("%d\n", first);

}

return 0;

}

**Makefile**

all:Race1 Race

CC= gcc

LOL = -lpthread

Race1: Race1.c

$(CC) -o $@ Race1.c $(LOL)

Race: Race.c

$(CC) -o $@ Race.c $(LOL)

1. **자료 출처**

<https://jeong-pro.tistory.com/93>

<http://egloos.zum.com/rousalome/v/10003459>

<https://kuroikuma.tistory.com/26>