fork speed up

과목명	시스템프로그래밍
교수명	최종무 교수님
학 과	소프트웨어학과
팀 명	상민혁
팀 원	박민혁/이상민
학 번	32151671/32153180
제출일	2018.12.12

DKU 단국대학교 DANKOOK UNIVERSITY

목 차

Ⅰ. 프로젝트 개요

1. 목표 및 필요성

프로젝트에 들어가기 전, 수업 시간에 Loop splitting 배운 적이 있다. 따라서 Loop splitting의 개념을 토대로 아이디어가 시작되었다. 이것은 컴파일러 최적화 기술 중 하나로, 종속성을 제거하기 위한 것이다. Wikipedia에 있는 정의를 참조하면 다음과 같다.

> Loop splitting is a compiler optimization technique. It attempts to simplify a loop or eliminate dependencies by breaking it into multiple loops which have the same bodies but iterate over different contiguous portions of the index range

앞서 배운 fork()를 복습하던 도중 fork()를 이용해 flow control을 2개로 나누어 계산하면 더 빨라질 것 같다는 생각을 했다. 그래서 「fork()를 이용한 수행 속도 향상」을 프로젝트 주제이자 목표로 설계했다.

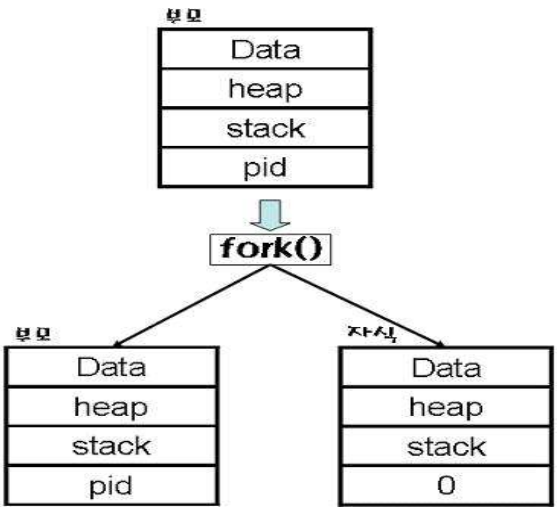
2. 프로젝트 기본개념

(1) fork() system call

① 새로운 task 생성 : 기존에 있던 task를 복사한다.
Existing task = parent task
New task = child task

② flow control이 2개로 나누어진다.

③ return 값이 2개다.
parent task : return child pid (always lager than 0)
child task : 0



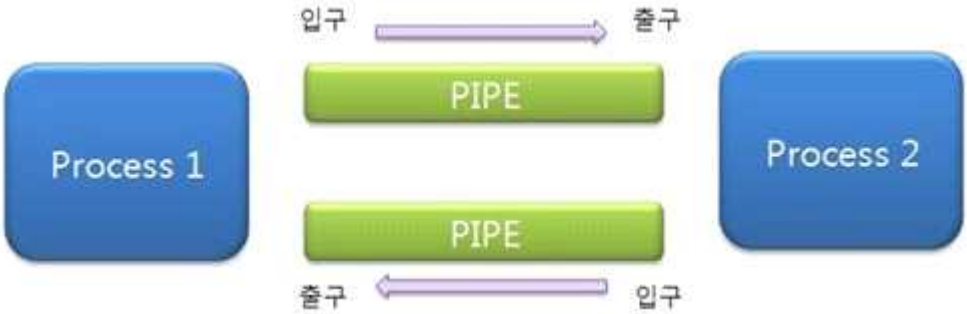
parent task와 child task는 각각 독립적인 task이기 때문에 어느 한쪽 변수의 값이 변해도 다른 task 변수의 값은 변하지 않는다.

(2) pipe() system call

① IPC 기법 중 하나로써 단순한 통로 역할을 한다.

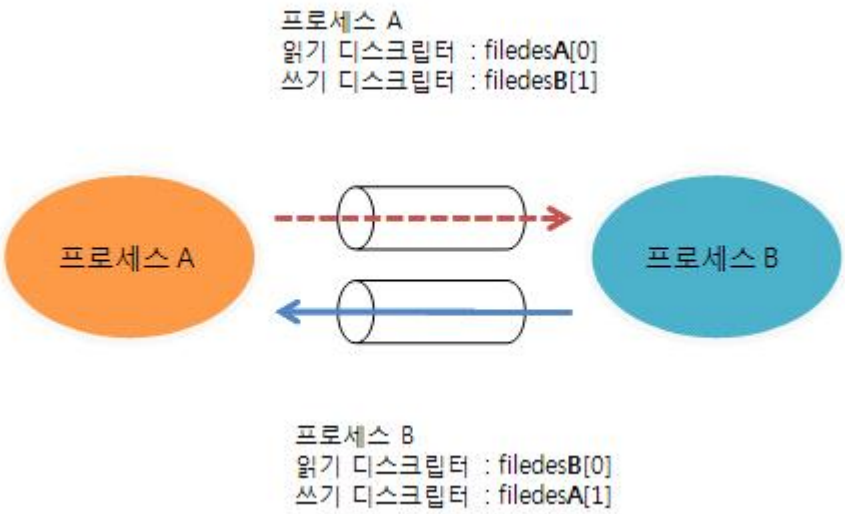
② 입구와 출구의 개념이 없다.

