2022년 1학기 운영체제 과제 #2

최종 수정: 2022-04-02

과제 개요

프로세스(Process)는 실행중인 프로그램의 인스턴스로, 운영체제에서 동시에 여러 개가 실행된다. 이를 통해 우리는 여러 개의 프로그램을 거의 동시에 사용할 수 있다. 스레드(Thread)는 한 프로세스 내에서 여러 작업을 동시에 처리하기 위한 기법으로, 빠른 처리를 위해 널리 사용되는 기법이다. 일반적으로 **멀티 프로세스(Multi-process) 프로그램**은 fork와 exec 계열의 **시스템 콜(System call)**을 이용하여 구현되고, **멀티 스레딩(Multi-threading) 프로그램**은 pthread 함수를 통해 구현된다. 이 과제에서는 합성곱 신경망(Convolutional neural network; CNN)의 여러 레이어 중 하나인 **풀링 계층(Pooling layer)**의 기본적인 연산 과정을 멀티 프로세스와 멀티 스레딩 기법을 통해 구현한다. 마지막으로, **각 기법을 적용한 프로그램과 적용하지 않은 프로그램의 성능 차이를 비교 분석**한다.

1 과제 내용 및 목적

- ✓ fork()와 exec()등의 시스템 콜을 이해하고, 이를 이용하여 멀티 프로세스 프로그램을 구현한다.
- ✓ 스레드의 동작 원리를 이해하고, pthread 함수를 이용하여 멀티 스레딩 프로그램을 구현한다.
- ✓ 멀티 프로세스와 멀티 스레딩을 적용했을 때와 적용하지 않았을 때의 차이를 이해한다.

2 제출 기한

- ✓ **2020년 4월 20일(수) 오후 6:00 까지**

3 제출 방법

- ✓ 과제는 보고서 과제와 실습 과제로 이루어진다.
- ✓ 보고서 과제 파일 (pdf) 및 실습 과제 파일 (tar) 각각을 learnUs 과제 제출 게시판에 제출한다. **(제출 양식 반드시 엄수)**

4 제한 사항

- ✓ 운영체제는 리눅스를 사용한다. 리눅스 종류와 버전은 제한이 없다.
  - **채점 서버 OS: Ubuntu 20.04, 채점 서버 컴파일러: gcc 9.4.0, locale: UTF-8** (euc-kr 등 사용 금지)
- ✓ 프로그래밍 언어는 반드시 C 또는 C++를 사용한다.
  - **스레드 생성은 pthread만을, 프로세스 생성은 fork()만**을 사용해야한다.
  - 그 외 구현에서는 C 표준 라이브러리 또는 C++ 표준 라이브러리 (STL 포함)내에서 자유롭게 사용할 수 있다.
- ✓ 프로그램 구조
  - 제출물의 압축을 해제하면 생성되는 폴더 내에 Makefile과 구현한 프로그램들의 소스 코드가 존재해야 한다.
  - make 명령을 실행하면 각각의 소스 코드들이 컴파일 되어 `program1, program2, program3`라는 이름의 실행 가능한 바이너리가 생성되어야 한다. (터미널에서 `./program1`로 프로그램이 실행 됨)
  - **컴파일시 필요한 옵션(라이브러리, C++11 등)은 Makefile에 반드시 기재**하여 채점 서버에서 컴파일이 되도록 해야 한다.
  - make clean 명령을 실행하면 make를 통해 생성된 빌드 결과물을 삭제해야 한다.

