Homework2 사전 조사 보고서

2018147558_김정주

A. 프로세스와 스레드

1. Process : 실행 중인 프로그램의 인스턴스
    - 구성 : 독자적인 자료구조인 PCB로 관리
        - Image : code, data, stack, heap
        - Process context : data register, program counter, stack pointer
        - Kernel context
    - 구현(리눅스)
        - task_struct(image: mm, program context : thread, kernel context: other)
2. Thread : 프로세스 내에서 실행되고 있는 일련의 명령어
    - 구성 : process에서 공유하는 자원을 제외한 나머지
        - Program counter
        - Stack, stack pointer
        - Data register
    - 구현(리눅스)
        - One-to-one 모델
        - Clone()이라는 경량 프로세스를 만드는 명령어를 통해 지원
        - Pthread 지원
3. Process와 Thread의 차이
    - Process
        - 정적인 독립체
        - Data를 내부적으로 공유하지 않는다.(별도의 IPC 필요)
        - 생성과 전환의 비용이 높다.
    - Thread

- 동적인 독립체
- Data의 일부분을 공유
- 생성과 전환의 비용이 낮다.

B. 멀티 프로세싱과 멀티 스레딩

1. 멀티 프로세싱 : 하나의 작업을 여러 프로세스로 실행
    - 구현(리눅스)
        - Fork()를 통해 먼저 프로세스를 복제하고 exec()로 새로운 코드를 로드하는 것으로 새로운 프로세스를 생성하지 않고, 프로세스의 속성을 상속하여 그래도 유지시키는 방식
2. 멀티 스레딩 : 하나의 프로세스를 여러 스레드로 실행
    - 구현(리눅스)
        - Pthread API를 지원함으로, pthread_create로 스레드를 생성하고, 스레드 함수를 실행시킨다. 실행이 끝나면 pthread_exit로 스레드를 종료한다. 만약 종류 된 후 스레드의 종료를 확인하고 싶은 경우 pthread_join을 통해 컨트롤 한다.

C. exec() 계열 함수

1. int execl(const char *path, const char *arg0, ..., const char *argn, NULL)
    - Path에 파일을 매개변수 arg0 – argn을 인자로 실행
    - 마지막 매개변수는 인자의 끝을 알리는 NULL로 지정
2. int execv(const char *path, char *const argv[])
    - 포인터의 배열인 argv를 인자로 전달하여 실행, 배열의 마지막은 NULL
3. int execle(const char *path, const char *arg0, …, const char *argn, NULL, char *const envp[])
    - 포인터의 배열 envp는 새로운 환경 변수 설정을 위해 존재
4. int execve(const char *path, char *const argv[], char *const envp[])
    - 포인터의 배열 argv와 envp를 전달
5. int execlp(const char *file, const char *arg0, …, const char *argn, NULL)

- file에서 지정한 파일을 매개변수 arg0-argn을 인자로 실행

6. int execvp(const char *file, char *const argv[])
    - file에 지정한 파일을 argv를 인자로 실행

D. 참조문헌

1. https://docs.oracle.com/cd/E36784_01/html/E36872/exec-2.html#scrolltoc
2. https://velog.io/@gyrbs22/%EC%A0%84%EC%82%B0%ED%95%99-%EB%A6%AC%EB%88%85%EC%8A%A4-exec-%ED%95%A8%EC%88%98-IPC%EA%B8%B0%EB%B2%95-Copy-on-Write-mmap-%ED%95%A8%EC%88%98
3. https://probe29.tistory.com/39
4. Operating System Concepts, 10th edition, Abraham silberschatz

