

**운영체제 과제 3**

**염선욱 물리학과 201921786**

단계	완료여부
관련 매크로 및 구조체 파악	O
page_reference_sequence를 읽기 위한 access_unit 구조체 추가	O
hw1과 os-gen.c를 참고해서 load_process() 구현	O
실제 메모리에 프레임 단위로 할당된 "pas"와 각 프레임을 page_table_entry 단위로 type-casting한 "cur_pte" 사이의 관계 파악	O
프로세스마다 8개의 프레임 할당	O
각 프로세스의 page_table에서 page번호를 입력으로 받아서 frame번호를 출력시키는 방법(Address-Translation) 구현	O
page_fault v_1구현(3-2 이후 변경)	O
프로세스들의 할당된 프레임 수, PF 횟수, reference 횟수를 세기 위한 구조체 ps_result(3-2에서는 report로 변경) 추가	O
Start() 구현	O
프로세스를 순회하면서 valid pte의 정보만 출력하는 print_all() 구현	O
과제 3-1 완료 (JOTA 확인)	O
물리 메모리 상에서의 3-1과 3-2 상황을 그려서 문제 완벽하게 이해하기	O
이진파일로부터 프로세스를 읽을 때마다 프레임 하나를 할당해서 L1PT 1개씩 만들 수 있도록 load_process() 변경	O
OOM 처리 방식 변경(v_2)	O
V_1처럼 OOM 처리하면 안 되는 이유 파악	X
L1PT, L2PT의 demand-paging 구현(3-1과 Address-Translation 다름)	O
Start() 구현	O
print_all() 구현	O
과제 3-2 완료 (JOTA 확인)	O

## 201921786-3-1.c 전체 코드

hw3 > C os3-1.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define PAGESIZE (32)
5  #define PAS_FRAMES (256)    //fit for unsigned char frame in PTE
6  #define PAS_SIZE (PAGESIZE*PAS_FRAMES) //32*256 = 8192 B
7  #define VAS_PAGES (64)
8  #define VAS_SIZE (PAGESIZE*VAS_PAGES) //32*64 = 2048 B
9  #define PTE_SIZE (4)    //sizeof(pte)
10 #define PAGETABLE_FRAMES (VAS_PAGES*PTE_SIZE/PAGESIZE) //64*4/32 = 8 consecutive frames
11
12 #define PAGE_INVALID (0)
13 #define PAGE_VALID (1)
14 #define MAX_REFERENCES (256)
15
16 typedef struct{
17     unsigned char frame;    //allocated frame
18     unsigned char vflag;    //valid-invalid bit
19     unsigned char ref;      //reference bit
20     unsigned char pad;      //padding
21 } pte; //Page Table Entry (total 4 Bytes, always)
22
23 typedef struct{
24     int pid;
25     int ref_len;    //Less than 255
26     unsigned char *references;
27 } process_raw;
28
29 typedef struct{
30     unsigned char b[PAGESIZE];
31 } frame;
32
33 typedef struct{
34     unsigned char page;
35 } access_unit;
36
37 //별도의 구조체 필요? PT: PTE 정의하여 사용 --> PAS에서 저장&관리 ==> 3-1에서 필요 없었음
38 // typedef struct{
39 //     pte entries[VAS_PAGES];
40 // } PageTable;
41
42 typedef struct{
43     int F; //Frames
44     int PF; //Page-Faults
```

hw3 > C os3-1.c

