

OS Assignment2 사전 보고서

컴퓨터과학과

2015147533 유현석

1. Process 와 Thread

1-(1) Process란

여러 분야에서 과정 또는 처리라는 뜻으로 사용되는 용어로, 컴퓨터 분야에서는 ‘실행중인 프로그램’이라는 뜻으로 쓰인다. 프로그램 또는 그 일부를 뜻하기도 하고, 데이터의 입력이나 출력 등을 조작하거나 처리하는 것을 말하기도 한다.

부모프로세스(parent process)라는 상위계층과 자식프로세스(child process)라는 하위계층이 존재한다. 부모프로세스는 프로그램이나 명령어에 의해 시작되며, 자식프로세스는 부모프로세스에 의해 만들어지는 것을 말한다. 하나의 부모프로세스는 여러 개의 자식프로세스를 관리하는데, 여러 개의 자식프로세스가 하나의 CPU에서 동시에 처리되는 것처럼 보이는 것을 멀티태스킹(multitasking)이라고 한다.

1-(2) Thread란

CPU가 독립적으로 처리하는 하나의 작업 단위를 뜻합니다.

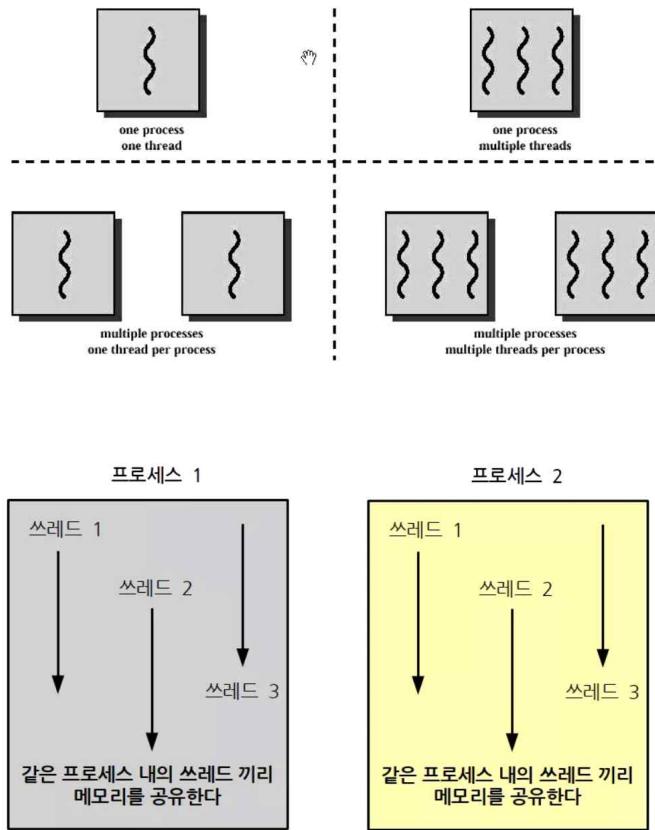
컴퓨터 프로그램 수행 시 프로세스 내부에 존재하는 수행 경로, 즉 일련의 실행 코드. 프로세스는 단순한 껍데기일 뿐, 실제 작업은 스레드가 담당한다. 프로세스 생성 시 하나의 주 스레드가 생성되어 대부분의 작업을 처리하고 주 스레드가 종료되면 프로세스도 종료된다. 하나의 운영 체계에서 여러 개의 프로세스가 동시에 실행되는 환경이 멀티태스킹이고, 하나의 프로세스 내에서 다수의 스레드가 동시에 수행되는 것이 멀티스레딩이다.

1-(3) Process 와 Thread의 차이점

1. Thread는 프로세스와 다음과 같은 차이점을 가진다.
2. 프로세스는 독립적이다. 쓰레드는 프로세스의 서브셋이다.
3. 프로세스는 각각 독립적인 자원을 가진다. 쓰레드는 stat, memory 기타 다른 자원들을 공유한다.
4. 프로세스는 자신만의 주소영역을 가진다. 쓰레드는 주소영역을 공유한다.
5. 프로세스는 IPC(:12)를 이용해서만 통신이 가능하다.
6. 일반적으로 쓰레드의 문맥교환(context switching)은 프로세스의 문맥교환보다 빠르다.

1-(4) 리눅스에서의 구조와 구현

Process와 Thread 구조 및 구현



프로세스는 서로의 메모리를 접근할 수 없지만, 같은 프로세스 내에 쓰레드끼리는 메모리를 공유한다

2. Multi-Process 와 Multi-Threading

2-(1) Multi-Process 구조 및 구현

<실제 예시>

```
pid_t pid[div];
for(int t=0;t<div;t++){
    pid[t] = fork();
    if(pid[t]==0){
        char c[]="./program1";
        char* const args[]={c,NULL};

        bwr1=t_to_string(t);
        cwr1=awr1+bwr1;
        ewr1=dwr1+bwr1;
        input1 = cwr1.c_str();
        output1=ewr1.c_str();

        in = open(input1,O_RDONLY);
        out = open(output1,O_WRONLY | O_TRUNC | O_CREAT,S_IRUSR | S_IWRUSR | S_IWGRP | S_IWUSR);
        dup2(in,out);
        dup2(out,1);
        close(in);
        close(out);
        execvp("./program1",args);
    }
}
```

1. fork() : system call 함수로써 Process를 생성한다.

헤더 파일 : <unistd.h>, <sys/types.h>

<sys/types.h>는 pid_t라는 구조형 때문에 헤더 파일에 포함해야한다.

fork는 다른 프로세스를 만들고 그때 현재 프로세스는 부모 프로세스로 하고 만들어진 다른 프로세스를 자식 프로세스라고 한다. 리턴하는 값이 pid_t는 프로세스를 구분할 수 있는 기준이 되기 때문에 중요하다. pid_t가 0보다 크면 부모, 0이면 자식, -1이면 Error가 발생한 것이다.

