운영체제 과제 #3

3-1. Demand Paging

3-2. Demand Paging with 2-level Hierarchical Page Table

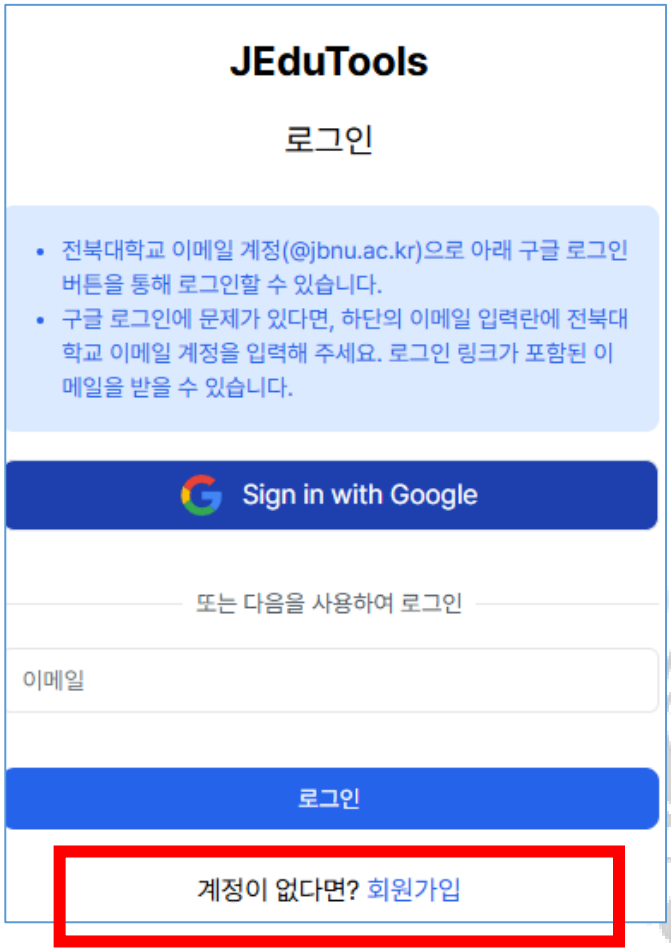
과제 내용 및 제출 방법

- 프로그램 작성: /home/coder/project/hw3/os3-*.c (반드시 지정된 이름으로 작성)
  - (3-1) Demand Paging
  - (3-2) Demand Paging with 2-level Hierarchical Page Table
- LMS "과제3" 제출
  - 보고서와 소스코드 2개를 함께 압축 (zip) 해서 하나의 파일로 제출
  - 소스코드는 각 부분 과제 별로 모두 각각 제출
  - 파일 : (os3-1.c, os3-2.c, .pdf)를 zip
- 보고서 내용
  - 표지 포함(필수) 총 A4 12장 이하, PDF 형식으로 제출 (이외 형식은 10%p 감점)
  - 주석이 포함된, 전체코드에 대한 주요한 부분 서술(주석만 존재할 시 감점, 서술 필수)
  - os3-1.c, os3-2.c : test3.bin 을 이용한 수행 결과
  - 학번이 보이게 Litmus "운영체제 과제 3-1, 3-2" 제출 결과 캡처 (분량에 포함되지 않음)
  - 과제 수행 시 어려웠던 점 및 해결 방안
- 기한: 5/30 (금) 23:59 (지각 감점: 5%p / 12H, 6/6(금)23:59 이후 제출 불가)

과제 수행 및 제출 전체 과정

- JCode 홈페이지로 접속
  - https://jcode.jbnu.ac.kr
  - JEduTools 통합로그인 서비스에 가입 후 로그인 진행
    - https://jedutools.jbnu.ac.kr
    - @jbnu.ac.kr 학교 웹메일 계정으로만 사용 가능
  - 수업 참가 코드로 수업에 참가 (LMS 참고)
  - 매뉴얼: https://jhelper.jbnu.ac.kr

- Litmus 홈페이지 접속
  - https://litmus.jbnu.ac.kr/
  - [운영체제 5분반] 과제 3
  - 과제 참가 코드로 과제 참여 (lms 참고)



