

# 2021년 1학기 운영체제 과제 #3

## 과제 개요

여러 프로세스가 한 CPU를 놓고 경쟁하는 상황이 발생하면, 운영체제는 스케줄러(Scheduler)를 통해 이러한 상황을 적절하게 관리한다. 또한 운영체제는 프로세스가 실제 메모리의 물리적인 용량보다 큰 가상 주소 공간을 제공받아 이를 참조할 수 있게 한다. 이러한 기법을 가상 메모리(Virtual Memory)라고 하며, 메모리 관리(Memory Management) 기법으로 페이징(Paging)과 버디 시스템(Buddy System)이 있다. 이번 과제에서는 스케줄러와 메모리 관리 기법에 대해 알아보고, 이를 단순한 형태의 시뮬레이터를 제작한다.

## 1 과제 목적

- ✓ 스케줄러의 동작 원리를 이해한다.
- ✓ 가상 메모리의 구조와 동작에 대해 이해하고, 메모리 관리 기법인 페이징과 버디 시스템을 이해한다.

## 2 제출 기한

- ✓ 2021년 6월 14일(월) 오전 11:55 까지 (**늦은 제출 절대 받지 않음**)

## 3 제출 방법

- ✓ 과제는 보고서 과제와 실습 과제로 이루어진다.
- ✓ 보고서 과제 파일 및 실습 과제 파일: 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)
  - 위에서 언급한 환경이 아닌 다른 환경을 사용하여 컴파일이 되지 않거나 동작하지 않는 경우 큰 감점
- ✓ 프로그래밍 언어는 C 또는 C++을 사용한다.
  - C 표준 라이브러리 또는 C++ 표준 라이브러리는 자유롭게 사용 가능하나 그 외의 외부 라이브러리의 사용은 금지한다.
- ✓ 실습 과제 프로그램 구조
  - 제출물의 압축을 해제하면 생성되는 폴더 내에 Makefile과 구현한 프로그램의 소스 코드가 존재해야 한다.
  - make 명령을 실행하면 각각의 소스 코드들이 컴파일 되어 **project3**라는 이름의 실행 가능한 바이너리가 생성되어야 한다. (터미널에서 **./project3**로 프로그램이 실행 됨)
  - 컴파일 시 필요한 옵션(라이브러리, C++14, C++17 등)은 Makefile에 반드시 기재하여 채점 서버에서 컴파일이 되도록 해야 한다.
  - make clean 명령을 실행하면 make를 통해 생성된 빌드 결과물을 삭제해야 한다.
  - 이 구조를 따르지 않는 경우 크게 감점이 된다.
- ✓ 1인 1프로젝트이며 각자의 환경에서 작업한다.

## 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
- ✓ 과제 제출 방법, 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
- ✓ 과제에 대한 문의 사항이 있을 경우 learnUs 질문 게시판으로 문의 (**반드시 공개 질문으로 올릴 것 비밀질문에는 답변하지 않음**)
- ✓ 이 과제 명세에 대한 내용을 웹 사이트에 배포하거나 질문한 것이 발각되는 경우 0점 처리

# 과제 세부 사항

과제는 실습 과제와 보고서 과제로 이루어져 있다.

## 실습 과제

실습 과제에서는 스케줄러와 메모리 관리 기법이 결합된 시뮬레이터를 만든다.

