2021년 1학기 운영체제 과제 #2

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. 이 과제에서는 간단한 쉘(Shell)의 구현을 통해 프로세스의 생성과 동작을 이해하고, 병합 정렬(Merge Sort)을 여러 프로세스/스레드를 사용하는 병렬 프로그래밍(Parallel Programming)기법을 통해 구현하여 성능을 비교해 볼 것이다.

1 과제 내용 및 목적

✓ 쉘(Shell)의 구조와 명령어의 동작 방식을 이해하고, fork()와 exec()을 사용하여 직접 구현한다.

✓ 프로세스와 스레드의 구조와 동작을 이해하고, 실제 리눅스 환경에서 멀티 프로세스/스레드 프로그램을 구현한다.

2 제출 기한

✓ **2020년 4월 21일(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 18.04, 채점 서버 컴파일러: gcc 7.5.0 / clang 6.0.0, locale: UTF-8 (euc-kr 등 사용 금지)

✓ 프로그래밍 언어는 반드시 C 또는 C++를 사용한다.

- 스레드 생성은 pthread만을, 프로세스 생성은 fork()만을 사용해야한다.
- 그 외 구현에서는 C 표준 라이브러리 또는 C++ 표준 라이브러리내에서 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 `miniShell, program1, program2, program3`라는 이름의 실행 가능한 바이너리가 생성되어야 한다. (터미널에서 ./program1로 프로그램이 실행 됨)
- 컴파일시 필요한 옵션(라이브러리, C++11 등)은 Makefile에 반드시 기재하여 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

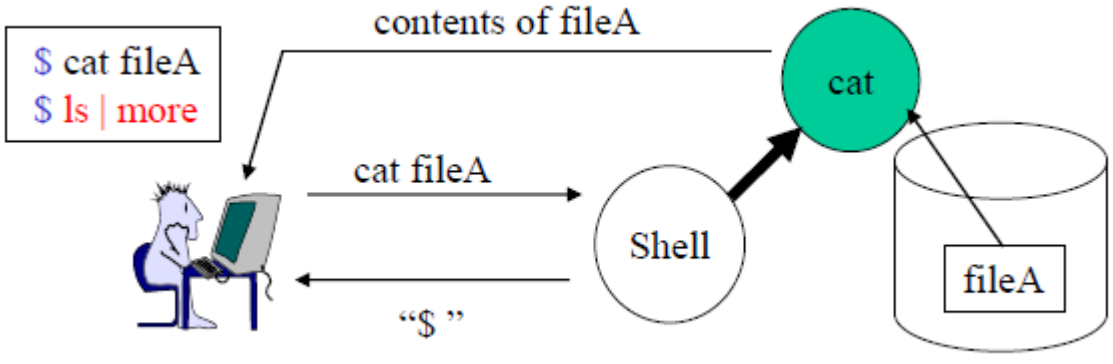
✓ 제출한 과제가 컴파일이 되지 않는 경우 0점 처리

✓ 과제에 대한 문의 사항이 있을 경우 learnUs의 Q&A게시판에 **'공개 게시글'**로 문의

사전 지식

쉘 (Shell)

쉘은 운영체제(Windows, Linux, etc.)의 커널과 사용자를 사이를 이어주는 인터페이스로 사용자의 명령어를 입력 받아 이를 해석하여 대신 실행하는 역할은 한다. 쉘의 명령문은 명령문과 함께 다양한 인자 값 또는 옵션을 포함할 수 있다. 예를 들어 아래 그림 보면, 쉘은 사용자에게 명령어를 사용할 수 있는 인터페이스를 제공한다. 사용자가 'fileA'의 내용을 출력하고 싶어 'cat fileA'라는 명령어를 입력하면, 쉘은 새로운 자식 프로세스를 생성하고 해당 명령을 수행한 후, 자식 프로세스의 실행이 끝나길 기다리다 해당 작업이 끝나면 이 정보를 받아 사용자에게 전달하게 된다.