그래서 2개의 task가 통신하기 위해서 읽기 전용과 쓰기 전용 2개의 pipe를 사용한다.



③ fd[0] : 읽기 file descriptor

fd[1] : 쓰기 file descriptor



Ⅱ. 프로젝트 시행

1. 프로젝트 체계

박민혁	이상민
프로젝트 아이디어 제시	프로젝트 아이디어 구현
프로젝트 목표 및 필요성 작성	프로젝트 설계 및 코딩
프로젝트 결과 및 확장방향 작성	프로젝트 시행착오 작성
프로젝트 기본개념 작성(공동)	

2. 프로젝트 설계

```c
/* for loop comparison by.sangmin & minhyuk */
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/types.h>

int main()
{
        long i;
        struct timeval stime, etime, gap;


        gettimeofday(&stime, NULL);
        for(i = 0; i < 100000; i++);
        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);

        pid_t pid;
        pid = fork();

        gettimeofday(&stime, NULL);
        if(pid == 0) for(i = 50000; i < 100000; i++);
        else {
                for(i = 0; i < 50000; i++);
                exit(1);
        }
        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);

        return 0;
}
```

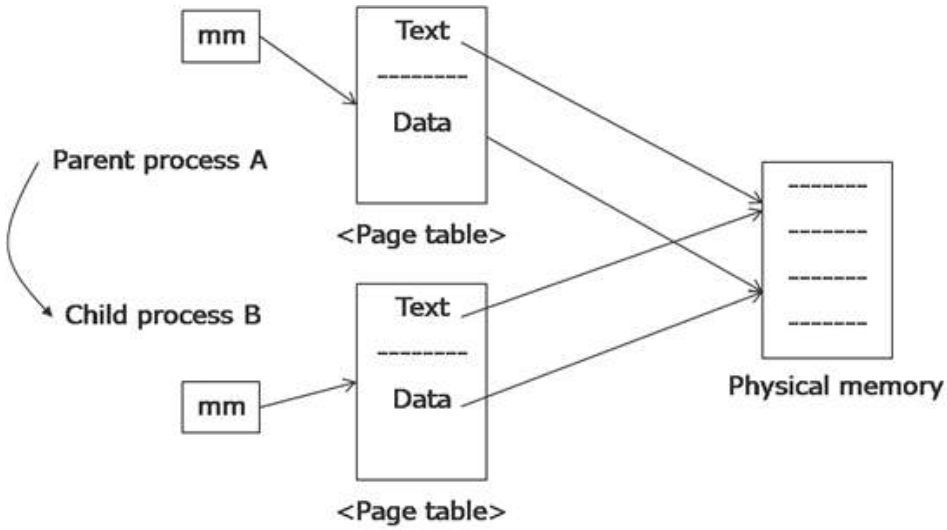
(초기 설계 코드)

기본개념에서 서술한 것처럼 fork()는 flow control을 2개로 나눠준다. 따라서 parent task와 child task 각각에서 for loop를 반반씩 수행한다면, 일반적인 for loop보다 수행시간이 빠르지 않을까 생각했다. 예를 들어 for loop가 100번 반복한다고 가정을 하면, 일반적인 for loop에서 100번 반복하는 것과 fork()를 사용해 parent task에서 50번, child task에서 50번 반복하는 것의 수행 시간을 비교해보기로 했다. 단, parent task가 child task를 기다리지 않고 동시에 수행해야 한다는 조건이 붙는다. 따라서 wait()를 사용하지 않기로 결정했다.

시간 측정을 위해 가장 최근 수업시간에 배운 gettimeofday()를 사용했다. 첫 번째 gettimeofday()에서는 일반적인 for loop의 수행 시간을 측정하고, 두 번째에서 fork()에서의 for loop 수행 시간을 측정했다. 공정한 수행 시간 측정을 위해 error handling은 다루지 않았다.

