1. pthread의 전역변수 concurrency 문제

1. 개요

리눅스 환경에서 pthread의 동시 접근 문제로 인한 전역변수 깨짐 현상을 구현하고 mutex로

문제를 해결해본다.

2. 프로그램 구조 설명

2.1 함수에 대한 설명

-create.c ( concurrency 문제를 발생시키는 코드입니다.)

void\* func1(void\* args)

=>thread로 돌아가는 함수입니다. 전역변수 num에 1증가 명령을 일정 횟수만큼 반복합니다.

-mutex.c ( mutex 사용 )

=> mutex를 사용하는 것 이외에 create.c와 다른 것이 없습니다.

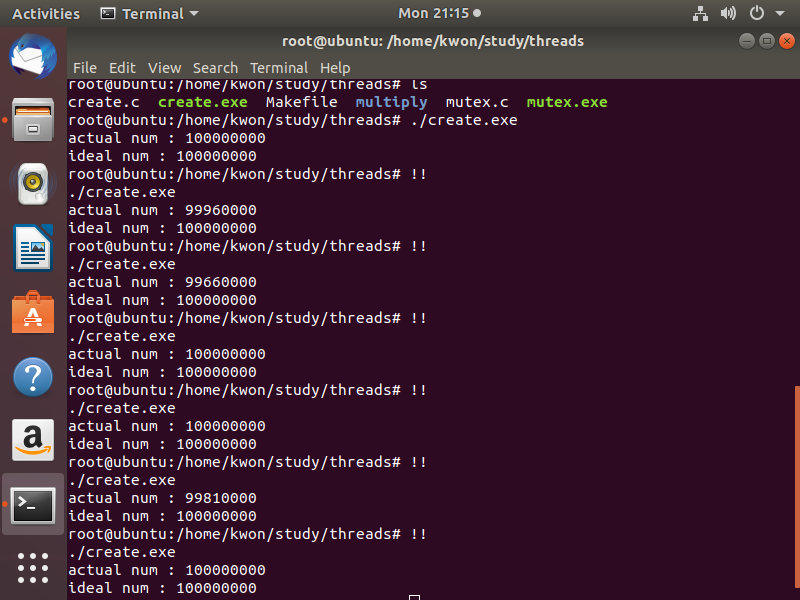
2.다이어그램

텍스트, 화이트보드이(가) 표시된 사진

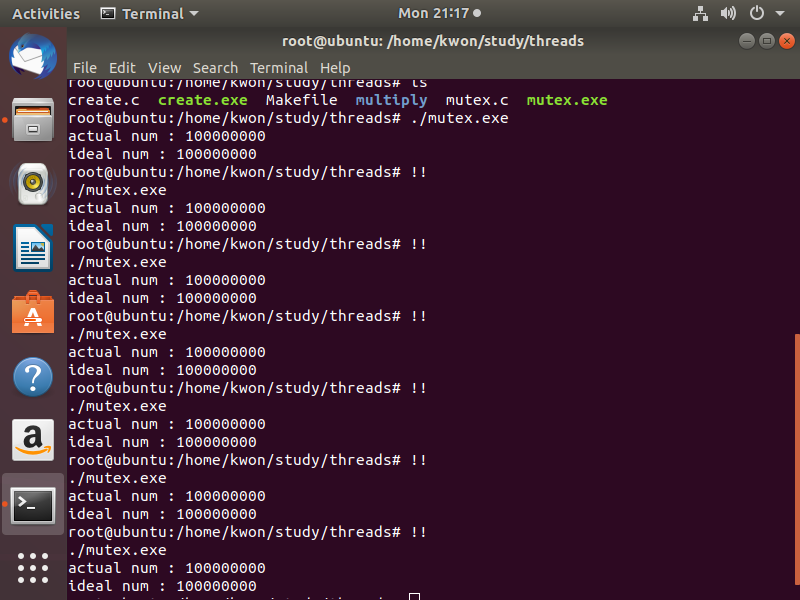
자동 생성된 설명

3. 실행 결과

- create.c 결과



- mutex.c 결과



**(실행결과는 자신의 컴퓨터환경에서 실행한 결과 캡쳐(그림파일)해서 문서에 첨부하여 제출합니다. 실행결과가 성공한 경우뿐만 아니라 실패한 경우도 포함되어야 함.**

**만약 과제로 올려서 실행한 결과와 실행 결과로 캡쳐한 결과가 다를 경우 감점합니다.)**

4. 고찰 (제일 중요함 - **점수배점이 가장 높음!** 자세히 쓸 것)

이번 코딩도 꽤나 흥미로운 코딩이었습니다. 어셈블리어 수준에서 일어나는 전역변수 동시접근

문제를 고의적으로 발생시키려면 엄청나게 많은 명령어가 단시간에 일어나야 될 것이라 예상은

했었습니다. 하지만 각자 전역변수에 1씩 더하는 명령을 만 번 수행하는 쓰레드를 만 개를 수행

시켜야 겨우 일어나는 모습을 보고 cpu의 성능은 대단하구나 새삼 느꼈습니다.