<예시| fork() 함수>

```
#include <sys/types.h>
#include <unistd.h>

void main()
{
    pid_t pid;

    printf("Before the Fork()\n");
    pid = fork();

    if(pid == 0)
        printf("This Is Child. And the value of PID = %d\n", pid);
    else if(pid > 0)
        printf("This is Parent. And the value of PID = %d\n", pid);
    else
        printf("Fork is Failed!!!\n");
}
```

2. exec : system call 함수로써 fork로 생성된 Process를 다르게 변경해준다.

exec는 함수명보다는 비슷한 일을 하는 family로 보는 것이 더 정확합니다. exec family가 하는 일은 현재 실행되는 프로세스에서 다른 프로세스 일을 하게 하는 것입니다. 예를 들어 fork()로 생성한 후 그 생성된 자식 프로세스를 다른 일을 하게 끔

변경할 수 있게 되는 것입니다.

```
#include <sys/types.h>
#include <unistd.h>

main()
{
    printf("executing ls\n");
    execl("/bin/ls", "ls", "-l", (char*) 0);
    perror("execl failed to run ls");
    exit(1);
}
```

2-(2) Multi-Threading 구조 및 구현

<실제 예시>

```
if(each_num==0){
    for(int k=0;k<else_num;k++){
        gettimeofday(&th_start,NULL);
        starttimenil[k] = (th_start.tv_sec*1000)+(th_start.tv_usec)/1000;
        adding.push_back(thread(&calculate,k,(k+1)));
    }
    for(int t=0;t<else_num;t++){
        adding[t].join();
        gettimeofday(&th_end,NULL);
        endtimenil[t]=(th_end.tv_sec*1000)+(th_end.tv_usec)/1000;
    }
}
else{
    for(int k=0;k<thread_num;k++){
        if(else_num==0)
            remain = 0;
        if(k==0){
            insertstart=0;
            insertend=each_num+remain;
        }
        else{
            insertstart=insertend;
            insertend=insertstart+each_num+remain;
        }
        gettimeofday(&th_start,NULL);
        starttimenil[k] = (th_start.tv_sec*1000)+(th_start.tv_usec)/1000;
        adding.push_back(thread(&calculate,insertstart,insertend));
        else_num= else_num-1;
    }
    for(int t=0;t<thread_num;t++){
        adding[t].join();
        gettimeofday(&th_end,NULL);
        endtimenil[t]=(th_end.tv_sec*1000)+(th_end.tv_usec)/1000;
    }
}
```

C++ ▾ 빌드 번호: 8 ▾ 166행, 18열 ▾ 삽입

Thread 함수란 multi - thread를 하기 위함으로 같은 일을 할 수 있는 여러개의 thread를 생성하는 함수입니다. 예시로 pthread가 아닌 thread 함수는 사용할 함수와 그 함수의 매개변수에 해당하는 변수로 입력하여 생성합니다.

<thread 생성 과정>

```
	thread(worker, data.begin() + i * 2500, data.begin() + (i + 1) * 2500,
           &partial_sums[i])
```

`thread` 생성자의 첫번째 인자로 함수 (정확히는 `Callable` 은 다 됩니다) 를 전달하고, 이어서 해당 함수에 전달할 인자들을 쭈르릉 써주면 됩니다.

Join() 함수란 앞서 생성한 thread가 끝날때까지 기다린 후 첫 번째가 리턴되면 그 다음 thread를 리턴하고 순서대로 리턴될 수 있게 도와주는 함수입니다.

<Join 과정>

```
■ C/C++ □ 확대 □ 축소

for (int i = 0; i < 4; i++) {
    workers[i].join();
```

3. Exec 계열의 시스템콜

3-(1) Exec 계열의 시스템 콜에는 어떤 것들이 있는지

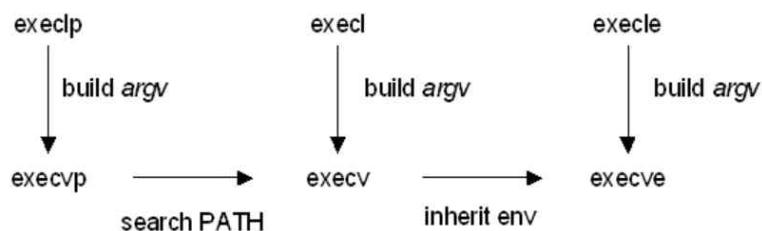
Exec Family

```
1)int execl(const char *path, const char* arg0, ..., const char* argv, (char*) 0);
2)int execlp(const char* file, const char* arg0, ..., const char* argv, (char*) 0);
3)int execle(const char* path, const char* arg0, ..., const char* argv, char* const envp[]);
4)int execv(const char* path, char* const argv[]);
5)int execvp(const char* file, char* const argv[]);
6)int execve(const char *path, char* const argv[], char* const envp[]);
```

3-(2) 조사한 시스템 콜의 리눅스에서의 구현 및 동작하는 방법

위에 그림에 해당하는 path, file에 대한 설명입니다.

path -> 실행파일까지 포함한 경로명 ex)/bin/ls <- ls가 실행파일이며, 경로명에 꼭 포함되어야 합니다.
file -> 실행파일만 ex)ls



<실제 예시>

```
in =open(input1,O_RDONLY);
out = open(output1,O_WRONLY | O_TRUNC | O_CREAT,S_IRUSR | S_IWGRP | S_IWUSR);
dup2(in,0);
dup2(out,1);
close(in);
close(out);
execvp("./program1",args);
```

4. 출처

[네이버 지식백과] 프로세스 [process] (두산백과)

[네이버 지식백과] 스레드 [thread] (IT용어사전, 한국정보통신기술협회)

https://www.joinc.co.kr/w/Site/system_programing/Book_LSP/ch07_Thread

<https://channelofchaos.tistory.com/55>

<https://modoocode.com/269>

OS Assignment2 수행 결과 보고서

컴퓨터과학과
2015147533 유현석

1. 프로그램의 동작 과정 및 구현 방법

1-1 Program 1 (기본 구현)

Program 1을 구현 과정은 크게 4개로 나누어집니다.

