운영체제 과제 #2

: 멀티 프로세스 프로그램 구현

과 목	운영체제 _02
전 공	컴퓨터공학과
학 번	20190850
이 름	이수진

목 차

Ⅰ. 문제 기술

1. 문제 정의 및 해결

연립방정식의 해를 구하는 프로그램을 작성한다. 연립방정식 풀이는 Gaussian elimination과 back substitution을 이용하고, Multi-process 방식을 사용하여야 한다.

이 문제는 반복문을 이용해 다중으로 fork()를 생성하고, 여러 프로세스는 연립방정식을 구하는 두 알고리즘을 각각 병렬 처리함으로 해결해야 한다. 이때 행렬과 두 벡터는 공유 메모리에 공유해야 하며, 데이터의 일관성을 유지하기 위해 프로세스 동기화가 필요하다.

Ⅱ. 세부 구현사항

1. 구현 시 어려웠던 코드

(1) shared memory

파일을 읽을 때 share memory에 행렬과 벡터를 공유하고자 했을 때, ‘bad address’ 혹은 ‘세그멘테이션 오류(core dumped)’ 에러가 발생했었다. [1][4] 이는 함수 mmap()과 공유 메모리를 잘못 이해한 것으로 mmap()의 두 번째 인자를 size로 변경하니 해결되었다.

```
int size = n1*n1*sizeof(float) + 2*n1*sizeof(float);
ftruncate(fd, size);
float *a = mmap(NULL, size, PROT_READ | PROT_WRITE,
                MAP_SHARED, fd, 0);
```

(2) 다중 fork 생성 및 파이프라인

[4] 변수 pid와 배열 pids, p2c와 c2p를 각각 선언한다. 이때 p2c, c2p는 p*2 크기의 2차원 배열이다. p2c, c2p는 pipe로, 1개 생성 시 동기화가 제대로 작동하지 않았다.

```
pid_t pid;
pid_t pids[p];

pid_t p2c[p][2];
pid_t c2p[p][2];
```

pid, pids는 각각 fork의 반환 값을 저장하기 위한 변수, 자식 프로세스의 ID를 알고자 하는 배열이다. 처음엔 프로세스 간 구분을 위하여 fork를 생성하는 for-문 안에 Gaussian-elimination을 계산했다. 그랬더니 프로그램이 중단되는 에러가 발생했다.

[2][4] 그래서 분리하여 fork 생성을 main에 두고, 함수 안에서도 프로세스를 구분할 수 있도록 pids로 프로세스 ID를 저장해 함수에게 전달하는 방식으로 해결하였다.

[3] 그리고 fork가 여러 번 생성되어 x의 값이 올바로 나오지 않는 오류 또한 발생

했는데, 이는 자식 프로세스일 때 break를 넣어 해결하였다.

```
int i;
for (i=0;i<p;i++)
{
    pid = fork();
    if (pid == 0) // child process
    {
        pids[i] = getpid();
        break;
    }
    …
}
```

2. 알고리즘

(1) Parallel Gaussian elimination

과제 설명에 있는 알고리즘을 사용하였다. Gaussian elimination은 연립방정식 수식을 upper triangle 형태의 방정식으로 변경하는 과정이다.

먼저 함수 getpid()와 배열 pids을 이용해서 현 프로세스가 부모인지, 혹은 어느 자식 프로세스의 index인지를 알아낸다.

0부터 n-2인 for-l 반복문을 돌 때 부모 프로세스와 자식 프로세스인지 구분한다. 다음 l에 들어가기 전, write()와 read() 함수를 이용해 부모 프로세스인 경우, 자식 프로세스들로부터 동기화 신호를 모두 모아 다시 자식 프로세스에게 알린다. 이때 자식 프로세스 모두에게 알려야 하므로 for문을 사용한다. 자식 프로세스인 경우, for-i, for-j 문을 돌고 난 후 부모 프로세스에게 동기화 신호를 보낸 뒤, 동기화 신호를 되받는다.

