운영체제 과제 #3

3-1 Demand Paging

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

컴퓨터공학부
202246109
김기현

단계	완료 여부○▲
프로세스, 페이지 테이블, pte, pas 구조 파악하기	○
프로세스 로딩후 출력해보기	○
PAS 구현 및 초기화	○
페이지 접근해서 출력해보기	○
동작구조 파악 및 test3-1.txt 분석하기	○
Page fault 구현하기	○
out of memory 구현하기	○
Out of memory 계속 뜨는 오류 고치기	○
demandPaging() 구현 및 재구성	○
Out of memory 3-1.txt 처럼 구현하기	○
3-1 주요 기능 함수화	○
accessMemory() 구현	○
loadProcess() 구현	○
init() 구현	○
finalReport() 구현	○
과제 3-1완료 (JOTA 제출)	○
2 level page table 구조 파악하기	○
L2PT 접근해서 출력해보기	○
accessMemory() 재구성	○
finalReport() 재구성	○
Reference count miss	X
과제 3-2 완료 (JOTA 제출)	○

3-1.c 코드설명

```c
typedef struct{
    unsigned char b[PAGESIZE];
} frame;
```

사용되는 구조체 입니다.

```c
typedef struct{
    unsigned char frame;
    unsigned char vflag;
    unsigned char ref;
    unsigned char pad;
} pte;
```

PAS는 frame으로 구성되어있고

Page table을 구성하는 pte 구조체입니다.

```c
typedef struct{
    int pid;
    int ref_len;
    unsigned char *references;
    int allocatedFrame;
    int pfCount;
    int refCount;
} process;
```

Process 구조체를 선언하여 프로세스당 할당된 프레임의 수,

Page Fault count, reference count를 관리합니다.

```c
process *psArray[10];
int psNum = 0;
int usedFrame = 0;
```

전역변수 입니다.

프로세스의 수가 10개로 제한되었기 때문에 psArray 배열에서 관리합니다.

psNum은 Binary 파일로부터 읽어온 프로세스의 수를 기록합니다.

usedFrame 에 동작하면서 사용된 frame의 수를 기록합니다.

```c
void loadProcess(){…
}
```

Binary 파일로부터 모든 프로세스를 로드 합니다

```c
void init(frame* pas){…
}
```

페이지 테이블을 초기화합니다.

```c
bool accessMemory(frame* pas, process* cur, int pos){…
}
```

Reference sequence에 따라 메모리에 접근하는 함수입니다.

```c
void demandPaging(frame* pas){…
}
```

Demand paging을 처리합니다.

```c
void finalReport(frame* pas){…
}
```

각 프로세스의 페이지 테이블을 출력합니다.

```c
int main(){
    frame* pas = (frame*)malloc(PAS_SIZE);

    // LOAD
    loadProcess();
    //printf("%d", usedFrame);

    // INIT
    init(pas);

    // Demand Paging
    demandPaging(pas);

    // Final Report
    finalReport(pas);

    /*
    pte* PTE = (pte*)&pas[0];
    for (int i = 0; i < VAS_SIZE; i++) {…
    }
    */

    // FREE
    free(pas);
    for(int i = 0; i < psNum; i++){
        free(psArray[i]->references);
        free(psArray[i]);
    }
    return 0;
}
```

메인 함수입니다.

PAS 입니다 실제 메모리를 할당해서 사용합니다.

프로그램의 로직은

프로세스 로딩 -> 초기화 -> demand paging -> report -> 메모리 해제

순서로 동작합니다.

모든 처리가 끝나고 메모리를 해제합니다.

```
void loadProcess(){
    //printf("load_process() start\n");
    for(; (psArray[psNum] = (process*)malloc(sizeof(process))) && fread(psArray[psNum], sizeof(int) * 2, 1, stdin); psNum++){
        //ref_len 만큼 메모리 할당
        psArray[psNum]->references = (unsigned char*)malloc(psArray[psNum]->ref_len);
        fread(psArray[psNum]->references, psArray[psNum]->ref_len, 1, stdin);
        psArray[psNum]->allocatedFrame = 8; // page 하나는 8개의 프레임
        usedFrame += 8;
        psArray[psNum]->pfCount = 0;
        psArray[psNum]->refCount = 0;
    }
    /*
    // debug
    printf("load_process() end\n");
    for(int i = 0; i < psNum; i++){
        printf("%d %d\n", psArray[i]->pid, psArray[i]->ref_len);
        for(int j = 0; j < psArray[i]->ref_len; j++){
            printf("%d ", psArray[i]->references[j]);
        }
        printf("\n");
    }
    */
    return;
}
```

loadProcess 함수는 binary 파일로부터 프로세스를 읽어와 로드합니다.