1. 주어진 Input 데이터를 저장할 배열을 동적 할당하는 것
2. 그 데이터를 받아서 각각에 해당하는 배열 저장하는 과정
3. 주어진 데이터 배열들을 이용하여 계산하는 과정 및 결과를 만드는 과정
4. 그 계산한 결과 값을 출력하는 과정

이와 같이 나누어집니다.

1. 그중 주어진 Input 데이터를 저장할 배열을 만들고 동적 할당하는 과정은 다음과 같습니다.

먼저 표준 입출력으로 ./program1 < (Input 받을 파일명) 이라는 실행 과정을 통해서 Input 받을 파일을 명시 해줍니다. 그 후, Input 파일 첫 번째 줄을 보면 filter , row , column 의 숫자를 알 수 있습니다. 그 3개의 숫자들과 각 filter는 3개로 이루어졌기 때문에 4차 Int 배열인 Filter [filter_num] [3] [filter_row] [filter_column]을 만들 수 있습니다.

마찬가지로 Data 역시 3차 Int 배열인 Data [3] [data_row+2] [data_column+2]을 만들 수 있습니다. Data 배열은 여러 개가 아닌 하나뿐이라 Filter와 다르게 개수를 만들어 줄 필요가 없어 3차 배열로 충분합니다. 2를 더하는 이유는 Zero-패딩을 해야하기 때문입니다.

결과를 출력할 Result 배열이 역시 필요합니다. 저는 Result 배열을 2개로 만들었습

니다. Result [filter_num] [3] [result_row] [result_column]과 Real_Result [filter_num] [result_row] [result_column]으로 이루어진 것들입니다. result_row와 column은 각각 Data의 row와 column에서 Filter의 row와 column을 빼준 후 3을 더한 값들입니다.

```
result_row = data_row - filter_row + 3
```

```
result_column = data_column - filter_column + 3
```

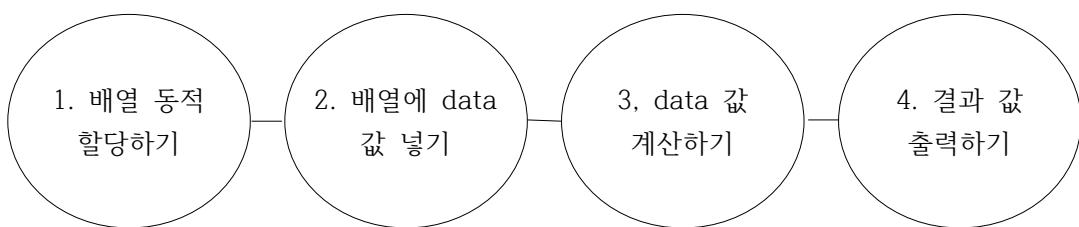
Result 배열과 Real_Result 배열의 차이점은 Filter와 Data를 계산한 결과를 저장한 것이 Result이고 3개로 이루어진 Result를 하나로 합치기 위해 각각의 값을 더해서 저장한 것이 Real_Result 배열입니다. 즉 출력할 배열은 바로 Real_Result 파일입니다.

이제 마지막으로 각각의 배열에 해당하는 크기만큼 4차 배열이면 4번의 동적 할당, 3차 배열이면 3번의 동적 할당 과정을 통해서 배열을 마무리해줍니다.

2. 데이터를 받아서 각각의 배열의 저장하는 과정은 해당하는 변수를 통해서 하면 됩니다. 예를 들어 Filter는 filter_num, 3, filter_row, filter_column을 이용한 4중 for문을 통해 cin 함수를 이용하여 저장 받고 Data는 3, data_row+2, data_column +2 를 이용한 3중 for문을 통해 저장 받습니다.

3. 주어진 데이터 배열들을 이용하여 계산하는 과정 및 결과를 만드는 과정은 6중 배열을 이용하여 만든 Calculate 함수를 통해 계산을 합니다. 함수는 Zero 패딩을 완료한 data 값과 filter 값을 합성곱 함수를 통해 Result 배열에 저장하는 과정을 담고 있습니다.

4. 결과 값을 출력하는 과정을 위해서는 3개로 이루어진 Result 배열을 합을 구해 Real_Result 배열에 저장한 후 그 값이 0보다 작으면 0으로 만들어주는 Max(0, 값)을 이용하여 마무리한다. 그 저장한 Real_Result 배열은 cout 함수를 이용하여 형식에 맞게 출력합니다.



1-2 멀티 프로세스 구현

Program 2을 구현 과정은 크게 6개로 나누어집니다.

1. 주어진 Input 데이터를 저장할 배열을 동적 할당하는 것
2. 그 데이터를 받아서 각각에 해당하는 배열 저장하는 과정
3. 주어진 데이터와 필터 배열들을 이용하여 새로운 Input 파일 만드는 과정
4. fork랑 execvp를 이용하여 program1를 만드는 과정
5. program1를 통해서 새로운 Input 파일에 해당하는 Output 파일을 만들어준다
6. 새로운 Output 파일을 형식에 맞게 출력하는 과정

이와 같이 나누어집니다.