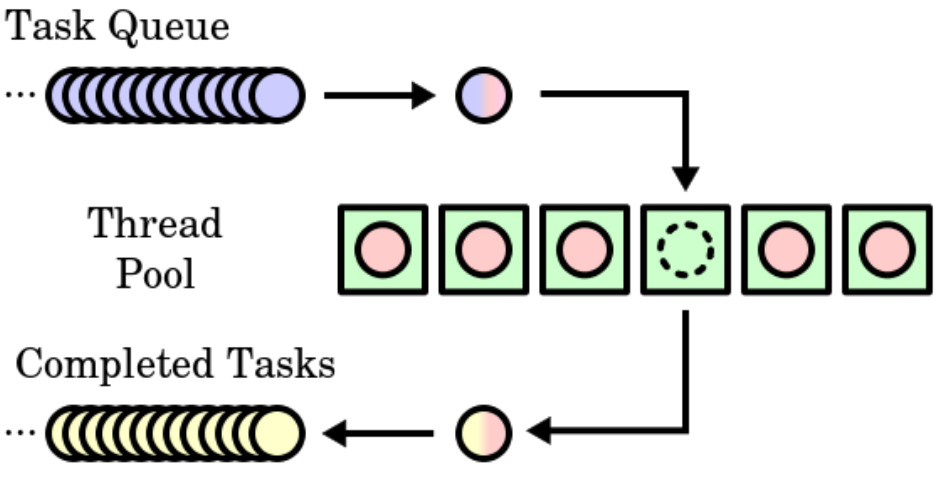
5 유의 사항

- ✓ **적절한 사유 없이 Delay 될 경우 절대 받지 않음**
- ✓ 그림, 코드 조각을 포함한 모든 참고 자료는 내용은 반드시 출처를 명시 (출처 명시 안했을 시 감점)
- ✓ 타 수강생의 보고서 및 과제를 카피 또는 수정하여 제출하였을 경우 모두 0점 처리
- ✓ 제출 파일 생성 방법과 프로그램 구조(특히 프로그램의 이름)를 지키지 않을 경우 감점 또는 0점 처리
- ✓ 제출한 소스에는 반드시 주석을 달 것, 주석의 내용이 불분명하거나 없을 경우 감점 처리
- ✓ 제출한 과제가 컴파일이 되지 않는 경우 0점 처리
- ✓ 과제에 대한 문의 사항이 있을 경우 learnUs의 Q&A게시판에 **'공개 게시글'**로 문의

Background

멀티 프로세스 (Multi-process)와 멀티 스레딩 (Multi-threading)

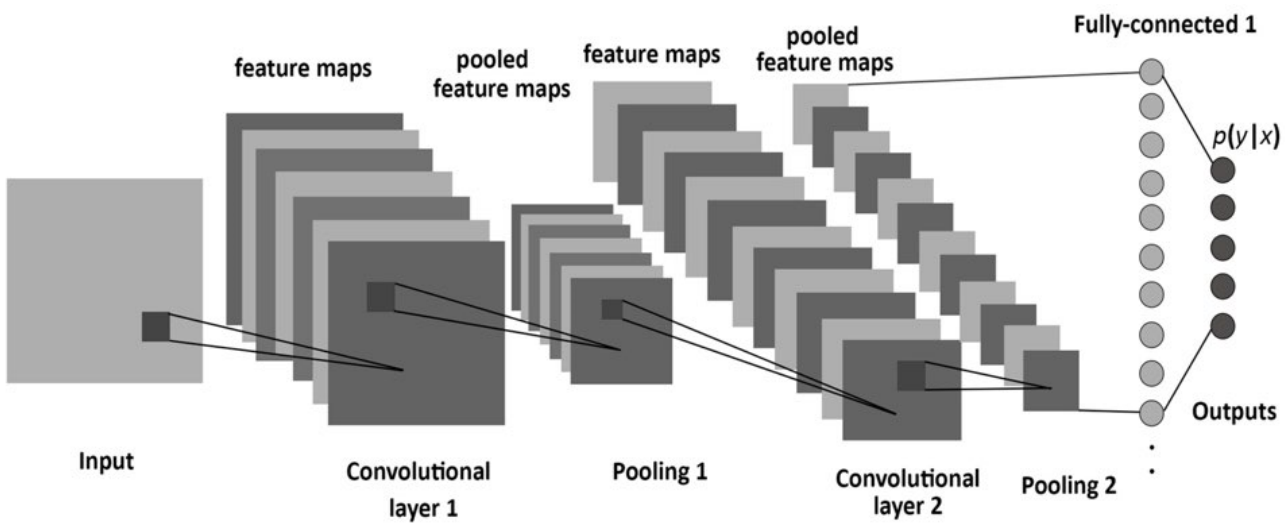
멀티 프로세스는 한 프로그램을 여러개의 프로세스로 쪼개어 각 프로세스가 각자의 작업을 동시(Concurrency)에 처리하도록 하는 기법이다. 구글 크롬(Google Chrome)과 같은 프로그램은 실제로 여러개의 프로세스로 동작하는 것을 확인할 수 있다. 주로 멀티 프로세스 기법은 한 프로세스의 결함이 다른 프로세스에 영향을 미치지 않도록 분리하기 위하거나, 연산 작업을 기다리지 않고 동시에 따로 실행하기 위하여 사용한다. 멀티 스레딩은 한 프로세스를 여러 스레드로 쪼개어 각 스레드가 각자의 작업을 동시에 처리하도록 하는 기법이다. 이 때, 스레드 풀(Thread Pool)을 만들어 필요한 작업을 비어있는 스레드에 맡기는 방법이 주로 사용된다.