### 시뮬레이터 프로그램

- 스케줄러, 버디 시스템과 페이징이 결합된 가상의 시스템을 시뮬레이션 한다. 입력으로 프로세스가 사용하는 메모리 연산(Memory Operation)이 들어오고 명령어 한개 단위로 연산을 처리하는 과정을 출력하는 시스템이다.
- 만약 다른 조건이 같다면 먼저 Run queue에 들어온 프로세스가 우선순위가 더 높은 것으로 한다. CPU를 사용하는 프로세스와 Run queue에 있는 프로세스의 우선순위가 같다면 현재 CPU를 사용중인 프로세스에 우선순위가 있다.
- 스케줄러는 Input에 따라 프로세스를 생성하고, 주어진 스케줄링 알고리즘으로 대기중인 프로세스들을 관리한다.
- 스케줄러는 **Multilevel Queue Scheduling**으로 동작하며, 0부터 9까지의 priority에 해당하는 run queues(= ready queues)가 있다.
  - 공통 조건
    - 현재 실행중인 프로세스보다 높은 priority를 갖는 프로세스가 run queue에 존재한다면, 현재 실행중인 프로세스의 실행은 즉시 유예되고 더 높은 priority를 갖는 프로세스가 CPU를 선점한다.
    - Fixed-priority이다. 즉 프로세스의 priority는 처음에 주어진 것에서 변하지 않는다.
    - Priority 숫자가 작을 수록 높은 우선순위를 갖는다.
  - **Priority 0-4: FCFS (First-come First-Served)**
    - Run queue에 들어온 순서대로 CPU가 선택한다.
    - CPU에서 동작하고 있는 프로세스는 (1) Sleep이나 IO로 인하여 스스로 block되거나 (2) 스스로 종료되거나 (3) 우선순위가 높은 프로세스가 runqueue에 존재하기 전까지 계속 CPU를 사용한다.
    - Block이 끝난 프로세스는 Run queue의 맨 뒤에 삽입된다.
  - **Priority 5-9: RR (Round Robin)**
    - FCFS와 마찬가지로 CPU는 run queue의 맨 앞에 있는 프로세스를 선택한다.
    - 주어진 time quantum을 다 사용한 프로세스는 CPU 사용을 중단하고 Run queue의 맨 뒤에 삽입된다.
      - 즉, CPU에서 동작하고 있는 프로세스는 (1) Sleep이나 IO로 인하여 스스로 block되거나 (2) 스스로 종료되거나 (3) 우선순위가 높은 프로세스가 runqueue에 존재하거나 (4) 남은 time quantum이 0이 되기 전까지 계속 CPU를 사용한다.
    - 주어진 time quantum을 다 사용하지 못하고 실행 도중 block되면, 다시 run queue에 삽입될 때 time quantum이 초기화된다. Run queue에 삽입될 때에는 항상 queue의 맨 뒤에 삽입된다.
    - 이 과제에서는 **time quantum을 10 cycle로 한다.**
- 시스템의 설정값, 몇 번째 cycle에 어떤 Code가 실행되는지, 어떤 프로세스에 IO작업이 이루어지는지 등이 **Input**으로 주어진다.
  - 프로세스는 독자적인 가상 메모리(Virtual Memory) 공간을 가지며, 본 시스템은 물리 메모리(Physical Memory)가 존재한다.
  - 프로세스의 가상 메모리는 페이지 단위로 나뉘어 관리가 되며, 물리 메모리 또한 프레임 단위로 관리된다.
    - 페이지 테이블(Page Table)은 각 페이지의 **Page ID**와 각 페이지가 물리 메모리에 할당될 시 갖게 되는 **Valid bit, Allocation ID (Frame index), Reference bit**를 갖는다.
      - Page ID는 각 프로세스의 가상 메모리마다 따로 관리되며, 0부터 시작한다. 새로운 페이지가 가상 메모리에 할당될 때 마다 부여받으며, 해당 가상 메모리의 마지막 Page ID보다 1 큰 값으로 받는다.
      - Allocation ID는 전체 시스템이 하나로 관리한다. 즉, 물리 메모리에서 관리하며, 0부터 시작한다. 페이지가 물리 메모리에 처음 할당될 때 부여받으며, 마지막 Allocation ID보다 1 큰 값으로 받는다.
      - Reference bit는 페이지 교체 알고리즘에 의존적이므로, 해당 설명 부분을 참고한다.
    - 가상 메모리 할당의 경우 요청하는 크기가 들어가는 첫 번째 공간으로 할당한다.
    - 가상 메모리는 하위 주소부터 메모리를 채워나간다.
  - 각 프로세스는 별개의 instruction이 주어진다.
  - 메인 input은 프로세스가 언제 CPU에 들어오는지를 알려준다. 시스템의 스케줄러는 이를 바탕으로 동작한다.
    - 프로세스는 실행되는 순서대로 PID (Process ID)가 부여되며, PID는 0부터 시작한다. 한 번 사용된 ID는 재사용되지 않는다.
- Code의 Instruction은 **Memory Allocation, Memory Access, Memory Release, Sleep, IOwait** 그리고 **Non-memory instruction**으로 총 8가지 명령으로 이루어져 있다. **모든 명령은 1 cycle을 소모한다.**