```
37 typedef struct{
38     int F; //Frames
39     int PF; //Page-Faults
40     int REF; //Reference-Count
41 } ps_result;
42
43 int num_ps = 0; //number of loaded process
44 int frame_number = 0;
45 ps_result res[10];
46 process_raw procs[10]; //max: 10
47 frame *pas;
48
49 void load_process(){
50     // printf("load_process() start\n");
51     process_raw ps;
52     int size = 2*sizeof(int);
53     while(fread(&ps, size, 1, stdin) == 1){
54         procs[num_ps].pid = ps.pid;
55         procs[num_ps].ref_len = ps.ref_len;
56         procs[num_ps].references = (unsigned char*) malloc(sizeof(unsigned char) * procs[num_ps].ref_len);
57         // printf("%d %d\n", procs[num_ps].pid, procs[num_ps].ref_len);
58         access_unit seq;
59         for(int i=0; i<procs[num_ps].ref_len; i++){
60             if(fread(&seq, sizeof(seq), 1, stdin) == 1){
61                 procs[num_ps].references[i] = seq.page;
62                 // printf("%02d ", procs[num_ps].references[i]);
63             }
64         }
65         // printf("\n");
66         num_ps++;
67     }
68     // printf("load_process() end\n");
69 }
70
71 void Start(){
72     //ps#1: pte* cur_pte = (pte*) &pas[8]
73     //typecast PAS into PTEs: frames -> PTEs
74     //Address-Translation: frame_number = 8*i + procs[i].ref[j]/8, cur_pte_idx = procs[i].ref[j]%8
75     // printf("Start() start\n");
76     frame_number += num_ps*PAGETABLE_FRAMES; //PT: 연속된 frame 8개 할당
77     for(int i=0; i<num_ps; i++){
78         res[i].F += PAGETABLE_FRAMES;
79     }
80     for(int j=0; j<MAX_REFERENCES; j++){
81         for(int i=0; i<num_ps; i++){
82             if(j>=procs[i].ref_len) continue; //skip finished_process
83             pte* cur_pte = (pte*) &pas[PAGETABLE_FRAMES*i + procs[i].references[j]/PAGETABLE_FRAMES];
84             int idx = procs[i].references[j]%PAGETABLE_FRAMES;
85             // printf("[PID %02d REF:%03d] Page access %03d: ", procs[i].pid, res[i].REF, procs[i].references[j]);
86             if(cur_pte[idx].vflag == PAGE_INVALID){ //Page-Fault
87                 if(frame_number==PAS_FRAMES){ //OutOfMemory
88                     if(cur_pte[idx].vflag == PAGE_INVALID){
89                         printf("Out of memory!!\n");
90                         return;
91                     }
92                 }
93                 // printf("PF, Allocated ");
94                 res[i].F++;
95                 res[i].PF++;
96                 cur_pte[idx].frame = frame_number++; //allocate new frame
97                 cur_pte[idx].vflag = PAGE_VALID; //update PT
98             }
99             // printf("Frame %03d\n", cur_pte[idx].frame);
100             res[i].REF++;
101             cur_pte[idx].ref++;
102         }
103     }
104     // printf("Start() end\n");
105 }
106
107 void print_all(){
```

```

hw3 > C os3-1.c
106 }
107 void print_all(){
108     int tot_F=0, tot_PF=0, tot_REF=0;
109     for(int i=0; i<num_ps; i++){
110         printf("*** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n", procs[i].pid, res[i].F, res[i].PF, res[i].REF);
111         tot_F += res[i].F;
112         tot_PF += res[i].PF;
113         tot_REF += res[i].REF;
114         for(int j=0; j<VAS_PAGES; j++){
115             pte* cur_pte = (pte*) &pas[PAGETABLE_FRAMES*i + j/PAGETABLE_FRAMES];
116             int idx = j%PAGETABLE_FRAMES;
117             if(cur_pte[idx].vflag == PAGE_INVALID) continue;
118             printf("%03d -> %03d REF=%03d\n", j, cur_pte[idx].frame, cur_pte[idx].ref);
119         }
120     }
121     printf("Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n", tot_F, tot_PF, tot_REF);
122 }
123 int main(int argc, char* argv[]){
124     pas = (frame*)malloc(PAS_SIZE); //실제 메모리 할당
125
126     load_process();
127     Start();
128     print_all(); //print all PT's info(valid only PTEs)
129
130     for(int i=0; i<num_ps; i++){
131         free(procs[i].references);
132     }
133     free(pas);
134     return 0;
135 }

```

## Jcloud test3.bin 실행결과

```

// print("\n");
num_ps++;
}
// printf("load_process() end\n");
}

void Start(){
    /*1 frame = 8 PTEs (page frame size = 32 B, PTE_SIZE = 4 B)*/
    /*ps#1: pte* cur_pte = (pte*) &pas[8]
    //typecast PAS into PTEs: frames -> PTEs
    //Address-Translation: frame_number = 8*i + procs[i].ref[j]/8, cur_pte_idx = procs[i].ref[j]%8
    // printf("Start() start\n");
    frame_number += num_ps * PAGETABLE_FRAMES; //PT: 연속된 frame 8개 할당
    for(int i=0; i<num_ps; i++){
        res[i].F += PAGETABLE_FRAMES;
    }
    for(int j=0; j<MAX_REFERENCES; j++){
        for(int i=0; i<num_ps; i++){
            if(j>procs[i].ref_len) continue; //skip finished process
            pte* cur_pte = (pte*) &pas[PAGETABLE_FRAMES*i + procs[i].references[j]/PAGETABLE_FRAMES];
            // printf("%03d -> %03d REF=%03d\n", j, cur_pte[idx].frame, cur_pte[idx].ref);
        }
    }
}

```