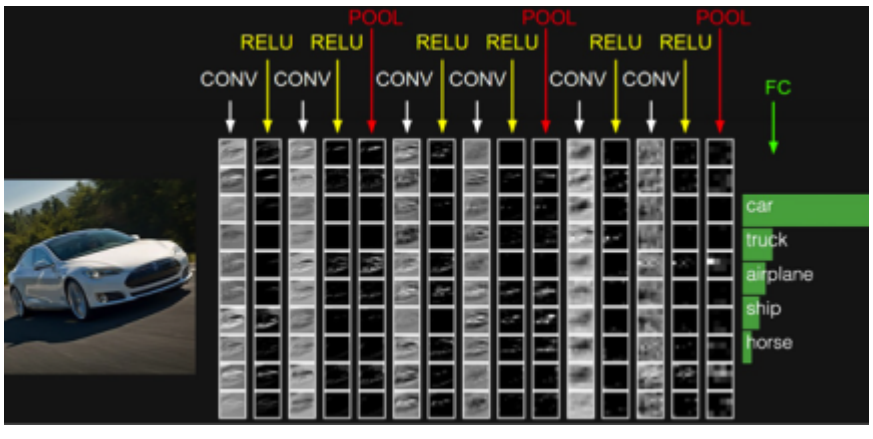


이번 과제에서는 프로세스 풀(Process Pool) 혹은 스레드 풀(Thread Pool)을 만든 다음, 주어지는 연산 작업을 적절히 프로세스 혹은 스레드에 분할하여 분할된 작업을 동시에 처리하도록 하는 것이 목표이다. 그리고 각각의 기법의 차이를 실행 시간 관점에서 비교 분석하는 것도 목표이다. CPU 환경(싱글 코어, 멀티 코어)에 따라서 결과의 차이가 발생할 수 있는데, 이를 확인해보는 것을 권장한다.

합성곱 신경망 (Convolutional Neural Network; CNN)

합성곱 신경망은 컴퓨터 비전를 비롯한 다양한 인공지능 분야에서 각광받고 있는 딥 러닝(Deep learning) 알고리즘 중 하나다. 실제 합성곱 신경망의 구조는 아래 그림과 같이 여러번의 합성곱(Convolution) 연산을 비롯한 다양한 계층(Layer)으로 이루어져있다. 합성곱 신경망을 이루는 계층의 종류로는 합성곱 계층(Convolution layer), 활성화 함수(Activation function), 풀링 계층(Pooling layer), 그리고 완전 연결 계층(Fully connected layer)등이 있다.