- **Memory Allocation** : 프로세스가 원하는 연속된 페이지 개수 만큼 페이지 테이블에 메모리 할당(Page Table Allocation)을 수행한다
  - 프로세스는 원하는 페이지의 개수 만큼을 가상 메모리의 페이지 테이블에 할당하고, 이를 Page ID를 통해 관리한다.
    - Page ID는 가상 메모리 내에서는 중복되지 않는 유일한 값이다. 다른 가상 메모리와는 중복될 수 있다.
  - 페이지 테이블 할당 연산의 경우, 물리 메모리에 프레임을 할당하지 않고 페이지 테이블에만 할당한다. (아래의 Memory Access 참고)
- **Memory Access** : 프로세스가 이전에 할당한 페이지와 그것에 연결된 프레임에 접근(Access)한다. 이 때 Page ID를 통해 접근한다.
  - Allocation ID는 새로 할당되는 순서대로 0부터 ID가 부여된다. 한 번 사용된 ID는 재사용되지 않는다.
    - 새롭게 접근되는 페이지에 대해서 새로운 Allocation ID가 부여되며, 기존에 Allocation ID를 부여받은 페이지의 경우에는 ID가 갱신되지 않는다.
  - 페이지에 할당된 프레임이 없는 경우에는 버디 시스템을 사용하여 할당해야 할 프레임의 수를 계산한 뒤, 물리 메모리 상에서 연속된 프레임을 할당한다.
    - 버디 시스템에 따라 작은 크기를 갖는 공간에 우선순위가 생기며, 같은 크기를 갖는 공간이 여러개일 때에는 하위 주소부터 메모리를 채워나간다.
    - 프로세스가 메모리 접근 연산을 수행하면, 요청한 페이지들은 버디 시스템에 의해 할당해야 할 페이지의 개수를 계산한 뒤 해당 페이지들을 가상 메모리 상에 해당 크기만큼 할당 가능한 연속된 공간에 순차적으로 할당된다.
  - 버디 시스템에 의해 프로세스로부터 요청된 페이지들을 물리 메모리에 할당할 때, 물리 메모리에 할당 가능한 공간이 없다면 바로 아래에 서술되어있는 **페이지 교체 알고리즘**에 따라 기존에 할당되어 있는 프레임을 해제한다.
  - **페이지 교체 알고리즘(Page Replacement Algorithms)**
    - 프레임을 해제할 때에는 같은 Allocation ID를 갖는 연속된 프레임을 모두 해제한다.
    - 요청된 연속된 페이지들이 할당될 수 있는 물리 메모리 내의 가용 공간이 확보될 때까지, 페이지 교체 알고리즘에 의한 해제 과정을 반복한 후 할당한다.
    - 요청된 페이지들이 물리 메모리에 할당되면 Allocation ID를 부여받고, 해당 Allocation ID가 프로세스의 페이지 테이블에 기록된다. 페이지 테이블은 페이지 교체 알고리즘에 의해 해제된 프레임에 대한 사항을 업데이트 한다.
    - 만약 알고리즘에 의해 선택될 수 있는 프레임이 두개 이상이라면, Allocation ID가 작은 것부터 선택해 이를 해제후 교체한다.
    - 옵션에 따라 다양한 알고리즘으로 동작할 수 있다. 각 알고리즘의 자세한 설명은 강의노트(L9-virtualmemory)를 참고하도록 한다.
      - **LRU 알고리즘 (-page=lrn)**
        - Stack 기반으로 구현해 보는 것을 권장한다.
        - 기본 LRU의 경우에는 Reference bit는 항상 비어있다.
      - **Sampled LRU 알고리즘 (-page=sampled)**
        - Time interval은 8 Cycles로 한다. 즉, 8 Cycle의 배수가 되는 시점마다 Reference byte가 갱신된다.
          - 해당 Cycle에 명령어가 실행되기 전에 갱신한다.
          - 물리 메모리에 있지 않은 페이지도 갱신된다.
        - 한 Interval내에 Memory Access가 발생한 페이지의 경우 Reference bit가 1이 된다.
        - Reference byte는 강의노트에 있는 것처럼 1 byte(8 bits)로 한다.
        - 교체되어 나가는 페이지의 경우에는 Reference bit가 0이 되며, reference byte는 유지된다.
      - **Clock 알고리즘 (-page=clock)**
        - Two-handed가 아닌 기본적인 Second chance 알고리즘 기반으로 동작한다.
        - Memory Access가 발생하면 Reference bit가 1이 된다.
        - Clock이 해당 페이지를 가리킬 때 Reference bit가 1이면 0이 되고 다음 page로 넘어가며, Reference bit가 0이면 해당 page를 교체시킨다.
        - Clock은 현재 물리 메모리에 할당 되어 있는 프레임들 (즉, Valid bit이 1인 페이지들)에 대해서 순회하며, Allocation ID가 0인 프레임부터 시작하여 Allocation ID가 커지는 방향으로 순회한다.
        - 예를 들어, Allocation ID가 8인 프레임이 선택되어 교체되었다면 다음 Clock 순회는 해당 프레임의 다음 순서에 해당하는 프레임부터 시작한다.
  - **Memory Release** : Process가 이전에 할당한 Page들에 대해 Memory Release 수행.
    - Process가 Memory Release를 수행하면, Release를 요청한 Page들은 Virtual Memory 상에서 해제된다. 또한 Physical Memory 상에서 해당 Page들이 할당되어 있다면 이 또한 함께 해제된다.
    - Page ID를 통해 연속된 Page들을 해제한다.
  - **Non-memory instruction** : 특별한 동작 없이 1 cycle을 소모한다.

- **Sleep** : 인자로 주어진 cycle만큼 sleep한 후에 대기상태로 들어간다.
  - n cycle에서 sleep명령을 수행할 때, 인자 값이 10이면 그 Process는 n+10 cycle의 시작지점에서 대기상태로 들어간다.
  - 즉, scheduler.txt의 sleep list에서 10번 등장한다.
  - 마지막 명령이 sleep인 경우에는 해당 명령을 실행하지 않고 바로 종료한다. 단, scheduler.txt와 memory.txt에는 해당 명령을 출력한다.
- **IOwait** : IO작업이 이루어질때까지 대기한다.
  - IO작업이 해당 Process에 이루어질 경우 프로세스를 Run queue에 삽입한다.
  - 마지막 명령이 IO wait인 경우에는 해당 명령을 실행하지 않고 바로 종료한다. 단, scheduler.txt와 memory.txt에는 해당 명령을 출력한다.