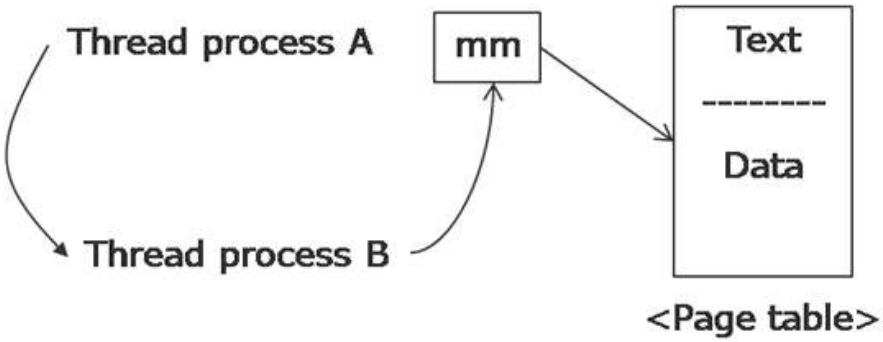
3. 프로젝트 시행착오

초기 설계 코드를 컴파일해서 실행시켜본 결과 정말로 수행 시간에 차이가 있었다. wait()를 하지 않아 parent task와 child task가 동시에 각각 for loop를 수행했고, 그 결과 일반적인 for loop보다 수행 시간이 짧게 측정 되었다. 그래서 바로 덧셈과 곱셈 연산을 비교해보기로 했다. 하지만 성공하지 못했는데, 이유는 다음과 같다.



(fork()에서의 메모리 공유 방식)

위의 그림처럼 fork() 후 parent task와 child task는 변수를 공유하지 않는다. 그래서 덧셈 연산 시 parent task과 child task에서 각각 더한 값을 마지막에 합치지 못하게 된다. 고민 끝에 처음으로 내린 결론은 thread를 사용하는 것이었다. 다음 그림처럼 thread는 fork()와 달리 변수를 공유한다.