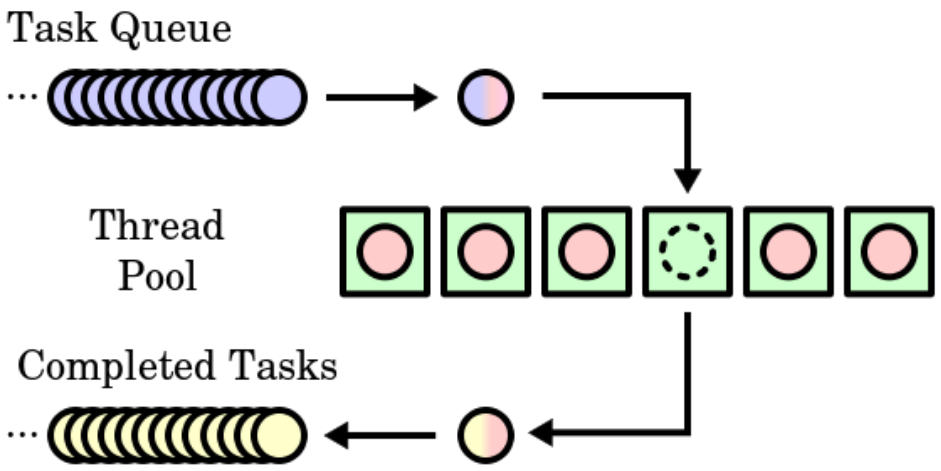
병합 정렬 (Merge Sort)

병합 정렬은 폰 노이만이 개발한 정렬 알고리즘으로 분할 정복 알고리즘 기반으로하며 O(nlogn)의 계산 복잡도를 갖는다. 이 정렬 알고리즘은 유명한 알고리즘으로 혹시 모르는 내용이 있는 경우 아래 링크의 위키 피디아 내용을 참고할 수 있다.

https://en.wikipedia.org/wiki/Merge_sort

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.

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 과제1 프로그램에 대해 서술
    - 실습 과제2에서 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 과제 1의 동작 검증**
      - 과제의 요구사항대로 잘 동작하는지 확인
    - **실습 과제 2의 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 1. MiniShell 구현

사용자의 입력을 한줄씩 받으면서 처리하는 쉘을 만든다. 엔터를 통해 입력받은 한줄의 명령어를 처리하고, 다시 명령어를 기다린다. 이 때 명령어 중 표준 유닉스 프로그램을(위 예시의 cat) 불러와 사용하는 경우, fork를 통해 자식 프로세스를 만들고 cat의 바이너리를 실행시킨 후 자식 프로세스가 끝날 때까지 기다린다. 우리가 구현하는 쉘은 주로 표준 유닉스 프로그램을 실행시키기 위한 용도이고, 이때 인자를 전달할 수 있다.

- 쉘은 다음과 같은 명령어로 실행된다. 이 프로그램의 파일명은 반드시 'miniShell' 이어야 한다.

```
./miniShell
```

- 쉘을 실행했을 때의 출력 양식은 다음과 같도록 한다.

```
[HH:MM:SS]`username`@`pwd`$
```

- 출력 예제는 다음과 같다.

```
parallels@ubuntu:~/Desktop/assign2/test$ ./miniShell
[18:19:59]parallels@/home/parallels/Desktop/assign2/test$ls
dysh.c  miniShell
[18:20:03]parallels@/home/parallels/Desktop/assign2/test$rm dysh.c
[18:20:11]parallels@/home/parallels/Desktop/assign2/test$ls
miniShell
[18:20:14]parallels@/home/parallels/Desktop/assign2/test$ls -al
total 24
drwxrwxr-x 2 parallels parallels  4096 Mar 28 18:20 .
drwxrwxr-x 4 parallels parallels  4096 Mar 28 18:18 ..
-rwxrwxr-x 1 parallels parallels 14471 Mar 28 18:19 miniShell
[18:20:18]parallels@/home/parallels/Desktop/assign2/test$
```

