2022년 1학기 운영체제 과제 #3

Last update: 2022-05-23

과제 개요

운영 체제는 실제 메모리의 용량보다 큰 가상 주소 공간을 제공하여, 실행되는 프로세스들이 물리적으로 존재하는 것보다 많은 메모리를 참조할 수 있게 한다. 이러한 기법을 가상 메모리(Virtual Memory)라고 한다. 이번 과제에서는 이러한 가상 메모리에서 주로 사용되는 메모리 관리 기법인 페이징(Paging)과 버디 시스템(Buddy System)에 대해 알아보고, 단순한 형태의 시뮬레이터를 제작한다.

1 과제 목적

- ✓ 가상 메모리의 구조와 동작에 대해 이해한다.
- ✓ 메모리 관리 기법인 페이징과 버디 시스템을 이해한다.

2 제출 기한

- ✓ 2022년 6월 10일(금) 오후 6:00 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 환경 OS: Ubuntu 20.04, 채점 환경 컴파일러: gcc 9.4.0, 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 주석**을 달 것, 주석의 내용이 불분명하거나 없을 경우 감점 처리
- ✓ 과제에 대한 문의 사항이 있을 경우 learnUs 질문 게시판으로 문의 **(반드시 공개 질문으로 올릴 것)**
- ✓ **이 과제 명세에 대한 내용을 웹 사이트에 배포하거나 질문한 것이 발각되는 경우 0점 처리**

과제 세부 사항

과제는 보고서 과제와 실습 과제로 이루어져 있다.

보고서 과제

보고서는 자유 형식으로 작성하되, 아래의 내용을 반드시 포함하도록 한다.

- ✓ 작성한 프로그램의 동작 과정과 구현 방법
  - 소스 코드 줄별 설명보다는 전체적인 동작 방식에 대한 설명
  - 각 알고리즘에 대하여 **순서도나 블록 다이어그램 등 도식화 자료**를 반드시 이용할 것
- ✓ 개발 환경 명시
  - uname -a 실행 결과, CPU, 메모리 정보 등
- ✓ **결과 화면 스크린샷과 그것에 대한 토의 내용 (중요!)**
  - 자신이 만든 다양한 페이지 교체 알고리즘의 동작 확인
  - 각 페이지 교체 알고리즘의 성능을 비교 분석
  - input 파일 스크린샷, 그래프, 그림, 또는 도표등을 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 코드 의 특정 부분을 설명하기 위한 일부 첨부는 허용)
- ✓ 그림을 비롯한 참조 내용은 반드시 출처를 명시할 것
- ✓ 항목을 분명하게 나누어 명료하게 서술하는 것이 좋다

실습 과제

실습 과제에서는 버디 시스템과 페이징이 결합된 가상의 시스템을 시뮬레이션한다.