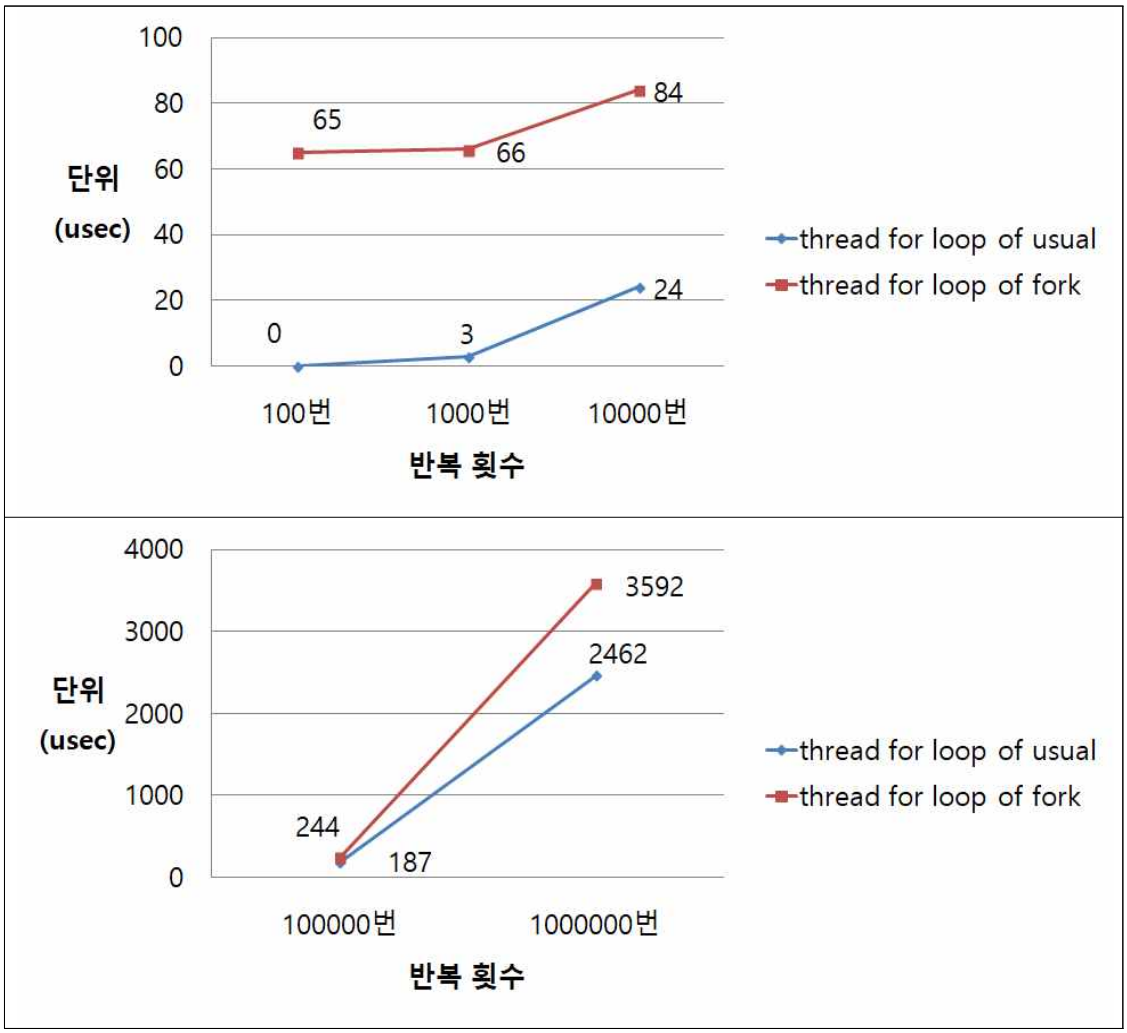


(thread에서의 메모리 공유 방식)

```c
/* thread comparison by.sangmin & minhyuk */
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/types.h>

long j = 0;
void *thread(void *unused) {
        for(j = 0; j < 50; j++);
}

int main()
{
        long i;
        struct timeval stime, etime, gap;

        gettimeofday(&stime, NULL);
        for (i = 0; i < 100; i++);
        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("\n%ldsec : %ldusec\n\n", gap.tv_sec, gap.tv_usec);

        pthread_t tid;

        gettimeofday(&stime, NULL);
        pthread_create(&tid, NULL, &thread, (void *)NULL);
        for(i = 50; i < 100; i++);
        pthread_join(tid, NULL);
        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("\n%ldsec : %ldusec\n\n", gap.tv_sec, gap.tv_usec);

        return 0;
}
```

(thread 설계 코드)

그래서 수업 때 배운 pthread_create()를 사용해서 구현해 보았다. 이는 fork()와 같은 역할을 하는데, 차이점이 있다면 thread는 이를 생성한 task와 모든 자원을 공유한다. 따라서 pid 또한 생성한 task와 동일하다. 위의 코드에서 pthread_join()이 wait() 역할을 해준다. fork()에서는 wait()를 뺐는데 thread에서는 pthread_join()을 하니 당연히 수행 속도가 빨라질 리가 없었다. 그런데 반복횟수를 계속 늘려본 결과 신기한 결과를 도출했다.



이처럼 반복 횟수가 엄청나게 많아지면 일반 for loop보다 수행 시간이 빠르게 측정되었다. 어쨌든 우리의 목표에 부합하지 않아 다시 고민을 하고 마지막 결론에 도달했다. 바로 pipe를 사용하는 것이다. pipe는 IPC(Inter Process Communication)로써 프로세스 간 통신을 가능하게 해주고, 결론적으로 계속해서 해결하지 못했던 변수 공유 문제를 가능하게 했다.