Homework2 결과 보고서

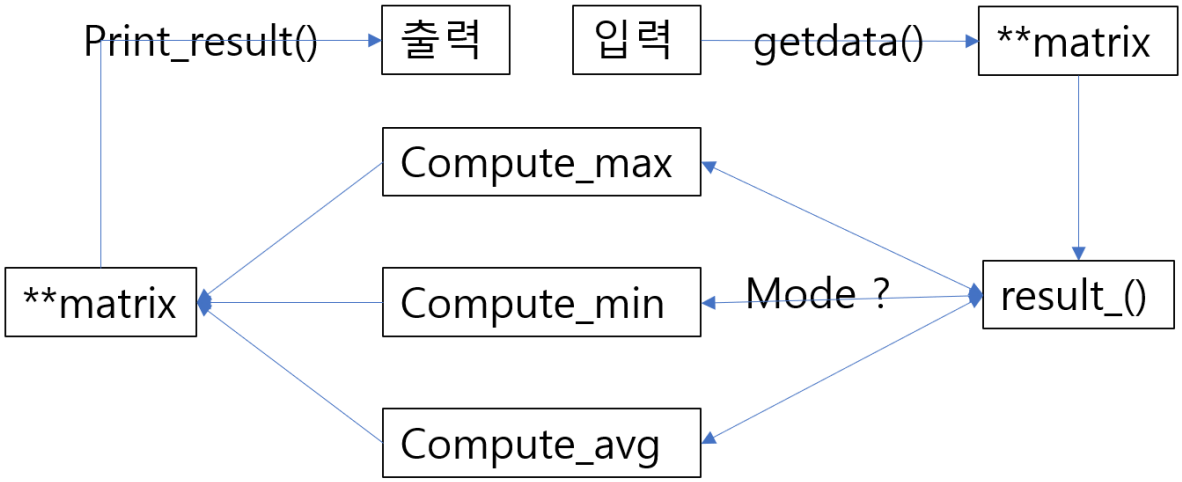
2018147558. 김정주

1. 프로그램 스펙

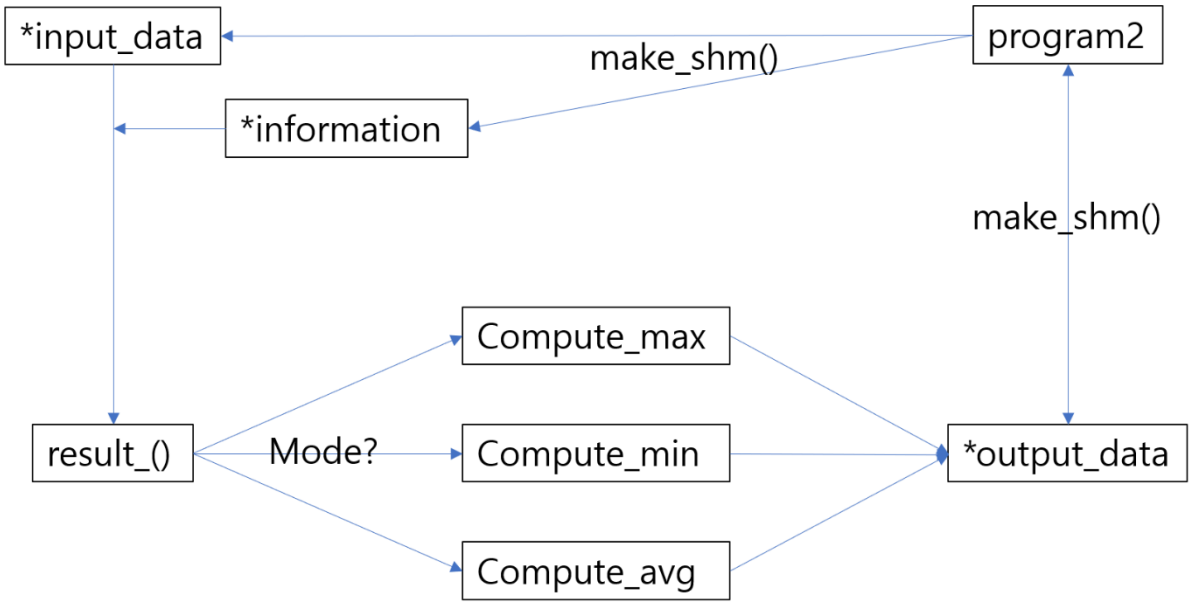
   A. Program1

      i. 프로그램 도식도

      - Kernel이 프로그램을 실행시켰을 때



      - Program2가 호출하였을 때



ii. 프로그램 구조

- 변수
  - **matrix : input 데이터를 저장하기 위한 변수, 데이터가 행렬이기 때문에 이중 포인터를 써서 데이터의 입력을 저장했다. 추가로 input의 입력을 받아오는 getdata()함수를 통해 입력을 받아오고 결과값을 해당 인덱스에 맞게 저장하기 위한 변수. 이는 input 데이터가 한번 쓰이고 다시는 쓰이지 않기에 이 같은 방식을 취했다.
  - *input_data : 공유 메모리에서 input data를 담고 있는 영역으로 program2에 의해 불렸을 때, 데이터를 가져오는 필드이다.
  - *output_data : 공유 메모리면서 결과 데이터를 저장하는 필드이다.
  - *information : 공유 메모리이며, 입력데이터의 행, 열, 커널 크기의 정보를 가져오기 위한 변수로 쓰인다.
- 함수
  - Print_result() : 커널에 의해서 직접 불렸을 때, 결과값을 출력하는 함수이다.
  - Make_shm(key_t key) : key값에 맞게 공유 메모리를 만들고 해당 공유메모리의 세그먼트 포인터 값을 리턴한다. 공유메모리 크기의 경우, program2에서 이미 설정해주었기에 설정해줄 필요는 없다.
