Assignment 3

OS LAB

Index

Overview

- **Input**
  - Page reference sequence를 가지고 있는 여러 프로세스(이진 파일에 포함되어 있음)

- **Processing**
  - 모든 프로세스를 이진파일로부터 로드
  - 모든 프로세스에 page table을 할당하고 초기화
  - 각 프로세스가 PID 순서대로 돌아가며 한 번에 하나씩 메모리(page)를 접근
  - 메모리 접근에 대해, demand paging에 따라 처리
  - 모든 프로세스의 처리가 끝나거나, 할당할 메모리가 부족한 경우 종료

- **Output**
  - 종료 시, 각 process의 page table 출력

1. Demand Paging

1.1 Binary format

- **PID (4B), Length of reference sequence (4B), Page reference sequence (variable)**
  - PID <10, Ref_len < 256, Page number < 64
  - Reference sequence: Locality를 가짐
- **이진 파일: 프로세스 정보가 저장됨**
  - 예시 파일 - test3.bin: 2개의 프로세스 정보 포함
    - 0번 프로세스: PID=0, Ref_len=8
    - 1번 프로세스: PID=1, Ref_len=7
      - 예시- test3.bin

        0 8

        52 52 51 53 50 17 53 51

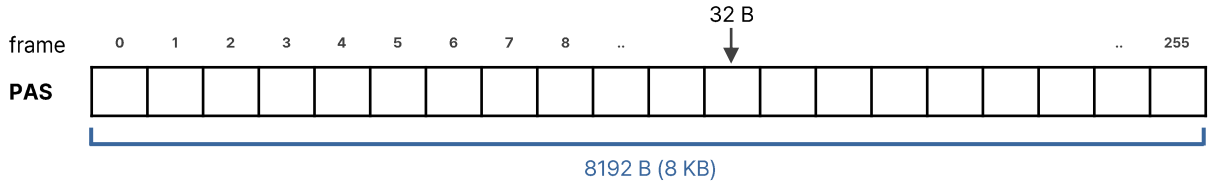
        1 7

        07 04 06 04 05 07 21
  - 테스트 파일 - test3-x.bin: x 는 프로세스 개수

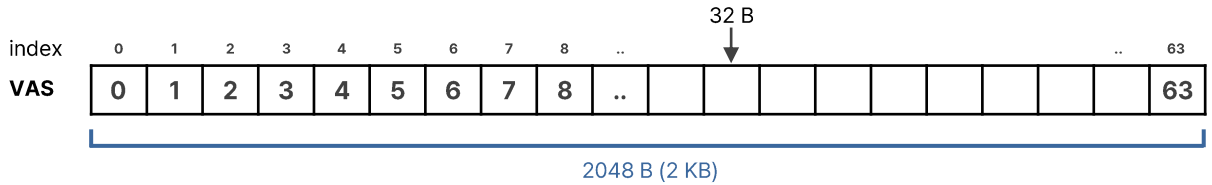
1.2 Demand Paging: System Parameters

- **Frame, Page size = 32 B**

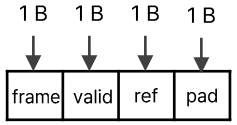
- **Physical address space**
  - Size = 32B * 256 frames = 8KB
  - frame *pas = (frame*) malloc(PAS_SIZE);