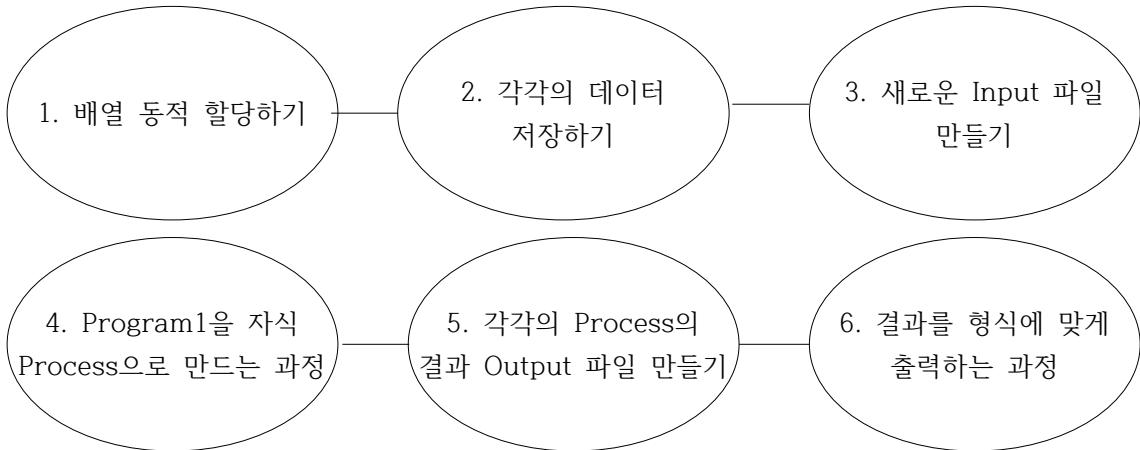
1, 2번 과정은 1-1의 Program1에서 했던 과정과 거의 일치합니다. 차이점이라고는 Data 배열을 만들 때 Zero 패딩을 위해 data_row와 data_column에 +2를 했던 것을 안하는 것입니다. 그 이유로는 새로운 파일만을 만들기 위한 데이터 저장의 역할만을 수행하기 때문에 있는 그대로의 data 값을 전달해줘야하기 때문입니다.

3번 과정은 1,2번을 통해서 저장된 Filter와 Data 배열에 있는 정보를 입력받은 Process의 숫자의 맞게 만들어주는 과정입니다. 여기서 만들어진 데이터들로 새로운 NewInput 파일들을 만들었습니다.

4번 과정인 fork랑 execvp를 이용하여 program1를 만드는 과정은 for문을 통해서 Process의 숫자만큼 fork를 해줘 자식 Process를 만들고 execp를 통해서 해당 Process를 Program1로 바꾸어줍니다. dup2 함수를 사용하여 각각의 Program1의 Input을 위에서 만든 NewInput으로 바꾸어줍니다.

5번 과정인 program1를 통해서 새로운 NewInput 파일에 해당하는 Output 파일을 만들어주는 과정은 각각의 Process에서의 표준출력 결과 값을 NewOutput이라는 파일로 적어주는 과정입니다.

6번 과정으로 각각의 Process에서 만들어진 NewOutput 파일들을 그 해당하는 개수만큼 읽어와 형식에 맞게 출력하는 과정입니다. 각각의 NewOutput 파일에 저장된 소요된 시간 결과 값들은 Time이라는 배열로 만들어서 저장해준 뒤 모든 결과들이 출력된 후에 표현해줍니다.



1-3 멀티 스레딩 구현

Program 3을 구현 과정은 크게 5개로 나누어집니다.

1. 주어진 Input 데이터를 저장할 배열을 동적 할당하는 것
2. 그 데이터를 받아서 각각에 해당하는 배열 저장하는 과정
3. Thread를 입력받은 개수만큼 생성하는 과정
4. 주어진 데이터 배열들을 Thread의 개수만큼 분할하여 계산하는 과정 및 결과를 만드는 과정
5. 그 계산한 결과 값을 출력하는 과정

이와 같이 나누어집니다.

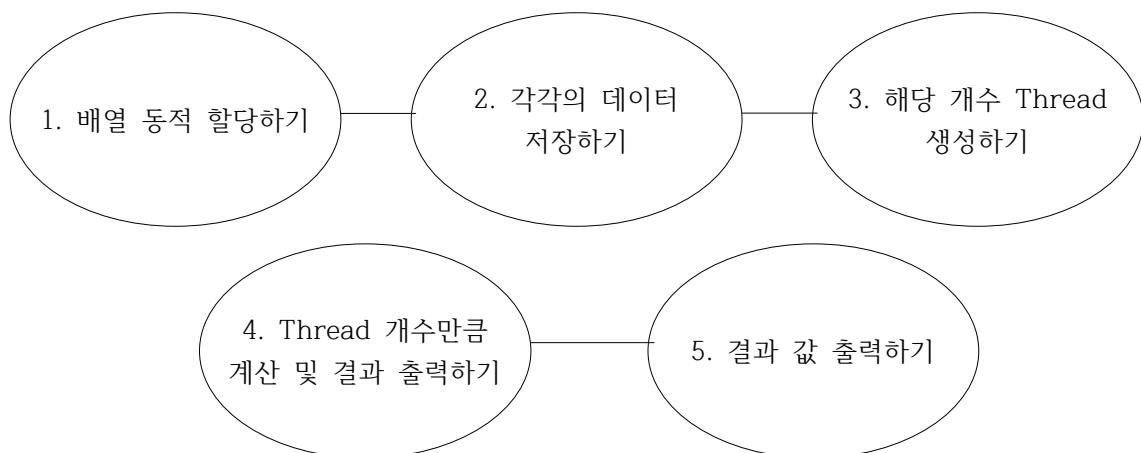
1,2번 과정을 Program1에서 했던 것과 일치합니다. 3번 프로그램에서 1번과 달라진 점은 연산 과정을 멀티 스레딩으로 구현했다는 점입니다. 그 외에 과정은 1번과 거의 유사합니다.

3번 과정을 2번과 마찬가지로 입력받은 숫자만큼 Threading을 하는 것입니다. For 문을 통해 해당 Thread 숫자만큼 반복한 후 목표 함수를 설정해줍니다. 여기서 목표 함수라고 하면 Program1에서 설명했던 calculate 함수입니다. Program1과 달라진 점이라고는 첨부터 끝까지 돌렸던 전과 다르게 Program3 에서는 시작해야 하는 Filter의 시작점과 끝나는 점을 지정해야 합니다.

