**Operating System HW3**

**학번:** 2016113832

**이름:** 윤동준

**담당교수:** 강순주

1. **개요**

Argument로 특정 정수를 받아서, 10,000개의 thread를 만들었고, 5000개의 thread에서는 add라는 함수를 실행하였고, 다른 5,000개의 thread에선 sub라는 함수를 실행하였다. 실행한 것을 30번 반복하여 마지막으로 나온 전역변수 first의 값을 출력해보았다.

1. **프로그램 구조 설명**

**2.1 함수에 대한 설명**

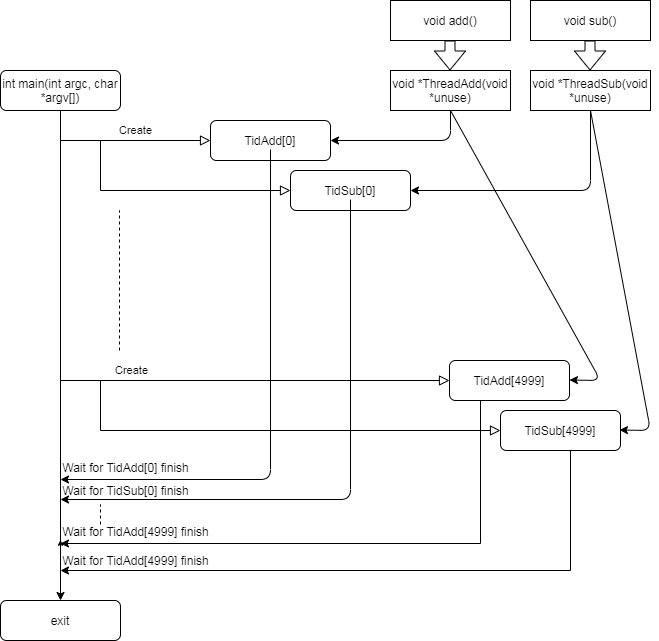
**-void add():** while문을 활용하여 (count<x) x값이 결정되면 전역변수 first가 얼마나 더해질지 결정된다.

**-void sub():** while문을 활용하여 (count<x) x값이 결정되면 전역변수 first가 얼마나 질지 결정된다.

**-void \*ThreadAdd(void \*unuse):** thread함수를 만들어서 add함수를 넣어줬다. Pthread\_create의 세 번째 인자의 반환값이 void \*타입이라 매개변수를 void\*로 선언하였다.

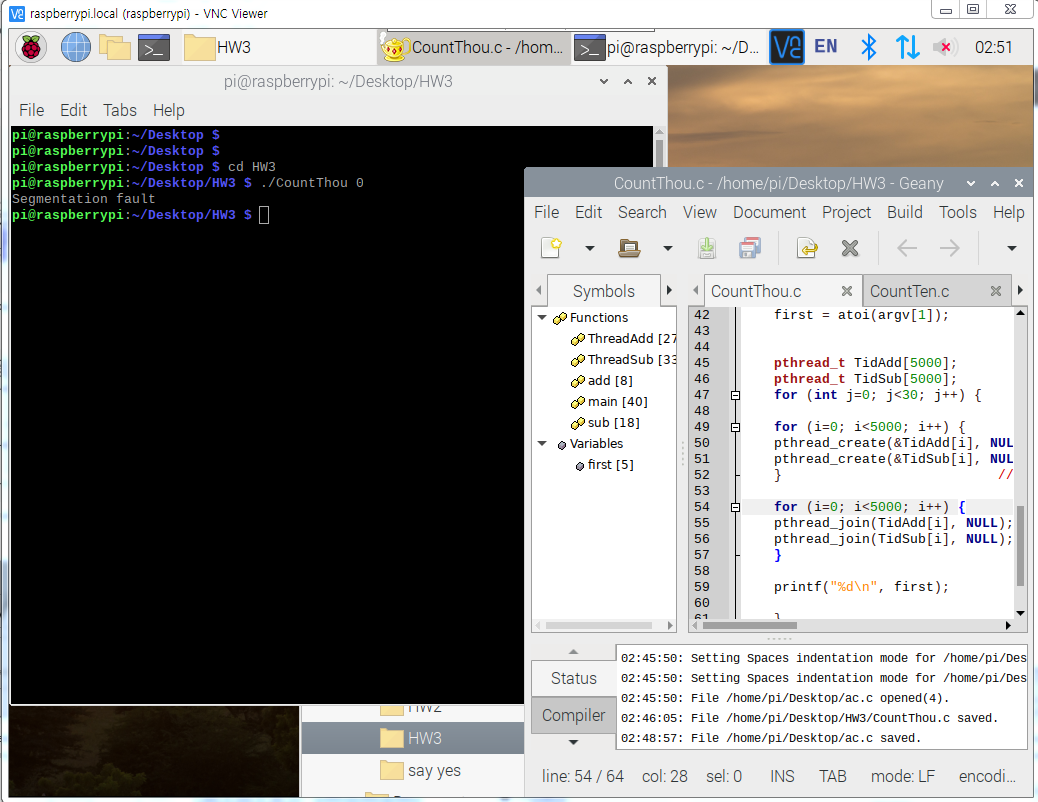
-**void \*ThreadSub(void \*unuse):** thread함수를 만들어서 sub함수를 넣어줬다. Pthread\_create의 세 번째 인자의 반환값이 void \*타입이라 매개변수를 void\*로 선언하였다.