시뮬레이터 프로그램

- 시뮬레이터 프로그램은 입력으로 프로세스가 사용하는 메모리 연산(Memory Operation)이 들어오고 명령어 한개 단위로 연산을 처리하는 과정을 출력한다.
- 본 시스템의 전체 구성은 아래와 같다.
- 프로세스는 독자적인 **N** bytes 크기의 가상 메모리(Virtual Memory) 공간을 갖는다. **N**은 32의 배수이다.
- 본 시스템은 **M** bytes 크기의 물리 메모리(Physical Memory)가 존재한다. **M**은 32의 배수이며, 2의 거듭제곱이다.
- 프로세스의 가상 메모리는 **32 bytes의 페이지 단위**로 나뉘어 관리되며, 물리 메모리 또한 **32 bytes 프레임 단위**로 관리된다.
    - 가상 메모리 할당의 경우 요청하는 크기가 들어가는 첫 번째 공간으로 할당한다.
    - 가상 메모리는 하위 주소부터 메모리를 채워나간다.
    - 각 프로세스의 페이지 테이블(Page Table)은 각 페이지 별로 다음 세가지 정보를 갖는다.
        - (1) 해당 페이지가 프로세스의 몇번째 가상 메모리 할당 연산으로 생성되었는지를 나타내는 Allocation ID (AID)
            - Allocation ID는 프로세스별로 관리하며, 시스템 전체에서는 프로세스가 다르면 중복될 수 있다.
            - Allocation ID는 새로 할당되는 순서대로 0부터 ID가 부여된다. 한 번 사용된 ID는 재사용되지 않는다.
            - 예를 들어, Process 1은 AID가 0, 1, 2, … 순서로 할당되고, Process 2도 AID가 0, 1, 2, …로 할당된다.
        - (2) 해당 페이지 영역이 물리 메모리의 프레임에 존재하는지를 나타내는 Valid bit
            - Valid bit는 물리 메모리의 프레임에 존재하지 않으면 0, 존재하면 1이다.
            - 같은 AID로 엮인 페이지 영역이라면 같은 Valid bit 값을 갖는다.
        - (3) 해당 페이지 영역이 물리 메모리의 어느 프레임부터 시작하는지를 나타내는 Frame index (FI)
            - Frame index는 0부터 (M/32)-1 까지 존재한다.
            - Valid bit가 1인 경우에만, Frame index가 존재한다.
            - 같은 AID로 엮인 페이지 영역이라면 같은 Frame index를 갖는다.
- 프로세스는 다음과 같은 **세가지 메모리 연산**을 수행할 수 있다:
    - **(1) 메모리 할당 (Memory Allocation;** Operation ID = 0**)**
        - 프로세스가 원하는 연속된 페이지 개수 만큼 가상 메모리에 할당 연산을 수행한다.
            - 반드시 비어있는 연속적인 공간에 할당하며, 하위 메모리 주소부터 할당한다.
        - 인자로 (1) Process ID와 (2) 몇개의 페이지를 할당할 것인지가 주어진다.
            - 가상 메모리의 전체 크기나 물리 메모리의 전체 크기보다 많은 페이지를 요구하는 경우는 고려하지는 않는다.
        - 메모리 할당 연산의 경우, 물리 메모리에 프레임을 할당하지 않고 가상 메모리에만 할당한다.
        - 메모리 할당 연산의 결과는 페이지 테이블에 기록된다.
        - 프로세스는 원하는 페이지의 개수 만큼을 페이지 테이블에 할당하고, 해당 페이지 영역은 Allocation ID를 통해 관리한다.
    - **(2) 메모리 접근 (Memory Access;** Operation ID = 1**)**
        - 프로세스가 이전에 할당한 페이지와 그것에 연결된 프레임에 접근(Access)한다.
        - 인자로 (1) Process ID와 (2) Allocation ID가 주어진다.
            - 이 Allocation ID를 통해 어떤 페이지 영역에 메모리 접근 연산을 사용할지가 결정된다.
            - 해당 Process ID에 존재하지 않는 Allocation ID에 접근하는 경우는 고려하지 않는다.
        - 페이지에 할당된 프레임이 없는 경우에는 **버디 시스템 (Buddy System)**을 사용하여 할당해야 할 프레임의 수를 계산한 뒤, 물리 메모리 상에서 연속된 프레임을 할당한다.
            - 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 시스템은 작은 크기를 갖는 공간에 우선순위가 생긴다. 같은 크기를 갖는 공간이 여러개일 때에는 낮은 주소를 갖는 영역부터 메모리를 채워나간다.
        - 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.
    - **(3) 메모리 해제 (Memory Release; ;** Operation ID = 2**)**
        - 프로세스가 Allocation ID를 기준으로 해당 페이지 영역을 가상 메모리에서 해제한다.
        - 인자로 (1) Process ID와 (2) Allocation ID가 주어진다.
            - 해당 Process ID에 존재하지 않는 Allocation ID를 해제하는 경우는 고려하지 않는다.
        - 해당 페이지가 물리 메모리에도 존재하는 경우, 물리 메모리의 프레임도 해제한다.