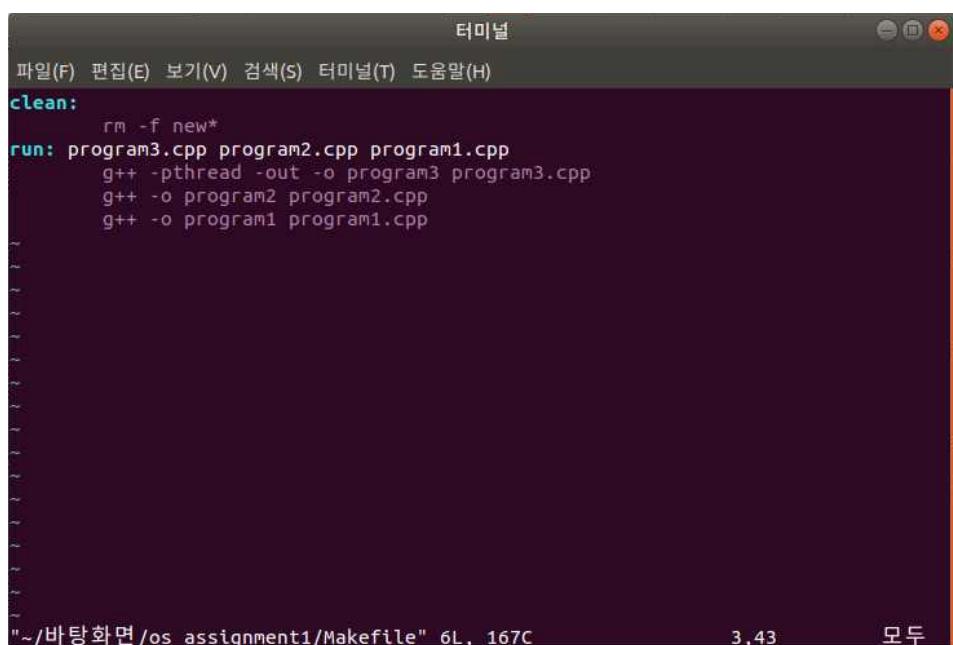
4번 주어진 데이터 배열들을 Thread의 개수만큼 분할하여 계산하는 과정 및 결과를

만드는 과정은 3번에서 설명한 거처럼 시작점과 끝나는 점이 지정되어 그 해당하는 Filter 배열들만이 Data 배열들과 계산됩니다. 그 결과 값은 Program1과 마찬가지로 Result 배열에 저장됩니다. 그 후 역시 Real_Result에 저장해줍니다.

5번 과정은 Program1과 마찬가지로 Real_Result를 형식에 맞게 출력합니다.



2. 작성한 Makefile에 대한 설명



```
터미널
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)

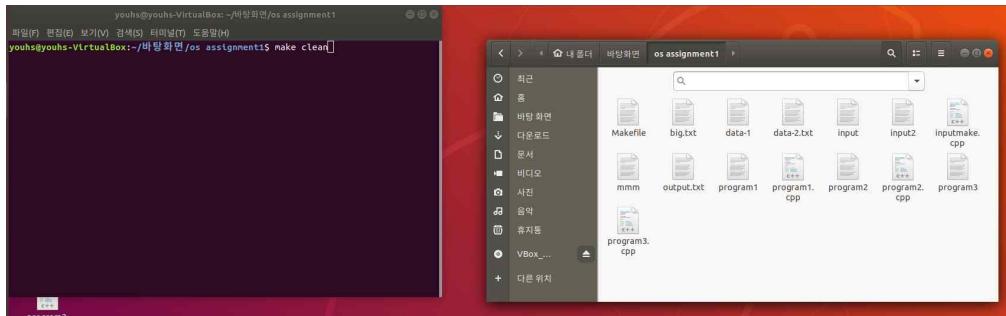
clean:
    rm -f new*
run: program3.cpp program2.cpp program1.cpp
    g++ -pthread -out -o program3 program3.cpp
    g++ -o program2 program2.cpp
    g++ -o program1 program1.cpp

~/바탕화면/os assignment1/Makefile" 6L, 167C      3,43      모두
```

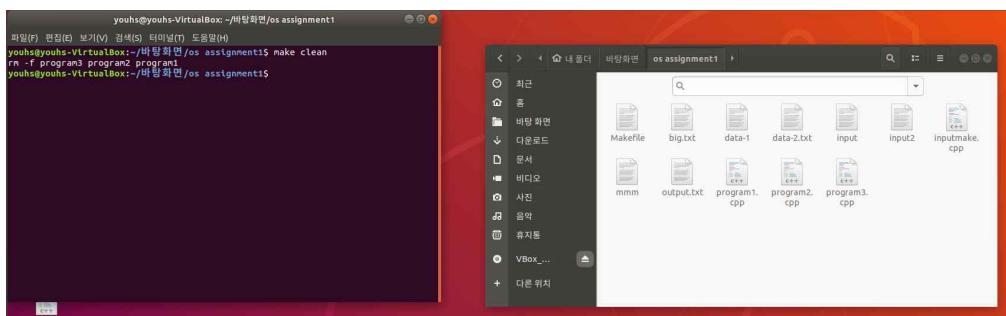
제가 작성한 Makefile은 2가지 기능을 가지고 있습니다.

1. make clean을 실행 시 Program1.cpp, Program2.cpp, Program3.cpp을 컴파일 하는 과정에서 생성된 Program1, Program2, Program3이라는 파일들을 삭제하는 과정입니다.