또 이번에 사소한 실수지만 critical한 문제가 있었습니다. 제가 쓰레드를 2개를 만들었다가

만 개로 늘렸는데 pthread\_join는 늘려주지 않은 것이 문제였습니다. 그 문제 때문에 mutex를

써도 예상값과 계속 다른 현상이 나타났습니다. Mutex를 처음 써보는 것이라 처음에는 mutex를

제가 잘못 쓴 줄로만 알고 한참을 고민하다 join의 문제인 것을 알았습니다. 앞으로는 좀 더

사소한 부분에 신경을 쓰면서 코딩을 하도록 하겠습니다.

5. 프로그램 소스 파일 (주석을 꼭 달 것)

-create.c

#include <stdio.h>

#include <stdbool.h>

#include <pthread.h>

#include <unistd.h>

#define NUM 0 // original number

#define SIZE1 10000 // the number of threads

#define SIZE2 10000 // the number which is subtracted per thread

volatile int num = NUM;

void\* func1(void\* args){ // num을 1을 증가시키는 명령을 만 번 수행

for(int i=0; i<SIZE2; i++){

num++;

}

pthread\_exit(0);

}

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

int i;

pthread\_t id[SIZE1];

for(i = 0; i<SIZE1; i++){

if(pthread\_create(&id[i],NULL,func1,&i)){

printf("error\n");

return 0;

}

}

for(i = 0; i<SIZE1; i++){ //쓰레드 각각 join을 걸어준다.

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

}

//나와야 하는 값과 실제 값을 출력

printf("actual num : %d\n",num);

printf("ideal num : %d\n",NUM+SIZE1\*SIZE2);

return 0;

}

-mutex.c (create.c와 거의 같으나 mutex부분이 추가)

#include <stdio.h>

#include <stdbool.h>

#include <pthread.h>

#include <unistd.h>

#define NUM 0 // original number

#define SIZE1 10000 // the number of threads

#define SIZE2 10000 // the number which is subtracted per thread

volatile int num = NUM;

pthread\_mutex\_t mutex;

void\* func1(void\* args){

for(int i=0; i<SIZE2; i++){

pthread\_mutex\_lock(&mutex); //mutex lock을 검

num++; // critical section

pthread\_mutex\_unlock(&mutex); //mutex unlock

}

pthread\_exit(0);

}

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

int i;

pthread\_mutex\_init(&mutex,NULL); //mutex 초기화

pthread\_t id[SIZE1];

for(i = 0; i<SIZE1; i++){

if(pthread\_create(&id[i],NULL,func1,&i)){

printf("error\n");

return 0;

}

}

for(i = 0; i<SIZE1; i++){

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

}

printf("actual num : %d\n",num);

printf("ideal num : %d\n",NUM+SIZE1\*SIZE2);

pthread\_mutex\_destroy(&mutex); //mutex 제거

return 0;

}

6. 자료 출처 (어떤 책을 참고했으며 어떤 웹사이트를 참고했는지 출처를 적습니다)

대부분 교수님의 ppt를 참고했습니다.

밑에서 계속됩니다.

2. 행렬 곱셈을 단일 쓰레드와 멀티쓰레드로 수행하고

수행시간을 비교해보자

1. 개요

리눅스 환경에서 행렬 곱셈을 단일 쓰레드와 멀티 쓰레드로 수행하고 수행시간을

비교해보고 수행시간이 차이나는 원인에 대해 고찰해보자

2. 프로그램 구조 설명

2.1 함수에 대한 설명

-multi.c ( 단일 쓰레드 행렬 곱셈 코드입니다. )

void multiply()

=>전역변수로 설정된 행렬 A와 B를 곱하여 행렬 ans에 결과를 저장합니다.

-mutex2.c ( 멀티 쓰레드 행렬 곱셈 코드입니다. )

void multiply()

=>행렬 A와 B의 곱셈을 6개의 쓰레드에 분담하여 수행시키고 연산 결과를 출력합니다.

void thread1()~void thread6()

=>행렬 A와 B의 곱셈을 분담하여 수행하는 함수들입니다.