- **Virtual address space**
  - Size = 32B * 64 pages = 2KB

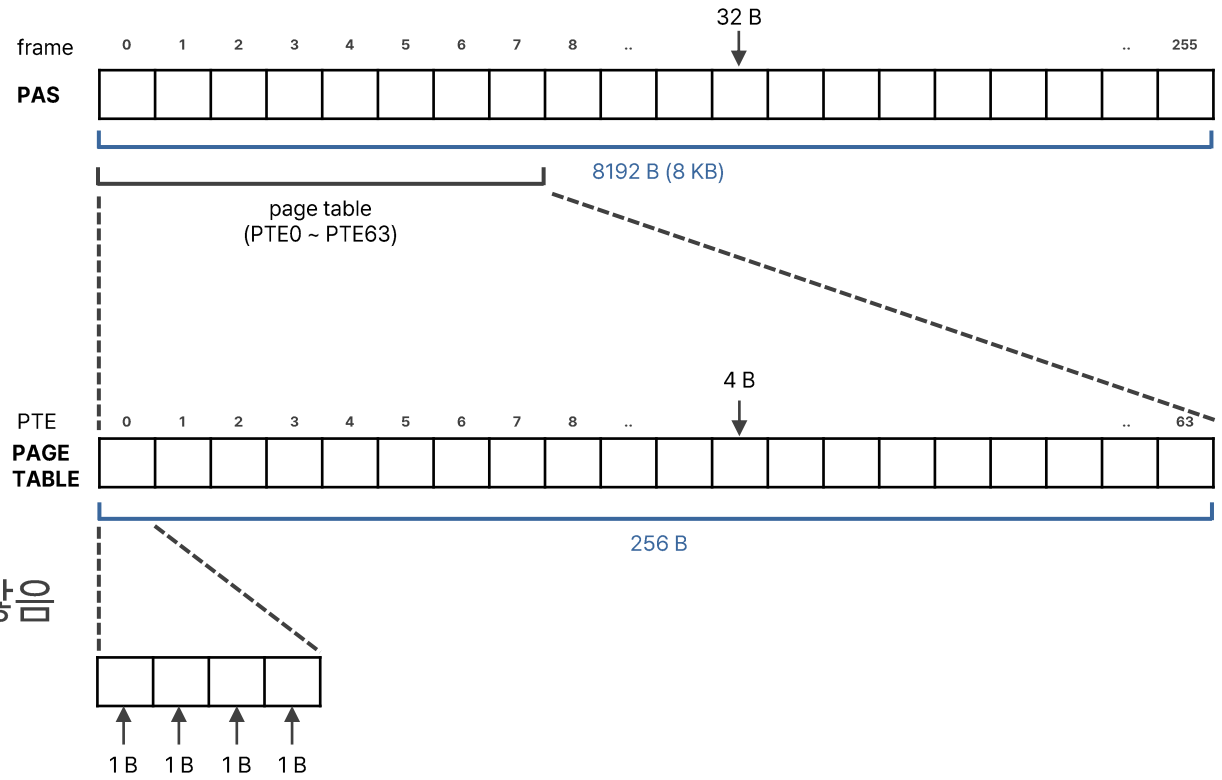


- **Page Table Entry: 4B**
  - Frame number, valid bit, reference bit, padding로 구성
  - 하나의 프로세스는 64개의 PTE로 구성
    - 8 consecutive frames(= 64 pages * 4B PTE / 32B PAGESIZE)이 필요



1.2 Demand Paging: Physical Memory Management

- **물리 메모리(PAS) 관리**
  - 실제 메모리를 할당하여 사용
    - Frame size(32 B) * 256 Frames = 8192 B
  - Frame 을 구분하여 관리
  - Page table도 PAS에서 프레임을 할당하여 저장, 관리함
    - PTE: Page Table Entry (4B)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: 8 consecutive frame 할당
    - Page: 1 frame 할당
  - No page replacement: 한 번 할당된 frame 을 다시 해제하지 않음
    - Frame이 부족한 경우, Out of memory (OOM) 에러를 출력하고 종료

1.2 Demand Paging: Page Fault

- **각 프로세스의 메모리 접근(페이지 단위)**
  - 해당 프로세스의 페이지 테이블을 검색

  - 1) 해당 페이지에 이미 물리 프레임이 할당되어 있는 경우,
    - 해당 프레임 번호로 접근: 해당 PTE 에 reference count 증가
    - (접근에 따른 실제 데이터 처리는 없음)