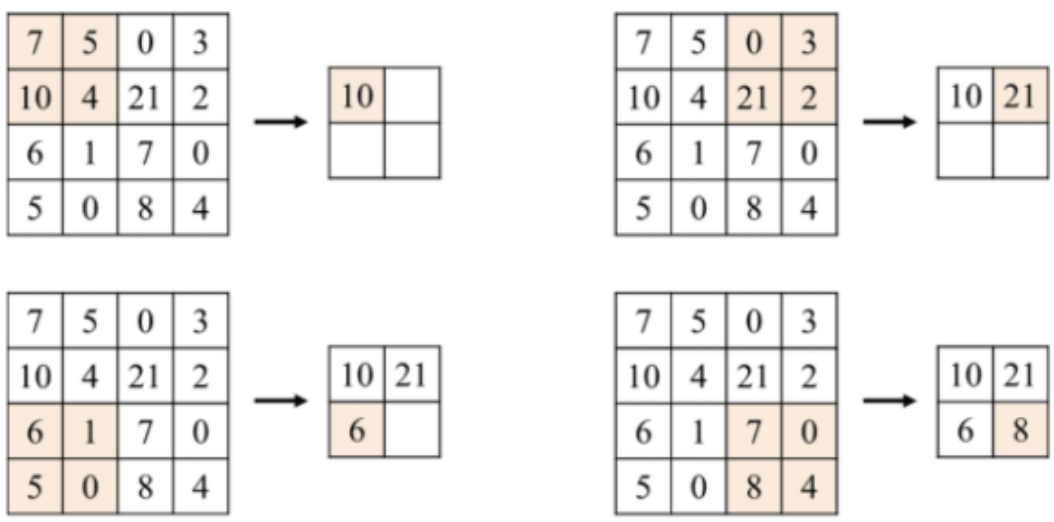
이번 과제에서는 합성곱 신경망을 이루는 다양한 계층 중 풀링 계층(Pooling layer)의 기본적인 연산만을 다룬다. 다른 계층은 이번 과제에서 다루지 않는다.

풀링 계층 (Pooling layer)

풀링 계층은 입력으로 주어지는 입력 데이터로 주어지는 특성 맵(Feature map)의 크기를 줄여주는 이른바 다운샘플링을 하는 과정이다. 풀링을 하는 방법으로는 크게 다음과 같이 세가지가 있다.

- 최대 풀링 (Max pooling)
- 평균 풀링 (Average pooling)
- 최소 풀링 (Min pooling)

합성곱 신경망에서는 주로 최대 풀링을 사용하는데, 이는 뉴런이 가장 강한 신호에 반응하는 것과 유사하기 때문이라고 한다. 최대 풀링은 각 영역에서 최댓값을 찾는 연산의 반복이다. 아래 그림들은 모두 **스트라이드(Stride) 값**이 2로 설정된 최대 풀링 과정을 나타낸다. 일반적으로 최댓값을 찾는 영역의 높이와 너비는 스트라이드 값과 동일하게 설정된다. 이번 과제에서도 이렇다고 가정한다. 즉, 최댓값을 찾는 각 영역의 크기는 2x2가 된다.



평균 풀링과 최소 풀링은 위의 최대 풀링 연산 과정에서 최댓값을 찾는 과정을 평균값 또는 최솟값으로 대체하면 된다.

과제 세부 사항

보고서 과제

보고서는 **사전 조사 보고서**와 **프로그래밍 수행 결과 보고서**로 구성한다.

- **사전 조사 보고서**
  - 프로세스와 스레드
    - 프로세스와 스레드의 차이점
    - 프로세스와 스레드의 리눅스에서의 구조 및 구현 등
  - 멀티 프로세스와 멀티 스레딩
    - 멀티 프로세스와 멀티 스레딩의 개념 및 구현 방법
  - exec 계열의 시스템 콜
    - exec 계열의 시스템 콜에는 어떤 것들이 있는지
    - 조사한 시스템 콜의 리눅스에서의 구현, 리눅스에서 동작하는 방법 등
  - 참조 문헌