자식 프로세스일 때 for-i, for-j 과정에서 첫 번째 단계는 $A_{0,0}$을 제외한 0-행의 모든 계수 $A_{i,0}$을 0으로 만드는 것이다. 두 번째 단계는 변경된 방정식을 바탕으로 $A_{1,1}$을 제외한 1-행의 모든 계수 $A_{i,1}(i>1)$을 0으로 만든다. 이를 반복하여 k-번째 단계에서 $A_{k,k}$를 제외한 k-열의 모든 계수 $A_{i,k}(i>k)$을 0으로 만든다. 이 단계는 n-2까지 반복된다.

이때 하나의 단계에서 하나의 아래쪽 행을 0으로 만드는 과정은 순서가 중요하지 않기 때문에 병렬로 수행할 수 있다. l+l~n-1의 열들을 p_0~p_{p-1} 프로세스에게 분배하는데, 프로세스 pi가 담당할 시작 열과 마지막 열을 계산하여 for-i 문, for-j 문을 돈다.

```
function Gaussianelimination(n, A, b, p, pids, p2c, c2p)
n <- 행과 열의 개수
A, b <- 행렬과 벡터
p <- 프로세스 개수
pids <- 모든 자식 프로세스의 ID 배열
p2c, c2p <- 부모와 자식 프로세스 간의 pipe
start, end <- 프로세스 pi가 담당할 시작 열과 마지막 열의 index
m <- 병렬화해야 하는 블록의 개수
pi <- 자식 프로세스 index. 단, 부모일 때는 자식 프로세스 개수
id <- 현재 프로세스의 id
data <- 정수 크기의 초기화 하지 않은 데이터

        pi <- p
        id <- getpid()

        for i <- 0 to p do
            if id == pids[i] then
                pi <- i
            end if
        end for

        for l <-0 to n-1 do
            if pi >= p then
                for k <- 0 to p do
                    read(c2p[k][0], &data, sizeof(data))
                end for
            else
                m <- n-l-1
                start <- (pi*m)/p+1+l
                end <- ((pi+1)*m)/p+l;
                for i <- start to end+1 do
                    for j <- 1+l to n do
                        A[i*n+j] <- A[i*n+j] - (A[i*n+l] / A[l*n+l]) * A[l*n+j]
                    end for
                    b[i] <- b[i] - (A[i*n+l] / A[l*n+l]) * b[l]
                end for
                write(c2p[pi][1], &data, sizeof(data))
                read(p2c[[pi][0], &data, sizeof(data)]
            end if
        end for

End Gaussianelimination()
```

[4] 동기화 과정 시 처음엔 작동했더니 프로그램 실행마다 여러 값이 나오는 오류가 발생했다. 이는 파이프라인을 두 개 생성하고, sync child 시 for 문으로 모든 자식 프로세스와 통신/동기화하여 해결하였다.

(2) Back Substitution

과제 설명에 있는 알고리즘을 사용하였다. Back Substitution은 후진 대입법으로 맨 밑의 있는 행부터 x의 값을 구해 오는 과정이다.