- **기타 조건**

- Cycle은 0 Cycle부터 시작한다.
- Process 종료시 해당 프로세스의 메모리는 모두 반환되어야 한다. 즉, Physical Memory에 해당 Process가 가지고 있는 Frame들은 모두 할당 해제되어야 한다.
- 해당 프로세스가 여러 메모리를 할당받은 상태로 종료되어 여러 메모리가 동시에 반환된다면 Page Table의 앞부터 입력되어있는 Allocation ID 순서대로 해제한다.
- 메모리를 해제 할 때, Virtual Memory의 Page ID와 valid bit 값만 바꾸는 것이 아니라 Physical Memory안의 값들도 해제해야 한다.
- 프로세스의 code나 page table, Run queue 등의 내용은 Physical Memory에 올라가지 않는다고 가정한다. 요청받은 Memory Allocation만 신경쓰면 된다.
- 메모리 할당과 관련하여, 0 페이지 할당이나 음수 페이지 할당 등은 고려하지 않아도 된다.
- 시뮬레이터 프로그램의 종료는 인풋으로 주어진 모든 작업이 끝났을 때이다.
- **한 Cycle이 시작되고 나서 진행되는 작업들의 우선순위는 다음과 같다.**
  - Sleep된 프로세스의 종료 여부 검사 (Sleep이 끝난 프로세스는 즉시 Run queue 맨 뒤에 삽입)
  - Input으로 주어진 IO 작업의 시행 (IO 작업이 시행된 프로세스는 즉시 Run queue 맨 뒤에 삽입)
  - Input으로 주어진 Process 생성 작업의 시행 (해당 프로세스는 즉시 Run queue 맨 뒤에 삽입)
  - 이번 Cycle에 실행 될 Process 결정
    - 현재 수행중인 Process가 없는 경우, 스케줄러가 실행될 Process를 결정
    - 현재 수행중인 Process가 있는 경우, 현재 Process보다 높은 priority를 갖는 Process가 있는지 확인.
      - 현재 더 높은 Priority를 갖는 run queue에 대기중인 Process가 있다면, 해당 Process로 교체.
  - 해당 Cycle에 수행할 Process의 명령이 무엇인지 확인하여 명령을 실행
  - **해당 Cycle 수행에 대한 scheduler.txt와 memory.txt 정보 출력**
  - 예시 (Process1과 Process2가 같은 priority)
    - 1 cycle에서 Process1 생성 후 9 cycle만큼 sleep하는 코드 실행, 그 후 이 결과를 반영하는 내용 출력
    - 2 cycle에서 Process2 생성 후 IOwait하는 코드 실행
    - 10 cycle에서
      - sleep이었던 Process1을 Run queue의 맨 뒤에 삽입
      - input 파일에 의한 Process2에 대한 IO명령 실행되어 Process2를 Run queue의 맨 뒤에 삽입
      - input 파일에 의한 Process3 생성, Process3은 Run queue의 맨 뒤에 삽입
      - 스케줄러에 의해 Process1이 선택 (알고리즘에 따라 다를 수 있음)
      - Process1 코드 실행
      - scheduler.txt와 memory.txt 정보 출력

## 프로그램 실행 방법

- 프로그램은 “파일 입출력”을 통해 입력값을 받고 결과를 파일로 출력한다.
- 프로그램 실행 시 -dir 옵션을 지정할 수 있다.
  - 인풋 파일이 위치하는 디렉토리 및 출력 파일이 위치할 디렉토리를 설정한다. (반드시 상대경로가 아닌 절대경로)
  - 기본값은 project3 프로그램이 위치한 디렉토리이다.
- 프로그램 실행 시 -page 옵션을 지정할 수 있다.
  - 시뮬레이터의 페이지 교체 알고리즘이 어떤 알고리즘으로 동작할지를 명시한다.
  - 기본값은 lru이다.

### • 실행 예시

```
./project3 -page=lru -dir=/home/os/input  
./project3
```

## 입력 파일 구성

- 입력은 하나의 [input]과 여러 개의 [Code]으로 이루어진다. **실제 파일 구성은 아래 예시 입력을 참고하자.**
  - 메인 input 파일의 이름은 ‘input’이며 여러 code 파일의 이름은 ‘input’ 안에서 주어진다. (“input.txt”가 아님에 주의)
- 이 입력 파일들의 경로는 프로그램 실행 시 -dir 옵션으로 주어진다.
- [input] 파일의 구성
  - 첫 번째 줄에는 다음 내용이 `[Total event num] [VM size(byte)] [PM size(byte)] [Page/Frame size(byte)]`의 형태로 한 칸씩 공백을 두고 순서대로 주어진다.
    - Total even num: 시스템이 종료될 때까지 발생할 이벤트(작업)의 수
    - VM size(byte): Virtual Memory의 크기
    - PM size(byte): Physical Memory의 크기
    - Page/Frame size(byte): Page(Frame)의 크기
  - 조건
    - Virtual/Physical Memory의 크기와 Page(Frame)의 크기는 모두  $2^n$ 이다.
    - Virtual/Physical Memory의 크기는 항상 Page(Frame)의 크기의 4의 정수배이다.
  - 두 번째 줄부터는 실행과 관련된 내용이 주어진다. Instruction은 실행 코드인지, I/O 작업 코드인지로 구분된다. 단, 입력 파일의 [Time]은 오름차순으로 정렬되어 있다고 가정한다. 이 때, 같은 Time에 여러 작업이 들어올 수도 있다.
    - 실행 코드의 경우 `[Time] [Code] [Priority]`의 형태로 입력이 주어진다.
      - [Time] : [Code]가 실행되는 CPU cycle time
      - [Code] : 실행될 Code(파일)의 이름, 최대 20글자
        - 예를 들어 파일 이름이 program1이면, -dir 옵션으로 주어진 디렉토리 안에서 “program1”이라는 파일을 읽어야 한다.
        - 프로세스는 생성되는 순서대로 PID가 부여되며, PID는 0부터 시작한다. 한 번 사용된 ID는 재사용되지 않는다.
      - [Priority] : 실행될 코드의 우선순위, 0부터 9까지의 정수
    - I/O 작업의 경우 `[Time] "INPUT" [PID]`의 형태로 입력이 주어진다.
      - [Time] : IO가 발생하는 CPU cycle time
      - [PID] : 대상 Process ID
  - 예시
    - 3 1024 ... => 총 3개의 작업이 존재, VM 크기가 1024 ...
    - 2 abc 7 => 2 CPU cycle에 ‘abc’의 코드가 우선순위 7인 process로 생성됨 (PID는 0, 스케줄러는 RR)
    - 4 def 1 => 4 CPU cycle에 ‘def’의 코드가 우선순위 1인 process로 생성됨 (PID는 1, 스케줄러는 FCFS)
    - 5 INPUT 1 => 5 CPU cycle에 PID 1인 Process에 IO가 발생