기존에는 while문을 사용했지만 이번엔 for문을 사용해봤습니다.

Page table 하나 당 8개의 프레임이 사용되어서

해당 프로세스의 allocatedFrame을 Page table 크기인 8로 초기화 해줍니다.

```
void init(frame* pas){
    pte* cur_pte = (pte*)&pas[0];
    for(int i = 0; i < VAS_SIZE; i++){
        // 초기화
        cur_pte->vflag = PAGE_INVALID;
        cur_pte->ref = 0;
        cur_pte++; // 다음 pte
    }
}
```

Page table은 64개의 pte로 구성되어있습니다.

처음부터 순회하면서 모든 pte를 초기화 합니다.

```
void demandPaging(frame* pas){
    int pos = 0;

    while(1){
        // reference sequence 끝에 도달
        if(pos >= MAX_REFERENCES){
            return;
        }

        // 프로세스 순회하면서 페이지 접근
        for(int i = 0; i < psNum; i++){
            process* cur = psArray[i];
            // 프레임 부족하면
            if(usedFrame >= PAS_FRAMES){
                //cur = psArray[i % psNum];
                //printf("[PID %02d REF:%03d] Page access %03d: PF,",cur->pid, cur->refCount, cur->references[pos]);
                printf("Out of memory!!\n");
                return;
            }
            // 프레임 부족하면 break;
            if(cur->ref_len > pos && accessMemory(pas, cur, pos)){
                break;
            }
        }
        pos++;
    }
}
```

demandPaging 함수입니다. 메모리 접근에 대해, demand paging에 따라 처리합니다.

1. 프로세스의 reference sequence 끝에 도달 시

if 문의 조건을 통해 reference sequence 끝에 도달했는지 확인합니다.

Reference sequence 끝에 도달하면 demand paging을 종료합니다.

2. 프로세스 순회 및 페이지 접근

for루프를 통해 각 프로세스의 reference를 처리합니다.

3. 모든 프레임이 사용중이라면

프레임의 한계에 도달하면 out of memory를 출력하고 demand paging을 종료합니다.

4. Pos 가 Reference sequence의 범위를 벗어나지 않았다면 메모리에 접근합니다.

accessMemory() 를 수행하며 알맞는 프레임에 접근하여 reference를 처리합니다.

pos++를 통해 다음 reference로 이동합니다.

```
bool accessMemory(frame* pas, process* cur, int pos){
    pte* cur_pte = (pte*)&pas[cur->pid * 8];
    // valid pte 일 때
    if(cur_pte[cur->references[pos]].vflag == PAGE_VALID){
        //printf("[PID %02d REF:%03d] Page access %03d: Frame %03d\n", cur->pid, cur
        cur_pte[cur->references[pos]].ref++;
        cur->refCount++;
    }
    // unvalid pte 일 때 -> page fault
    if(cur_pte[cur->references[pos]].vflag == PAGE_INVALID){
        //page fault
        if(usedFrame >= PAS_FRAMES){
            //printf("break");
            return true;
        }
        cur_pte[cur->references[pos]].frame = usedFrame++;
        cur_pte[cur->references[pos]].vflag = PAGE_VALID;
        cur->pfCount++;
        cur->allocatedFrame++;
        //printf("[PID %02d REF:%03d] Page access %03d: PF,Allocated Frame %03d\n",c
        cur_pte[cur->references[pos]].ref++;
        cur->refCount++;
    }
    return false;
}
```

Bool 형태의 accessMemory() 함수는 프레임이 모자란 경우 true를 리턴하여 demandPaging을 중단합니다.

cur_pte 를 사용하여 현재 프로세스의 pte를 가리킵니다.

프로세스의 page table은 8개의 frame으로 구성되어 있으므로 pas[pid * 8] 로 접근할 수 있습니다.

1. Valid pte 처리

cur_pte의 vflag를 확인해 valid pte면 reference 처리를 합니다.

2. Page fault 처리

vflag를 확인해 Invalid pte라면 page fault가 발생합니다.

사용된 프레임 수가 256개를 넘어간다면 true를 리턴하여 demandPaging() 함수에서 break문을 실행합니다.