  - Mat_del(int* shm) : 공유메모리의 세그먼트 포인터를 받아서 해당 세그먼트 포인터의 메모리 링크를 해제하는 역할을 수행한다.
  - Compute_max(int l, int k) or Compute_max(int l, int k, int s): 연산을 하는 필드에서 인덱스 l, k의 max값을 계산하여 해당값을 반환하는 함수이다. Int s는 쓰이는 매개변수는 아니지만 함수를 나눠주기 위해서 사용하며 int s가 있는 함수는 input값을 *input_data에서 읽어들여 계산한다.
  - Compute_avg(int l, int k) or Compute_avg(int l, int k, int s): 연산을 하는 필드에서 인덱스 l, k번째의 평균값을 계산하여 해당값을 반환하는 함수이다. . Int s는 쓰이는 매개변수는 아니지만 함수를 나눠주기 위해서 사용하며 int s가 있는 함수는 input값을 *input_data에서 읽어들여 계산한다.
  - Compute_min(int l, int k) or Compute_min(int l, int k, int s): 연산을

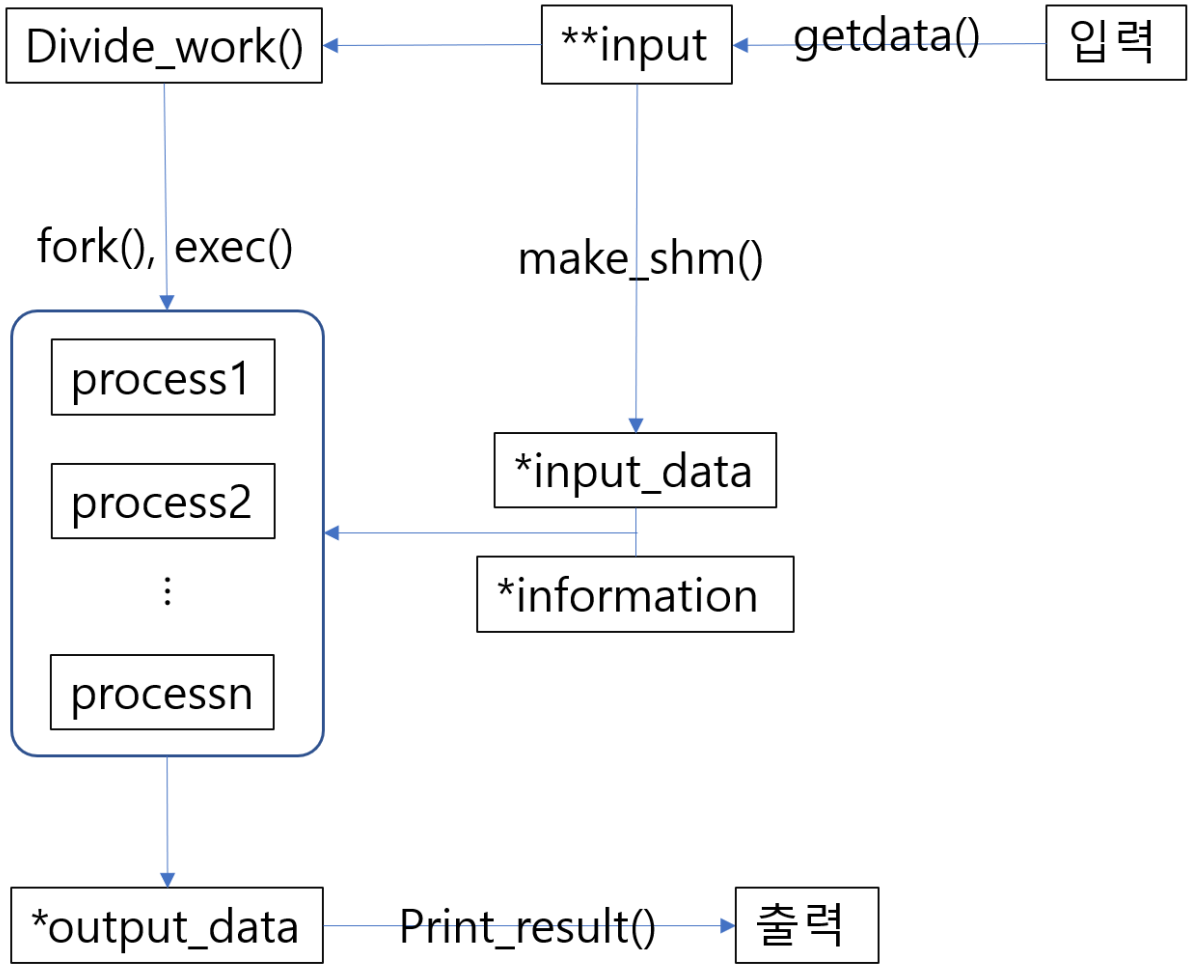
하는 필드에서 인덱스 I, k의 min값을 계산하여 해당값을 반환하는 함수이다. . Int s는 쓰이는 매개변수는 아니지만 함수를 나눠주기 위해서 사용하며 int s가 있는 함수는 input값을 *input_data에서 읽어들여 계산한다.