```

(L2PT) 007 -> 005 REF=002
(L1PT) 002 -> 014
(L2PT) 021 -> 015 REF=001
Total: Allocated Frames=016 Page Faults/References=014/015
** 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
ubuntu@jcode-client1-175:~/hw3$

```

솔직히 말하면, paging의 개념을 거의 모르는 상태에서 과제 3-1을 시작했다. 3-1은 억지로 끼워 맞추는 식으로 풀리긴 했다. 하지만 3-2를 풀기 위해서는 demand-paging과 hierarchical page table을 이해해야 했고, 역으로 3-1 문제가 풀리는 원리도 이해할 수 있었다. 처음에는 과제 안내에 있는 코드를 최대한 활용해서 이해해보려 했다. LAS가 없는 이유를 몰라서 헤매기도 했다. 시뮬레이션을 위해서는 PT를 PAS에 할당해주기만 하면 되고, 접근할 때는 Address-Translation(AT)을 위한 계산이 필요하겠단 정도는 보였다. PT를 구조체를 추가해

서 구현해야 되는 것인지도 헷갈렸지만, 아니었다. 과제 안내처럼 "frame \*pas = (frame\*) malloc(PAS\_SIZE)"를 통해서 실제 메모리를 할당하고, 그 PAS의 한 frame에 8개의 PTE들이 있다는 걸 알 수 있었다. 각 프로세스마다 64개의 PTE들이 있고, user-data에 접근하기 위해서는 우선 page번호를 찾아야 한다. 해당 프로세스에서(ex. ps#0이라면 PAS에서는 frame#0~7 사이) 각 reference가 page 자체이므로, 일단 그 page를 8로 나눠야 frame#(0~7 중 하나)를 찾을 수 있다. 다음 과정이 중요하다. "8\*프로세스+앞에서 8로 나눈 값"이 인덱스인 PAS에서의 frame 하나 자체를 PTE 단위로 캐스팅한다(pte\* cur\_pte = (pte\*)&pas[frame\_number]의 의미). 정확한 page를 찾기 위해선 reference를 8로 나눈 나머지를 cur\_pte의 인덱스로 접근(cur\_pte[idx])해야 하면 PTE 접근하는 것은 일단 성공이다. 그 이후엔 PF(프레임 할당 작업, vflag 변경 작업), OOM, ref\_cnt 증가 등의 작업을 하면 Start()를 구현할 수 있다.

JOTA 첫 제출에 성공하지는 못했다. ref\_cnt가 잘못 출력되었고, 이는 OOM 처리를 잘못된 것처럼 보였다. Line81을 수정해줘야 했다. 각 프로세스의 reference sequence는 256개 미만이어야 한다는 조건(j<MAX\_REFERENCES)을 VAS\_PAGES인 64개로 뒤서(j<VAS\_PAGES) 생긴 문제였었다. 그리고 초반에는 paging을 제대로 이해하지 못한 상태여서 프레임을 할당한다는 뜻이 헷갈렸다. PT를 위한 프레임을 할당하는 코드는 line77~80처럼 그냥 전역변수인 frame\_number를 8\*프로세스 개수만큼 증가(1 frame = 8 PTEs 이므로)하면 됐다. user-data를 위한 프레임을 할당하기 위해선 그냥 line97처럼 "frame\_number++;" 하면 됐다. 최종 출력을 할 때는 line114에서 for문 조건식이 중요했다. 예를 들어서 ps#0의 PTEs(총 64개) 중 valid한 PTEs의 정보를 출력해야 하므로 line117의 if문을 통해서 valid한 PTE만 출력한다.

load\_process(), Start(), print\_all() 세 함수를 다 수행하고 main으로 돌아온 뒤에는 전역으로 선언했던 procs와 pas의 동적할당을 해제하면 과제 3-1이 끝난다.

201921786-3-2.c 전체 코드

hw3 > C os3-2.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define PAGESIZE (32)
5  #define PAS_FRAMES (256)    //fit for unsigned char frame in PTE
6  #define PAS_SIZE (PAGESIZE*PAS_FRAMES) //32*256 = 8192 B
7  #define VAS_PAGES (64)
8  #define VAS_SIZE (PAGESIZE*VAS_PAGES) //32*64 = 2048 B
9  #define PTE_SIZE (4) //sizeof(pte)
10 #define PAGETABLE_FRAMES (VAS_PAGES*PTE_SIZE/PAGESIZE) //64*4/32 = 8 consecutive frames
11
12 #define PAGE_INVALID (0)
13 #define PAGE_VALID (1)
14 #define MAX_REFERENCES (256)
15
16 typedef struct{
17     unsigned char frame;    //allocated frame
18     unsigned char vflag;    //valid-invalid bit
19     unsigned char ref;      //reference bit
20     unsigned char pad;      //padding
21 } pte; //Page Table Entry (total 4 Bytes, always)
22 typedef struct{
23     int pid;
24     int ref_len;    //Less than 255
25     unsigned char *references;
26 } process_raw;
27 typedef struct{
28     unsigned char b[PAGESIZE];
29 } frame;
30 typedef struct{
31     unsigned char page;
32 } access_unit;
```