넘지 않은 경우 페이지를 할당하고 유효 플래그를 설정하며 참조 횟수를 증가시킵니다.

```
void finalReport(frame* pas){
    int totalFrame = 0;
    int totalPF = 0;
    int totalRef = 0;

    for(int i = 0; i < psNum; i++){
        //process
        printf("** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n",
         psArray[i]->pid, psArray[i]->allocatedFrame, psArray[i]->pfCount, psArray[i]->refCount);

        totalFrame += psArray[i]->allocatedFrame;
        totalPF += psArray[i]->pfCount;
        totalRef += psArray[i]->refCount;

        pte* cur_pte = (pte*)&pas[i * 8];
        // valid pte만 출력
        for(int j = 0; j < VAS_PAGES; j++){
            if(cur_pte[j].vflag == PAGE_VALID){
                printf("%03d -> %03d REF=%03d\n", j, cur_pte[j].frame, cur_pte[j].ref);
            }
        }
    }
    printf("Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n", totalFrame, totalPF, totalRef);
}
```

각 프로세스의 page table을 출력하는 finalReport 함수 입니다.

totalFrame, totalPf, totalRef에 전체 allocated frame, page faults, reference count를 저장합니다.

pte의 vflag를 확인하여 valid pte만 frame과 ref횟수를 출력합니다.

3-2.c 코드설명

```
void loadProcess(){
    //printf("load_process() start\n");
    for(; (psArray[psNum] = (process*)malloc(sizeof(process))) && fread(psArray[psNum], sizeof(int) * 2, 1, stdin); psNum++){
        //ref_len 만큼 메모리 할당
        psArray[psNum]->references = (unsigned char*)malloc(psArray[psNum]->ref_len);
        fread(psArray[psNum]->references, psArray[psNum]->ref_len, 1, stdin);
        psArray[psNum]->allocatedFrame = 1; //L1PT용 프레임
        usedFrame++;
        psArray[psNum]->pfCount = 0;
        psArray[psNum]->refCount = 0;
    }
```

3-2.c 의 코드입니다. 3-1에서와 달리 프로세스를 로드할 때 프레임을 1개만 할당해줍니다.

3-1 에서 accessMemory()와 finalReport() 함수를 제외하고 나머지 부분은 동일하기 때문에 두 함수에 대해서 설명하겠습니다.

```
bool accessMemory(frame* pas, process* cur, int pos){
    /*
    {…
    }
    */
    pte* L1PT = (pte*)&pas[cur->pid];
    int L1_index = cur->references[pos] / PAGETABLE_FRAMES;
    int L2_index = cur->references[pos] % PAGETABLE_FRAMES;

    // valid pte 일 때
    if(L1PT[L1_index].vflag == PAGE_VALID){
        //[PID 00 REF:001] Page access 052: (L1PT) Frame 002,
        //printf("[PID %02d REF:%03d] Page access %03d: (L1PT
    }
    //L1PT 할당
    else if(L1PT[L1_index].vflag == PAGE_INVALID){
        // L1_index page fault
        if(usedFrame >= PAS_FRAMES){
            //printf("break");
            return true;
        }
        L1PT[L1_index].frame = usedFrame++;
        L1PT[L1_index].vflag = PAGE_VALID;
        cur->pfCount++;
        cur->allocatedFrame++;
        //printf("[PID %02d REF:%03d] Page access %03d: (L1PT
    }

    pte* L2PT = (pte*)&pas[L1PT[L1_index].frame];
    // valid pte 일 때
    if(L2PT[L2_index].vflag == PAGE_VALID){
        L2PT[L2_index].ref++;
        cur->refCount++;
        //(L2PT) Frame 003
        //printf("(L2PT) Frame %03d\n", L2PT[L2_index].frame);
    }
    // unvalid pte 일 때 -> page fault
    else if(L2PT[L2_index].vflag == PAGE_INVALID){
        if(usedFrame >= PAS_FRAMES){
            //printf("break");
            return true;
        }
        L2PT[L2_index].frame = usedFrame++;
        L2PT[L2_index].vflag = PAGE_VALID;
        cur->pfCount++;
        cur->allocatedFrame++;

        //PF,Allocated Frame 006
        //printf("(L2PT) PF,Allocated Frame %03d\n", L2PT[L2_index].frame);
        L2PT[L2_index].ref++;
        cur->refCount++;
    }
    return false;
}
```

accessMemory 함수입니다.

3-1 에선 1-level 페이지 테이블을 사용했지만

3-2에선 2-level의 페이지 테이블을 사용합니다.

하나의 페이지 테이블은 8개의 pte로 구성되어 있습니다.

L1PT는 reference / 8 의 pte를 확인하고

L2PT는 reference % 8 의 pte를 확인합니다.

L1PT의 pte.frame은 8개중 하나의 L2PT를 가리키고

L2PT의 pte.frame에 찾아가 reference를 카운트 합니다.

L1PT 가 invalid pte라면 page fault 가 발생하며 프레임을 할당해줍니다.