- [Code] 파일의 구성

- [Code]의 첫 줄에는 해당 코드의 명령어 개수가 주어진다.
- [Code]의 2번째 줄부터 '[Opcode] [Argument]'가 주어진다.
  - [Opcode] : 명령어의 종류
    - 0: Memory Allocation
    - 1: Memory Access
    - 2: Memory Release
    - 3: Non-memory instruction
    - 4: Sleep
    - 5: IOwait
  - [Argument]
    - Opcode가 0일때: Memory Allocation의 경우 요청하는 Page 개수
    - Opcode가 1일때: Memory Access의 경우 접근 요청하는 Page ID
    - Opcode가 2일때: Memory Release의 경우 해제 요청하는 Page ID
    - Opcode가 3일때: Non-memory instruction의 경우 항상 0
    - Opcode가 4일때: Sleep의 경우 잠들 cycle의 수
    - Opcode가 5일때: IOwait의 경우 항상 0

- 예시

- 8 => 명령어가 총 8개인 code
- 0 5 => 연속된 5개의 page를 memory에 allocation 요청
- 3 0 => memory 사용안함
- 4 7 => 7 cycle 후에 Run queue의 맨 뒤에 삽입됨
- 1 0 => Page ID가 0인 page에 접근요청
- 2 0 => Page ID가 0인 page를 Memory와 page table에서 할당해제
- 5 0 => IO작업이 올때까지 대기
- Memory Release 시, 이전에 할당했던 Page의 개수만큼 해제한다. 예를 들어 과거 0번 Page ID를 갖는 5개의 Page를 할당 했을 때, 0번에 대한 Release 명령이 들어온다면 반드시 이전에 할당했던 5개의 Page 모두 해제한다.

## 입력 예시

- 실행 옵션

```
./project3 -dir=/home/os/hw2/input
```

- 파일: /home/os/hw2/input/input

```
5 2048 1024 32
0 program1 6
5 program2 3
8 INPUT 0
9 program3 6
15 INPUT 1
```

- 파일: /home/os/hw2/input/program1

```
11
0 16
0 12
0 22
0 14
5 0
1 1
1 1
1 2
1 3
1 3
1 1
```

- 파일: /home/os/hw2/input/program2

```
20
3 0
3 0
5 0
0 11
0 9
0 20
0 10
0 14
1 0
1 1
1 0
3 0
3 0
3 0
3 0
3 0
1 1
1 3
4 2
2 3
```

- 파일: /home/os/hw2/input/program3

```
1
3 0
```

## 출력 형식

- 스크립트를 이용한 자동채점이 진행되므로 아래의 형식을 정확하게 따라야 한다. 이에 따르지 않는 경우 점수에 불이익이 있을 수 있으며, “반드시” 이를 지켜야만 한다.
- 출력 파일은 -dir 옵션으로 주어진 위치에 생성되어야 한다.
  - 출력 파일의 이름은 각각 “scheduler.txt”, “memory.txt”이다.
  - (예시) /home/os/hw2/input 디렉토리에 “scheduler.txt”, “memory.txt” 파일 생성
- scheduler.txt
  - 스케줄러에 의해 스케줄링이 수행된 기록과 시뮬레이터 시스템의 전반적인 모든 활동을 출력한다.
  - 매 Cycle이 지날 때마다 출력한다. 자세한 출력 내용은 아래와 같다.
  - 해당 Cycle에서 프로세스가 명령을 실행한 결과를 출력한다. (한 Cycle내의 작업 우선순위 참조)
    - 단, Running process 관련 line에서는 해당 cycle에서 어떤 코드를 실행시켰는지를 출력한다.
  - Line 1: 스케줄러가 스케줄링을 수행한 기록을 출력한다
    - [[Time] Cycle] Scheduled Process: [Process ID] [Process name]의 형태로 출력한다.
      - [Time] : schedule이 된 시점의 CPU cycle
      - [Process ID] : schedule된 프로세스의 ID
      - [Process name] : Process의 이름(Process가 실행된 Code의 이름)
    - 스케줄링된 Process가 없을 경우에는
      - [[Time] Cycle] Scheduled Process: None을 출력
  - 주의사항
    - (RR) run queue에 대기중인 프로세스가 없어 한 프로세스가 지속적으로 CPU를 사용할 때, time quantum을 다 사용하여 time quantum이 초기화 되었다면 새롭게 scheduling 된 것이다. 즉, 현재 실행중인 프로세스는 run queue에 한 번 등장해야 하며, scheduled process에도 이를 기록해야 한다.

```
// Line 1
fprintf(out, "[%d Cycle] Scheduled Process: ", cycle);
if (Scheduled Process Exists)
    fprintf(out, "%d %s (priority %d)\n", pid, name, priority);
else
    fprintf(out, "None\n");
```