  - 2) 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

Related macros and structures

- os3-1.c 파일에 정의 (주석 참고)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PAGESIZE (32)
#define PAS_FRAMES (256) //fit for unsigned char frame in PTE
#define PAS_SIZE (PAGESIZE * PAS_FRAMES) //32 * 256 = 8192 B
#define VAS_PAGES (64)
#define PTE_SIZE (4) //sizeof(pte) = 4B
#define PAGETABLE_FRAMES (VAS_PAGES * PTE_SIZE / PAGESIZE) //64 * 4 / 32 = 8 consecutive frames
#define PAGE_INVALID (0)
#define PAGE_VALID (1)
#define MAX_REFERENCES (256)
#define MAX_PROCESSES (10)

typedef struct {
    unsigned char frame; //allocated frame
    unsigned char vflag; //valid bit
    unsigned char ref; //reference bit
    unsigned char pad; //padding
} pte;

typedef struct {
    int pid;
    int ref_len; //less than 255
    unsigned char *references;
    pte *page_table;
    int page_faults;
    int ref_count;
} process;
```

과제#1

- **Stdin으로부터 Binary 형태의 프로세스 정보와 Page reference sequence를 읽어들여, 아래와 같이 출력하시오.**
  - 'os_hw2' Github : os3-1.c 파일 수정 (https://github.com/woosy123/os_hw2)
  - 출력
    - load_process(): 이진 파일에 있는 프로세스 정보 및 page reference sequence

      [PID] [REF_LEN]

      [REF0] [REF1] ... [REFn]
    - simulate(): 각 프로세스가 PID 순서대로 돌아가며 메모리 접근 및 프레임 할당하는 과정

      [PID %02d IDX:%03d] %03d Page access: Frame %03d
      [PID %02d IDX:%03d] %03d Page access: PF -> Allocated Frame %03d(Page fault인 경우)
    - simulate(): 메모리가 부족해서 종료하는 상황: Out of memory!!
    - print_page_tables(): 종료 시, 각 프로세스별 페이지 테이블 상태
      - Allocated frames, Page faults/Reference count(실제 수행된 reference 개수)

        ** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d
      - Page table : Page number, allocated frame number, reference count (Valid PTE 에 대해서만 출력)

        [PAGE] %03d -> [FRAME] %03d REF=%03d
      - 전체 Allocated frames, Page faults/Reference count 개수 출력

        Total: Allocated Frames=%03d Page Faults/References=%03d/%03d

Test3.bin 수행 결과

- **cat test3.bin | ./a.out**