- 실행 되어야 하는 필수 명령어는 다음과 같다.
  - cd: 미니 쉘이 제어하는 디렉토리는 실제 운영체제의 디렉토리와 같다. 즉, 가상 환경을 만드는 것이 아니다.
  - 표준 유닉스 프로그램 (ls, mkdir, rm, echo, cat, head, tail, time 등 환경변수에 이름이 있는 프로그램)
    - ls를 통해 나타나는 파일 목록은 실제 운영체제의 디렉토리 내부의 파일 목록이다.
  - exit: 이 명령어를 치면 미니 쉘 프로그램이 종료된다.
  - 출력 리다이렉션 (stdout redirection; >)
  - 입력 리다이렉션 (stdin redirection; <)
  - 백그라운드 명령 (&)
    - 명령어 맨 뒤에 &이 붙어있을 경우 해당 명령어는 쉘 프로그램과 별도로 백그라운드에서 동작한다.
    - &이 명령어에 있는 경우 쉘은 해당 프로그램의 종료를 기다리지 않고 동작한다. 없는 경우는 프로그램의 종료를 기다린다.
  - 프로그램의 실행 (.)
    - 일반 쉘 환경과 같이 ./a.out 과 같은 방식으로 프로그램을 실행시킨다.
    - **반드시 fork()와 exec()을 이용**하여 프로그램을 실행해야 한다.
      - exec 계열의 시스템 콜이 다양하므로 그중 가장 적절한 시스템 콜을 골라야 한다.
    - exec 계열의 시스템 콜로 argc, argv 입력값을 넘겨주는 과정이 곤란할 수 있다. 이 때 입력 파일을 임시로 생성한 다음 dup/dup2 시스템 콜을 이용하여 표준 입력을 임시 입력 파일로 redirection 하거나 파이프(Pipe)를 이용 하는 것이 팁이다. 단, 중간에 임시로 생성되는 파일은 프로그램이 종료되면 반드시 삭제되어야 한다. 이외에 다양한 방법을 시도할 수 있으나 채점 환경에서 정상적으로 동작하는 경우에만 점수를 부여받을 수 있다.

```c
pid_t pid = fork();
if (pid == 0) {
  int in, out;
  char *args[] = {"./program1", NULL};
  // open input and output files
  in = open("input file name", O_RDONLY);
  out = open("output file name", O_WRONLY | O_TRUNC | O_CREAT, S_IRUSR | S_IRGRP | S_IWGRP | S_IWUSR);
  // replace standard input with input file
  dup2(in, 0);
  // replace standard output with output file
  dup2(out, 1);
  // close unused file descriptors
  close(in);
  close(out);
  // execute program 1
  execvp("./program1", args);
}
```

- **주의사항**
  - miniShell 프로그램에서 **운영체제에 존재하는 기존 쉘을 호출해 사용하는 것은 절대 금지**한다.
    - 직접 시스템 콜이나 기본 API 함수를 사용하여 쉘의 기능을 만들어내야 한다.
  - 없는 명령어의 호출이나 존재하지 않는 프로그램의 실행인 경우에도 쉘의 동작은 멈추지 않아야 한다.
- 필수 조건 외 기타 구현은 실제 리눅스 쉘(bash shell)과 유사하게 한다.
  - 단, 위에서 명시한 기본 동작 외의 사항은 채점 대상이 아니다.

실습 과제 2. 병합 정렬(Merge sort) 구현

실습 과제2 에서 구현해야 하는 프로그램은 **총 3개**이다.

- program1: 기본 프로그램 (프로세스 생성 X, 스레드 생성 X)
- program2: 멀티 프로세스를 이용하는 프로그램
- program3: 멀티 스레딩을 이용하는 프로그램

Program 1. 기본 구현

이 프로그램(program1)은 주어진 임의의 숫자들을 병합 정렬(Merge sort)을 통해 정렬하는 간단한 프로그램이다.