**-int main (int argc, char\* argv[]):** atoi를 활용하여 입력받은 숫자를 integer형식으로 바꿔서 전역변수에 저장을 하고, pthread\_t TidAdd[10000], TidSub[10000]라는 자료형을 선언 해주었다. pthread\_t는 pthread의 자료형이고 그 뒤는 스레드의 이름이다. atoi를 사용하여 터미널에서 쓴 2번째 인자를 integer로 바꿔서 전역변수 first에 저장했다. 그리고 for문을 사용하여 pthread\_create(&TidAdd[i], NULL, ThreadAdd, NULL), pthread\_create(&TidSub[i], NULL, ThreadSub, NULL) thread 10,000개를 만들어 주었다. Pthread\_create의 첫 번째 인자에는 pthread 식별자로 thread가 성공적으로 생성되면 thread 식별값이 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 인자값이 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**

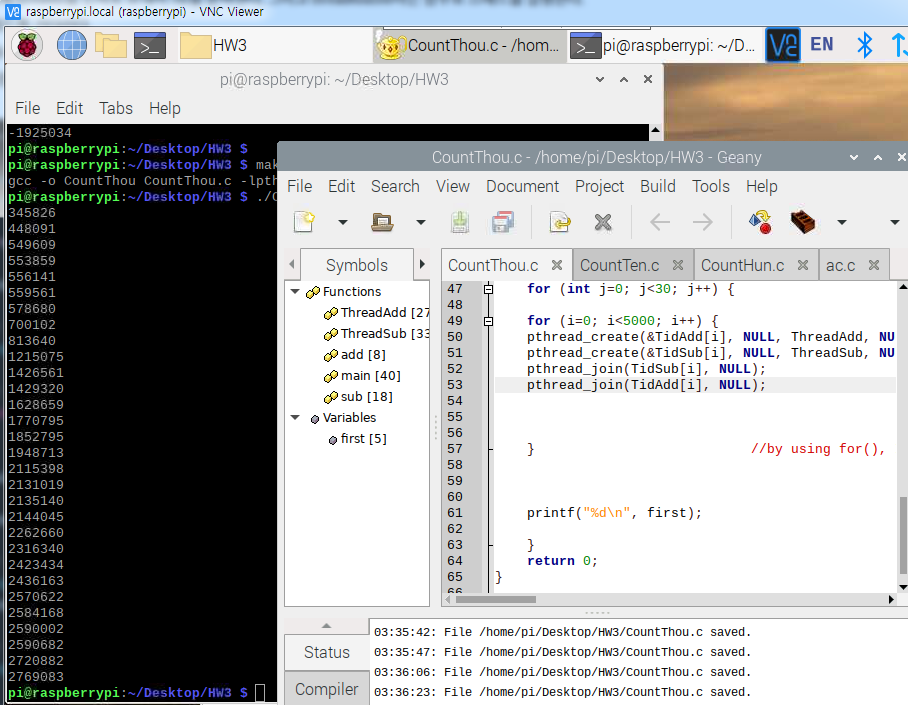
1. **실행결과**

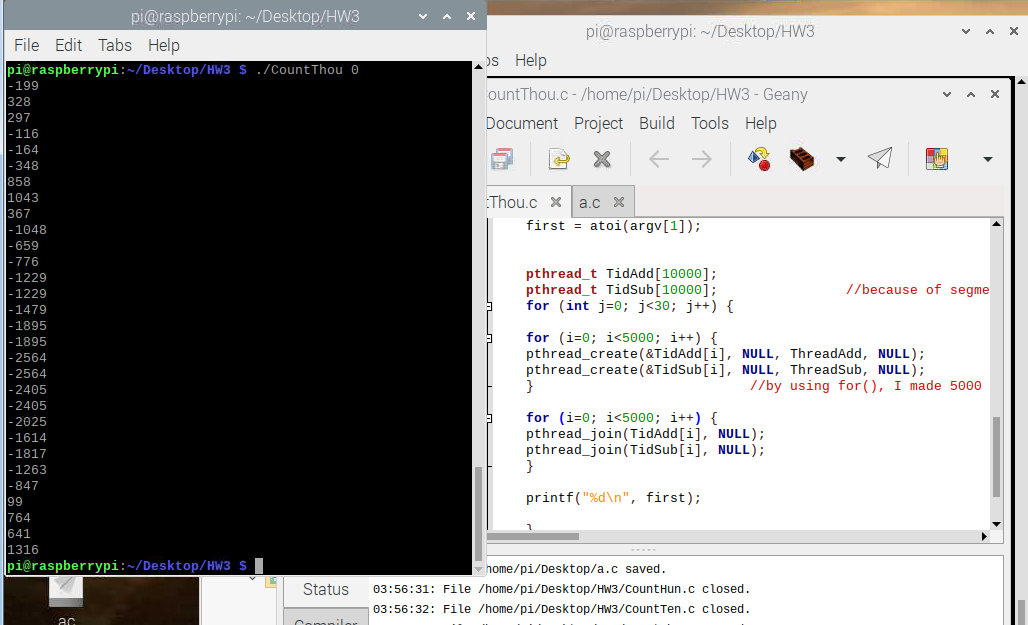
**-오류**



* 자료형의 배열 크기를 5000으로 잡았는데 segmentation fault가 나왔다.



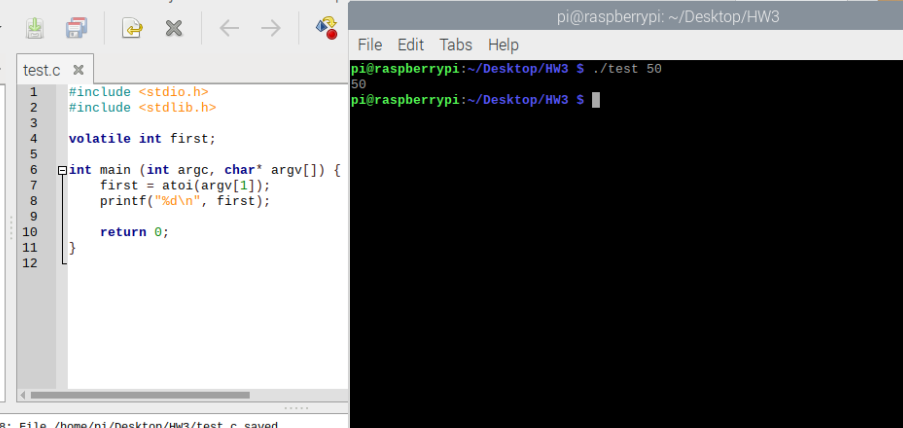
* 자료형의 배열의 크기를 10,000으로 늘려주니 프로그램이 실행이 되었다. 짐작하기로는 for문안에다가 join을 쓰지 않아서 5천개의 thread가 만들어지고 나서 실행 후 join을 해서 메모리 부족으로 인해 뜬 것 같다. 정확한 이유는 모르겠다.
* 
* Pthread\_join을 pthread\_create와 같이 for문에 넣게 되면 thread를 만들고 실행이 다 될때까지 기다렸다가 다음 thread가 실행되기 때문에 전역변수 first값이 천천히 출력이 된다.