- **프로그래밍 수행 결과 보고서**
  - 작성한 프로그램의 동작 과정과 구현 방법
    - 실습 과제에서 구현할 3개의 프로그램에 대해 각각 정리하여 서술
    - 순서도나 블록 다이어그램 등 도식화 자료를 반드시 이용할 것
  - 작성한 Makefile에 대한 설명
  - 개발 환경 명시
    - uname -a 실행 결과
    - 사용한 컴파일러 버전, **CPU (특히 몇 코어인지)**, 메모리 정보 등
  - 과제 수행 중 발생한 애로사항 및 해결방법
  - [중요] **결과 화면과 결과에 대한 토의 내용**
    - **실습 과제의 동작 검증**
      - **특히 기본 프로그램(program1), 멀티 프로세스(program2), 멀티 스레딩(program3)의 차이를 비교 분석**
    - 특히 성능 관점에서 어떤 결과가 나왔는지 설명
    - 결과에 대한 이유 분석
    - 그래프와 도표등을 반드시 활용하여 결과 분석을 성의있게 할 것

실습 과제

실습 과제에서 구현해야 하는 프로그램은 **총 3개**이다.

- `program1`: 기본 프로그램 (프로세스 생성 X, 스레드 생성 X)
- `program2`: 멀티 프로세스를 이용하는 프로그램, 내부적으로 `program1` 실행
- `program3`: 멀티 스레딩을 이용하는 프로그램, `program1`을 수정하여 구현

Program 1. 기본 구현

이 프로그램은 2차원 배열로 주어진 입력 데이터에 풀링 연산을 적용해, 그 결과를 출력하는 간단한 프로그램이다.

구현 사항

- `program1`은 리눅스 쉘에서 아래와 같은 명령어를 통해 실행된다. 어떤 풀링 연산을 할 지가 인자로 주어진다.
  (참고: C에서 메인 함수의 argc, argv)

```
./program1 [type]
```

- 인자로 전달 가능한 풀링 연산의 종류는 최대 풀링(max), 평균 풀링(avg), 최소 풀링(min)이 있다. 각각 아래와 같이 실행된다.
  이 세가지 외에 다른 입력은 없다고 가정한다.

```
./program1 max
./program1 avg
./program1 min
```

- 입력값과 출력값은 **파일 입출력이 아닌 표준 입출력 (Standard I/O)**을 통해 처리한다.
  과제 채점시에는 아래 예시처럼 입출력 리다이렉션을 이용해 입력값 및 출력값을 전달함에 유의한다.
  input 파일 및 output 파일의 이름, 확장자, 디렉토리 등은 아래 예시와 같이 다양할 수 있음에 주의해야 한다.

```
./program1 max < input > /home/osuser/result/output
./program1 max < input.txt > output
./program1 avg < ../input/osinput > ../output
./program1 min < /home/osuser/input/input > output.txt
```

- 입력값은 다음과 같이 주어진다.
  - 첫 줄에는 입력 배열의 **높이 H, 너비 W, 풀링 연산의 스트라이드 값 N**이 주어진다. 각 값은 공백 한 칸으로 구분된다.
    - H, W, N은 모두 정수이다.
    - $0 < H \leq 25000$, $0 < W \leq 25000$, $0 < N \leq 25000$
    - H와 W는 M의 양의 정수배이다.
  - 풀링 연산을 적용할 입력 배열에 대한 정보로 **두번째 줄부터 H+1번째 줄**에는 **각 W개의 음이 아닌 정수**(int 범위 내)가 주어진다.
    각 정수는 공백 한 칸으로 구분된다.
  - 입력값 예시 (H = 4, W = 6, N = 2)

```
4 6 2
7 5 0 3 10 0
10 4 21 2 0 4
6 1 7 0 21 2
5 0 8 4 3 16
```

- 출력값은 반드시 다음과 같이 출력한다. 채점은 자동 스크립트에 의해 진행되므로, **출력 양식**이 잘못된 경우 많은 감점을 받을 수 있다.
  - 첫 줄에는 프로그램을 수행하는데 걸린 시간을 밀리초(ms) 단위의 정수형으로 출력한다.
  - 두번째 줄 부터는 풀링 연산 결과를 정수형으로 출력한다. 결과 배열의 높이는 H/N, 너비는 W/N일 것이다.
    - 평균 풀링의 경우 모든 연산을 int형으로 진행한다. 예를 들어, "1, 10, 5, 6"의 평균을 구해야 하는 경우 (1+10+5+6)/4 를 int로 계산한 5가 출력되어야 한다.
  - 출력값 예시

```
101
10 21 10
6 8 21
```