```
load_process() start
0 8
52 52 51 53 50 17 53 51
1 7
07 04 06 04 05 07 21
load_process() end
simulate() start
[PID 00 IDX:000] 052 Page access: PF -> Allocated Frame 016
[PID 01 IDX:000] 007 Page access: PF -> Allocated Frame 017
[PID 00 IDX:001] 052 Page access: Frame 016
[PID 01 IDX:001] 004 Page access: PF -> Allocated Frame 018
[PID 00 IDX:002] 051 Page access: PF -> Allocated Frame 019
[PID 01 IDX:002] 006 Page access: PF -> Allocated Frame 020
[PID 00 IDX:003] 053 Page access: PF -> Allocated Frame 021
[PID 01 IDX:003] 004 Page access: Frame 018
[PID 00 IDX:004] 050 Page access: PF -> Allocated Frame 022
[PID 01 IDX:004] 005 Page access: PF -> Allocated Frame 023
[PID 00 IDX:005] 017 Page access: PF -> Allocated Frame 024
[PID 01 IDX:005] 007 Page access: Frame 017
[PID 00 IDX:006] 053 Page access: Frame 021
[PID 01 IDX:006] 021 Page access: PF -> Allocated Frame 025
[PID 00 IDX:007] 051 Page access: Frame 019
simulate() end
```

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

Test3.bin 수행 결과 그림

- **PAS**

frame	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	..		..	255
																	52	07	04	51	06	53	50	05	17	21				

Process0 page table (PTE0 ~ PTE63)

Process1 page table (PTE0 ~ PTE63)

- **VAS**

index	0	1	2	3	4	5	6	7	8	..																..	62	63
page	0	1	2	3	4	5	6	7	8																		62	63

- **PTE (process 0 & 1)**

page	0	1	2	3	4	5	6	7	..	..	17	18	19	20	21	..			..	50	51	52	53	..				..	63
frame											24									22	19	16	21						
valid											1									1	1	1	1						
ref											1									1	2	2	2						
padding																													

page	0	1	2	3	4	5	6	7	..	..	17	18	19	20	21	..			..	50	51	52	53	..				..	63
frame					18	23	20	17						25	25														
valid					1	1	1	1						1	1														
ref					2	1	1	2						1	1														
padding																													

2. 2-level Hierarchical Page Table

1-level Page Table의 문제점

- **메모리 낭비**
  - 예: 페이지 12개만 사용하는 경우에도 64개 전체 PTE를 생성
  - 사용하지 않는 페이지에 대해서도 PTE를 할당
    - 메모리 낭비 발생

- **연속된 메모리 공간 요구**
  - 페이지 테이블은 연속된 물리 메모리 공간이 필요
  - 단편화(fragmentation)로 인해 연속된 메모리 공간 확보가 어려움
    - 메모리 재배치 필요
    - 페이지 테이블 생성 실패

Hierarchical Page Table

- **Hierarchical Page Table 구조**
  - 기존에는 8개의 연속된 프레임에 64개 PTE가 배치 : 메모리에서 큰 부분을 차지함
  - 불필요한 PTE의 할당과 PT의 연속된 메모리 공간 문제를 해결하기 위해, 계층적 페이지 테이블 구조로 변경

- **Level 1 Page table (L1 PT)**
  - 프로세스 로드 시, L1 PT 를 위해 한개의 프레임을 할당하고 초기화
    - 해당 프레임에는 8개 PTE (4B x 8=32B)가 있고,
    - 각 PTE는 8개의 level 2 page table (L2 PT) 을 가리킬 수 있음
    - On demand 방식으로 L2 PT를 할당하고 PTE를 valid로 설정(page fault)

- **Level 2 Page table (L2 PT)**
  - 8x8=64 PTEs
  - 총 8개의 프레임이 on demand 방식으로 할당되고, 기존과 같이 PTE 관리

동작 예시

- **28번 페이지 접근**
  - L1 PT
    - 28/8(L1 PT entries) = 3번 PTE를 확인
    - page fault 인 경우, frame 할당(n) 및 valid 설정
  - L2 PT
    - L1 3번 PTE를 확인 후, 3번 PTE 가 가리키는 frame(n) 확인
    - 해당 frame은 다시 8개의 PTE 로 구성
    - 28%8(L2 PT entries) = 4번 PTE를 확인
      - Page fault인 경우,
        - frame 할당(i) 및 valid 설정 후 해당 frame 으로 접근, reference count 증가
      - Page fault가 아닌 경우,
        - 해당 frame 으로 접근, reference count 증가

- **L1 PT**

page/8	0	1	2	3	4	5	6	7
frame				n				
valid				1				
ref								
padding								

- **L2 PT**

page%8	0	1	2	3	4	5	6	7
frame					i			
valid					1			
ref					1			