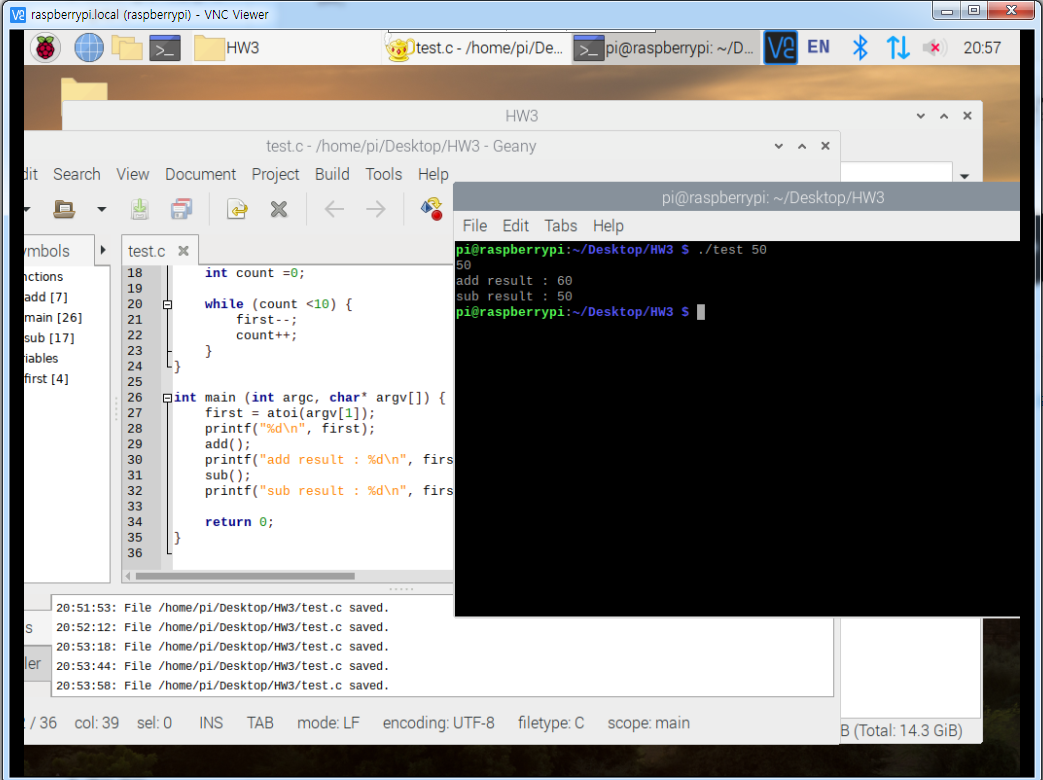


* Join을 따로 만들어 주었을 때는 전역변수 first를 출력하는 속도가 매우 빨라졌다.

**-실행결과**

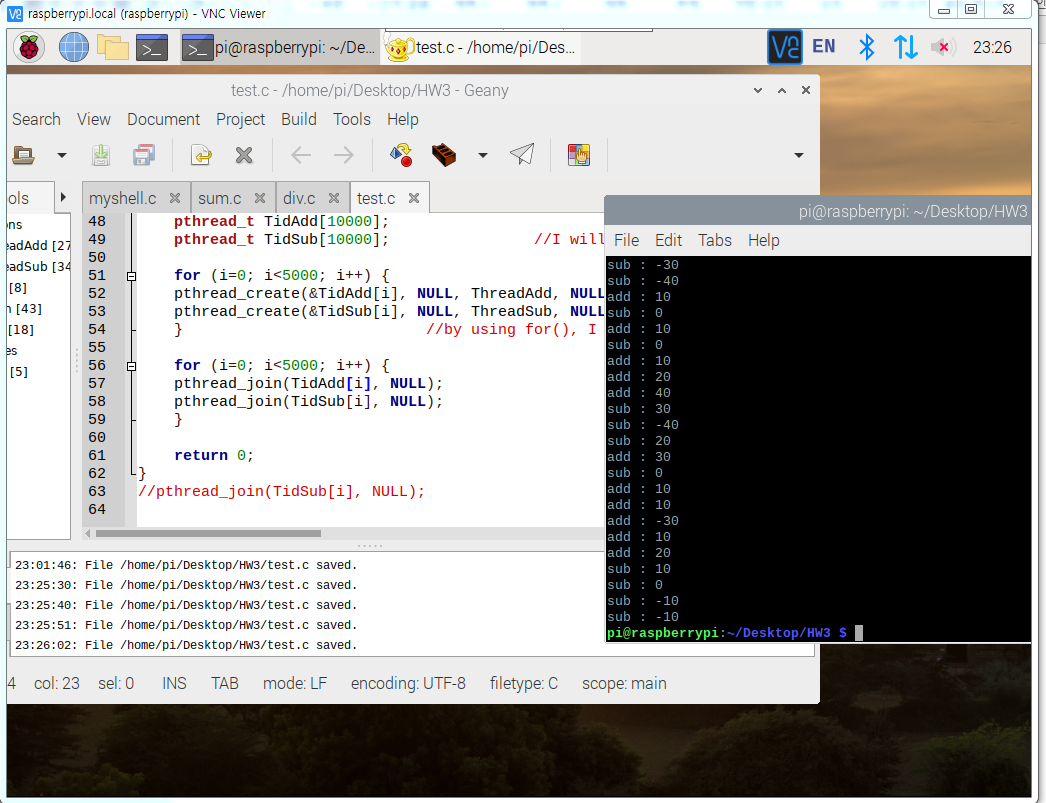
1. argument로 특정 정수를 받는 프로그램을 만들었다.