- Result_(int mode) : 커널에서 프로그램1이 직접 실행됐을 때 쓰는 함수로 mode의 값으로 min, avg, max의 계산을 결정하여 결과값을 **matrix쓰는 방식으로 구현된다.
- Result_(int mode, int i, int j, int work) : 공통적으로 쓰이는 함수로 결과 값의 계산을 전달하고, 결과 값에 저장하는 함수이다. Mode는 min, avg, max를 정하는 값이고, work로 주어진 숫자만큼만 연산을 수행한다. 그리고 I,j는 수행을 마치고 결과값에 데이터를 입력하는 좌표이다.

- main함수
  - 이 프로그램은 kernel에 의해 직접 불리때와 프로그램2에 의해 불렸을 때를 나누어서 처리를 해줘야 함으로, if문을 사용해서 매개변수의 개수에 차이를 두어 이를 control해주었다.
  - 커널에 의해 직접 불릴 때는, input을 직접 받고, mode에 따라 연산을 달리하여 결과값만 출력해주면 끝이다.
  - 프로그램2에 의해서 불렸을 경우, 인자로 총 처리해야 하는 일의 수를 나타낸 worknum, 일을 해야하는 input데이터의 index값을 subrow와 subcol에 각각 받는다. 그리고 공유 메모리를 위한 key값을 설정하고 해당 key값으로 공유메모리를 생성하여 데이터를 읽고 맞는 연산을 수행해서 다시 공유메모리에 결과값을 작성하는 방식으로 수행된다.

B. Program2

i. 프로그램 도식도



ii. 프로그램 구조

- 변수
  - **input : input 데이터를 저장하기 위한 변수, 데이터가 행렬이기 때문에 이중 포인터를 써서 데이터의 입력을 저장했다. 추가로 input의 입력을 받아오는 getdata()함수를 통해 입력을 받아온다.
  - *input_data : 공유 메모리에서 input data를 담고 있는 필드이다.
  - *output_data : 공유 메모리면서 결과 데이터를 저장하는 필드이다.
  - *information : 공유 메모리이며, 입력데이터의 행, 열, 커널 크기의 프로세스에 전달하기 위해 쓰인다.
  - Maxnumcp : 계산을 하는 최대 숫자를 위한 변수이다.
  - **workindex : 각 프로세스가 진행할 input의 index를 담고 있는 데