위의 알고리즘과 같이 함수 getpid()와 배열 pids를 이용해서 현 프로세스의 정보를 읽는다. 다음에 오는 for-i는 부모와 자식 프로세스를 구분하여 계산한다. 다음 I에 넘어가기 전, 부모 프로세스인 경우, 통신 및 동기화만 수행하고, 자식 프로세스인 경우, 계산 후 통신 및 동기화를 수행한다. 자식 프로세스일 때 계산에서 for-j문은 병렬화할 수 있다. n-1~0의 열들을 p_0~p_{p-1} 프로세스에게 분배하는데, 프로세스 pi가 담당할 시작 열과 마지막 열을 계산하여 for-j 문을 돈다.

```
function BackSubstitution(n, A, b, x, p, pids, p2c, c2p)
n <- 행과 열의 개수
A, b <- 행렬과 벡터
x <- 해 벡터
p <- 프로세스 개수
pids <- 모든 자식 프로세스의 ID 배열
p2c, c2p <- 부모와 자식 프로세스 간의 pipe
start, end <- 프로세스 pi가 담당할 시작 열과 마지막 열의 index
pi <- 자식 프로세스 index. 단, 부모일 때는 자식 프로세스 개수
id <- 현재 프로세스의 id

        pi <- p
        id <- getpid()

        for i <- 0 to p do
            if id == pids[i] then
              pi <- i
            end if
        end for
```

```
        for i <- n-1 to -1 do
            x[i] <- b[i] / a[i*n+i]
            if pi >= p then
              for k <- 0 to p do
                   read(c2p[k][0], &data, sizeof(data))
              end for
              for q <- 0 to p do
                   write(p2c[q][1], &data, sizeof(data))
              end for
            else
                 start <- (pi*i)/p
                 end <- ((pi+1)*i)/p
                 for i <- start to end do
                    for j <- start to end do
                       b[j] <- b[j] - (x[i]*a[j*n+i])
                    A[j*n+i] <- 0
                    end for
                 end for
                 write(c2p[pi][1], &data, sizeof(data))
                 read(p2c[[pi][0], &data, sizeof(data)]
            end if
        end for

End BackSubstitution()
```

Ⅲ. 체크 포인트

1. 수행 결과

```
leesujin@ubuntu:~/os/hw2$ make
gcc -o hw2 hw2.o -lrt
leesujin@ubuntu:~/os/hw2$ ./hw2 4 a.dat b.dat c.dat
leesujin@ubuntu:~/os/hw2$ ./diffCk c.dat x.dat 0.001
Two file (c.dat, x.dat) are the same.
```

[그림 1] 수행 결과 스냅샷

2. 결과 분석

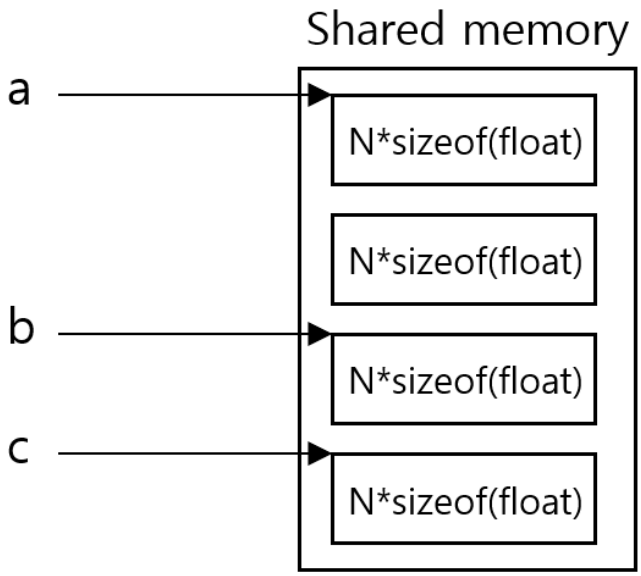
(1) p개의 child process를 생성하려면 어떤 방식으로 fork()를 구성해야 하는가?

다음과 같이 for 문을 이용해서 fork를 생성해야 한다.

```
for (i=0;i<p;i++) pid = fork();
```

(2) Shared memory를 사용할 자료구조들은 shared memory 상에서 어떤 자료구조를 가지는 것이 바람직한가?

다음과 같이 float형 포인터 a, b, c가 shared memory를 가리키는 구조여야 한다. (단, a는 1차원 배열)



[그림 2] shared memory 구조

(3) 전체 프로그램에서 총 몇 개의 ordinary pipe가 필요한가?

부모에서 자식 프로세스로 가는 pipe, 자식에서 부모 프로세스로 가는 pipe로 2개가 필요하다.