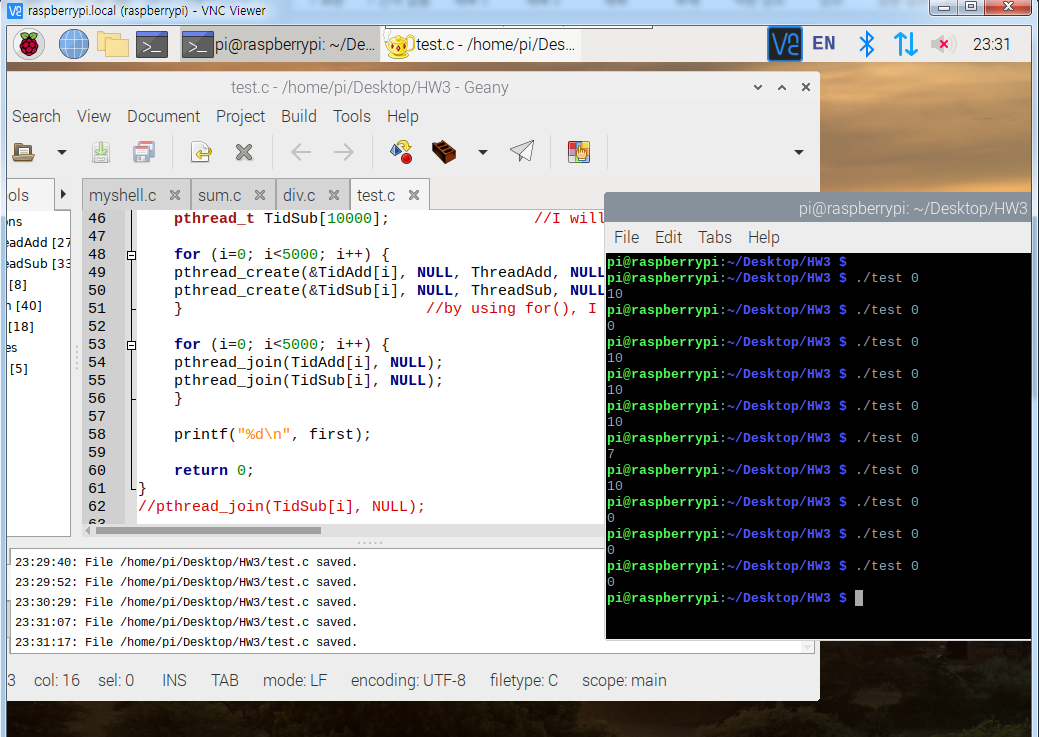
2. 위의 전역변수에 (1)10번씩 +1을 반복하는 ‘add()’ 함수, -1을 반복하는 ‘sub()’ 함수를 만들었다.

Sub에 50이 나오는 이유는 add된 상태에서 --가 되어서 60-10=50이 된 것이다.

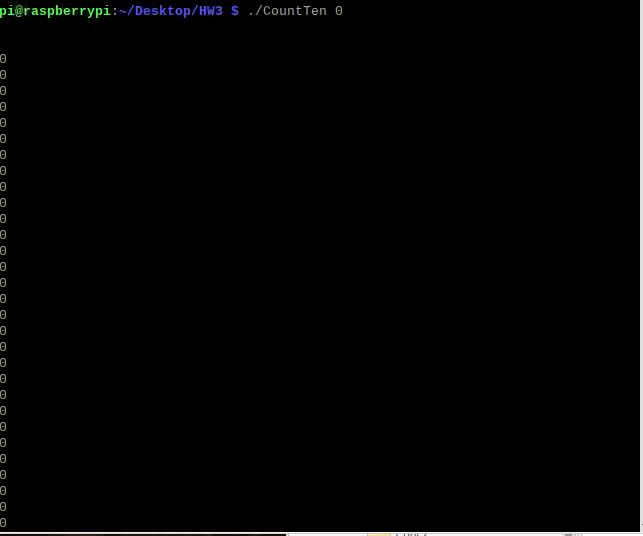
3. add,sub thread를 5000개를 만들었다.