구현 사항

- program1은 실습 과제1에서 구현한 **miniShell**에서 아래와 같은 명령어를 통해 실행된다. 입력값은 **파일 입력이 아닌 표준 입력(Standard input)**을 통해 주어지며, 과제 채점시에는 입력 리다이렉션을 이용해 입력값을 전달한다. input 파일의 이름과 디렉토리는 아래와 같이 다양할 수 있음에 주의해야 한다.

```
./program1 < input
./program1 < input.txt
./program1 < ../input/osinput
```

- 입력값은 다음과 같이 주어진다.
  - 첫 줄에는 정수의 개수 N이 주어진다. N은 int 범위 내의 수이다.
  - 두번째 줄에는 N개의 정수가 주어진다. 각 정수는 공백 한칸으로 구분되며, 모두 int 범위 내의 수이다.

```
8
10 3 123 523490 213 -12312 213 -768
```

- 결과는 반드시 다음 의사 코드와 같은 양식으로 **파일 출력이 아닌 표준 출력(Standard output)**을 통해 출력한다. 과제 채점시에는 출력 리다이렉션을 이용해 출력값을 만들어냄에 주의한다. **출력 양식이 틀린 경우 점수를 받지 못할 수 있으니 주의한다**.
  - 첫 줄에는 정렬의 결과를 내림차순으로 출력한다. 각 숫자 사이는 공백 한칸으로 구분한다.
  - 두번째 줄에는 병합 정렬을 수행하는데 걸린 시간을 ms 단위로 출력한다.

```
523490 213 213 123 10 3 -12312 -768
1231
```

Program 2. 멀티 프로세스 구현

이 프로그램은 위에서 구현한 program1을 이용하여 멀티 프로세스 기법을 사용한다. **fork() 시스템 콜**로 total_process_num 으로 주어진 개수만큼의 자식 프로세스를 생성한다. 전체 입력 데이터를 total_process_num 만큼 적절히 나눈 후, 나누어진 입력 데이터를 program1을 실행할 때 전달한다. 그 후, 여러번 실행된 program1의 데이터를 수합해 최종 결과를 만들어낸다.

구현 사항

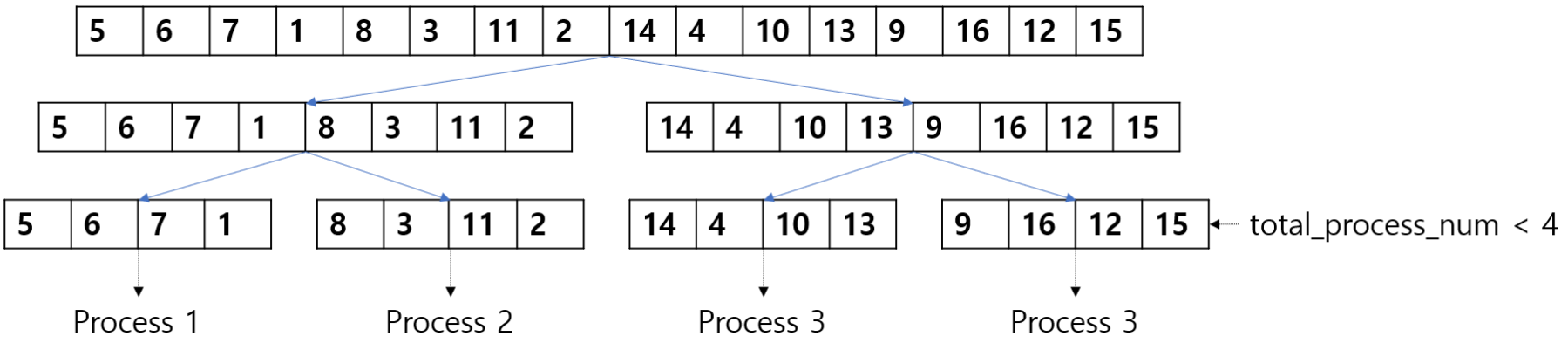
- program2은 리눅스 쉘에서 아래와 같은 명령어를 통해 실행된다. (참고: C에서 메인 함수의 argc, argv)