참고: 과제 평가 주안점

항목	설명	감점 (미수행 및 수준 미달)
과제 요구 사항 만족	각 과제 별 요구사항을 만족했는지 여부	항목 별 10%p 감점
동적 메모리 할당 및 해제	적절한 동적 메모리 할당 및 할당된 메모리를 모두 해제하였는지 여부	항목 별 5%p 감점
주석	코드의 주요한 부분에 대한 주석 (불필요한 부분까지 요구하는게 아님)	
성능	현저하게 성능을 낮출 수 있는 아키텍쳐, 자료구조, 알고리즘의 사용	
코드 품질	현저하게 낮은 readability, 적절하게 함수를 정의해서 사용하지 않고 main()에 모든 코드를 쓰는 것, 적절하지 않은 구조체의 선언 및 활용, 암호같은 변수명 등	

3-1. Demand Paging

Overview

- 입력
  - Page reference sequence 를 가진 여러 프로세스 (최대 10개)
  - PAGESIZE(4B), PAS_FRAMES(4B), VAS_PAGES(4B)
  - Binary format for a process
    - PID (4B), Length of reference sequence (4B), Reference sequence (variable)
- 처리
  - 모든 프로세스를 이진 파일로부터 로드하고,
  - 모든 프로세스에 1-Level 페이지 테이블을 할당하고 초기화
  - 각 프로세스가 PID 순서대로 돌아가며 한 번에 하나씩 페이지를 접근
    - ex) 3개의 프로세스 p1, p2, p3이면 p1 -> p2 -> p3 -> p1 … 순으로 접근
  - 페이지 접근에 대해, demand paging 에 따라 처리
  - 모든 프로세스의 처리가 끝나거나, 할당할 메모리가 부족한 경우 종료
- 출력
  - 종료 시, 각 프로세스의 페이지 테이블을 출력

Binary format

- PID (4B), Length of reference sequence (4B), Page reference sequence (variable)
  - PID < 10, Ref_len < 256, 0 <= page number <= VAS_PAGES - 1
- test3.bin: 2개의 프로세스 정보가 저장됨
  - PAGESIZE, PAS_FRAMES, VAS_PAGES
  - 0번 프로세스: PID=0, Ref_len=8
  - 1번 프로세스: PID=1, Ref_len=7

```
32 256 64
0 8
52 52 51 53 50 17 53 51
1 7
07 04 06 04 05 07 21
```

- 이전 과제와 다른 점
  - Arrival time 없음: 모두 동시에 시작하고, PID 순서대로 번갈아가며 페이지를 하나씩 접근
  - Idle process 불필요.
  - CPU, IO 처리 없음

Demand Paging: Physical Memory Management

- 물리 메모리 (PAS) 관리
  - 실제 메모리를 할당하여 사용
    - Ex) test3.bin : Page size(INPUT : 32) * PAS_FRAMES (INPUT : 256) = 8192 B
  - Frame 을 구분하여 관리하고,
  - Page table 도 PAS에서 프레임을 할당하여 저장, 관리함
    - PTE: Page Table Entry (4 B) 를 정의하여 사용
  - 각 프로세스가 접근하는 페이지는 프레임을 할당하되, 실제 어떤 유저 데이터를 기록하지는 않음
  - 기타 시뮬레이터 수행을 위한 데이터는 PAS에 저장하지 않음
    - 예) PCBs 저장을 위한 자료 구조 등
- Free Frame의 관리
  - 0부터 순서대로 증가하며 필요한 frame 만큼 할당
    - Page table: 연속된 frame (VAS_PAGES*PTE_SIZE/PAGESIZE) 개
    - Page: 1개
  - No page replacement: 한 번 할당된 frame 을 다시 해제하지 않음
    - Frame 이 부족한 경우, Out of memory (OOM) 에러를 출력하고 종료

Demand Paging: Page Fault

- 각 프로세스의 메모리 접근(페이지 단위)에 대해,
  - 해당 프로세스의 페이지 테이블을 검색하여
  - 해당 페이지에 이미 물리 프레임이 할당되어 있는 경우,
    - 해당 프레임 번호로 접근: 해당 PTE 에 reference count 증가
    - (접근에 따른 실제 데이터 처리는 없음)
  - 해당 페이지에 물리 프레임이 할당되어 있지 않은 경우,
    - Page fault 처리
    - 새로운 물리 프레임을 하나 할당받고,
    - 페이지 테이블을 업데이트하고,
    - 해당 프레임 번호로 접근: 해당 PTE 에 reference count 증가

Demand Paging: System Parameters

- Test3.bin 예시
  - Page size = 32 B
  - Physical address space
    - Size = 8 KB = 32 B * 256 frames
    - PAS_SIZE = (PAGESIZE*PAS_FRAMES) //32*256 = 8192 B
  - Virtual address space
    - Size = 2048 B = 32 B * 64 pages
    - Page Table: 64 PTEs (8 consecutive frames = 64 pages * 4B PTE / PAGESIZE)
    - Page Table Entry: 4 B
      - Frame number, valid-invalid bit, reference bit, padding