Add 와 sub thread에 정확한 값이 들어가는지 확인하였지만 정확한 값이 들어가지 않았다.

4. 결과값 하나를 출력하였다.

그리고 결과값 하나를 출력하는 과정을 for를 사용하여, 30번 시행해 보았다. (파일이름을 test에서 countTen으로 변경하였다,)



1~4번 과정을 (2)100번식 실행하였다.

1~4번 과정을 (3)1000번씩 실행하였다.



1. **고찰**

main함수의 thread와 add, sub스레드를 만들면 다 같이 돌아가는데, 메모리 할당, 우선권을 주는거는 OS역할이다. 이 스레드들이 공유하는 것이 있는데 그것이 바로 volatile int first라고 선언해준 전역변수이다. 스레드를 사용하는 이유는 fork()와 같은 병렬처리의 문제점을 많이 해결해준다. Fork()는 새로운 프로세스를 생성하게 되면, 5가지의 구성요소(code, data, stack, fileI/O 그리고, single table) 모두 복사가 되서 비용이 많이 나오는 반면, thread는 새로운 프로세스를 만들지 않고, 같은 프로세스에서 특정 코드만 병렬로 실행할 수 있게 해준다는걸 알게 되었다. 첫번째 과정에서 volatile으로 변수를 선언하면 더 잘된다고 하셔서 volatile 변수를 코딩도장에 검색해서 확인해보니, “변수를 최적화에서 제외되어 항상 메모리에 접근하도록 만듬”이라고 설명되어 있던데, 내 생각으로는 전역변수로 volatile 변수를 설정하여서, 전역변수의 값이 add와 sub함수로 인해 값이 언제든지 바뀔 수 있어서 항상 메모리에 접근 할 수 있도록 volatile 변수를 사용한 것 같다. 그리고 처음 코드를 다 짜고 실행해보니segmentation fault가 생겨서 구글링을 해보니, pthread\_t TidAdd(Sub)[5000]에서 배열 방의 크기를 늘리라고 되어 있어서 6000으로 바꿔봤지만 에러가 떠서 7000으로 배열의 크기를 잡으니 실행이 되었고, phtread\_t TidAdd(Sub)[5000]={0}으로 초기화하니 실행이 되었다. 이거에 대해서 생각을 많이 해봤지만 왜 그런건지 잘 모르겠다.