Program 2. 멀티 프로세스 구현

이 프로그램은 위에서 구현한 **program1**을 내부적으로 여러번 실행하는 멀티 프로세스 기법을 사용한다. **fork() 시스템 콜**로 total_process_num으로 주어진 개수 만큼의 자식 프로세스를 생성한다. 이 때, 반드시 **exec 계열의 시스템 콜**로 **program1**을 호출하여 멀티 프로세스를 수행해야 한다. **전체 입력 데이터를 total_process_num 만큼 적절히 나눈 후**, 나누어진 입력 데이터를 **program1**을 실행할 때 전달한다. 그 후, 여러번 실행된 **program1**의 데이터를 수합해 최종 결과를 만들어낸다.

구현 사항

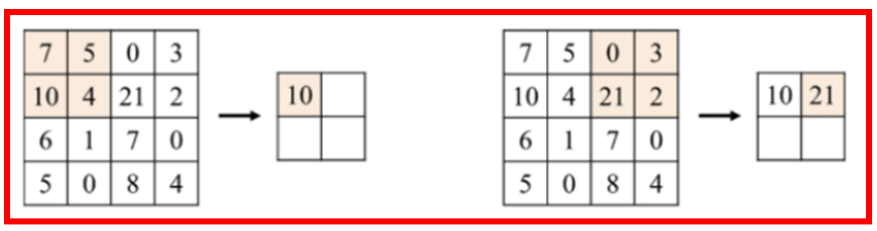
- **program2**은 리눅스 쉘에서 아래와 같은 명령어를 통해 실행된다. 총 프로세스 수(total_process_num)이 새로운 인자로 추가되었다.

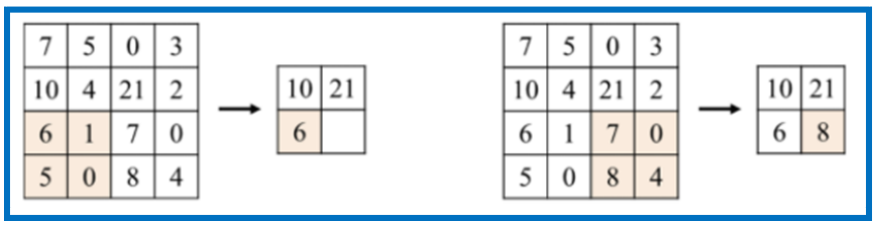
```
./program2 [type] [total_process_num] < input
```

- **입력 예시** (생성되는 자식 프로세스의 수가 2)

```
./program2 max 2 < input.txt
```

- **입력 양식과 출력 양식은 program1과 같다.**
- **구현 방법**
  메인 프로세스는 total_process_num만큼 자식 프로세스를 fork하고, 자식 프로세스는 입력 데이터를 분배받아 풀링 연산을 실행한다. 예를 들어, total_process_num가 2이면 메인 프로세스와 2개의 자식 프로세스로 프로그램이 진행된다. 메인 프로세스는 자식 프로세스가 처리한 결과를 병합하는 역할만을 진행한다. 풀링 연산 과정을 분배하는 방식은 아래 그림과 같이 진행하면 된다.





- 자식 프로세스와 메인 프로세스가 통신하기 위해서 다양한 **프로세스간 통신 방법(IPC)**를 사용할 수 있다. 이를 위해서, **program1**은 (1) **program2**에 의해 호출될 때와, (2) 쉘에서 바로 시작될 때를 구분하여 IPC를 적절히 핸들링하도록 구현하는 것이 한가지 방법일 수 있다.
- **참고 사항:** 프로세스의 동작은 다음과 같이 확인할 수 있다

```
user@ubuntu:~$ ps -eaT | grep program1
```

Program 3. 멀티 스레딩 구현

이 프로그램은 위에서 구현한 **program1**을 **수정**하여 멀티 스레딩 기법을 이용한다. **pthread 함수**를 이용하여 스레드를 여러개 생성한 후, 입력 데이터를 각 스레드에 적절히 분배하여 각 스레드에서 연산을 처리하도록 한다. **pthread 함수** 외의 다른 스레딩 함수를 사용하는 것은 허용되지 않는다.