Related macros and structures

```
PAGESIZE, VAS_PAGES, PAS_FRAMES  (4B 씩 입력 받기)

#define PAS_SIZE (PAGESIZE*PAS_FRAMES)  //test3.bin : 32*256 = 8192 B

#define VAS_SIZE (PAGESIZE*VAS_PAGES)   // test3.bin : 32*64 = 2048 B

#define PTE_SIZE (4)    //sizeof(pte)


// test3.bin : 64*4/32 = 8 consecutive frames

#define PAGETABLE_FRAMES (VAS_PAGES*PTE_SIZE/PAGESIZE)


#define PAGE_INVALID (0)

#define PAGE_VALID (1)


#define MAX_REFERENCES (256)

typedef struct{

    unsigned char frame;   //allocated frame

    unsigned char vflag;    //valid-invalid bit

    unsigned char ref;      //reference bit

    unsigned char pad;      //padding

} pte;  // Page Table Entry (total 4 Bytes, always)
```

```
typedef struct{

    int pid;

    int ref_len;    //Less than 255

    unsigned char *references;
} process_raw;


typedef struct {

    //b[PAGESIZE]

    unsigned char *b;

} frame;
```

Output

- 과제 3 출력
  - 메모리가 부족해서 종료하는 상황: “Out of memory!!\n”
  - 종료 시, 각 프로세스 별 페이지 테이블 상태 출력
    - Allocated frames, Page faults/Reference count
    - Page table 출력: Page number, allocated frame number, reference count
      - Reference count: 실제 수행된 reference 개수 (OOM 상황에는 ref. seq. 보다 적음)
      - Valid PTE 에 대해서만 출력

```
Out of memory!!
** Process 000: Allocated Frames=055 PageFaults/References=047/134
001 -> 152 REF=001
005 -> 228 REF=001
006 -> 211 REF=002
007 -> 061 REF=002
008 -> 040 REF=004
```

  - 전체 Allocated frames, Page faults/Reference count 개수 출력

```
Total: Allocated Frames=256 Page Faults/References=216/620
```

test3.bin 상세 결과 (1/2)

- (실제출력 X) 디버깅 메시지

```
Start() start
[PID 00 REF:000] Page access 052: PF,Allocated Frame 016
[PID 01 REF:000] Page access 007: PF,Allocated Frame 017
[PID 00 REF:001] Page access 052: Frame 016
[PID 01 REF:001] Page access 004: PF,Allocated Frame 018
[PID 00 REF:002] Page access 051: PF,Allocated Frame 019
[PID 01 REF:002] Page access 006: PF,Allocated Frame 020
[PID 00 REF:003] Page access 053: PF,Allocated Frame 021
[PID 01 REF:003] Page access 004: Frame 018
[PID 00 REF:004] Page access 050: PF,Allocated Frame 022
[PID 01 REF:004] Page access 005: PF,Allocated Frame 023
[PID 00 REF:005] Page access 017: PF,Allocated Frame 024
[PID 01 REF:005] Page access 007: Frame 017
[PID 00 REF:006] Page access 053: Frame 021
[PID 01 REF:006] Page access 021: PF,Allocated Frame 025
[PID 00 REF:007] Page access 051: Frame 019
Start() end
```

test3.bin 상세 결과 (2/2)

```
** Process 000: Allocated Frames=013 PageFaults/References=005/008
017 -> 024 REF=001
050 -> 022 REF=001
051 -> 019 REF=002
052 -> 016 REF=002
053 -> 021 REF=002
** Process 001: Allocated Frames=013 PageFaults/References=005/007
004 -> 018 REF=002
005 -> 023 REF=001
006 -> 020 REF=001
007 -> 017 REF=002
021 -> 025 REF=001
Total: Allocated Frames=026 PageFaults/References=010/015
```