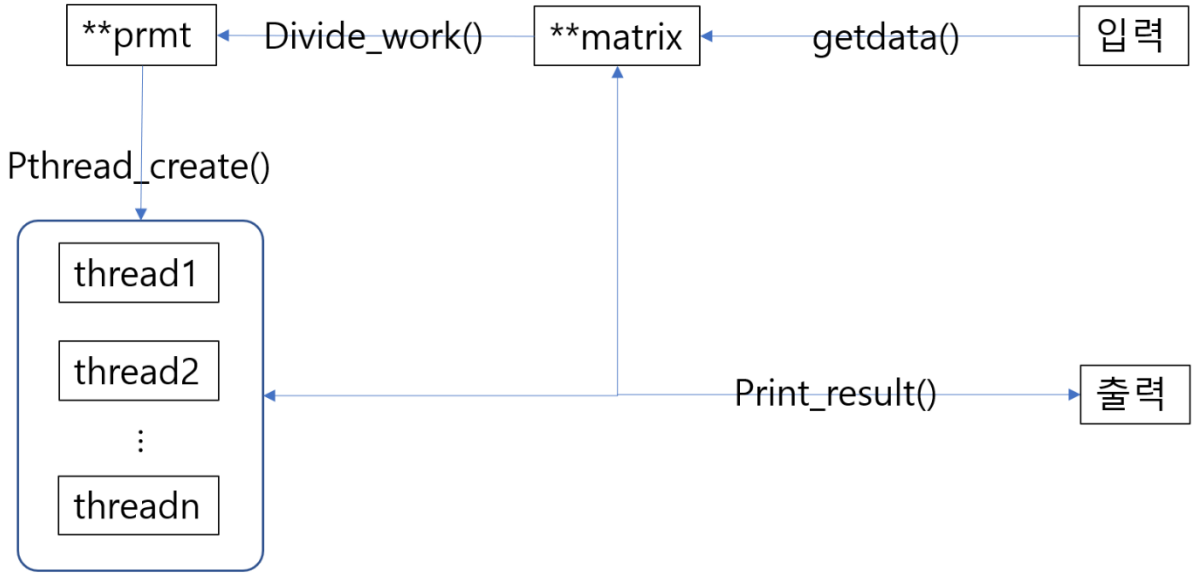
이터이다.

- MaxProcess : 최대 프로세스 개수를 담고 있는 변수이다.
- *numperpro : 프로세스 마다의 계산을 해야하는 숫자를 담고 있는 변수이다.

- 함수
  - Divide_work() : input의 필드를 쪼개서 각 프로세스 별로 수행해야 하는 계산의 수와 index를 나눈다.
  - Print_result() : 결과값을 출력하는 함수
  - make_shm(key_t key, int size) : 공유 메모리를 만들고, 해당 공유 메모리의 ID를 리턴한다.
  - Make_shm_sg(int shm) : 공유 메모리의 ID를 제공하고 해당 ID의 메모리 세그먼트를 리턴하는 함수이다.
  - Del_shm(int shm) : 공유 메모리의 ID를 받아 해당 공유메모리를 삭제하는 함수이다.
- Main 함수
  - 먼저 MacProcess에 인자로 주어진 프로세스 수를 저장하고, 프로세스의 개수만큼 프로세스를 생성한다.
  - 입력 데이터를 전달하는 공유메모리와 결과 데이터를 받는 공유메모리 그리고 행, 열, kernel의 크기를 담고있는 공유메모리르 각각 생성한다.
  - Getdata()함수를 호출하여 데이터를 받고, 해당 데이터를 입력 공유메모리에 저장하고, 각 프로세스 별로 일을 나눈다.
  - 나눠진 일에 대한 정보를 인자로 제공하고, 자식프로세스를 생성하고, 처리를 시작하고, 자식 프로세스가 모드 끝날 때까지 기다렸다가 모두 마치면, 결과를 담고있는 공유 메모리에서 데이터를 출력한다.

C. Program3

i. 프로그램 도식도



ii. 프로그램 구조

- 변수
  - **matrix : input 데이터를 저장하기 위한 변수, 데이터가 행렬이기 때문에 이중 포인터를 써서 데이터의 입력을 저장했다. 추가로 input의 입력을 받아오는 getdata()함수를 통해 입력을 받아온다. 그리고 결과값을 저장하는 행렬이기도 하다. 이는 input 데이터가 한번 쓰이고 그 이상 쓰이지 않기 때문에 이런 방식을 취했다.
  - Maxnumcp : 계산을 하는 최대 숫자를 위한 변수이다.
  - **workindex : 각 스레드가 진행할 input의 index를 담고 있는 데이터이다.
  - MaxThread : 최대 스레드 개수를 담고 있는 변수이다.
  - *numperpro : 스레드 마다의 계산을 해야하는 숫자를 담고 있는 변수이다.
  - Prmt : 스레드에 줄 인자를 담고 있는 변수이다.
- 함수
  - Divide_work() : input의 필드를 쪼개서 각 프로세스 별로 수행해야 하는 계산의 수와 index를 나눈다.

