쓰레드 프로그램 시간 측정

과목명	시스템프로그래밍
담당교수	최종무 교수님
학과	소프트웨어학과
학번	32191105
이름	김지민
제출일자	20.12.09

[계기]

처음에 무슨 프로그램을 할지 고민하다가 map-reduce와 암달의 법칙에 대한 퀴즈를 했던 것이 떠오르면서 combine7로 병렬수행을 해보는 게 어떨까 하는 생각이 들었다. 배운 것 중에 병렬수행하면 떠오르는 게 fork()와 그와 비슷한 쓰레드, 그리고 백그라운드 실행이었다. 그 중 쓰레드가 메모리들을 공유하니까 병렬수행에 적합하다고 생각해서 멀티쓰레드로 combine7을 만들어 성능을 측정해보기로 했다. 처음에는 쓰레드 3개만 만들어서 할 생각이었다. 그러다가 쓰레드의 개수가 다르면 어떤지 궁금해서 2개를 만들어서 해봤는데 3개와 별 차이가 없었다. 초기 프로그램은 이렇다.

```
sys32191105@embedded: ~/homework
/*gettimeofday() program by kim jm DKU software*/
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>

void *combine(void *data)
{
        int i;
        int x0=0;
        int x1=0;
        int x2=0;
        int sum;
        int v[1000000];
        char* thread_name=(char*)data;
        for(i=0;i<1000000;i++)
        {
                v[i]=i;
        }
        if(thread_name=="thread_m")
        {
                for(i=0;i<300000;i++)
                {
                        x0=x0+v[i];
                }
        }
        if(thread_name=="thread_1")
        {
                for(i=300000;i<600000;i++)
                {
                        x1=x1+v[i];
                }
        }
        if(thread_name=="thread_2")
        {
                for(i=600000;i<1000000;i++)
                {
                        x2=x2+v[i];
                }
        }
        sum=x0+x1+x2;
}

int main()
{
        struct timeval stime,etime,gap;
        pthread_t p_thread[2];
        char p1[]="thread_1";
        char p2[]="thread_2";

        char pM[]="thread_m";

        pthread_create(&p_thread[0],NULL,combine,(void *)p1);
        pthread_create(&p_thread[1],NULL,combine,(void *)p2);
        gettimeofday(&stime,NULL);
        combine((void *)pM);
        gettimeofday(&etime,NULL);
        gap.tv_sec = etime.tv_sec - stime.tv_sec;
        gap.tv_usec = etime.tv_usec - stime.tv_usec;
        if (gap.tv_usec < 0) {
                gap.tv_sec = gap.tv_sec-1;
                gap.tv_usec = gap.tv_usec + 1000000;
        }
        printf("%ldsec : %ldusec\n\n", gap.tv_sec, gap.tv_usec);
}
```

[과정]

쓰레드를 생성할 때 두 번째 인자에 들어가는 attr의 속성을 detach로 바꿔보았는데도 똑같았다. 쓰레드 개수를 4, 5개로도 해봤는데 오히려 시간이 더 오래 걸려서 이상함을 느끼고 sum을 출력해봤더니 0이 나왔다. 인터넷에 검색해봤더니 쓰레드는 지역변수는 공유하지 않고 전역변수만 공유하는 특징이 있었다. 그래서 지역변수를 전역변수로 바꿔주었다. 이후 또 sum이 0이 나와서 이번엔 detach를 빼고 join을 해주었다. join을 해주지 않으면 쓰레드가 종료되기도 전에 메인함수가 종료돼서 오류가 날 수 있다는 것을 알았다. 그런데도 sum이 0이 나왔는데 오기가 생겨서 이 프로그램을 꼭 완성시키고 싶었다. 아무리 봐도 combine함수에서 조건문으로 쓰레드 이름을 비교하는 부분 빼고는 모두 맞는 코드 같았다. 그래서 비교하는 연산 '==' 대신에 strcmp 함수를 사용했는데 sum이 제대로 출력됐다. 왜 '=='는 안되는지 조금 의문이 들었다. 이후 다시 쓰레드를 2개부터 5개까지 만들어봤다. 최종 코드는 아래와 같다. 쓰레드 2개부터 5개는 모두 같은 방식의 코드라 쓰레드 2개인 코드만 캡쳐했다.