<make clean 실행 전>

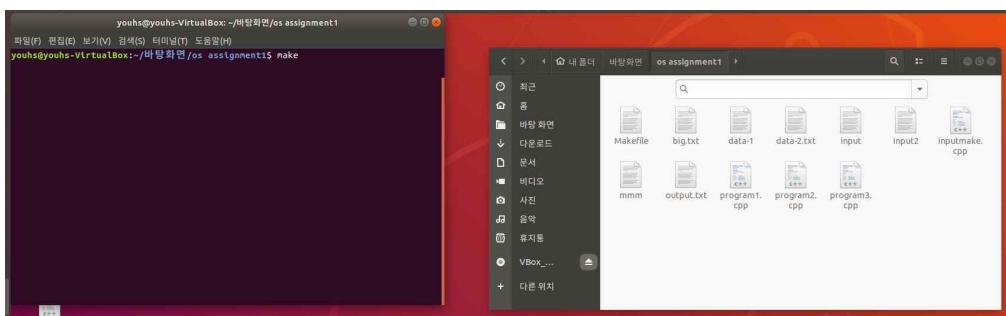


<make clean 실행 후>

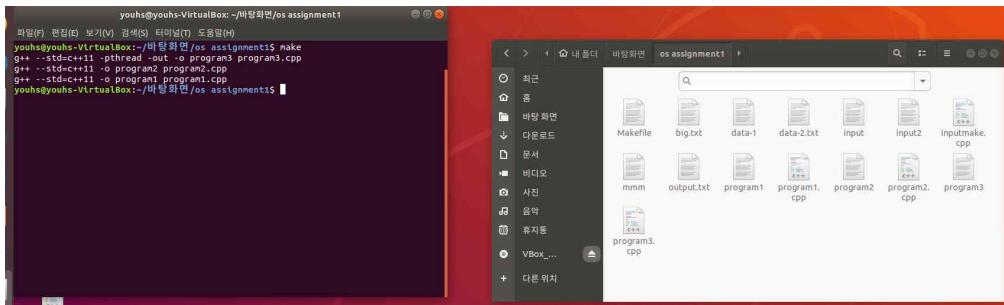


2. make을 실행 시 Program1.cpp, Program2.cpp, Program3.cpp 의 3개의 파일들이 컴파일 되고 그 이름은 각각 a.out 대신에 program1, program2, program3으로 바뀌어져 저장됩니다.

<make 실행 전>

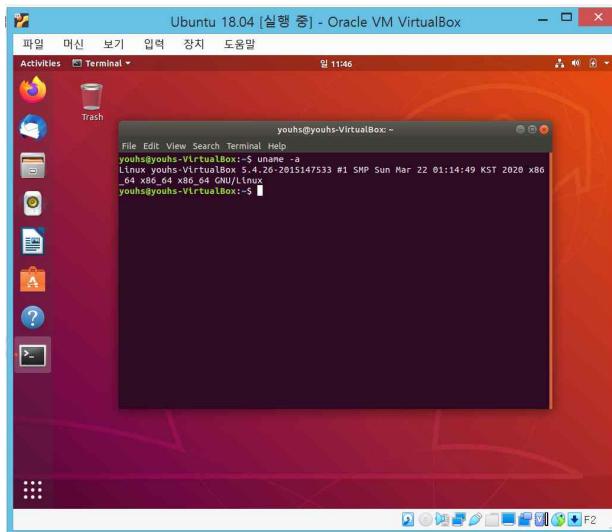


<make 실행 후>



3. 개발 환경 명시

3-1 uname -a 실행 결과



3-2 사용한 컴파일 버전, CPU, 메모리정보

The image consists of three vertically stacked screenshots of a Linux terminal window. The terminal has a dark background with light-colored text and a standard window title bar.

Screenshot 1: The terminal shows the output of the command `gcc --version`. The output indicates that gcc version 7.5.0 is running on an Ubuntu 18.04 system. It includes the standard copyright notice from the Free Software Foundation.

```
youhs@youhs-VirtualBox:/$ gcc --version
gcc (Ubuntu 7.5.0-3ubuntu1-18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

youhs@youhs-VirtualBox:/$
```

Screenshot 2: The terminal shows the output of the command `cat /proc/cpuinfo`. This command displays detailed information about the system's processor, including its model name (Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz), stepping, MHz, cache size, and various flags.

```
youhs@youhs-VirtualBox:/$ cd proc
youhs@youhs-VirtualBox:/proc$ cat cpuinfo
processor       : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 61
model name     : Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz
stepping        : 4
cpu MHz        : 1995.380
cache size     : 3072 KB
physical id    : 0
siblings        : 1
core id        : 0
cpu cores      : 1
apicid         : 0
initial apicid : 0
fpu             : yes
fpu_exception   : yes
cpuid level    : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_goo
```

Screenshot 3: The terminal shows the output of the command `cat /proc/meminfo | grep MemTotal`. This command outputs the total amount of memory available to the system, which is 4030904 kB.

```
youhs@youhs-VirtualBox:/$ cat /proc/meminfo | grep MemTotal
MemTotal:        4030904 kB
youhs@youhs-VirtualBox:/$
```

4. 과제 수행 중 발생한 애로사항 및 해결방법

4-1 동적 할당을 하지 않았을 때 오류

문제점 : program1을 코딩할 때 처음에는 아래의 사진과 같이 int filter[fnum][3][frow][fcolumn]으로 선언 후 동적 할당 없이 포인터를 이용하여 자리를 정해준 후 데이터를 대입하였습니다.

<예시 Filter 배열 선언과정>

```
gettimeofday(&start, NULL);
long militime;
long microtime;
cin >> fnum; cin >> frow; cin >> fcolumn;
int filter [fnum][3][frow][fcolumn];

int read = 0;
```

<예시 calculate함수 만들 시 포인터로 위치 설정과정>



```
newoutput98 x program2.cpp x program1.cpp x
}

//read input the data for filter array

void calculate(int * ex_filter,int * ex_data,int * ex_result){
    for(int d=0;d<fnum;d++){
        for(int a=0;a<3;a++){
            for(int b=0;b<frow;b++){
                for(int c=0;c<rcolumn;c++){
                    for(int t=0;t<frow;t++){
                        for(int k=0;k<fcolumn;k++){
                            if(t==0 && k==0){
                                *(ex_result+(d*3*frow*rcolumn)+(a*frow*rcolumn)+(b*rcolumn)+c)=*(ex_data+(a*(drow+2)*(dcolumn+2))+((b+t)*(dcolumn+2))+(c+k))*(*(ex_filter+(d*3*frow*fcolumn)+(a*frow*fcolumn)+(t*fcolumn)+k));
                            }
                            else{
                                *(ex_result+(d*3*frow*rcolumn)+(a*frow*rcolumn)+(b*rcolumn)+c)=*(ex_result+(d*3*frow*rcolumn)+(a*frow*rcolumn)+(b*frow*fcolumn)+(c+k))*(*(ex_filter+(d*3*frow*fcolumn)+(a*frow*fcolumn)+(b*frow*fcolumn)+(c+k)));
                            }
                        }
                    }
                }
            }
        }
    }
    //calculating filter and data and put the result into result array
}

int main(){
    struct timeval start,end;
    gettimeofday(&start, NULL);
}
```