- Print_result() : 결과값을 출력하는 함수
- Compute_max(int l, int k) : 연산을 하는 필드에서 k번째의 max값을 계산하여 해당값을 반환하는 함수이다.
- Compute_avg(int l, int k) : 연산을 하는 필드에서 k번째의 평균값을 계산하여 해당값을 반환하는 함수이다.
- Compute_min(int l, int k) : 연산을 하는 필드에서 k번째의 min값을 계산하여 해당값을 반환하는 함수이다.
- Calculate_result(void* paramt) : 함수로 결과 값의 계산을 스레드에 전달하고, 결과 값에 저장하는 함수이다. Paramt는 해당 스레드가 연산을 함에 필요한 정보를 담고 있는 void* 변수이고, 해당 인자를 받아서 연산을 수행한다.

- main함수
  - 프로그램1과 비슷하다. 하지만 각 스레드에 일을 적절히 분배함에서 조금 달라진다.
  - Divide_work()로 주어진 일을 스레드 개수에 맞게 나눈다.
  - 나눠진 일을 prmt에 저장하여 각 스레드을 만들 때 인자로 부여한다.
  - Pthread_create를 최대 스레드 수에 맞춰 반복해서 호출해주면 각각의 일을 부여받은 스레드가 생성된다.
  - 생성된 스레드를 pthread_join을 통해 모든 스레드가 일을 마칠 때까지 대기하여 모든 일을 맞추면, 결과 데이터를 출력하여 프로그램은 종료된다.

D. Makefile

i. program1 program2 program3:

```
g++ -o program1 program1.cpp

g++ -o program2 program2.cpp

g++ -pthread -o program3 program3.cpp
```

- program1, program2, program3을 모두 생성한다. 생성은 -o로 하여, 바로 프로그램이 생성될 수 있게 하였다.

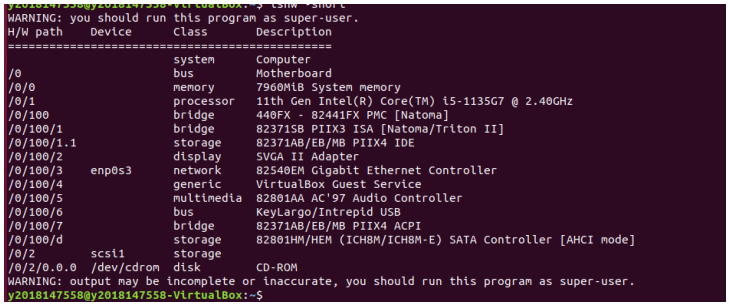
- Pthread를 사용하는 program3의 경우는 -pthread를 명시함으로 라이브러리를 사용한다는 것을 알려 컴파일 에러가 나지 않도록 하였다.

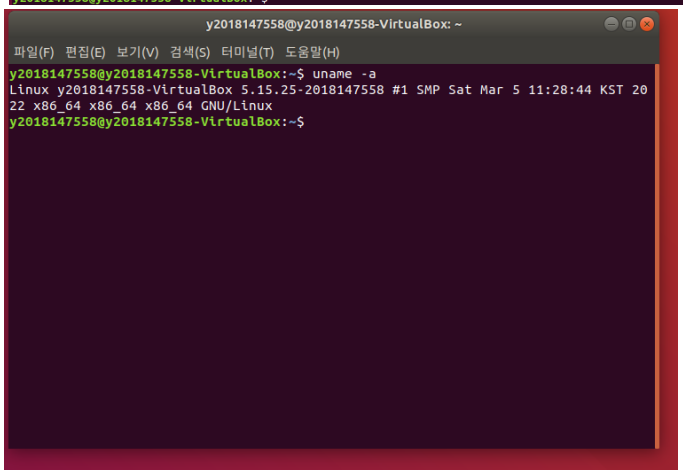
ii. Clean:

Rm program1 program2 program3

- Make clean 명령어를 입력하면 프로그램들이 모두 삭제된다.