hw3 > C os3-2.c

```
33 typedef struct{
34     int F; //Frames
35     int PF; //Page-Faults
36     int REF; //Reference-Count
37 } report;
38
39 int num_ps = 0; //number of loaded process
40 process_raw procs[10]; //max: 10
41 frame *pas;
42 int frame_number = 0; //frame-cursor for "new PT or user-data" in PAS
43 report res[10];
44
45 void load_process(){ //include allocating L1PT for all process
46     // printf("load_process() start\n");
47     process_raw ps;
48     int size = 2*sizeof(int);
49     while(fread(&ps, size, 1, stdin) == 1){
50         procs[num_ps].pid = ps.pid;
51         procs[num_ps].ref_len = ps.ref_len;
52         procs[num_ps].references = (unsigned char*) malloc(sizeof(unsigned char) * procs[num_ps].ref_len);
53         // printf("%d %d\n", procs[num_ps].pid, procs[num_ps].ref_len);
54
55         //allocate new frame(of PAS) for L1PT
56         // pte* cur_pte = (pte*) &pas[frame_number++]; //warning: unused variable. useless code
57         frame_number++;
58         res[num_ps].F++;
59
60         access_unit seq;
61         for(int i=0; i<procs[num_ps].ref_len; i++){
62             if(fread(&seq, sizeof(seq), 1, stdin) == 1){
63                 procs[num_ps].references[i] = seq.page;
64                 // printf("%02d ", procs[num_ps].references[i]);
65             }
66         }
67         // printf("\n");
68         num_ps++;
69     }
70     // printf("load_process() end\n");
71 }
```

hw3 > C os3-2.c

```
70 // printf("load_process() end\n");
71 }
72
73 void Start(){ //On-demand_Paging of L1PT->L2PT & L2PT->PAS
74     /*=====AT_V2(3-2)=====*/
75     /*ps#0: frame#0{8 PTEs}, ps#1: frame#1, ...*/
76     /*in L2PT ==> 0th PTE: page#0~7, 1st PTE: page#8~15, ...*/
77     //L1PT: idx(i) -> frame(cur_pte[idx].frame) {w. invalid->valid}
78     //L2PT: page(procs[i].ref[j]) -> frame(cur_pte[idx].frame) {w. ref_cnt++}
79     //L1PT: frame_number = i, cur_pte_idx = procs[i].ref[j]/8
80     //L2PT: frame_number = L1PT's output-frame#, cur_pte_idx = procs[i].ref[j]%8
81     // printf("Start() start\n");
82     for(int j=0; j<MAX_REFERENCES; j++){
83         for(int i=0; i<num_ps; i++){
84             if(j>procs[i].ref_len) continue; //skip finished_process
85             //OutOfMemory --> complicated to handle here!!!
86             /*if(frame_number==PAS_FRAMES){
87                 pte* cur_pte = (pte*) &pas[i]; //AT
88                 int idx = procs[i].references[j]/PAGETABLE_FRAMES;
89                 if(cur_pte[idx].vflag == PAGE_INVALID){ //check access to invalid "L2PT"
90                     printf("Out of memory!!\n");
91                     return;
92                 }
93                 int frame_num = cur_pte[idx].frame;
94                 cur_pte = (pte*) &pas[frame_num];
95                 idx = procs[i].references[j]%PAGETABLE_FRAMES;
96                 if(cur_pte[idx].vflag == PAGE_INVALID){
97                     printf("Out of memory!!\n");
98                     return;
99                 }
100             } //escape OOM: restore index of L1PT*/
101             pte* cur_pte = (pte*) &pas[i]; //AT
```