- 본 시스템은 다음과 같은 **일곱가지 페이지 교체 알고리즘(Page Replacement Algorithms)**중 하나를 사용한다:
  - **(1) FIFO 알고리즘** (-page=fifo)
  - **(2) Stack 기반 LRU 알고리즘** (-page=stack)
  - **(3) Sampled LRU 알고리즘** (-page=sampled)
    - Reference bit가 갱신되는 time interval은 8번의 명령어 입력으로 한다.
      - Reference bit와 reference byte는 물리 메모리에 존재하는 페이지만 관리한다.
    - 한 Time Interval내에 Memory Access가 발생한 페이지의 경우 **Reference bit**가 1이 된다.
    - Reference byte는 강의노트에 있는 것처럼 1 byte (8 bits)로 한다.
  - **(4) Second-Chance Clock 알고리즘** (-page=second-chance)
    - Clock은 Circular Queue를 통해 관리된다.
      - 물리 메모리에 할당된 순서대로 Circular Queue에 존재한다.
      - Clock은 Circular Queue를 기준으로 순회한다.
    - Memory Access가 발생하면 Reference bit가 1이 된다.
    - Clock이 해당 페이지를 가리킬 때 Reference bit가 1이면 0이 되고 다음 페이지 영역으로 넘어간다.
    - Clock이 해당 페이지를 가리킬 때 Reference bit가 0이면 해당 페이지 영역을 교체시킨다.
    - Clock의 탐색은 Clock이 마지막으로 참조했던 Page에서부터 시작한다.
      - 마지막으로 참조했던 Page가 교체되면, 그 다음 Page에서부터 시작한다.
      - 예를 들어, Circular Queue가 [Page 0, Page 3, Page 5, Page 7]로 있다고 하자.
      - Page 5가 교체되고 Page 6이 새로 들어오면, Circular Queue는 [Page 0, Page 3, Page 7, Page 9]가 된다. 이후의 Clock 탐색은 Page 7에서부터 시작한다.
  - **(5) LFU 알고리즘** (-page=lfu)
  - **(6) MFU 알고리즘** (-page=mfu)
  - **(7) Optimal 알고리즘** (-page=optimal)
    - 강의노트에 서술되어 있는 대로 "the one that will not be used for the longest period of time"를 선택해서 교체한다.
    - 이는 모든 경우의 수를 탐색해 얻은 page fault rate가 최소가 되는 해와 다를 수 있다.
  - 알고리즘 공통 조건
    - 프레임을 해제할 때에는 같은 프로세스의 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, 해당 Frame index가 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, 먼저 물리 메모리에 할당된 영역을 교체한다. (FIFO)
    - 각 알고리즘에 대한 자세한 설명은 강의노트를 참고하도록 한다. (L9-virtualmemory)
    - 교체되어 나가는 페이지 영역의 모든 정보는 초기화된다.
      - Reference bit, reference byte, counter 정보 등
    - 각 페이지의 정보 갱신은 명령어 실행 이후에 이루어진다.
      - 예를 들어, reference bit의 갱신은 8번째 명령어 입력 실행 직후에 이루어진다.

프로그램 실행 방법

- 프로그램은 “표준 입출력”을 통해 입력값을 받고 결과를 출력한다.
- 프로그램 실행 시 **-page 옵션**을 지정한다.
    - 시뮬레이터의 페이지 교체 알고리즘이 어떤 알고리즘으로 동작할 지를 명시한다.
- **실행 예시**

```
./project3 -page=fifo
./project3 -page=stack < input > output
```

입력 구성

- 첫번째 줄에는 다음과 같은 기본적인 정보가 순서대로 주어지며, 각 정보는 공백을 통해 구분된다.
    - (1) 입력될 명령어의 수 (최대 10,000)
    - (2) 시뮬레이션에 사용될 프로세스의 수 (최대 10)
        - Process ID (PID)는 0부터 순서대로 부여된다. PID는 최대 9까지 존재 가능하다.
    - (3) 가상 메모리 크기 N
    - (4) 물리 메모리 크기 M
- 두번째 줄부터는 다음과 같이 명령어가 입력된다. 각 항목은 공백을 통해 구분된다.
    - [Operation ID] [Arguments …]
    - 예시
        - 0 0 10: PID 0인 프로세스에 10개의 페이지 할당
        - 1 0 0: PID 0인 프로세스의 Allocation ID가 0인 페이지에 접근
        - 2 0 0: PID 0인 프로세스의 Allocation ID가 0인 페이지를 가상 메모리에서 해제
- **입력 예시**

```
7 2 2048 1024
0 0 31
1 0 0
0 1 14
0 1 3
1 1 0
1 1 1
2 1 1
```

출력 구성

- 매 명령어 실행 이후마다 다음과 같은 규칙으로 현재 메모리 정보를 출력한다.
    - 각 명령어 실행 결과 사이에는 빈 줄을 하나씩 삽입하여 구분한다.
    - 물리 메모리 출력 (Line 1, Line 2)
        - 각 현황을 32 bytes 단위로 출력한다. 즉, 한 프레임 기준으로 출력한다.
        - 만약 해당 Frame에 할당된 Page가 없다면 “-“를 출력하고, 128 bytes 마다 “|”를 출력해 구분한다.

```
// Line 1
printf("%-30s", ">> Physical Memory (PID): ");
printf(각 Frame의 Process ID 출력);

// Line 2
printf("%-30s", ">> Physical Memory (AID): ");
printf(각 Frame의 Allocation ID 출력);
```