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

- **입력 예시** (생성되는 자식 프로세스의 수가 3)

```
./program2 3 < input.txt
```

- **입력 양식과 출력 양식은 program1과 같다.**
- **구현 방법**
  total_process_num만큼 자식 프로세스를 fork하고, 자식 프로세스는 입력 데이터를 분배받아 병합 정렬을 실행한다. 이 과제에서 구현하는 병합 정렬의 연산 과정은 여러가지 방법으로 병렬화 할 수 있지만, **이 과제에서는 배열을 여러번 분할하던 중 `total_process_num` 이상의 그룹이 생성되었을 때, 각 그룹에 대한 정렬을 각 프로세스에 맡기는 방식으로** 구현한다. `total_process_num`이 3일 때의 구체적인 동작 방식 예는 아래 그림과 같다.



- **참고 사항:** 프로세스의 동작은 다음과 같이 확인할 수 있다

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

Program 3. 멀티 스레딩 구현

이 프로그램(program3)은 위에서 구현한 program1을 수정하여 멀티 스레딩 기법을 이용하여 데이터를 여러 그룹으로 분할해 이를 동시에 처리할 수 있도록 한 프로그램이다. **pthread 함수**를 이용하여 스레드를 여러개 생성한 후, 입력 데이터를 각 스레드에 적절히 분배하여 각 스레드에서 연산을 처리하도록 한다.

구현 사항

- program3은 리눅스 쉘에서 아래와 같은 명령어를 통해 실행된다.

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

- **입력 예시** (생성되는 스레드의 수가 3)

```
./program3 3 < input.txt
```

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

  `total_thread_num`만큼 pthread 함수를 이용해 스레드를 생성하고 필터 연산을 적절히 분배해 처리하도록 한다. 구체적인 분할 방식은 program 2와 같다. 입력 데이터를 스레드에 전달할 때 파일을 임시로 생성해서는 안되고, 공유 메모리와 같은 방법으로 스레드에 인자를 전달해야 한다.

- **참고 사항** 스레드의 동작은 다음과 같이 확인할 수 있다

```
mobed@ubuntu:~$ ps -eaT | grep program3
 10461   10461 pts/4    00:00:00 program3
 10461   10462 pts/4    00:00:01 program3
 10461   10463 pts/4    00:00:01 program3
 10461   10464 pts/4    00:00:01 program3
```

실습 과제 컴파일 조건

실습 과제 1과 실습 과제 2는 하나의 **Makefile을 통해 컴파일되어야** 한다. 즉, make 빌드 시스템으로 빌드를 하게 되면 `miniShell, program1, program2, program3`라는 이름의 실행 가능한 파일이 총 4개 생성되어야 한다. make clean을 하게 되면 빌드 결과로 생성된 파일들이 모두 삭제되어야 한다. make 빌드 시스템에서 사용하는 gcc나 clang의 버전은 1페이지의 '제한 사항' 부분을 반드시 확인한다. 컴파일러의 버전이 다른 경우 컴파일이 안될 수 있어 점수에 불이익을 받을 수 있다.

(예시)

```
os@ubuntu:~$ cd 2021123123
os@ubuntu:~/2021123123$ make
os@ubuntu:~/2021123123$ ls
Makefile miniShell program1 program2 program3 …
os@ubuntu:~/2021123123$ 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 2021123123 이라고 가정하면, 아래의 일련의 명령어를 수행한 이 후에 hw2_2021123123.tar을 제출한다.
# 압축을 해제하였을 때 본인의 학번으로 된 폴더 하나가 나타나야 하고, 그 폴더 바로 아래에 Makefile이 있어야 함을 의미한다.
$ cd ~/hw2/..
$ mv hw2 2021123123
$ tar cvf hw2_2021123123.tar 2021123123
```

참고 문헌

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