- Line 2: 이번 Cycle에 실행된 Process의 ID 및 Code의 정보
  - 실행되고 있는 Process가 없는 경우, None을 출력한다.

```
// Line 2
fprintf(out, "Running Process: ");
if (Running Process Exists)
    fprintf(out, "Process#%d(%d) running code %s line %d(op %d, arg %d)\n"
              , pid, priority, name, line, op, arg);
else
    fprintf(out, "None\n");
```

- Line 3: Run queue 현황
  - Priority 별로 하나씩 출력한다. (총 10개)
  - Run queue에 Process가 없는 경우 “Empty”를 출력한다.
  - Run queue에 먼저 들어온 프로세스부터 출력한다.

```
// Line 3
for each RunQueue
    fprintf(out, "RunQueue %d: ", priority);
    if (RunQueue is Empty)
        fprintf(out, "Empty");
    else
        for each node
            fprintf(out, "%d(%s) ", pid, name);
    fprintf(out, "\n");
```

o Line 4: Sleep List 현황

- Sleep중인 Process가 없는 경우 “Empty”를 출력한다.
- Sleep List에 먼저 들어온 프로세스부터 출력한다.

```
// Line 4
fprintf(out, "SleepList: ");
if (SleepList is Empty)
    fprintf(out, "Empty");
else
    for each node
        fprintf(out, "%d(%s) ", pid, name);
fprintf(out, "\n");
```

o Line 5: IOWait List 현황

- IOWait중인 Process가 없는 경우 “Empty”를 출력한다.
- IOWait List에 먼저 들어온 프로세스부터 출력한다.
- 매 Cycle마다 위의 5 Line을 출력한 후, 개행 문자(‘\n’)를 출력하여 다음 Cycle 결과와 구분될 수 있도록 한다.

```
// Line 5
fprintf(out, "IOWait List: ");
if (IOWaitList is Empty)
    fprintf(out, "Empty");
else
    for each node
        fprintf(out, "%d(%s) ", pid, name);
fprintf(out, "\n");

// 매 사이클이 끝나면 다음 사이클과 구분을 위해 개행문자 필요
fprintf(out, "\n");
```

- 출력 예시

```
[0 Cycle] Scheduled Process: 0 program1 (priority 6)
Running Process: Process#0(6) running code program1 line 1(op 0, arg 16)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[1 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 2(op 0, arg 12)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[2 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 3(op 0, arg 22)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[3 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 4(op 0, arg 14)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[4 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 5(op 5, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 0(program1)

[5 Cycle] Scheduled Process: 1 program2 (priority 3)
Running Process: Process#1(3) running code program2 line 1(op 3, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
```

```
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 0(program1)

[6 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 2(op 3, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 0(program1)

[7 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 3(op 5, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 0(program1) 1(program2)

[8 Cycle] Scheduled Process: 0 program1 (priority 6)
Running Process: Process#0(6) running code program1 line 6(op 1, arg 1)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 1(program2)

[9 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 7(op 1, arg 1)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: 2(program3)
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 1(program2)

[10 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 8(op 1, arg 2)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: 2(program3)
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
```

```
SleepList: Empty
IOWait List: 1(program2)

[11 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 9(op 1, arg 3)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: 2(program3)
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 1(program2)

[12 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 10(op 1, arg 3)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: 2(program3)
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 1(program2)

[13 Cycle] Scheduled Process: None
Running Process: Process#0(6) running code program1 line 11(op 1, arg 1)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: 2(program3)
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 1(program2)

[14 Cycle] Scheduled Process: 2 program3 (priority 6)
Running Process: Process#2(6) running code program3 line 1(op 3, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: 1(program2)

[15 Cycle] Scheduled Process: 1 program2 (priority 3)
Running Process: Process#1(3) running code program2 line 4(op 0, arg 11)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[16 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 5(op 0, arg 9)
RunQueue 0: Empty
RunQueue 1: Empty
```

```
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[17 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 6(op 0, arg 20)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[18 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 7(op 0, arg 10)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[19 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 8(op 0, arg 14)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[20 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 9(op 1, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[21 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 10(op 1, arg 1)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
```

```
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[22 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 11(op 1, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[23 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 12(op 3, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[24 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 13(op 3, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[25 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 14(op 3, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[26 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 15(op 3, arg 0)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[27 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 16(op 3, arg 0)
RunQueue 0: Empty
```

```
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[28 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 17(op 1, arg 1)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[29 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 18(op 1, arg 3)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: Empty
IOWait List: Empty

[30 Cycle] Scheduled Process: None
Running Process: Process#1(3) running code program2 line 19(op 4, arg 2)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: 1(program2)
IOWait List: Empty

[31 Cycle] Scheduled Process: None
Running Process: None
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
RunQueue 8: Empty
RunQueue 9: Empty
SleepList: 1(program2)
IOWait List: Empty

[32 Cycle] Scheduled Process: 1 program2 (priority 3)
Running Process: Process#1(3) running code program2 line 20(op 2, arg 3)
RunQueue 0: Empty
RunQueue 1: Empty
RunQueue 2: Empty
RunQueue 3: Empty
RunQueue 4: Empty
RunQueue 5: Empty
RunQueue 6: Empty
RunQueue 7: Empty
```

```
RunQueue 8: Empty  
RunQueue 9: Empty  
SleepList: Empty  
IOWait List: Empty
```