10+1=11 10+1=11 11+1=12 10+1=11 11+1=12

TidAdd

read

11-1=10 12-1=11 11-1=10

TidSub

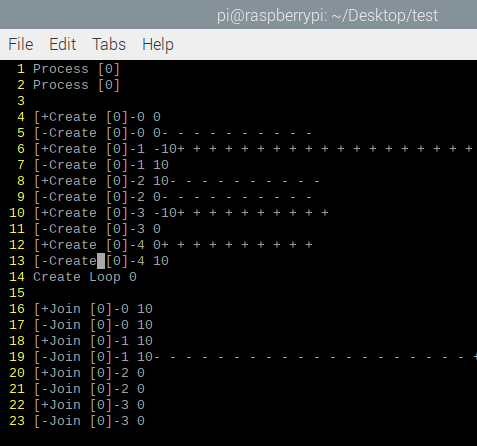
write

read

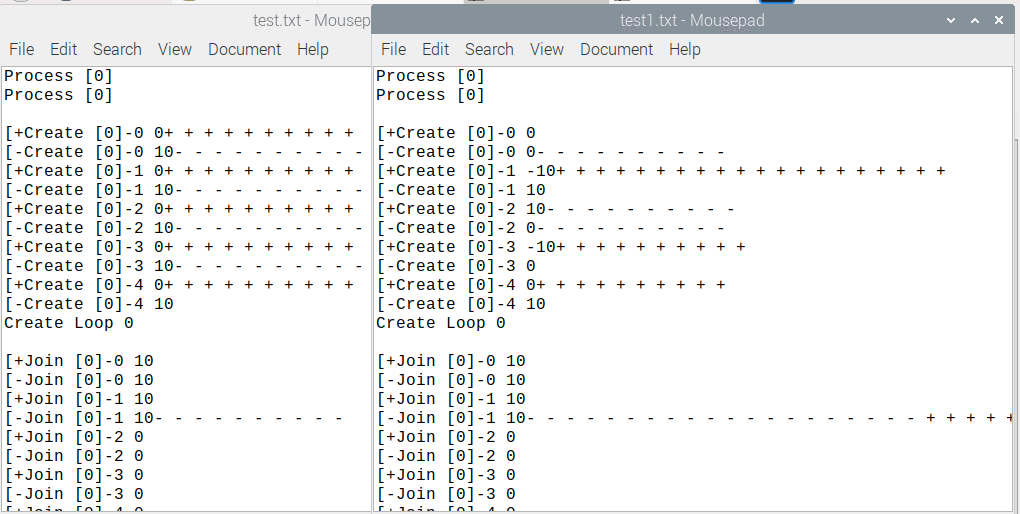
10 11 10 11 12 11 10 11 first=12

int first

Thread가 어떻게 동작하는지에 대해 그림을 그려보았다. 우리가 원하는 연산 순서는 TidAdd한번 TidSub한번씩해서 처음 first에 10을 넣으면 thread실행이 끝나고 받는 값이 10이길 원하는데, race condition으로 thread충돌이 일어나서 원하는 값이 안 나왔다. Race condition이 일어나는 이유는 Add thread와 Sub thread가 공유자원(first)을 접근하는 코드의 일부분을 동시에 실행하려고 해서 생기는 것이다. Race condition의 쉬운 현실적인 예제로는 계좌에 30만원이 들어있는데, 같은 카드를 가지고 있는 3명의 사람이 동시에 현금 인출 5만원을 같이 뽑게 되면 원래는 3명의 사람이 5만원을 뽑으면 계좌에 15만원이 남아야 하는데 25만원이 계좌에 남는 것과 유사하다.

Thread의 연산과정을 보기 위해 test.c라는 코드를 작성하였다. (코드는 프로그램 소스파일 맨 밑에 있다. )

Process가 실행이 되면 process [first]를 출력해서 process가 2번 출력이 된 것이고, create된 것을 출력하여 어떻게 실행되는지 확인하였다. 6번줄에서는 20번 +연산이 되었고, 7번줄은 실행이 되지 않았다. 맨 마지막 13줄에서 create가 join보다 더 빨리 끝나서 join도중에 sub()함수가 실행되면서 first의 값이 0이 되었다. Test코드를 여러 번 실행하여 ./test 0 3 > test.txt명령어를 사용하여 실행파일의 결과값을 test.txt파일을 현재 다이렉토리에 만들어서 저장하였다.



./test1 0 3를 여러 번 하니 처음에 연산이 되지 않고 바로 create로 넘어가는 경우가 생겼다. 이 경우를 race condition이라고 한다. 프로세스가 어떤 순서로 데이터에 접근하냐에 따라서 결과 값이 달라질 수 있다. 그래서 위의 실행파일 text같이 코드 작성자가 원하는 대로 안 흘러간다.

이 후 thread가 어떻게 동작하는지에 대해 더 세밀하게 파고 들기 위해 횟수도 늘리고 4일 동안 진짜 거짓말하지 않고 thread만 보았지만 소프트웨어적 사고가 부족하여 한계를 느끼고 더 이상 못하겠다. 이 후 교수님에게 정확하게 어떻게 동작하는지 여쭤봐야겠다.

**프로그램 소스 파일**

**CountTen.c(10번 더해주고, 10번빼주는 코드)**

#include <stdio.h>

#include <stdlib.h> //to use atoi function

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

volatile int first;

void add () { //to increase first 10 times by using while

int count =0;

while (count < 10) {

first++;

count++;

}

}

void sub () {

int count =0; //to decrease first 10 times by using while

while (count <10) {

first--;

count++;

}

}

void \*ThreadAdd(void \*unuse) { //Made Threadfunction that run add function

add();

}

void \*ThreadSub(void \*unuse) { //Made Threadfunction that run sub function

sub();

}

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

int i;

first = atoi(argv[1]); //char to integer

pthread\_t TidAdd[10000]; //because of segmentation fault, I made array size 10000

pthread\_t TidSub[10000];

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

for (i=0; i<5000; i++) { //by using for(), I made 5000 thread for add and sub

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

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

}

for (i=0; i<5000; i++) {

pthread\_join(TidAdd[i], NULL); //wait until the TidAdd[i] ends

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

}

printf("%d\n", first); //print first after run TidAdd and TidSub

}

return 0;

}

**CountHun.c (100번 더해주고 100번 빼주는 코드)**

#include <stdio.h>

#include <stdlib.h> //to use atoi function

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

volatile int first;

void add () { //to increase first 10 times by using while

int count =0;

while (count < 100) {

first++;

count++;

}

}

void sub () {

int count =0; //to decrease first 10 times by using while

while (count <100) {

first--;

count++;