hw3 > C os3-2.c

```
100 //escape OOM: restore index of L1PT*/
101 pte* cur_pte = (pte*) &pas[i]; //AT
102 int idx = procs[i].references[j]/PAGETABLE_FRAMES;
103 // printf("[PID %02d REF:%03d] Page access %03d: ",procs[i].pid, res[i].REF, procs[i].references[j]);
104 //L1PT: idx->frame
105 // printf("(L1PT) ");
106 if(cur_pte[idx].vflag == PAGE_INVALID){ //Page-Fault
107     if(frame_number == PAS_FRAMES){ //OOM: can't make "L2PT"
108         printf("Out of memory!!\n");
109         return;
110     }
111     // printf("PF, Allocated Frame %03d -> %03d, ", idx, frame_number);
112     cur_pte[idx].frame = frame_number++; //allocate new frame for L2PT(logically "page" in L2)
113     cur_pte[idx].vflag = PAGE_VALID; //update PT
114     res[i].F++;
115     res[i].PF++;
116 }
117 // else{ //not PF
118 //     // printf("Frame %03d, ", cur_pte[idx].frame);
119 // }
120 int frame_num = cur_pte[idx].frame; //L2PT's frame# in PAS
121
122 //L2PT: page->frame
123 // printf("(L2PT) ");
124 cur_pte = (pte*) &pas[frame_num]; //AT: update frame-cursor(allocated frame for L2PT)
125 idx = procs[i].references[j]%PAGETABLE_FRAMES;
126 if(cur_pte[idx].vflag == PAGE_INVALID){ //Page-Fault
127     if(frame_number == PAS_FRAMES){ //OOM: can't allocate new frame for "user-data"
128         printf("Out of memory!!\n");
129         return;
130     }
131     // printf("PF, Allocated ");
132     cur_pte[idx].frame = frame_number++; //allocate new frame for user-data: move PAS_frame-cursor here
133     cur_pte[idx].vflag = PAGE_VALID; //update PT
134     res[i].F++;
135     res[i].PF++;
136 }
137 // printf("Frame %03d\n", cur_pte[idx].frame);
138 cur_pte[idx].ref++;
139 res[i].REF++;
```

hw3 > C os3-2.c

```
136 }
137 // printf("Frame %03d\n", cur_pte[idx].frame);
138 cur_pte[idx].ref++;
139 res[i].REF++;
140 }
141 }
142 // printf("Start() end\n");
143 }
144
145 void print_all(){
146     int tot_F=0, tot_PF=0, tot_REF=0;
147     for(int i=0; i<num_ps; i++){
148         printf("*** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n", procs[i].pid, res[i].F, res[i].PF, res[i].REF);
149         tot_F += res[i].F;
150         tot_PF += res[i].PF;
151         tot_REF += res[i].REF;
152         for(int j=0; j<PAGETABLE_FRAMES; j++){ //L1PT has 8 PTEs: j==index of L1PT
153             pte* cur_pte = (pte*) &pas[i]; //L1PT로 이동(반복문 밖에 뒀서 출력 틀렸었음)
154             if(cur_pte[j].vflag == PAGE_INVALID) continue; //only check valid PTEs
155             printf("(L1PT) %03d -> %03d\n", j, cur_pte[j].frame);
156             int frame_num = cur_pte[j].frame; //frame# of L2PT
157             cur_pte = (pte*) &pas[frame_num];
158             for(int k=0; k<PAGETABLE_FRAMES; k++){//L2PT also has 8 PTEs for specific L1PT's 1 PTE(demand-paging)
159                 if(cur_pte[k].vflag == PAGE_INVALID) continue;
160                 printf("(L2PT) %03d -> %03d REF=%03d\n", j*PAGETABLE_FRAMES+k, cur_pte[k].frame, cur_pte[k].ref);
161             }
162         }
163     }
164     printf("Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n", tot_F, tot_PF, tot_REF);
165 }
166
167 int main(int argc, char* argv){
168     pas = (frame*)malloc(PAS_SIZE);
169
170     load_process();
171     Start();
172     print_all(); //print all valid-PTEs' info
173
174     for(int i=0; i<num_ps; i++){
```

```

hw3 > C os3-2.c
167 int main(int argc, char* argv[]){
168     pas = (frame*)malloc(PAS_SIZE);
169
170     load_process();
171     Start();
172     print_all(); //print all valid-PTEs' info
173
174     for(int i=0; i<num_ps; i++){
175         free(procs[i].references);
176     }
177     free(pas);
178     return 0;
179 }