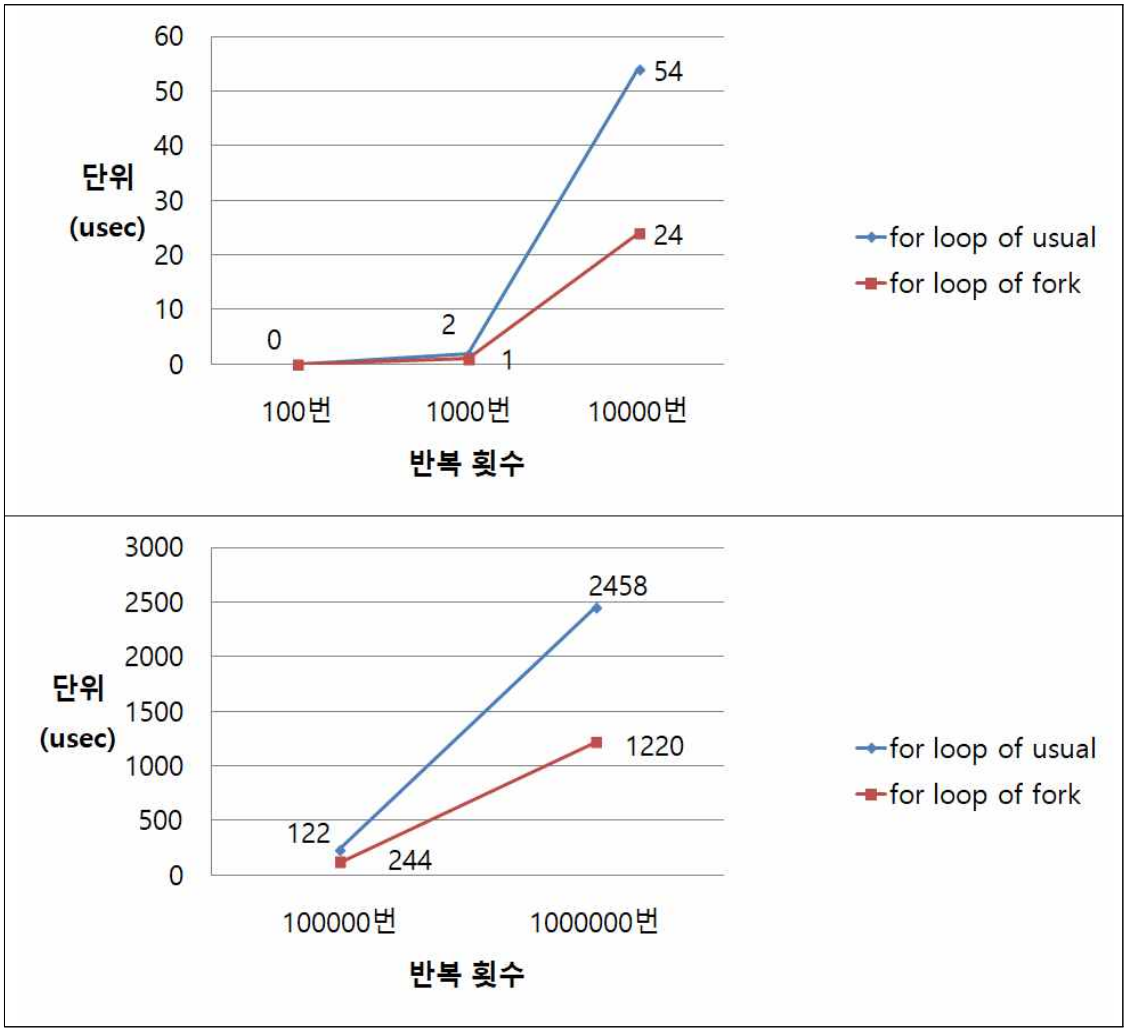
Ⅲ. 프로젝트 결과

1. 프로젝트 코딩

```c
/* add comparison using pipeline by.sangmin & minhyuk */
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/types.h>

int main()
{
        long i, sum = 0;
        struct timeval stime, etime, gap;
        gettimeofday(&stime, NULL);
        for(i = 1; i <= 100000; i++) {
                sum += i;
        }
        printf("%d\n", sum);
        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);


        sum = 0;
        long sum1 = 0, sum2 = 0;
        pid_t pid;
        int fd1[2], fd2[2];
        pipe(fd1);
        pipe(fd2);
        pid = fork();
        gettimeofday(&stime, NULL);
        if(pid == 0) {
                for(i = 50000; i <= 100000; i++) {
                        sum1 += i;
                }
                write(fd1[1], &sum1, sizeof(sum1));
        }
        else {
                for(i = 1; i < 50000; i++) {
                        sum2 += i;
                }
                write(fd2[1], &sum2, sizeof(sum2));
                exit(1);
        }
        read(fd1[0], &sum1, sizeof(sum1));
        read(fd2[0], &sum2, sizeof(sum2));
        printf("%d\n", sum1 + sum2);
        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);

        return 0;
}
```