}

}

void \*ThreadAdd(void \*unuse) { //Made Threadfunction that run add function

add();

}

void \*ThreadSub(void \*unuse) { //Made Threadfunction that run sub function

sub();

}

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

int i;

first = atoi(argv[1]); //char to integer

pthread\_t TidAdd[10000]; //because of segmentation fault, I made array size 10000

pthread\_t TidSub[10000];

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

for (i=0; i<5000; i++) { //by using for(), I made 5000 thread for add and sub

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

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

}

for (i=0; i<5000; i++) {

pthread\_join(TidAdd[i], NULL); //wait until the TidAdd[i] ends

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

}

printf("%d\n", first); //print first after run TidAdd and TidSub

}

return 0;

}

**CountThou.c (1000번더해주고 1000번빼는 코드)**

#include <stdio.h>

#include <stdlib.h> //to use atoi function

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

volatile int first;

void add () { //to increase first 10 times by using while

int count =0;

while (count < 1000) {

first++;

count++;

}

}

void sub () {

int count =0; //to decrease first 10 times by using while

while (count <1000) {

first--;

count++;

}

}

void \*ThreadAdd(void \*unuse) { //Made Threadfunction that run add function

add();

}

void \*ThreadSub(void \*unuse) { //Made Threadfunction that run sub function

sub();

}

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

int i;

first = atoi(argv[1]); //char to integer

pthread\_t TidAdd[10000]; //because of segmentation fault, I made array size 10000

pthread\_t TidSub[10000];

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

for (i=0; i<5000; i++) { //by using for(), I made 5000 thread for add and sub

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

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

}

for (i=0; i<5000; i++) {

pthread\_join(TidAdd[i], NULL); //wait until the TidAdd[i] ends

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

}

printf("%d\n", first); //print first after run TidAdd and TidSub

}

return 0;

}

**test.c (thread동작원리를 파악하기 위한 코드)**

#include <stdio.h>

#include <stdlib.h>

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

volatile int first;

void add () {

int count =0;

for (int j=0; j<10; j++) {

while (count < 1000) { //for delay

count++;

}

printf("+ "); //to see add works well

first++;

}

}

void sub () {

int count =0;

for (int j=0; j<10; j++) {

while (count < 1000) { //for delay

count++;

}

printf("- "); //to see sub works well

first--;

}

}

void PrintProcess() {

printf("Process [%d]\n", first);

}

void \*ThreadAdd(void \*unuse) {

add();

}

void \*ThreadSub(void \*unuse) {

sub();

}

void \*ThreadPrint(void \*unuse) {

PrintProcess(); //when the process starts, print process[first]

}

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

int i, j, End;

pthread\_t TidFirst, TidAdd[5000];

pthread\_t TidSub[5000];

if( argc <2 )

printf ("Usage: Prtest 0 100");

first = atoi(argv[1]);

End = atoi(argv[2]);

pthread\_create(&TidFirst, NULL, ThreadPrint, NULL); //made PrintThread

printf("Process [%d]\n", first);

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

for (i=0; i<End; i++) { //made Add Thread, Sub Thread

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

printf("\n[+Create [%d]-%d %d", j, i, first);

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

printf("\n[-Create [%d]-%d %d", j, i, first);

} //by using for(), I made 5000 thread for add and sub

printf("\nCreate Loop %d\n", j); //show the process of loop

for (i=0; i<End; i++){ //made Add Thread join, Sum Thread join

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

printf("\n[+Join [%d]-%d %d", j, i, first);

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

printf("\n[-Join [%d]-%d %d", j, i, first);

}

printf("\nJoin Loop %d\n", j);

}

pthread\_join(TidFirst, NULL);

return 0;

}

**6. 자료 출처**

<https://bitsoul.tistory.com/156?category=683199>

<https://dojang.io/mod/page/view.php?id=278>

[https://notnegative.tistory.com/entry/7%EC%9B%94-8%EC%9D%BC-pthreadh%EC%97%90-%ED%8F%AC%ED%95%A8%EB%90%98%EC%96%B4-%EC%9E%88%EB%8A%94-%ED%95%A8%EC%88%98-%EB%AA%87%EA%B0%9C-%EA%B7%B8%EB%A6%AC%EA%B3%A0-makefile](https://notnegative.tistory.com/entry/7ì-8ì¼-pthreadhì-í¬í¨ëì´-ìë-í¨ì-ëªê°-ê·¸ë¦¬ê³ -makefile)