```

## Jcloud test3.bin 실행결과

```

hw3 > C os3-2.c
64 }
65 // printf("\n");
66 num_ps++;
67
68 // printf("load_process() end\n");
69 }
70
71 void Start(){
72     /*1 frame = 8 PTEs (page frame size = 32 B, PTE_SIZE = 4 B)*/
73     //ps#1: pte* cur_pte = (pte*) &pas[8]
74     //typecast PAS into PTEs: frames -> PTEs
75     //Address-Translation: frame_number = 8*i + procs[i].ref[j]/8, cur_pte_idx = procs[i].ref[j]%8
76     // printf("Start() start\n");
77     frame_number += num_ps*PAGETABLE_FRAMES; //PT: 연속된 frame 8개 할당
78     for(int i=0; i<num_ps; i++){
79         res[i].F += PAGETABLE_FRAMES;
80     }
81     for(int j=0; j<MAX_REFERENCES; j++){
82         for(int i=0; i<num_ps; i++){
83             if(j>procs[i].ref_len) continue; //skip finished process
84             pte* cur_pte = (pte*) &pas[PAGETABLE_FRAMES*i + procs[i].references[j]/PAGETABLE_FRAMES];
85             // printf("Start() start\n");

```


```

ubuntu@jcode-client1-175:~/hw3$ gcc -Wall os3-2.c && 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
ubuntu@jcode-client1-175:~/hw3$

```

paging 개념을 잘 모르는 상태에서 3-1 문제를 맞췄다. 3-2 과제 안내를 보고 공부의 필요성을 느끼고, 3-1과 3-2 상황을 물리 메모리에 그려보았다(최하단에 사진 첨부). 사실 PAGESIZE가 PAGESIZE뿐만 아니라 FRAMESIZE도 똑같이 32 B를 의미하는 걸 이 때 알았다. 또한 VAS를 나타낼 필요는 없고, VAS\_PAGES가 결국 프로세스 입장에서 바라보는 LAS\_PAGES가 된다는 것도 이 때 알았다. 3-2에서는 2 level hierarchical PT를 이용한다. 한 개의 L1PT(8 PTEs)를 위해서는 1개의 frame이 필요하다. 한 개의 L2PT도 on demand(필요할 때만)로 1개의 frame에 할당된다. test3.bin의 첫 reference를 예로 들면 이해하기 쉬웠던 것 같다. PTE 자체를 위한 frame 개수는 결론적으로 3-1에서는 16개 프레임이 필요했지만, 3-2에서는 2(프로세스 개수) + alpha(L1PT에서 나눗셈 연산을 통해 구한 index에서의 on demand{invalid PTE인 경우에만})로 할당. 이 예시에선 4개(for ps#0: frame#2, 12, for ps#1: frame#4, 14)개의 프레임만 있으면 된다. 코드 자체는 의외로 3-1에서 추가할 코드가 많지 않았다. load\_process()에서 프로세스를 읽어올 때 프레임을 1개씩 할당(line55~58)하면 됐다. 처음엔 할당한다는 의미가 헛갈려서 line56처럼 쓰이지도 않고, 무의미한 코드를 썼지만, 에러코드를 통해서 수정할 수 있었다. 3-1에서와 다르게 PF 처리를 2번(invalid한 L2PT와 user-data로의 접근)해줘야 했다. L2PT에서의 index를