Single thread

sys32191105@embedded: ~/homework

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>

void combine()
{
        int i;
        int v[1000000];
        int x=0;
        int sum;
        for(i=0;i<1000000;i++)
        {
                v[i]=i;
        }
        for(i=0;i<1000000;i++)
        {
                x=x+v[i];
        }
        sum=x;
        printf("sum:%d\n",sum);
}

int main()
{
        struct timeval stime,etime,gap;
        gettimeofday(&stime,NULL);
        combine();
        gettimeofday(&etime,NULL);
        gap.tv_sec = etime.tv_sec - stime.tv_sec;
        gap.tv_usec = etime.tv_usec - stime.tv_usec;
        if (gap.tv_usec < 0) {
                gap.tv_sec = gap.tv_sec-1;
                gap.tv_usec = gap.tv_usec + 1000000;
        }
        printf("%ldsec : %ldusec\n\n", gap.tv_sec, gap.tv_usec);
}
~
~
```

1,1 All

single thread의 수행결과

```
sys32191105@embedded:~/homework$ ./singleThread
sum:1783293664
0sec : 8466usec

sys32191105@embedded:~/homework$
```

Thread 2개

sys32191105@embedded: ~/homework

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <pthread.h>

int x0=0;
int x1=0;
int sum;

void *combine(void *data)
{
        int i;
        int v[1000000];
        char* thread_name=(char*)data;
        for(i=0;i<1000000;i++)
        {
                v[i]=i;
        }
        if(strcmp(thread_name,"thread_m")==0)
        {
                for(i=0;i<500000;i++)
                {
                        x0=x0+v[i];
                }
        }
        if(strcmp(thread_name,"thread_1")==0)
        {
                for(i=500000;i<1000000;i++)
                {
                        x1=x1+v[i];
                }
        }
        sum=x0+x1;
}

int main()
{
        struct timeval stime,etime,gap;
        pthread_t p_thread[2];
        char p1[]="thread_1";
        char pM[]="thread_m";

        pthread_create(&p_thread[0],NULL,combine,(void *)p1);
        gettimeofday(&stime,NULL);
        combine((void *)pM);
        gettimeofday(&etime,NULL);
        gap.tv_sec = etime.tv_sec - stime.tv_sec;
        gap.tv_usec = etime.tv_usec - stime.tv_usec;
        if (gap.tv_usec < 0) {
                gap.tv_sec = gap.tv_sec-1;
                gap.tv_usec = gap.tv_usec + 1000000;
        }
        pthread_join(p_thread[0],(void *)NULL);

        printf("sum:%d\n",sum);
        printf("%ldsec : %ldusec\n\n", gap.tv_sec, gap.tv_usec);
}
```

17,1-8 Top

57,1 Bot

Thread 2개 수행결과

```
sys32191105@embedded:~/homework$ ./2_multiThread
sum:1783293664
0sec : 7571usec

sys32191105@embedded:~/homework$
```

Thread 3개 수행결과

```
sys32191105@embedded:~/homework$ ./3_multiThread
sum:1783293664
0sec : 6695usec

sys32191105@embedded:~/homework$
```

Thread 4개 수행결과

```
sys32191105@embedded:~/homework$ ./4_multiThread
sum:1783293664
0sec : 12781usec

sys32191105@embedded:~/homework$
```

Thread 5개 수행결과

```
sys32191105@embedded:~/homework$ ./5_multiThread
sum:1783293664
0sec : 17885usec

sys32191105@embedded:~/homework$
```

[결과 및 느낀점]

성능측정결과 thread1개(8466usec)보다는 2개가, 2개(7571usec)보다는 3개(6695 usec)가 빨랐다. 하지만 4개(12781 usec)부터는 프로그램 수행시간이 더 길어졌는데, 그 원인이 뭔지 생각해봤다. 내가 생각한 첫 번째 원인은 시분할 시스템이다. 프로그램을 번갈아가면서 수행하는 시분할 시스템의 성질 때문에 수행시간이 오히려 더 길어지지 않았을까 라는 생각을 했다. 두 번째로는 cpu수에 비해 쓰레드가 너무 많아서 동기화 비용이 많이 들어서일 수도 있겠다고 생각했다. 어쨌든 이번 과제를 통해 좀 더 최적화할 수 있는 방법에 대해 알았고, 쓰레드가 너무 많이 있어도 성능에 좋은 영향을 주지 못한다는 것을 깨달았다. 그리고 프로그램을 만들면서 최적화 하는 방법들에 대해 더 자세하게 공부하는 계기가 되었다. 제일 많이 알게 된 것은 쓰레드에 관련된 것들이었다. 수업때 fork()와 thread를 비교하면서 봤을 때 그냥 아 이런 차이가 있구나 하고 넘어갔었는데 직접 만들어보니까 더 와 닿는 기분이었다. 특히 전역변수는 공유하지만 지역변수는 공유하지 않는다는 것, 코드에는 사용하지 않았지만 조건문이 없을 때는 lock, unlock을 해줘야 한다는 것, thread를 만들 때 어떤 인자들을 사용해야하는지에 대해 정확히 알게 되었다. 또 정말로 쓰레드의 개수에 따라 프로그램 속도에 차이가 있다는 것이 신기했다. 처음 목표는 combine7을 만드는 거였는데, thread를 통해 비교하려하다 보니까 함수가 많이 변형되긴 했지만 그래도 뿌듯한 과제이다.