2. 개발환경





- 프로세스는 4코어, 메모리용량은 7960MiB이다.
- 컴파일러는 g++ 9.5.0을 사용하였다.

3. 과제 중 발생한 애로사항 및 해결법

A. Shared memory access error

i. 문제 : 프로그램1에서 공유메모리를 사용함에 있어서 어떤 실행 경우에는 데이터가 올바르게 나오지만 어떤 실행경우에는 데이터의 오류와 공유메모리 접근 오류가 발생했다.

ii. 원인 : 공유메모리를 사용하고 나서 공유메모리에 대한 링크 해제 및 삭제를 해주지 않아서 오류가 발생

iii. 해결 : 공유메모리를 모두 사용 후에 shmctl()함수를 사용하여 공유메모리의 삭제

B. Killed

i. 문제 : 프로그램2를 돌리는 데 있어서 testcase를 10000*10000을 하는데, 발생한 error, killed라는 문자가 뜨고 곧바로 프로세스 종료

ii. 원인 : dmesg | grep -E -I -B100 'killed process'를 통해 kill된 프로세스를 찾아보니 out of memory가 원인이었다. 스레드의 경우는 데이터를 공유함으로 이 같은 문제가 발생하지 않았으나 프로그램2의 각각의 프로세스가 데이터를 공유하지 않기에 이러한 문제가 발생하였다.

iii. 해결 : 명료한 해결책은 메모리를 늘리는 것이지만, 이 이상은 하드웨어의 스펙으로 인해 이상의 메모리를 부여하진 못했다. 그렇기에 결과값으로 사용하는 포 int포인터를 통합했고, 계산의 필드를 나누기 위해 존재했던 삼중 int 포인터의 변수도 삭제해서 다른 방식으로 구현하였다.