- memory.txt

- 가상, 물리 메모리의 현황과 Instruction 기록을 출력한다. 매 cycle마다 메모리 정보를 출력한다.
  - 예시 출력 파일을 참고하여 아래에 자세히 서술된 출력 포맷대로 출력을 한다. 양식을 지키지 않으면 채점이 불가능할 수 있음을 유의한다.
  - 출력 파일의 맨 마지막 줄에는 시뮬레이션에서 발생한 page fault의 개수를 출력한다.
  - 다음은 예제 입력에 대한 시뮬레이션 결과이다.
    - Physical Memory현황을 출력한다
      - Frame 단위로 Process의 ID와 Allocation ID를 출력한다.
      - 나누어진 Block 단위로 구분자('l')가 들어가야 한다.
    - 각 가상 메모리의 현황을 프로세스마다 출력한다.
      - Page 단위로 각 Page의 Page ID (PID)를 출력하며, 매 4번째 Page마다 구분기호로 “|”를 삽입한다.
      - Page 단위로 각 Page의 Allocation ID (AID)를 출력하며, 매 4번째 Page마다 구분기호로 “|”를 삽입한다.
      - Page 단위로 각 Page의 Valid bit (Valid)을 출력하며, 매 4번째 Page마다 구분기호로 “|”를 삽입한다.
      - Page 단위로 각 Page의 Reference bit (Ref)을 출력하며, 매 4번째 Page마다 구분기호로 “|”를 삽입한다.
        - 단, 기본 LRU에 경우에는 항상 Reference bit가 비어있다.
        - Sampled LRU의 reference byte는 reference bit와 별개이므로 따로 출력되지 않는다.
    - 페이지 테이블과 물리 메모리를 출력할때 왼쪽을 하위 주소로 표현한다.

```
[7 Cycle] Input: Pid[1] Function[IOWAIT]
>> Physical Memory: |---|---|---|---|---|---|---|
>> pid(0) Page Table(PID): |0000|0000|0000|0000|1111|1111|1111|2222|2222|2222|2222|2222|2233|3333|3333|3333|
>> pid(0) Page Table(AID): |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
>> pid(0) Page Table(Valid): |0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|0000|
>> pid(0) Page Table(Ref): |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
>> pid(1) Page Table(PID): |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
>> pid(1) Page Table(AID): |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
>> pid(1) Page Table(Valid): |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
>> pid(1) Page Table(Ref): |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
```





- 입력의 각 줄을 실행 후 결과를 위와 같이 출력한다. **채점을 위해 출력 형태는 아래의 fprintf문을 반드시 사용하여 형태를 맞춘다. 아래 fprintf 문을 사용하지 않아 출력 형태가 맞지 않을 경우 감점.**
- **주의사항**
  - 프로세스의 정보는 프로세스가 생성된 cycle부터 종료된 cycle까지 출력한다.
  - 프로세스가 종료되면 해당 프로세스가 소유하던 메모리는 모두 반환되어야 한다.
  - 프로세스가 종료된 cycle 이후에는 해당 프로세스의 Page Table이 출력되지 않는다.
- **Line 1 : 입력 파일의 각 줄에 대한 정보**
  - Function은 출력 예시처럼 대문자로 표기한다
    - ALLOCATION, ACCESS, RELEASE, NON-MEMORY, SLEEP, IOWAIT, NO-OP
    - NO-OP는 해당 Cycle에 어떤 명령도 실행되지 않을 때 출력한다.
  - 출력 포맷(ALLOCATION, ACCESS, RELEASE) :
    - Page number는 해당 Page ID에 해당하는 페이지의 개수

```
fprintf(out, "[%d Cycle] Input: Pid[%d] Function[%s] Page ID[%d] Page Num[%d]\n", 각 Parameter...);
```
  - 출력 포맷(NON MEMORY, SLEEP, IOWAIT) :
 

```
fprintf(out, "[%d Cycle] Input: Pid[%d] Function[%s]\n", 각 Parameter...);
```
  - 출력 포맷(NO-OP) :
 

```
fprintf(out, "[%d Cycle] Input: Function[NO-OP]\n");
```