(4) 동기화 작업을 위한 sync child와 sync parent는 각각 어떻게 구현해야 하는가?

sync child는 부모 프로세스가 모든 자식 프로세스에게 보내는 것이므로 다음과 같이 프로세스의 개수만큼 for 문으로 돌아야 한다. (이때 data는 초기화하지 않은

정수형 변수)

```
for(int k=0;k<p;k++)
    read(c2p[k][0], &data, sizeof(data));
for (int q=0;q<p;q++)
    write(p2c[q][1], &data, sizeof(data));
```

sync parent는 현 자식 프로세스에서 부모 프로세스에게 보내는 것이므로 다음과 같이 구현한다. (이때 pi는 자식 프로세스 index)

```
write(c2p[pi][1], &data, sizeof(data));
read(p2c[pi][0], &data, sizeof(data));
```

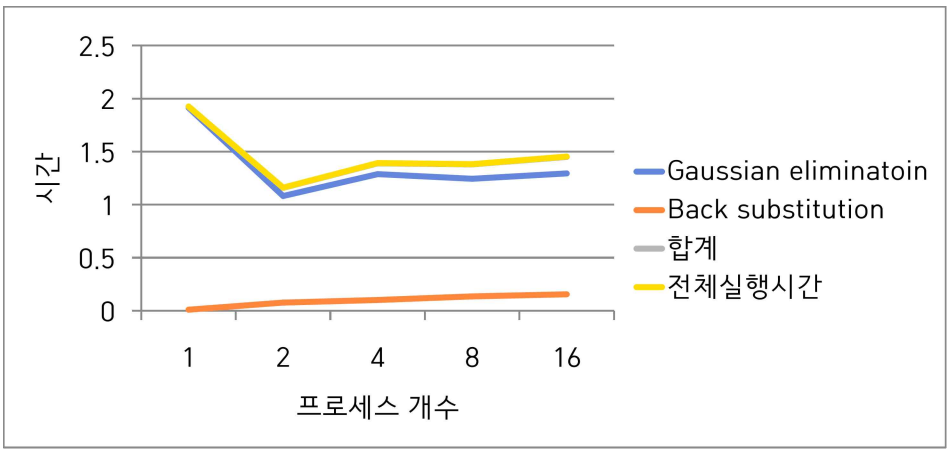
3. 성능 평가

(1) 자식 프로세스의 개수가 1, 2, 4, 8, 16개 일 때의 시간을 구해 표 1을 채우고 전체 실행 시간의 변화를 그래프로 표시하시오.

([그림 3]의 그래프에서 합계와 전체 실행 시간이 비슷하여 합계가 보이지 않는 문제가 있다.)

[표 1] 실행 시간 표

프로세스 개수	계산 시간			전체실행시간 (I/O 포함)
	Gaussian elimination	Back substitution	합계	
1	1.914257	0.010026	1.924283	1.926405
2	1.082001	0.076997	1.158998	1.160979
4	1.288317	0.101436	1.389752	1.392794
8	1.244527	0.135444	1.379971	1.382519
16	1.295109	0.155187	1.450296	1.454027



[그림 3] 실행 시간 그래프

Ⅳ. 결론

멀티 프로세스는 기존 과제 1에서 겪은 연립방정식 해결 알고리즘의 오래 걸리는 시간 문제를 해결한다. 그러나 프로세스가 너무 많으면 오히려 시간이 더 걸리게 되므로 적절한 수의 프로세스를 생성하는 것이 바람직함을 알 수 있었다.

Ⅳ. 참고문헌

[1] Latte 하늘이도우사, 『메모리 맵핑(mapping) - mmap() / mummap()』, mint&latte, 2021-04-06, https://mintnlatte.tistory.com/357
[2] crocus, 『소켓 프로그래밍-(8) 다중 fork() 사용과 이해』, crocus, 2021-04-06, https://www.crocus.co.kr/452
[3] MaybeMaster, 『wait 함수를 이용하여 모든 자식프로세스를 기다리는 방법』, egloos, 2021-04-07, http://voals.egloos.com/1678610
[4] 같이 수업을 듣는 학생 4명(최영민(19), 최은빈(19), 최연서(19))과 함께 share-memory의 사용, 함수에서 부모와 자식 process를 구분하는 문제, 파이프라인의 생성은 토론하여 작성하였다. 프로세스의 종료가 되지 않는 에러는 최연서(19) 학생의 도움을 받아 동기화와 exit(1) 지점을 수정하여 작성하였다.