접근하기 위해서 cur\_pte를 재정의해야 했고, 정확한 page로 접근하기 위해서 나머지 연산도 필요했다. print\_all()도 사소한 실수(line153에서 cur\_pte 정의하는 부분을 for문 밖인 line151 아래에 뒤서 생긴 예러) 제외하고는 어려운 점은 없었다. 위 내용들은 수월하게 진행할 수 있었는데, JOTA 첫 제출부터 걸리는 건 OOM 처리였다. 3-1 초반 제출(JOTA도 성공했었다)까지는 line86~100처럼 이중for문의 시작을 OOM 예외처리부터 했었다(시간 효율을 위해서였다). 논리적으로 문제는 없는 줄 알았는데, 3-2에서는 몇 번을 고쳐봐도 JOTA에서는 5개 testcase 중 최대 4개까지만 성공했었다. 그래서 전략을 바꿔서 frame\_number가 256인지 체크(PAS 꼭 참)하는 걸 먼저 하는 대신 우선 PF 처리를 할 때, OOM 처리(L1PT, L2PT의 demand-paging 할 때)를 하도록 수정했다. JOTA를 잘 성공할 수 있었다. v\_1의 OOM 처리(이중 for문 첫 작업으로 PAS가 full인지부터 확인) 방식이 논리적으로 틀린 것 같진 않다. 내 코드에 문제가 있는 것 같지만, 아직 해결하진 못했다. 해결하고 v\_1으로 실행하면 아마도 실행시간도 줄어들지 않을까 생각한다.

 PROBLEMS SUBMISSIONS USERS CONTESTS ABOUT

2024년 6월 2일 오후 7시 14분

os201921786의 운영체제 과제 3-1 제출

View source  
다시 제출

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: 0.016s, 1.02 MB

Maximum single-case runtime: 0.003s

Final score: 10/10 (10.0/10 points)

운영체제과제 - 27일 13:43:44

proudly powered by DMOJ | 한국어 (ko)




PROBLEMS

SUBMISSIONS

USERS

CONTESTS

ABOUT

 Hello, os201921786

os201921786의 운영체제 과제 3-2 제출

2024년 6월 2일 오후 7시 15분

C

[View source](#)

[다시 제출](#)

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: 0.010s, 1.02 MB

Maximum single-case runtime: 0.003s

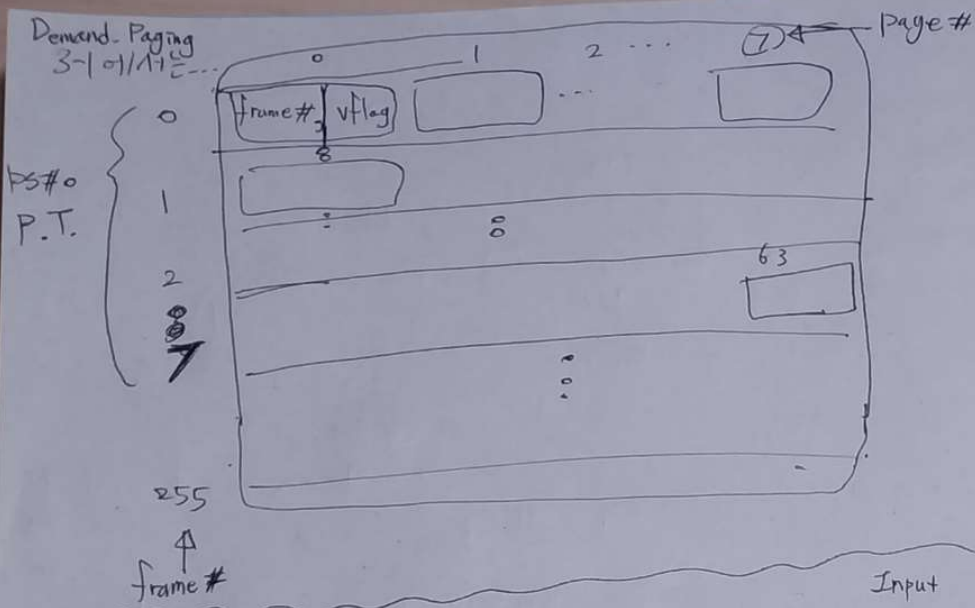
Final score: 10/10 (19.0/19 points)

운영체제과제 - 27일 13:42:34

proudly powered by DMOJ

한국어 (ko)

Demand Paging  
3-1 or 1/17E...



3-2: 2 Level: Hierarchical PT #

