

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/ubuntu/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 이름은 꼭 학번을 붙일 것 (20123456-3-1.c, 20123456-3-2.c)
- 보고서 내용
 - 표지 포함 총 A4 15장 이하, PDF 형식으로 제출 (이외 형식은 10%p 감점)
 - 간략한 자율 진도표 포함 (for Iterative and incremental development)
 - 주석이 포함된 전체 코드와 각 주요한 부분에 대한 서술, test3.bin 을 이용한 수행 결과
 - JOTA "운영체제 과제 3-1, 3-2" 제출 결과 캡처 (분량에 포함되지 않음)
 - 과제 수행 시 어려웠던 점 및 해결 방안
- 기한: 6/7 (금) 23:59 (지각 감점: 5%p / 12H, 6/14(금)23:59 이후 제출 불가)

Iterative and incremental development

- 반복적, 점진적 프로그래밍 습관을 기르기 위함
- 도달하고자 하는 최종 목표까지 단계를 잘게 나누어 하나씩 검증, 달성하며 진행
 - 단계별로 백업 파일을 남기는 것도 좋은 습관 (Git 사용 시, Commit 의 단위가 됨)
- 아래 예시를 참고하여 각자 **자율적으로 진도표**를 간략하게 작성해서, **보고서에 반드시 첨부할 것**

단계	완료 여부
List 자료구조 파악: 예제 수행	O
list_for_each_entry()를 이용한 순회	O
list_for_each_entry_safe_reverse()...	O
과제 1-1 완료 (JOTA 확인)	O
Idle process 구현	
...	
과제 1-2 완료 (JOTA 확인)	
...	
과제 1-3 완료 (JOTA 확인)	

과제 수행 및 제출 전체 과정

(준비) JCode 사용 신청 및 할당 후 접속 확인

1. 과제 요구 사항을 정확하게 파악
2. JCode (Web-IDE) 에서 코드 작성 및 테스트
3. JOTA 에 과제 제출
 - 통과하지 못하는 경우, 2,3번 반복
4. 보고서 작성 및 LMS 제출
5. 점수 확인

• 채점 과정 (교수)

1. 보고서 토대로 성취도를 평가
2. JPlag 를 이용해 과제 간 유사도 검증
(모든 학기의 제출 코드와 비교)
3. 유사도가 높거나, 보고서 내용이
부실한 경우, Watcher 를 통해 수집된
코드 액티비티 검사
4. 점수 산출 및 공개



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 아키텍처, 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)
 - 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 메모리(페이지)를 접근
 - 메모리 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)

- $PID < 10$, $Ref_len < 256$, page number < 64
- Locality 를 가진 레퍼런스 패턴이 나오도록 설정되어 있음 (os3-gen.c 참조)

- test3.bin: 2개의 프로세스 정보가 저장됨

- (test3-x.bin: x 는 프로세스 개수)
- 0번 프로세스: $PID=0$, $Ref_len=8$
- 1번 프로세스: $PID=1$, $Ref_len=7$

```
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
 - $\text{Page size}(32 \text{ B}) * 256 \text{ Frames} = 8192 \text{ B}$
 - 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 8 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

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

```
#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 VAS_SIZE (PAGESIZE*VAS_PAGES)    //32*64 = 2048 B

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

#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)

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 {

    unsigned char b[PAGESIZE];

} 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

```
ubuntu@41983:~/hw3$ gcc -Wall os3-1.c && cat test3-5.bin | ./a.out
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)

```
ubuntu@41983:~/hw3$ gcc -Wall os3-1.c && 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
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 Page Faults/References=010/015
```



JOTA 3-1

2021운영체제 과제 3-1

- LMS(구버전) 과제 3 참조

stdin 으로부터 Binary 형태의 프로세스 정보와 Page reference sequence 를 읽어들이고, 아래와 같이 출력하시오.

3-1.

Demand Paging 구현

출력 형식

Page reference 진행이 완료된 이후,

메모리 부족으로 완료된 경우: "Out of memory!!\n"

각 프로세스별 정보 출력: "** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n"

각 프로세스별 페이지 테이블 출력: "%03d -> %03d REF=%03d\n"

모든 작업 종료 후, 최종 리포트: "Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n"



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



Hierarchical Page Table

- Hierarchical Page Table 구조
 - 기존에는 8개의 연속된 프레임에 64개 PTE가 배치됨
 - 불필요한 PTE의 할당과 PT의 contiguous allocation 문제를 해결하기 위해, 계층적 페이지 테이블 구조로 변경
 - 그 외의 내용은 모두 그대로
- Level 1 Page table (L1 PT)
 - 프로세스 로드 시, L1 PT 를 위해 프레임 하나를 할당하고 초기화
 - 해당 프레임에는 8개 PTE ($4B \times 8 = 32B$)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)
 - $8 \times 8 = 64$ PTEs
 - 총 8개의 프레임이 on demand 로 할당되고, 기존과 같이 PTE를 관리함

Example and Output

- 동작 예 : 10번 페이지 접근 시
 - L1 PT
 - $10/8 = 1$ 번 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)