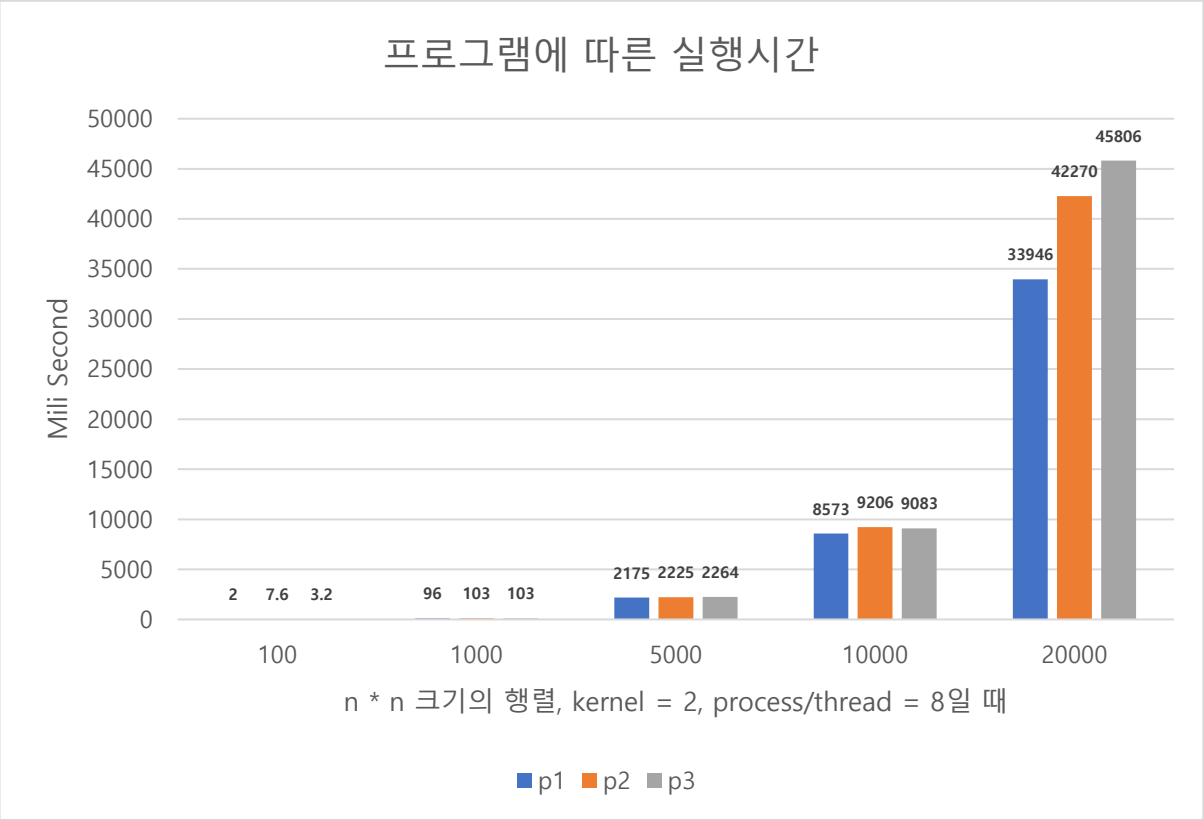
C. Pthread 컴파일

i. 문제 : 프로그램3를 컴파일하는 도중에 error가 발생하여 컴파일이 되지 않았다.

ii. 원인 : pthread를 사용하는 프로그램은 컴파일을 할 시 -pthread를 명시해줘야 하는데 이를 하지 않아서 발생

iii. 해결 : -pthread를 컴파일 할 때, 명시함으로서 해결

4. 결과 분석



- 먼저 처음부터 끝까지, 예상하던 값과는 상반되는 결과가 나온 것을 확인할 수 있다.
- 이에 구현문제, 컴파일 오류, 시스템 리소스 할당, 시간 측정 함수의 오류 등등 여러 요인들에 대해서 분석해본 결과 이상이 없는 것으로 결론지었다.
- 이러한 값이 나오게 된 이유는 false sharing때문이다. 캐싱 매커니즘을 채용하고 있는 멀티프로세싱 cpu에서 발생가능한 성능 저하 현상이다. Cpu의 캐시는 자주 사용될 가능성이 높은 데이터를 미리 저장해두고 사용한다. 여기서 직렬, 배치프로그램에서는 발생할 수 없는 문제가 멀티프로세싱에서는 발생한다. 이는 배열 구조의 데이터를 공유할 때 자주 발생하는데, 공유 데이터 배열 arr에 대해 x스레드가 arr[1,0] 변경하고 동시에 y스레드가 arr[0,1]을 변경한다고 가정하면 서로 같은 주소값을 공유하지만 캐시 내부의 값이 동기화가 맞지 않는 상태인 false sharing 상태를 가지게 된다. 이대로 데이터를 사용하면 데이터의 값이 이상하게 저장되지만 시스템 자체적으로 이를 해결하기 위해 같은 공유 데이터를 사용하는 주소값의 데이터를 동기화 해주는데, 여기서 성능저하가 발생하여 프로그램1보다 프로그램2, 3의 경우가 더 느려지게 되는 것이다. 내가 작성한 프로그램의 경우 *output_data가 공유 데이터 배열이며 구현 방식도 이 공유 데이터에 n개의 프로세스/스레드가 접근해서 각각의 데이터를 입력하는 방식을 취하기에 false sharing 현상이 발생하기에 필요충분조건이라고 사료된다.
- 이러한 현상이 발생하지 않는다면, 행렬의 n의 개수가 적으면 3개의 프로그램의 실행시

간의 차이가 크게 발생하지 않을 것 같으나, n의 개수가 기하급수적으로 늘어나면 늘어날수록 멀티 스레드/멀티 프로세싱의 실행시간이 더 짧을 것이라고 예상된다. 그리고 스레드 간의 전환이나 생성시의 비용, 시간보다 프로세스 전환 생성시의 비용, 시간이 너 많이 들기에 멀티 프로세싱보다 멀티 스레딩이 더 빠를 것이라고 예측할 수 있다.

5. 참고문헌

https://popcorntree.tistory.com/m/84

https://12bme.tistory.com/287

https://reakwon.tistory.com/96

https://young-cow.tistory.com/12

https://reakwon.tistory.com/56

https://ironmask84.tistory.com/348

https://wiserloner.tistory.com/1277

https://blog.naver.com/hermet/68290454

https://hwan-shell.tistory.com/230

https://en.wikipedia.org/wiki/False_sharing#:~:text=In%20computer%20science%2C%20false%20sharing,managed%20by%20the%20caching%20mechanism.

https://docs.oracle.com/cd/E19205-01/819-5270/aewcy/index.html

https://hijuworld.tistory.com/1

https://namneul.tistory.com/10

https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=remocon33&logNo=221123614816

https://www.youtube.com/watch?v=f_YckatmNCU&ab_channel=intrigano