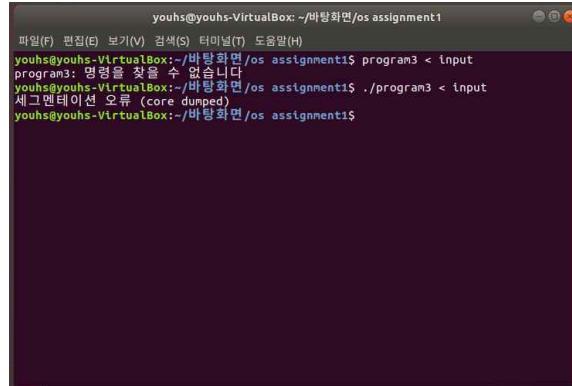
그 결과 Input 파일의 용량이 작아 데이터가 적을 경우에는 문제 없이 실행이 되었지만 자체적으로 만든

Filter [filter_num : 1000, filter_row : 100, filter_column : 100]

Data [data_row: 200 data_column: 200]

과 같이 큰 Input 파일을 실행했을 경우에는 아래와 같은 오류가 발생하게 되었습니다.

<예시 오류 출력과정>



A screenshot of a terminal window titled "youhs@youhs-VirtualBox: ~/바탕화면/os assignment1". The window shows the following command and its output:

```
youhs@youhs-VirtualBox:~/바탕화면/os assignment1$ program3 < input
program3: 명령을 찾을 수 없습니다
youhs@youhs-VirtualBox:~/바탕화면/os assignment1$ ./program3 < input
세그멘테이션 오류 (core dumped)
youhs@youhs-VirtualBox:~/바탕화면/os assignment1$
```

해결방안 : 위에서 프로그램들을 설명했던 거처럼 이러한 문제점은 모든 배열들을 동적 할당해주는 것으로 해결을 완료하였습니다. Input 파일이 커져서 배열 안에 포인터들 위치 값이 일정 크기를 초과하면 오류가 뜨는 것으로 확인하였습니다. 그리하여 작은 파일에서는 오류가 없었고 큰 파일에서만 문제가 생겼던 것이었습니다.

4-2 Process wait를 해주지 않아 생긴 문제점

문제점 : Program2를 multi-process를 이용하여 코딩하던 중 Process wait를 제대로 해주지 않아 결과 값들이 순서대로 나오지 않고 완료가 되는 순서대로 나오는 문제점이었습니다. 또한 wait 위치를 잘못 지정하여 process가 동시에 되지 않고 한 개가 끝나야지만 실행되었습니다.

해결방안 : 아래의 사진과 같이 wait를 for문을 통해서 모두 완료가 되게 해줌으로써 문제를 해결 할 수 있었습니다. 또한 위치 역시 적절하게 재설정해주었습니다.

<Process wait 해주는 과정>

```
for(int k=0;k<div;k++){
    int state;
    waitpid(pid[k],&state,0);
}
```

5. 결과화면과 결과에 대한 토의내용

결과화면과 결과에 대한 토의를 하기 전에 제 pc에 CPU 코어의 개수는 1개뿐이므로 Multi-process 및 Multi-threading을 하기에 적합하지 않다고 생각이 들어 CPU 코어의 개수가 2개인 동기의 PC에서 같은 파일을 돌려 비교 분석할 예정입니다.

또한 결과를 비교 분석하기 위해 같은 Input 파일을 사용할 예정입니다. Input 파일의 주어진 값은 Filter(10, 50, 50) Data(500, 500)입니다.

5-1 program1에 대한 결과값

<본인의 pc에서 돌린 program1>

<동기 pc에서 돌린 program1>

- > 코어 개수가 1개인 제 PC에서 돌렸을 때 걸린 시간은 278063 millisecond였고 코어 개수가 2개인 동기 PC에서 돌렸을 때 걸린 시간은 145343 millisecond였습니다.

5-2 program2에 대한 결과값

<본인의 pc에서 돌린 program2 (process_num : 1)>

<본인의 pc에서 돌린 program2 (process_num : 2)>

<본인의 pc에서 돌린 program2 (process_num : 4)>

<동기 pc에서 돌린 program2 (process_num : 4)>

- > 코어 개수가 1개인 제 PC에서 돌렸을 때 걸린 시간은 process_num에 따라
process_num (1개일 때) : 234523 millisecond
process_num (2개일 때) : 255119 millisecond
process_num (4개일 때) : 266255 millisecond 이 걸렸으며
코어 개수가 2개인 동기 PC에서 돌렸을 때 걸린 시간은 96134 millisecond
였습니다.

5-3 program3에 대한 결과값

<본인의 pc에서 돌린 program3 (thread_num : 1)>

<본인의 pc에서 돌린 program3 (thread_num : 2)>

<본인의 pc에서 돌린 program3 (thread_num : 4)>

<동기 pc에서 돌린 program3 (thread_num : 4)>

- > 코어 개수가 1개인 제 PC에서 돌렸을 때 걸린 시간은 thread_num에 따라
thread_num (1개일 때) : 236444 millisecond
thread_num (2개일 때) : 229795 millisecond
thread_num (4개일 때) : 306965 millisecond 이 걸렸으며
코어 개수가 2개인 동기 PC에서 돌렸을 때 걸린 시간은 150005 millisecond
였습니다.