    - 프로세스 별로 가상 메모리 출력 (Line 3부터 Line {2 + 프로세스 수 * 3} 까지)
        - 각 현황을 32 bytes 단위로 출력한다. 즉, 한 페이지 기준으로 출력한다.
        - 빈 영역은 “-“를 출력하고, 128 bytes 마다 “|”를 출력해 구분한다.
        - 같은 Allocation ID를 갖는 페이지들은 Valid bit와 Frame index도 같다.

```
// Line {3 + PID * 3}
printf(">> PID(%d) %-20s", PID, "Page Table (AID): ");
printf(각 Page의 Allocation ID 출력);

// Line {4 + PID * 3}
```

```
printf(">> PID(%d) %-20s", PID, "Page Table (Valid): ");
printf(각 Page의 Valid bit 출력);

// Line {5 + PID * 3}
printf(">> PID(%d) %-20s", PID, "Page Table (FI): ");
printf(각 Page의 Frame index 출력);
```

- 모든 명령어 실행이 끝나면, 마지막 줄에는 시뮬레이션 동안 발생한 페이지 폴트 (Page fault)의 수를 출력한다.
    - 페이지 폴트는 Memory Acess 명령 시 물리 메모리에 해당 페이지가 없을 때 1회 발생한 것으로 한다.
        - 즉, 얼마나 많은 페이지가 교체되어 나갔는지와는 무관하다.
- **출력 예시**
    - 페이지 폴트는 3회 발생하였다.

```
>> Physical Memory (PID):     |----|----|----|----|----|----|----|----|
>> Physical Memory (AID):     |----|----|----|----|----|----|----|----|
>> PID(0) Page Table (AID):   |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (Valid): |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (AID):   |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (Valid): |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

>> Physical Memory (PID):     |0000|0000|0000|0000|0000|0000|0000|0000|
>> Physical Memory (AID):     |0000|0000|0000|0000|0000|0000|0000|0000|
>> PID(0) Page Table (AID):   |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (Valid): |1111|1111|1111|1111|1111|1111|1111|111-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (FI):    |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (AID):   |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (Valid): |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

>> Physical Memory (PID):     |0000|0000|0000|0000|0000|0000|0000|0000|
>> Physical Memory (AID):     |0000|0000|0000|0000|0000|0000|0000|0000|
>> PID(0) Page Table (AID):   |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (Valid): |1111|1111|1111|1111|1111|1111|1111|111-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (FI):    |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (AID):   |0000|0000|0000|00--|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (Valid): |0000|0000|0000|00--|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

>> Physical Memory (PID):     |0000|0000|0000|0000|0000|0000|0000|0000|
>> Physical Memory (AID):     |0000|0000|0000|0000|0000|0000|0000|0000|
>> PID(0) Page Table (AID):   |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (Valid): |1111|1111|1111|1111|1111|1111|1111|111-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (FI):    |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (AID):   |0000|0000|0000|0011|1---|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (Valid): |0000|0000|0000|0000|0---|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

>> Physical Memory (PID):     |1111|1111|1111|1111|----|----|----|----|
>> Physical Memory (AID):     |0000|0000|0000|0000|----|----|----|----|
>> PID(0) Page Table (AID):   |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (Valid): |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (AID):   |0000|0000|0000|0011|1---|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (Valid): |1111|1111|1111|1100|0---|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (FI):    |0000|0000|0000|00--|----|----|----|----|----|----|----|----|----|----|----|----|

>> Physical Memory (PID):     |1111|1111|1111|1111|1111|----|----|----|
>> Physical Memory (AID):     |0000|0000|0000|0000|1111|----|----|----|
>> PID(0) Page Table (AID):   |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (Valid): |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (AID):   |0000|0000|0000|0011|1---|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (Valid): |1111|1111|1111|1111|1---|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (FI):    |0000|0000|0000|001616|16---|----|----|----|----|----|----|----|----|----|----|----|

>> Physical Memory (PID):     |1111|1111|1111|1111|----|----|----|----|
>> Physical Memory (AID):     |0000|0000|0000|0000|----|----|----|----|
>> PID(0) Page Table (AID):   |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (Valid): |0000|0000|0000|0000|0000|0000|0000|000-|----|----|----|----|----|----|----|----|
>> PID(0) Page Table (FI):    |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (AID):   |0000|0000|0000|00--|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (Valid): |1111|1111|1111|11--|----|----|----|----|----|----|----|----|----|----|----|----|
>> PID(1) Page Table (FI):    |0000|0000|0000|00--|----|----|----|----|----|----|----|----|----|----|----|----|

3
```

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 2022123123 이라고 가정하면, 아래의 일련의 명령어를 수행한 이 후에 hw3_2022123123.tar을 제출한다.
# 아래의 일련의 명령어를 수행한 이 후에 hw3_2022123123.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 2022123123
$ tar cvf hw3_2022123123.tar 2022123123
```

참고 사이트

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

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

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