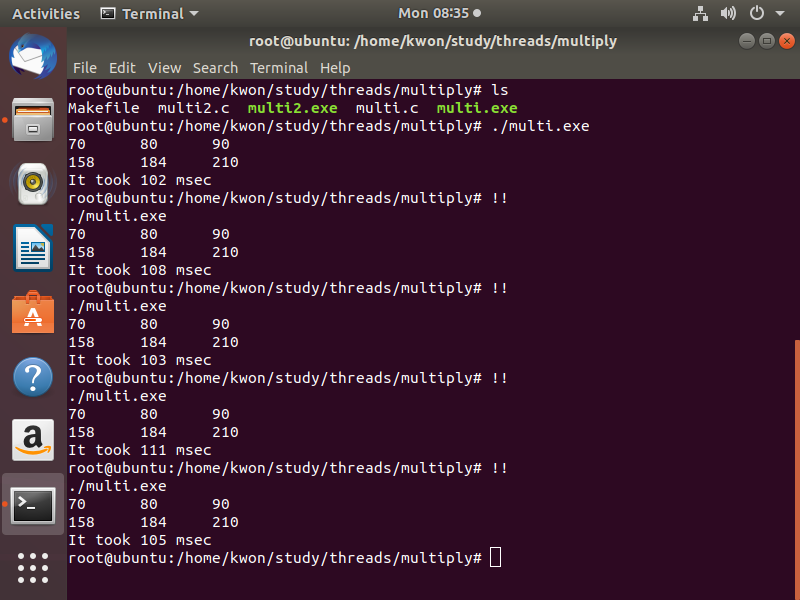
2.다이어그램

텍스트, 화이트보드, 실외이(가) 표시된 사진

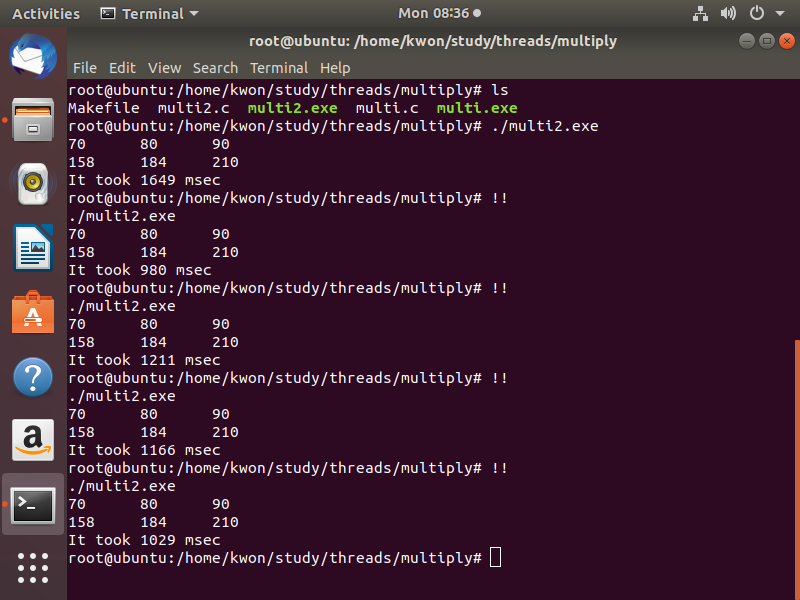
자동 생성된 설명

3. 실행 결과

- multi.c 결과



- multi2.c 결과



**(실행결과는 자신의 컴퓨터환경에서 실행한 결과 캡쳐(그림파일)해서 문서에 첨부하여 제출합니다. 실행결과가 성공한 경우뿐만 아니라 실패한 경우도 포함되어야 함.**

**만약 과제로 올려서 실행한 결과와 실행 결과로 캡쳐한 결과가 다를 경우 감점합니다.)**

4. 고찰 (제일 중요함 - **점수배점이 가장 높음!** 자세히 쓸 것)

‘단일 쓰레드와 멀티 쓰레드 중 뭐가 더 빠를까’에 대한 문제입니다. 이번 과제는 코딩에는

큰 어려움이 없었습니다. 단지 멀티 쓰레드로 구상을 어떻게 해야 할지를 조금 고민했습니다.