5-4 결과 분석

1. 코어에 따른 소요시간 비교 (Input 위와 동일)

(program1)

program1 소요시간 (단위 :millisecond)	코어 1개	코어 2개
기본	278063	145343

-> 코어 1개와 2개일 때 걸리는 시간은 위에 보다시피 약 2배가량 차이가 나고 코어 2개일 때의 속도가 월등히 높은 걸 확인 할 수 있습니다.

(program2)

program2 소요시간 (단위 :millisecond)	코어 1개	코어 2개
process 1개 (각각의 process 소요시간)	234523 (231627)	x
process 2개 (각각의 process 소요시간)	255119 2413355 251817	x
process 4개 (각각의 process 소요시간)	266255 (261874 261957 211907 211735)	96134 (93696 93595 75732 75646)

-> program1에서 비교했을 땐 코어 1개와 2개일 때 2배가량 차이가 났지만 process 4개로 했을 때 약 3배가량이 차이가 났습니다. 즉 코어가 2개일 때 process에 따른 속도 증가율이 높은 것을 확인할 수 있습니다. 또한 코어가 1개 일 때는 오히려 process의 개수가 늘어남에 따라 속도가 느려짐을 확인 할 수 있는데 이는 코어가 1개뿐이라 일어나는 process switching이 원인으로 추측하였습니다. 하지만 큰 속도의 차이점을 관찰할 순 없었습니다.

(program3)

program3 소요시간 (단위 :millisecond)	코어 1개	코어 2개
thread 1개 (각각의 thread 소요시간)	236444 (233850)	x
thread 2개 (각각의 thread 소요시간)	229795 (227362 227254)	x
thread 4개 (각각의 thread 소요시간)	306965 (299731 299917 234506 235310)	150005 (147924 145306 114455 114320)

-> program1에서 비교했을 땐 코어 1개와 2개일 때 2배가량 차이가 process 4개로 했을 때 여전히 2배가량이 차이가 났습니다. 즉 process와 다르게 thread는 코어 개수에 따른 속도 증가율이 크게 다르지 않다는 것을 확인할 수 있습니다. 또한 코어가 1개일 때는 오히려 thread의 개수가 늘어남에 따라 2개일 때는 속도가 빨라짐을 확인 할 수 있었지만 4개일 때는 오히려 속도가 감소하는 현상이 일어났

습니다. 이는 해당 프로그램을 돌리는 시간에 다른 컴퓨터 작업 때문에 일어나는 속도의 차이라고 추측하였습니다.

2. 기본 프로그램과 멀티프로세스 멀티스레딩 비교 (Input 위와 동일)

(코어 1개일 때)

소요시간 (단위 :millisecond)	기본 프로그램	멀티프로세스 (process num 4개)	멀티스레딩 (thread num 4개)
코어 1개	278063	266255	306965

(코어 2개일 때)

소요시간 (단위 :millisecond)	기본 프로그램	멀티프로세스 (process num 4개)	멀티스레딩 (thread num 4개)
코어 2개	145343	96134	150005

-> 코어 1개일 때와 2개일 때 모두 멀티프로세스를 할 때 시간이 줄어드는 것을 확인 할 수 있고 멀티스레딩을 할 때에는 시간이 늘어남을 확인 할 수 있다.
또한 위에서 언급했다시피 코어가 2개일 때의 멀티프로세스의 감소량은 코어가 1개일 때에 비해 눈에 띄게 변화함을 알 수 있습니다.
멀티스레딩에서는 Input 파일의 크기가 충분하지 않아서 오히려 스레드를 생성하고 기다리는 과정에서 더 많은 시간이 소요 됐을거라 추측하였고 코어의 개수 역시 영향을 주었음을 추측할 수 있습니다.

3. Input 크기에 따른 성능 비교 (위의 Input과 다른 Input도 사용)

기존에 사용하던 Filter(10, 50, 50) Data(500, 500)과 Input 파일의 크기에 따른 성능을 비교하기 위해 Filter(50, 50, 50) Data(500, 500)으로 프로그램을 실행시켜봤습니다.

->새로운 Input 파일에서는 Program1은 1281513millisecond, Program2은 1305073millisecond, Program3은 1205333millisecond이 걸렸습니다.

<본인의 pc에서 돌린 program1(New Input)>

<본인의 pc에서 돌린 program2 (process_num : 4)(New Input)>

<본인의 pc에서 돌린 program3 (thread_num : 4)(New Input)>

<코어 1개>

소요시간 (단위 :millisecond)	기본 프로그램	멀티프로세스 (process num 4개)	멀티스레딩 (thread num 4개)
Input 파일 Filter (10,50,50) Data (500,500)	278063	266255	306965
New Input 파일 Filter (10,50,50) Data (500,500)	1281513	1305073	1205333

-> Input 파일의 사이즈를 다르게 하여 측정을 했을 때 역시 기본 프로그램과 멀티 프로세스 멀티스레딩을 했을 때 큰 차이점은 찾기 어려웠습니다. 이는 앞에서 쭉 설명했다시피 코어 개수가 1개인 점이 크게 작동한거 같습니다. 그러나 사이즈가 커짐에 따라 소요되는 시간이 커져 시간의 변화의 폭은 증가한 것을 확인 할 수 있었습니다.

즉 코어가 1개일 때에는 기본 프로그램과 멀티프로세스 멀티스레딩에 따른 큰 변화를 기대하기 어렵고 시간 역시 큰 차이가 나지 않는 것을 확인 할 수 있었습니다. 부득이하게 코어가 1개인 노트북으로 과제를 수행함으로써 더 많은 데이터를 비교하고 분석하여도 큰 차이가 나지 않는 것을 확인할 수 있었습니다. 이에 코어가 2개인 다른 노트북과의 비교 분석을 통해 멀티프로세스와 멀티스레딩의 변화되는 결과를 볼 수 있었고 만약 코어가 더 커지면 멀티프로세스와 멀티스레딩의 결과 역시 더 달라질 것을 예측할 수 있었습니다. 더불어 Input 값이 커질수록 더 효율적으로 계산할 수 있을 거라는 예측 역시 할 수 있었습니다.