L2PT 에서도 valid pte라면 reference 처리를 하고

Invalid pte라면 page fault가 발생하며 프레임 할당 및 reference 처리를 합니다.

Page fault가 발생했을 때 모든 프레임이 사용되었다면

true를 리턴하여 demandPaging을 중지시킵니다.

```c
void finalReport(frame* pas){
    int totalFrame = 0;
    int totalPF = 0;
    int totalRef = 0;
    for(int i = 0; i < psNum; i++){
        // process
        printf("** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n",
         psArray[i]->pid, psArray[i]->allocatedFrame, psArray[i]->pfCount, psArray[i]->refCount);

        totalFrame += psArray[i]->allocatedFrame;
        totalPF += psArray[i]->pfCount;
        totalRef += psArray[i]->refCount;

        pte* L1PT = (pte*)&pas[i];
        for(int j = 0; j < PAGETABLE_FRAMES; j++){
            // valid L1PT
            if(L1PT[j].vflag == PAGE_VALID){
                //(L1PT) 002 -> 012
                printf("(L1PT) %03d -> %03d\n", j, L1PT[j].frame);
                pte* L2PT = (pte*)&pas[L1PT[j].frame];
                // valid L2PT
                for(int k = 0; k < PAGETABLE_FRAMES; k++){
                    if(L2PT[k].vflag == PAGE_VALID){
                        //(L2PT) 017 -> 013 REF=001
                        //printf("page access %d",psArray[i]->references[k]);
                        printf("(L2PT) %03d -> %03d REF=%03d\n", k + (j * PAGETABLE_FRAMES), L2PT[k].frame, L2PT[k].ref);
                        //printf("%d %d %d\n",i,j,k);
                        //printf("%d %d\n",L1PT[j].frame, L2PT[L1PT[j].frame].frame);
                    }
                }
            }
        }
    }
    printf("Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n", totalFrame, totalPF, totalRef);
}
```

finalReport에서 삼중 for문 으로 2-level page table에 접근하도록 변경되었습니다.

i 는 psArray를 순회하며 pid와 같습니다.

j 는 L1PT를 순회하며 valid pte의 frame을 출력합니다

k 는 L2PT를 순회하며 valid pte의 frame을 출력합니다.

K + (j * PAGETABLE_FRAMES) 에서 k는 나머지 j 는 몫 PAGETABLE_FRAMES 는 제수 reference는 피제수를 의미합니다.

단순히 생각하면 (제수 * 몫) + 나머지 연산을 통해 해당하는 pte에 접근할 수 있습니다.

test3.bin 수행 결과

Os3-1

```
ubuntu@jcode-client1-181:~/hw3$ cat test3.bin | ./a.out
** 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
```

os3-2

```
ubuntu@jcode-client1-181:~/hw3$ cat test3.bin | ./a.out
** 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
```

Os3-1 (full results)

```
ubuntu@jcode-client1-181:~/hw3$ cat test3.bin | ./a.out
load_process() start
0 8
52 52 51 53 50 17 53 51
1 7
7 4 6 4 5 7 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
** 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
```

os3-2 (full results)

```
ubuntu@jcode-client1-181:~/hw3$ cat test3.bin | ./a.out
load_process() start
0 8
52 52 51 53 50 17 53 51
1 7
7 4 6 4 5 7 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
** 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
```

10 / 10
AC | C

운영체제 과제 3-1
os202246109 5분 전

os202246109의 운영체제 과제 3-1 제출

View source
다시 제출

컴파일 경고

```
oshw31c.c: In function ‘loadProcess’:
oshw31c.c:43:9: warning: ignoring return value of ‘fread’ declared with attribute ‘warn_unused_result’ [-Wunused-result]
   43 |         fread(psArray[psNum]->references, psArray[psNum]->ref_len, 1, stdin);
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Execution Results

✔ ✔ ✔ ✔ ✔

Test case #1: AC [0.003s,1.02 MB] (2/2)
Test case #2: AC [0.003s,1.02 MB] (2/2)
Test case #3: AC [0.003s,1.02 MB] (2/2)
Test case #4: AC [0.003s,1.02 MB] (2/2)
Test case #5: AC [0.003s,1.02 MB] (2/2)

Resources: 0.015s, 1.02 MB
Maximum single-case runtime: 0.003s
Final score: 10/10 (10.0/10 points)

10 / 10
AC | C

os202246109 20초 전

os202246109의 운영체제 과제 3-2 제출

2024년 6월 7일 오후 6시 46분
C

View source
다시 제출

컴파일 경고