구현 사항

- **program3**은 리눅스 쉘에서 아래와 같은 명령어를 통해 실행된다. 각 인자는 **program2**와 유사하다.

```
./program3 [type] [total_thread_num] < input
```

- **입력 양식과 출력 양식은 program1과 같다.**
- **구현 방법**

  total_thread_num만큼 pthread 함수를 이용해 스레드를 생성하고 필터 연산을 적절히 분배해 처리하도록 한다. 구체적인 분할 방식은 **program2**와 같다. 입력 데이터를 스레드에 전달할 때 **임시 파일을 이용하는 것은 허용되지 않으며, 반드시 공유 메모리와 같은 방법**으로 스레드에 인자를 전달해야 한다.

실습 과제 컴파일 조건

실습 과제는 하나의 **Makefile을 통해 컴파일되어야** 한다. 즉, make 빌드 시스템으로 빌드를 하게 되면 make 명령어를 실행한 그 경로에 program1, program2, program3라는 이름의 실행 가능한 파일이 총 3개 생성되어야 한다. make clean을 하게 되면 빌드 결과로 생성된 파일들이 모두 삭제되어야 한다. make 빌드 시스템에서 사용하는 gcc의 버전은 1페이지의 '제한 사항' 부분을 반드시 확인한다. 컴파일러의 버전이 다른 경우 컴파일이 안될 수 있어 점수에 불이익을 받을 수 있다.

(예시)

```
os@ubuntu:~$ cd 2022123123
os@ubuntu:~/2022123123$ make
os@ubuntu:~/2022123123$ ls
Makefile program1 program2 program3 …
os@ubuntu:~/2022123123$ make clean
Makefile …
```

과제 제출 방법

보고서 pdf 파일(hw2_[학번].pdf)과 소스 압축 파일(hw2_[학번].tar)을 제출한다.

보고서 파일(hw2_[학번].pdf)

PDF 파일로 변환하여 제출한다. 파일의 이름은 반드시 hw2_[학번].pdf로 한다.

실습 과제 소스 압축 파일(hw2_[학번].tar)

반드시 다음 명령어를 사용해서 압축 파일을 생성한다. **다른 형식의 압축 또는 이중 압축은 절대 허용하지 않으며 보고서 파일을 실습 과제 파일과 함께 압축해서는 안된다.**

```
# 본인이 작업하는 디렉토리가 ~/hw2라고 가정하자.
이는 Makefile이 ~/hw2/ 에 있음을 의미한다. (vi ~/hw2/Makefile)
# 본인의 학번이 2022123123 이라고 가정하면, 아래의 일련의 명령어를 수행한 이 후에 hw2_2022123123.tar을 제출한다.
# 압축을 해제하였을 때 본인의 학번으로 된 폴더 하나가 나타나야 하고, 그 폴더 바로 아래에 Makefile이 있어야 함을 의미한다.
$ cd ~/hw2/..
$ mv hw2 2022123123
$ tar cvf hw2_2022123123.tar 2022123123
```

참고 문헌

CNN 관련

https://cs231n.github.io/convolutional-networks/

https://wikidocs.net/62306

기타

Using dup2 for I/O Redirection and Pipes http://www.cs.loyola.edu/~jglenn/702/S2005/Examples/dup2.html

Thread Pool https://en.wikipedia.org/wiki/Thread_pool

POSIX Threads Programming https://computing.llnl.gov/tutorials/pthreads/

Multi-Process Programming in C

http://home.deib.polimi.it/fornacia/lib/exe/fetch.php?media=teaching:aos:2016:aos201617_multiprocess_programming_updated20161223.pdf