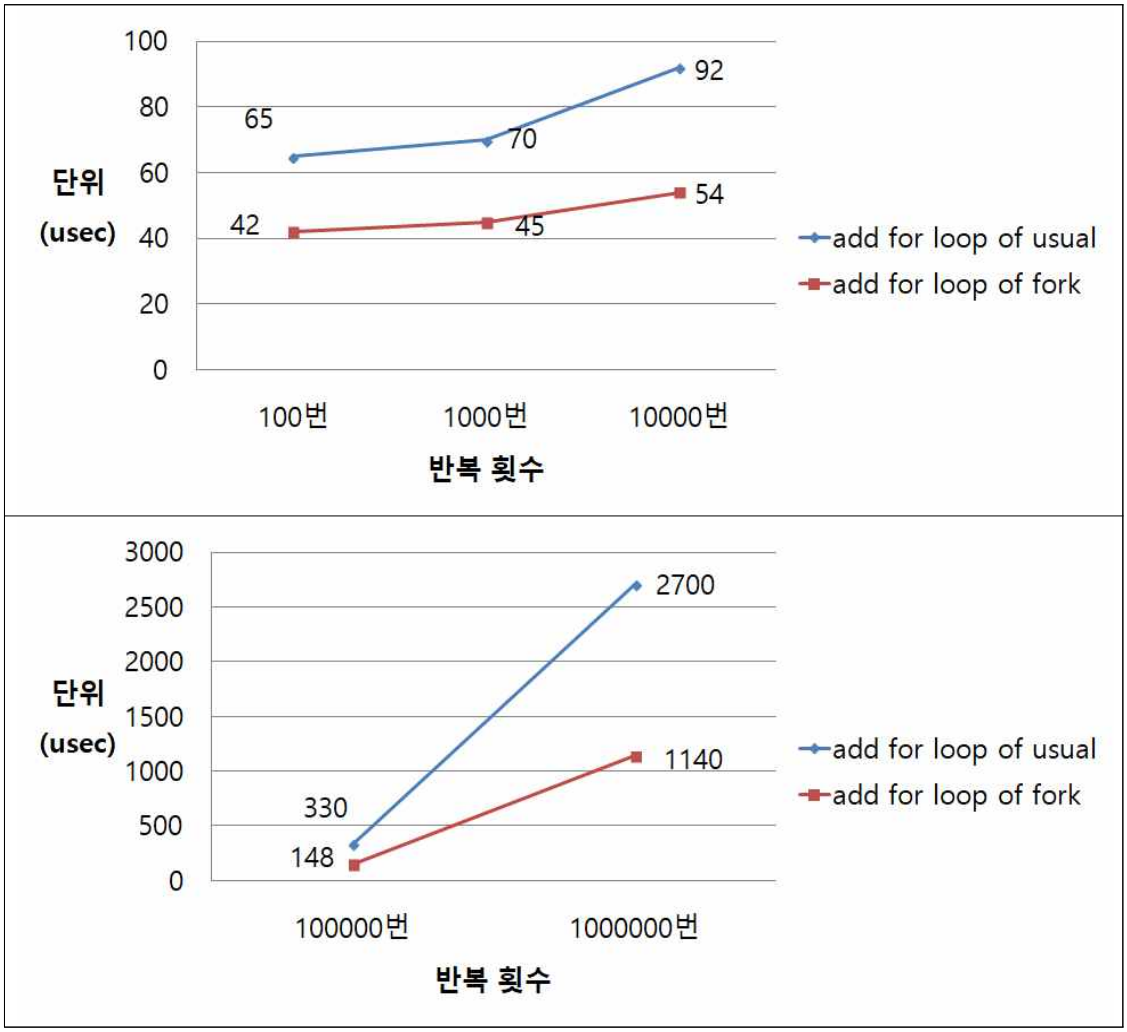
2. 프로젝트 결과

(1) 일반적인 for loop와 fork에서의 for loop 비교



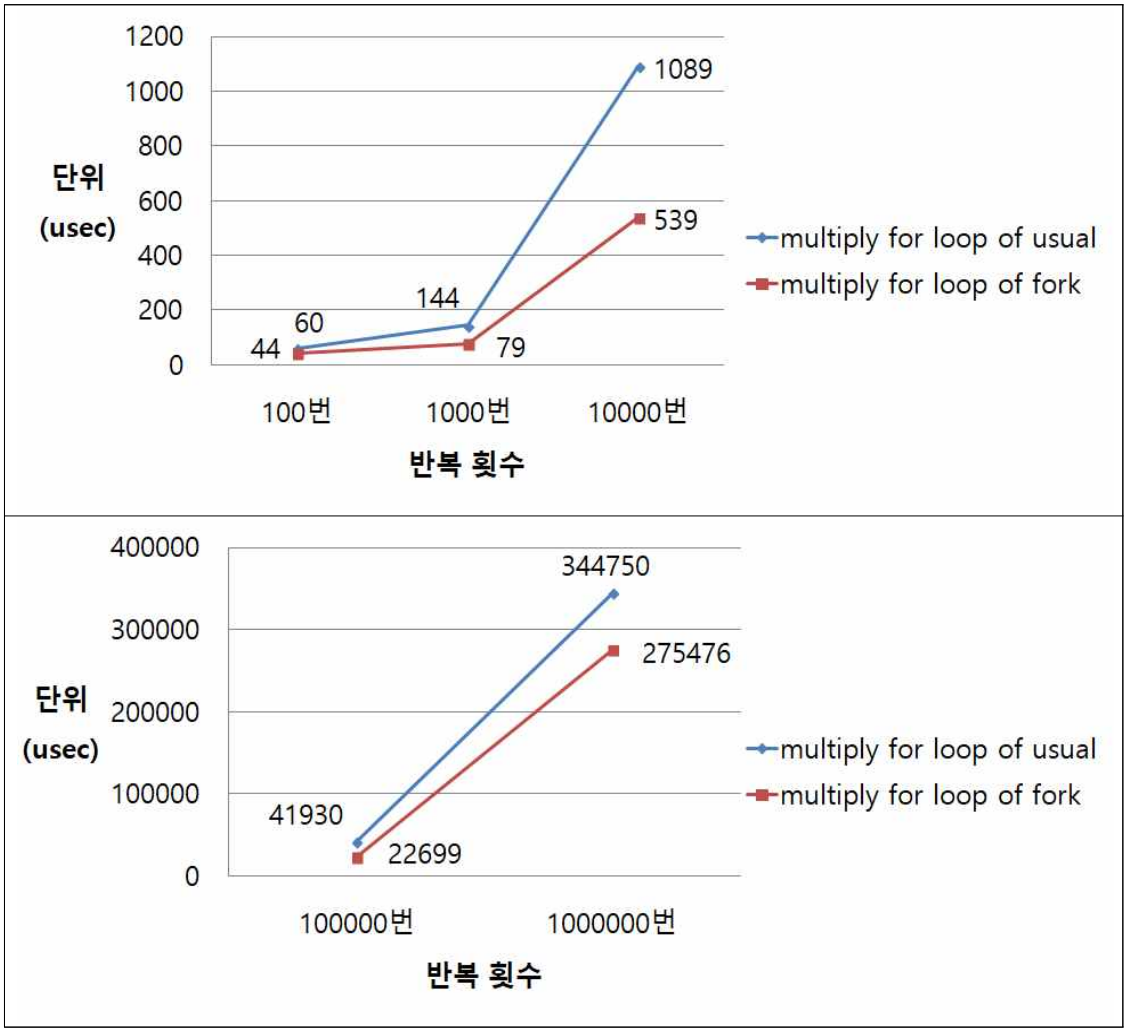
결과 : 그래프를 보면 fork()를 이용하여 속도가 빨라졌음을 보여준다. 처음 반복 횟수가 적을 때엔 비슷한 속도였지만, 반복 횟수가 10000개 이상으로 늘어나면 2배 이상의 속도 차이를 보이기 시작했다.