padding								

과제#2

- **Stdin으로부터 Binary 형태의 프로세스 정보와 Page reference sequence를 읽어들여, 아래와 같이 출력하시오.**
  - ‘os_hw2 ’ Github : os3-2.c 파일 수정 (https://github.com/woosy123/os_hw2)
  - 출력
    - load_process(): 과제#1과 동일
    - simulate():
      - 페이지 접근 성공: [PID %02d IDX:%03d] Page access %03d: (L1PT) Frame %03d, (L2PT) Frame %03d
      - L1 Page fault: [PID %02d IDX:%03d] Page access %03d: (L1PT) Frame %03d, (L2PT) PF -> Allocated Frame %03d
      - L1, L2 Page fault: [PID %02d IDX:%03d] Page access %03d: (L1PT) PF -> Allocated Frame %03d(PTE %03d),(L2PT) PF -> Allocated Frame %03d
      - 메모리가 부족해서 종료하는 상황: Out of memory!!
    - print_page_tables():
      - Allocated frames, Page faults/Reference count(실제 수행된 reference 개수) : 과제#1과 동일
      - L1 page table : (L1PT) [PTE] %03d -> [FRAME] %03d
      - L2 page table : (L2PT) [PAGE] %03d -> [FRAME] %03d REF=%03d
      - 전체 Allocated frames, Page faults/Reference count 개수 출력: 과제#1과 동일

Test3.bin 수행 결과(1/2)

- **cat test3.bin | ./a.out**

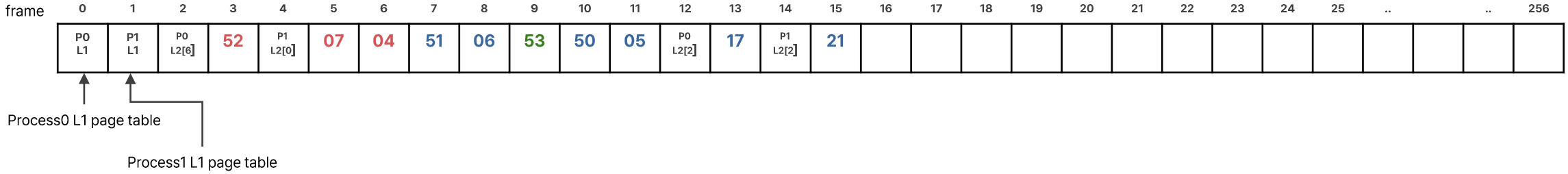
```
load_process() start
0 8
52 52 51 53 50 17 53 51
1 7
07 04 06 04 05 07 21
load_process() end
simulate() start
[PID 00 IDX:000] Page access 052: (L1PT) PF -> Allocated Frame 002(PTE 006), (L2PT) PF -> Allocated Frame 003
[PID 01 IDX:000] Page access 007: (L1PT) PF -> Allocated Frame 004(PTE 000), (L2PT) PF -> Allocated Frame 005
[PID 00 IDX:001] Page access 052: (L1PT) Frame 002, (L2PT) Frame 003
[PID 01 IDX:001] Page access 004: (L1PT) Frame 004, (L2PT) PF -> Allocated Frame 006
[PID 00 IDX:002] Page access 051: (L1PT) Frame 002, (L2PT) PF -> Allocated Frame 007
[PID 01 IDX:002] Page access 006: (L1PT) Frame 004, (L2PT) PF -> Allocated Frame 008
[PID 00 IDX:003] Page access 053: (L1PT) Frame 002, (L2PT) PF -> Allocated Frame 009
[PID 01 IDX:003] Page access 004: (L1PT) Frame 004, (L2PT) Frame 006
[PID 00 IDX:004] Page access 050: (L1PT) Frame 002, (L2PT) PF -> Allocated Frame 010
[PID 01 IDX:004] Page access 005: (L1PT) Frame 004, (L2PT) PF -> Allocated Frame 011
[PID 00 IDX:005] Page access 017: (L1PT) PF -> Allocated Frame 012(PTE 002), (L2PT) PF -> Allocated Frame 013
[PID 01 IDX:005] Page access 007: (L1PT) Frame 004, (L2PT) Frame 005
[PID 00 IDX:006] Page access 053: (L1PT) Frame 002, (L2PT) Frame 009
[PID 01 IDX:006] Page access 021: (L1PT) PF -> Allocated Frame 014(PTE 002), (L2PT) PF -> Allocated Frame 015
[PID 00 IDX:007] Page access 051: (L1PT) Frame 002, (L2PT) Frame 007
simulate() end
```

Test3.bin 수행 결과(2/2)

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

Test3.bin 수행 결과 그림(1/2)

- **PAS**



- **process0 L1 PT**

page/8	0	1	2	3	4	5	6	7
frame			12				2	
valid			1				1	
ref								
padding								

- **process1 L1 PT**

page/8	0	1	2	3	4	5	6	7
frame	4		14					
valid	1		1					
ref								
padding								

Test3.bin 수행 결과 그림(2/2)

- **process0 L2[6]**

page%8	0	1	2	3	4	5	6	7
frame			10	7	3	9		
valid			1	1	1	1		
ref			1	1	2	3		
padding								

- **process0 L2[2]**

page%8	0	1	2	3	4	5	6	7
frame		13						
valid		1						
ref		1						
padding								

- **process1 L2[0]**

page%8	0	1	2	3	4	5	6	7
frame					6	11	8	5
valid					1	1	1	1
ref					2	1	1	2
padding								

- **process1 L2[2]**

page%8	0	1	2	3	4	5	6	7
frame						15		
valid						1		
ref						1		
padding								