    - 각 Parameter는 입력 파일의 각 줄을 읽어서 실행결과 화면과 같이 출력한다.
- **Line 2 : 현재 물리 메모리 현황**
  - 출력 포맷 :
 

```
fprintf(out, "%-30s", ">> Physical Memory: ");
fprintf(out, 현재 Physical Memory 현황 출력);
```
  - 현재 물리 메모리 현황 : 각 현황을 Frame 단위로 출력한다. 각 숫자는 해당 Frame에 할당된 Page의 Allocation ID를 나타낸다. 만약 해당 Frame에 할당된 Page가 없다면 “-”를 출력, 4 Frame당 하나의 “|”를 출력하여 눈으로 인지할 수 있도록 한다.
- 그 다음 줄부터는 프로세스별로 두 줄씩 출력한다. 프로세스의 번호가 작은 것부터 출력한다.
- 프로세스의 첫번째 줄과 두번째 줄: 현재 페이지 테이블의 현황
  - 출력 포맷
 

```
fprintf(out, ">> pid(%d)%-20s", pid, " Page Table(PID): ");
fprintf(out, 현재 Page Table 현황 출력);
fprintf(out, ">> pid(%d)%-20s", pid, " Page Table(AID): ");
fprintf(out, 현재 Page Table 현황 출력);
fprintf(out, ">> pid(%d)%-20s", pid, " Page Table(Valid): ");
fprintf(out, 현재 Page Table 현황 출력);
fprintf(out, ">> pid(%d)%-20s", pid, " Page Table(Ref): ");
fprintf(out, 현재 Page Table 현황 출력);
```
  - 첫번째 줄에는 가상 메모리의 현황을 출력 형태와 같이 Page 단위로 출력한다. 각 숫자는 Page ID를 의미한다.
  - 두번째 줄에는 가상 메모리의 현황을 출력 형태와 같이 Page 단위로 출력한다. 각 숫자는 해당 Page가 할당되어 있다면 할당된 Allocation ID를 뜻한다. Allocation ID를 부여받지 못한 페이지의 경우에는 -로 출력한다.
  - 세번째 줄의 각 숫자는 할당된 각 페이지가 실제 물리 메모리에 할당되어 있는지에 대한 여부를 나타낸다. 해당 페이지가 물리 메모리에 할당되어 있다면 1, 아니면 0을 출력한다. 페이지가 없는 가상 메모리 공간만 -로 나타낸다.
  - 네번째 줄의 각 숫자는 Reference bit를 나타낸다. 기본 LRU의 경우에는 Reference bit가 없으므로 -로 출력한다. 다른 두 알고리즘의 경우에는 페이지의 Reference bit는 항상 0 아니면 1이다. 페이지가 없는 가상 메모리 공간만 -로 나타낸다.
- 위의 출력 이후 한줄 개행 문자 `fprintf(out, "\n");`을 추가로 출력하여 다음 입력 줄 처리 결과와 구분될 수 있도록 한다.
- 프로그램의 맨 마지막줄: 시뮬레이션 과정에서 발생한 총 페이지 폴트의 수 (물리 메모리에 해당 페이지가 없을 때 발생)
  - 출력 포맷 :
 

```
fprintf(out, "page fault = %d\n", page_fault);
```

## 보고서 과제

보고서는 자유 형식으로 작성하되, 아래의 내용을 반드시 포함하도록 한다.

- ✓ 작성한 프로그램의 동작 과정과 구현 방법
  - 소스 코드 출별 설명보다는 전체적인 동작 방식에 대한 설명
  - 각 알고리즘에 대하여 순서도나 블록 다이어그램 등 도식화 자료를 반드시 이용할 것
- ✓ 개발 환경 명시
  - uname -a 실행 결과, CPU, 메모리 정보 등
- ✓ 결과 화면 스크린샷과 그것에 대한 토의 내용 (중요!)
  - 자신이 만든 3가지 이상의 다른 input 파일로 스케줄러와 페이지 교체 알고리즘의 동작 확인
  - 이 스케줄러에 어떤 한계가 있는지 토의
  - 각 페이지 교체 알고리즘의 성능을 비교 분석
  - input 파일 스크린샷, 그래프, 그림, 또는 도표 등을 반드시 활용하여 결과 분석을 성의있게 할 것
- ✓ 과제 수행 시 겪었던 어려움과 해결 방법

보고서 작성 관련 유의사항은 다음과 같다.

- ✓ 자신의 소스 코드 전체를 첨부하지 말 것 (소스 코드의 특정 부분을 설명하기 위한 일부 첨부는 허용)
- ✓ 그림을 비롯한 참조 내용은 반드시 출처를 명시할 것
- ✓ 항목을 분명하게 나누어 명료하게 서술하는 것이 좋다

## 과제 제출 방법

보고서 pdf 파일(hw3\_[학번].pdf)과 소스 압축 파일(hw3\_[학번].tar)을 제출한다.

### 보고서 파일(hw3\_[학번].pdf)

PDF 파일로 변환하여 제출한다. 파일의 이름은 반드시 hw3\_[학번].pdf로 한다.

### 소스 압축 파일(hw3\_[학번].tar)

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

```
# 본인이 작업하는 디렉토리가 hw3라고 가정하자.  
# 작업하는 디렉토리가 hw3이라는 것은 hw3 폴더 바로 안에 Makefile이 있음을 의미한다.  
# 본인의 학번이 2021123123 이라고 가정하면, 아래의 일련의 명령어를 수행한 이 후에 hw3_2021123123.tar을 제출한다.  
# 아래의 일련의 명령어를 수행한 이 후에 hw3_2021123123.tar을 제출한다.  
# 이는 압축을 해제하였을 때 본인의 학번으로 된 폴더 하나가 나타나야 하고, 그 폴더 바로 아래에 Makefile이 있어야 함을 의미한다.  
$ cd ~/hw3/..  
$ mv hw3 2021123123  
$ tar cvf hw3_2021123123.tar 2021123123
```

## 참고 사이트

[https://en.wikipedia.org/wiki/Virtual\\_memory](https://en.wikipedia.org/wiki/Virtual_memory)

<https://en.wikipedia.org/wiki/Paging>

[https://en.wikipedia.org/wiki/Buddy\\_memory\\_allocation](https://en.wikipedia.org/wiki/Buddy_memory_allocation)