(2) 일반적인 for loop와 fork에서의 for loop 덧셈 연산 비교



결과 : 두 task간 통신이 가능하도록 pipe를 사용했다. pipe를 이용해 변수를 공유하고, 변수 공유를 해서 for loop를 이용한 것과 일반적인 for loop와 비교한 결과 그래프이다. 반복 횟수가 10000번이 될 때까지는 일정한 차이를 보여주는 반면, 100000번 이상이 넘어가서부터는 속도 차이가 크게 나는 것을 볼 수 있다.

(3) 일반적인 for loop와 fork에서의 for loop 곱셈 연산 비교



결과 : 곱셈의 경우도 덧셈과 비슷한 결과를 보여준다. 차이가 있다면 반복 횟수를 100번만 하더라도 거의 2배에 속도 차이가 난다는 점이다. 이로써 fork()와 pipe를 이용해 연산식의 속도를 향상시키는데 성공했다.

3. 프로젝트 확장방향

프로젝트 확장 방향으로는, 수업 시간에 배운 Loop unrolling과 우리가 진행한 프로젝트를 합치면 더 많은 속도 향상을 보여줄 것 같다. 또한 fork()의 사용 횟수를 늘려 flow control을 기존보다 훨씬 더 많이 나눈다면 코딩 자체는 복잡하겠지만, 현재 진행한 프로젝트보다 더욱 빠른 속도향상을 보여 줄 것 같다. 더 나아가 Loop unrolling도 하고, fork()의 사용 개수도 늘린다면 둘 중 한 가지를 이용하는 것 보다 수십 배 속도 향상 될 것이라고 생각한다.

Ⅳ. 프로젝트 논의

1. 박민혁 discussion

초기엔, 컴퓨터 구조에 대해 설계를 하려고 했다. 하지만, 컴퓨터 구조론 수업을 듣지 않는 내게는, 다른 친구들에 비해 아는 것이 부족했다. 그렇다고 해서, 좋은 아이디어를 내지 못 했던건 아니지만, 설계를 하기엔 지식이 부족했다. 그래서 프로젝트 팀원인 상민이와 좀 더 아이디어를 생각해 보자고 했다. 그러고 나서, 시스템 프로그래밍을 복습하기 시작했다. 복습을 하던 도중, 챕터 5의 첫 번째 이슈인, fork()로 재밌는 아이디어가 생길 것 같아서, 좀 더 깊게 생각해 보았다. 그러던 도중, Loop Splitting이 떠올랐고, 그것을 평범한 for loop에 적용시키는 것이 아닌, fork()를 사용해 적용해보자라고 아이디어를 제시했고, 같은 팀원인 상민이도 괜찮은 아이디어인 것 같아서, 추진하기 시작했다. 나는 코딩에 자신이 없어, 상민이를 위주로 코딩을 짜기 시작했다. 프로젝트를 진행 하면서 시행착오도 많았었다. 이것저것 배운 것들을 어려 번 넣어봤지만, 계속적으로 오류가 있었다. 하지만, 계속적으로 어떻게 하면 좋을까? 라는 생각을 계속했고, 그렇게 많은 시행착오 끝에 지금 코딩이 완성되었다. 코딩을 하면서, 상민이에게 배운 것이 많았고, 그것을 토대로 이번 프로젝트에 좀 더 노력을 기울이기 시작했다. 초기엔, 내가 과연 창의적인 아이디어를 낼 수 있을까?라는 생각을 많이 했고, 자신감도 없었다. 하지만 이번 프로젝트를 진행하면서, '나도 괜찮은 아이디어를 낼 수 있다.'라고 생각을 가지기 시작했고, 더불어 자신감도 생기기 시작했다. 무엇보다 task 관련 system call 이슈에 대해 더 깊게 공부할 수 있는 기회이기도 했으며, 이번 프로젝트를 진행하면서 나에게 의미 있는 프로젝트라고 생각하는 계기가 되었다.