결과를 보면 멀티 쓰레드로 수행한 프로그램이 단일 쓰레드로 수행한 프로그램보다 평균적으로

9~10배정도 느린 결과가 나왔습니다. 멀티 쓰레드가 더 느린 이유는 contact switching 시간 때 문으로 생각됩니다. 프로그램의 성격에 따라 단일 쓰레드가 유용할 때도 있고 멀티 쓰레드가 유 용한 때도 있다는 것을 알게 되었습니다.

5. 프로그램 소스 파일 (주석을 꼭 달 것)

-multi.c ( 단일 쓰레드 )

#include <stdio.h>

#include <sys/time.h>

#define MILLION 1000000

int A[2][4] = {{1,2,3,4},{5,6,7,8}};

int B[4][3] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

int ans[2][3]; // 계산 결과를 저장하기 위한 행렬

void multiply(){ // 행렬 곱셈을 수행하는 함수

int i,j,k;

int sum;

for(i=0; i<2; i++){

for(j=0; j<3; j++){

sum = 0;

for(k=0; k<4; k++){

sum += A[i][k]\*B[k][j];

}

ans[i][j] = sum;

printf("%d\t",ans[i][j]);

}

printf("\n");

}

}

void main(){ // 수행시간을 측정하고 출력합니다.

struct timeval tpstart, tpend;

long timediff;

int i;

gettimeofday(&tpstart, NULL);

multiply();

gettimeofday(&tpend, NULL);

timediff = MILLION\*(tpend.tv\_sec-tpstart.tv\_sec) + tpend.tv\_usec - tpstart.tv\_usec;

printf("It took %ld msec \n", timediff);

}

-multi2.c (멀티 쓰레드)

#include <stdio.h>

#include <sys/time.h>

#include <pthread.h>

#define MILLION 1000000

int A[2][4] = {{1,2,3,4},{5,6,7,8}};

int B[4][3] = {{1,2,3},{4,5,6},{7,8,9},{10,11,12}};

int ans[2][3];

// thread1 ~ thread6은 곱셈 결과 행렬인 2X3 행렬의 각 요소의 값 계산을 담당합니다.

void thread1(){

int k;

int sum = 0;

int i = 0;

int j = 0;

for(k=0;k<4;k++){

sum += A[i][k]\*B[k][j];

}

ans[i][j] = sum;

return;

}

void thread2(){

int k;

int sum = 0;

int i = 0;

int j = 1;

for(k=0;k<4;k++){

sum += A[i][k]\*B[k][j];

}

ans[i][j] = sum;

return;

}

void thread3(){

int k;

int sum = 0;

int i = 0;

int j = 2;

for(k=0;k<4;k++){

sum += A[i][k]\*B[k][j];

}

ans[i][j] = sum;

return;

}

void thread4(){

int k;

int sum = 0;

int i = 1;

int j = 0;

for(k=0;k<4;k++){

sum += A[i][k]\*B[k][j];

}

ans[i][j] = sum;

return;

}

void thread5(){

int k;

int sum = 0;

int i = 1;

int j = 1;

for(k=0;k<4;k++){

sum += A[i][k]\*B[k][j];

}

ans[i][j] = sum;

return;

}

void thread6(){

int k;

int sum = 0;

int i = 1;

int j = 2;

for(k=0;k<4;k++){

sum += A[i][k]\*B[k][j];

}

ans[i][j] = sum;

return;

}

void multiply(){ // 쓰레드 6개를 동시에 돌려서 행렬 곱계산을 마치고 결과값을 출력합니다.

pthread\_t id[6];

void\* ft[6];

ft[0] = thread1;

ft[1] = thread2;

ft[2] = thread3;

ft[3] = thread4;

ft[4] = thread5;

ft[5] = thread6;

int i,j,k;

k = 0;

for(i = 0; i<2; i++){

for(j = 0; j<3; j++){

pthread\_create(&id[k],NULL,ft[k],NULL);

k++;

}

}

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

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

}

for(i=0; i<2; i++){

for(j = 0; j<3; j++){

printf("%d\t",ans[i][j]);

}

printf("\n");

}

};

void main(){

struct timeval tpstart, tpend;

long timediff;

int i;

gettimeofday(&tpstart, NULL);

multiply();

gettimeofday(&tpend, NULL);

timediff = MILLION\*(tpend.tv\_sec-tpstart.tv\_sec) + tpend.tv\_usec - tpstart.tv\_usec;

printf("It took %ld msec \n", timediff);

}

6. 자료 출처 (어떤 책을 참고했으며 어떤 웹사이트를 참고했는지 출처를 적습니다)

대부분 교수님의 ppt를 참고했습니다.