```
oshw32c.c: In function ‘loadProcess’:
oshw32c.c:43:9: warning: ignoring return value of ‘fread’ declared with attribute ‘warn_unused_result’ [-
Wunused-result]
   43 |         fread(psArray[psNum]->references, psArray[psNum]->ref_len, 1, stdin);
      |         ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Execution Results

✔ ✔ ✔ ✔ ✔

Test case #1: AC [0.003s,1.02 MB] (2/2)
Test case #2: AC [0.003s,1.02 MB] (2/2)
Test case #3: AC [0.003s,1.02 MB] (2/2)
Test case #4: AC [0.005s,1.02 MB] (2/2)
Test case #5: AC [0.003s,1.02 MB] (2/2)

Resources: 0.018s, 1.02 MB
Maximum single-case runtime: 0.005s
Final score: 10/10 (19.0/19 points)

어려웠던 점 및 해결 방안

1. 프로세스 저장 구조

과제 1에선 리스트 자료구조를 사용하여 프로세스를 관리했는데 과제 3 설명에서 리스트 자료구조의 언급이 없어서 뭐로 관리해야할지 고민되었습니다. 과제 설명에서 프로세스를 10개로 제한한다고 했기 때문에 제일 직관적이고 간단한 자료구조인 배열로 관리해야겠다고 생각했습니다.

2. 초기화

초기화를 진행 할 때 pte를 어떻게 찾아가는지에 대한 고민을 많이 했습니다. pas를 실제로 뜯어보고 싶었는데 제대로 된 방법이 아닌거 같아 pte++ 로 다음 메모리로 이동하도록 수정했습니다.

3. Output 처리 및 프레임 찾아가기

```
frame *pas = (frame*) malloc(PAS_SIZE);
```

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

3-2에서 모든 demand paging 이 끝나고 final report에서 page num과 frame num이 잘 이해가 가지 않았습니다. 설명 영상과 수업 영상을 몇 번씩 돌려가면서 memory 파트에 관한 공부를 다시 했습니다. 생각보다 과제 설명에서 많은 도움을 받았습니다.

4. 의문의 out of memory 및 Txt 파일처럼 수정

Demand paging은 모든 프레임을 사용하거나 모든 reference를 처리했을 때 종료됩니다.
특히 모든 프레임을 다 사용했을 때 out of memory가 출력 되는데 프레임을 다 사용하지 않아도
out of memory 가 출력는 문제가 발생했습니다.
애매했던 while문의 조건식을 수정했고, 3-1.txt 에서 out of memory가 출력될 때 제 출력결과와 다른점이 있어서 DemandPaging() 함수의 구조를 처음부터 다시 재구성 했습니다.
demandPaging() 함수를 재구성 하면서 주요 코드들을 함수화 작업을 하면서 전체적인 재구성을 진행했습니다.

5. Ref 카운트 미스

```
Out of memory!!
** Process 000: Allocated Frames=032
PageFaults/References=031/062
(L1PT) 000 -> 146
(L2PT) 001 -> 223 REF=001
(L2PT) 004 -> 166 REF=004
(L2PT) 006 -> 147 REF=003
(L2PT) 007 -> 151 REF=003
(L1PT) 001 -> 007
(L2PT) 009 -> 031 REF=006
(L2PT) 010 -> 041 REF=006
(L2PT) 011 -> 037 REF=004
(L2PT) 012 -> 008 REF=007
(L2PT) 014 -> 215 REF=001
(L2PT) 015 -> 111 REF=002
(L1PT) 002 -> 077
(L2PT) 016 -> 095 REF=001
(L2PT) 017 -> 098 REF=002
```

```
Out of memory!!
** Process 000: Allocated Frames=032
PageFaults/References=031/063
(L1PT) 000 -> 146
(L2PT) 001 -> 223 REF=001
(L2PT) 004 -> 166 REF=004
(L2PT) 006 -> 147 REF=003
(L2PT) 007 -> 151 REF=003
(L1PT) 001 -> 007
(L2PT) 009 -> 031 REF=007
(L2PT) 010 -> 041 REF=006
(L2PT) 011 -> 037 REF=004
(L2PT) 012 -> 008 REF=007
(L2PT) 014 -> 215 REF=001
(L2PT) 015 -> 111 REF=002
(L1PT) 002 -> 077
(L2PT) 016 -> 095 REF=001
(L2PT) 017 -> 098 REF=002
```

JOTA 제출 결과에서 reference count 가 1개씩 어긋나는 경우가 발생했습니다.

Pos 또는 usedFrame 에서 문제가 발생하는 것 같아서 많은 수정을 해봤습니다.

그러나 도대체 어느 부분에서 문제가 생기는지 모르겠어서 해결하지 못했습니다…