```
ubuntu@41983:~/hw3$ gcc -Wall os3-2.c && 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
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: 프로세스별

```
ubuntu@41983:~/hw3$ gcc -Wall os3-2.c && cat test3.bin | ./a.out | grep "PID 00"
[PID 00 REF:000] Page access 052: (L1PT) PF,Allocated Frame 006 -> 002,(L2PT) PF,Allocated Frame 003
[PID 00 REF:001] Page access 052: (L1PT) Frame 002,(L2PT) Frame 003
[PID 00 REF:002] Page access 051: (L1PT) Frame 002,(L2PT) PF,Allocated Frame 007
[PID 00 REF:003] Page access 053: (L1PT) Frame 002,(L2PT) PF,Allocated Frame 009
[PID 00 REF:004] Page access 050: (L1PT) Frame 002,(L2PT) PF,Allocated Frame 010
[PID 00 REF:005] Page access 017: (L1PT) PF,Allocated Frame 002 -> 012,(L2PT) PF,Allocated Frame 013
[PID 00 REF:006] Page access 053: (L1PT) Frame 002,(L2PT) Frame 009
[PID 00 REF:007] Page access 051: (L1PT) Frame 002,(L2PT) Frame 007
ubuntu@41983:~/hw3$ gcc -Wall os3-2.c && cat test3.bin | ./a.out | grep "PID 01"
[PID 01 REF:000] Page access 007: (L1PT) PF,Allocated Frame 000 -> 004,(L2PT) PF,Allocated Frame 005
[PID 01 REF:001] Page access 004: (L1PT) Frame 004,(L2PT) PF,Allocated Frame 006
[PID 01 REF:002] Page access 006: (L1PT) Frame 004,(L2PT) PF,Allocated Frame 008
[PID 01 REF:003] Page access 004: (L1PT) Frame 004,(L2PT) Frame 006
[PID 01 REF:004] Page access 005: (L1PT) Frame 004,(L2PT) PF,Allocated Frame 011
[PID 01 REF:005] Page access 007: (L1PT) Frame 004,(L2PT) Frame 005
[PID 01 REF:006] Page access 021: (L1PT) PF,Allocated Frame 002 -> 014,(L2PT) PF,Allocated Frame 015
```



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
```



JOTA 3-2

2021 운영체제 과제 3-2

- LMS(구버전) 과제 3 참조

stdin 으로부터 Binary 형태의 프로세스 정보와 Page reference sequence 를 읽어들이고, 아래와 같이 출력하시오.

3-2.

Demand Paging with 2-level Hierarchical Page Table 구현

출력 형식

Page reference 진행이 완료된 이후,

메모리 부족으로 완료된 경우: "Out of memory!!\n"

각 프로세스별 정보 출력: "** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n"

각 프로세스별 Level 1 페이지 테이블 출력: "(L1PT) %03d -> %03d\n"

각 프로세스별 Level 2 페이지 테이블 출력: "(L2PT) %03d -> %03d REF=%03d\n"

모든 작업 종료 후, 최종 리포트: "Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n"



과제 주안점

- Demand paging, hierarchical page table 에 대한 이해!!
 - 슬라이드와 수업 동영상을 보면서 확실하게 이해할 것
- 아주 조금씩 확인해가며 진도를 나갈 것
 - 특히 계층적 PT 구현 시 L1, L2 PTE 각각을 주의해서 진행
- 숫자 하나 하나가 무슨 의미인지, 정확하게 파악하며 진행
 - Page 번호, Frame 번호, L1 PTE, L2 PTE 내용의 의미 등
- 할당한 메모리 공간을 다양한 포인터 형으로 캐스팅하여 사용하여야 함
 - 1 frame = 8 PTEs
 - `pte* cur_pte = (pte *) &pas[frame_number];`
 - `pte cur_pte[8]` 의 배열처럼 접근 가능하고 혹은 `cur_pte++` 로 순회 가능
- 마지막 할당받은 메모리 해제하는 것 잊지 말기!
- 과제 1, 2의 구조나 내용과 무관하므로, 과제 요구 사항을 만족하는 범위 내에서 자유롭게 가장 간단한 방법으로 구현할 것

Bonus: os3-gen.c

- Command-line parameter: 생성할 프로세스 개수
- stdout: binary file
- stderr: debug message

```
ubuntu@41983:~/hw3$ gcc os3-gen.c && ./a.out 1 > test.bin
Generate() start num_processes = 1
# PID=00
# REF_LEN=097
28 26 27 26 29 27 25 27 29 18 54 56 38 34 36 24 23 24 26 23 13 26 24 26 27 27 56 47 46 44 46
2 02 02 02 04 02 04 42 43 43 43 43 43 42 35 35 35 56 58 55 58 58 58 59 55 57 58 42 42 49 43
Generate() end
ubuntu@41983:~/hw3$ gcc os3-gen.c && ./a.out 2 > test.bin
Generate() start num_processes = 2
# PID=00
# REF_LEN=100
54 13 13 15 11 15 13 12 11 14 12 14 15 59 58 48 47 47 48 49 19 17 21 18 17 36 50 53 53 51 17
5 55 55 42 55 58 59 11 10 11 09 11 08 07 08 08 05 09 63 08 06 32 54 24 33 32 29 59 48 52 47
# PID=01
# REF_LEN=247
31 30 30 31 30 33 31 29 57 02 58 19 19 53 21 21 25 25 05 07 06 07 07 03 03 06 47 17 20 16 19
7 57 28 09 12 13 10 13 13 51 50 51 50 50 50 51 50 48 47 48 50 10 50 13 00 53 54 55 52 53 63
41 39 42 39 52 23 20 21 23 22 24 23 24 20 24 22 09 26 22 25 25 23 24 26 22 26 26 23 04 30 3
28 30 57 27 46 24 03 25 25 26 25 27 28 27 16 10 11 09 11 09 56 56 54 55 57 62 13 46 17 45 46
Generate() end
```