3-2. Demand Paging with 2-level Hierarchical Page Table

Hierarchical Page Table

- Hierarchical Page Table 구조
  - 기존에는 8개의 연속된 프레임에 대한 64개 PTE가 배치됨
    - test3.bin 기준
  - 불필요한 PTE의 할당과 PT의 contiguous allocation 문제를 해결하기 위해, 계층적 페이지 테이블 구조로 변경
  - 그 외의 내용은 모두 그대로
- test3.bin 예시(PAGESIZE : 32, PAS_FRAMES : 256, VAS_PAGES : 64)
  - Level 1 Page table (L1 PT)
    - 프로세스 로드 시, L1 PT 를 위해 프레임 하나를 할당하고 초기화
      - 해당 프레임에는 8개 PTE (4B x 8=32B)가 있고,
      - 각 PTE는 8개로 조각난 level 2 page table (L2 PT) 을 가리킬 수 있음
      - On demand 로 L2 PT를 할당하고, 그때 PTE를 valid로 설정 (이것도 page fault)
  - Level 2 Page table (L2 PT)
    - 8x8=64 PTEs
    - 총 8개의 프레임이 on demand 로 할당되고, 기존과 같이 PTE를 관리함

test3.bin : Example and Output

- 동작 예 : 10번 페이지 접근 시
  - L1 PT
    - 10/8 = 1번 PTE를 확인하고, page fault 인 경우, frame 할당 후, valid 설정
  - L2 PT
    - L1 1번 PTE를 확인 후, 해당 PTE 가 가리키는 frame 을 확인
    - 해당 frame 은 다시 8개의 PTE 로 구성되어 있고,
    - 10%8 = 2 번 PTE를 확인하고,
      - Page fault 인 경우, frame 할당 후, valid 설정 후 아래와 동일하게 처리
      - PF가 아닌 경우, 해당 frame 으로 접근, reference count 증가
- 출력: L1 PT를 순회하며, valid PTE에 대해 L2 PT 의 내용을 3-1과 같이 출력
  - L1 PT: Index -> Frame
  - L2 PT: Page -> Frame REF=해당 페이지에 대한 reference count

test3.bin 상세 결과 (1/2)

```
Start() start
[PID 00 REF:000] Page access 052: (L1PT) PF,Allocated Frame 006 -> 002,(L2PT) PF,Allocated Frame 003
[PID 01 REF:000] Page access 007: (L1PT) PF,Allocated Frame 000 -> 004,(L2PT) PF,Allocated Frame 005
[PID 00 REF:001] Page access 052: (L1PT) Frame 002,(L2PT) Frame 003
[PID 01 REF:001] Page access 004: (L1PT) Frame 004,(L2PT) PF,Allocated Frame 006
[PID 00 REF:002] Page access 051: (L1PT) Frame 002,(L2PT) PF,Allocated Frame 007
[PID 01 REF:002] Page access 006: (L1PT) Frame 004,(L2PT) PF,Allocated Frame 008
[PID 00 REF:003] Page access 053: (L1PT) Frame 002,(L2PT) PF,Allocated Frame 009
[PID 01 REF:003] Page access 004: (L1PT) Frame 004,(L2PT) Frame 006
[PID 00 REF:004] Page access 050: (L1PT) Frame 002,(L2PT) PF,Allocated Frame 010
[PID 01 REF:004] Page access 005: (L1PT) Frame 004,(L2PT) PF,Allocated Frame 011
[PID 00 REF:005] Page access 017: (L1PT) PF,Allocated Frame 002 -> 012,(L2PT) PF,Allocated Frame 013
[PID 01 REF:005] Page access 007: (L1PT) Frame 004,(L2PT) Frame 005
[PID 00 REF:006] Page access 053: (L1PT) Frame 002,(L2PT) Frame 009
[PID 01 REF:006] Page access 021: (L1PT) PF,Allocated Frame 002 -> 014,(L2PT) PF,Allocated Frame 015
[PID 00 REF:007] Page access 051: (L1PT) Frame 002,(L2PT) Frame 007
Start() end
```

test3.bin 상세 결과 (2/2)

```
** Process 000: Allocated Frames=008 PageFaults/References=007/008
(L1PT) 002 -> 012
(L2PT) 017 -> 013 REF=001
(L1PT) 006 -> 002
(L2PT) 050 -> 010 REF=001
(L2PT) 051 -> 007 REF=002
(L2PT) 052 -> 003 REF=002
(L2PT) 053 -> 009 REF=002
** Process 001: Allocated Frames=008 PageFaults/References=007/007
(L1PT) 000 -> 004
(L2PT) 004 -> 006 REF=002
(L2PT) 005 -> 011 REF=001
(L2PT) 006 -> 008 REF=001
(L2PT) 007 -> 005 REF=002
(L1PT) 002 -> 014
(L2PT) 021 -> 015 REF=001
Total: Allocated Frames=016 Page Faults/References=014/015
```