2. 이상민 discussion

이번 프로젝트는 시스템 프로그래밍 수업 마지막 과제로써 open topic이었다. 그래서 초반에 주제 정하는 것이 쉽지가 않았다. 교수님께서 예시로 들어

주신 disk 모든 track에 head를 달아 seek time을 없애는 방법처럼 구조적인 측면에서 프로젝트를 설계할까 고민을 했었다. 그러던 중 박민혁 학생이 fork()를 사용하면 flow control이 2개가 되기 때문에 parent task와 child task에서 각각 반복문을 수행하면 속도 향상이 되지 않을까하는 질문을 던졌다. 일리 있는 의견이라고 생각해서 곧바로 실행으로 옮겼다.

gettimeofday()를 사용해서 수행 시간을 재는 것은 어렵지 않았다. 또한 우리가 예상했던 것처럼 fork()에서 flow control이 2개로 나뉘어 parent task에서 반, child task에서 나머지 반만큼 반복문을 수행하는 것이 일반적인 반복문보다 빨랐다. 전제 조건이 충족되었으니 연산으로 응용을 해 증명하고 싶었다. 그러나 바로 난관에 봉착했다. fork()를 해서 생기는 parent task와 child task는 서로 변수를 공유하지 않는다는 점이었다. 연산을 반씩 수행하면 각자 수행한 결과를 마지막에 합쳐야 하는데 서로 다른 변수를 사용하다 보니 그것이 불가능했다. 이것을 해결하기 위해 많은 노력을 했다. 해결하지 못하면 프로젝트 주제 자체를 바꾸고, 처음부터 다시 시작해야 했기 때문이다. 그렇게 생각해낸 것이 바로 ppt chapter5 후반부에 간략히 나오는 thread였다. 수업 시간에 필기한 내용을 보니 thread와 fork()의 가장 큰 차이점은 바로 데이터 공유였다. ppt에는 간략하게 한 슬라이드로만 나와있어서 직접 구현을 위해 인터넷을 찾아보았다. 코드를 짜고 수행시켜본 결과 일반적인 for loop보다 현저히 느린 수행 속도를 보여주었다. 이유는 pthread_join 때문이라고 생각했다. 이것이 fork()에서 wait() 역할을 해주는데, 처음 fork()를 이용하여 코드를 짤 때 wait()를 사용하지 않았다. wait()는 parent task가 child task의 수행이 끝날때까지 기다리기 때문에 이를 사용하면 수행 속도를 향상시키지 못할 것이라 여겼기 때문이다. 그래서 pthread_join을 없애도 봤는데, 이 경우에는 제대로 실행되지 않았다.

고민에 고민을 거듭한 결과, 마지막으로 생각해낸 것이 바로 pipe였다. pipe는 프로세스 간 통신을 가능하게 해주는 IPC(Inter Process Communication) 기술 중 하나였다. 수업 시간에는 pipe를 항상 write와 read를 이용해서 썼다. 그래서 문자열만 주고 받을 수 있는 개념인 줄 알았는데, 인터넷을 찾아보니 변수를 주고 받는 것 또한 가능했다. 이로써 새로운 사실을 하나 더 알게 된 셈이었다.

결론적으로 pipe를 2개 사용하였고, 각각의 task에서 수행한 결과를 합칠 수 있게 되었다. 코딩을 하면서 포기하고 싶은 순간이 많았었는데 한순간에 그러

한 생각들을 모두 떨쳐냈다. 수준 높은 개발자들의 코드와 견주어보면 보잘 것 없겠지만 그 순간만큼은 나도 개발자가 된 기분이었다. 구현이 끝났기 때문에 이를 바탕으로 프로젝트 보고서를 작성하는 것은 일사천리였다. 이번 프로젝트 덕분에 새롭게 알게된 사실도 상당히 많고, 학습한지 오래되어 헷갈리는 개념 또한 바로잡을 수 있을 수 있어 유익한 시간이었다. 이번 프로젝트를 바탕으로 앞으로 있을 설계 과목들도 잘 해낼 수 있다는 자신감도 생겼다.