운영체제 과제 3 보고서

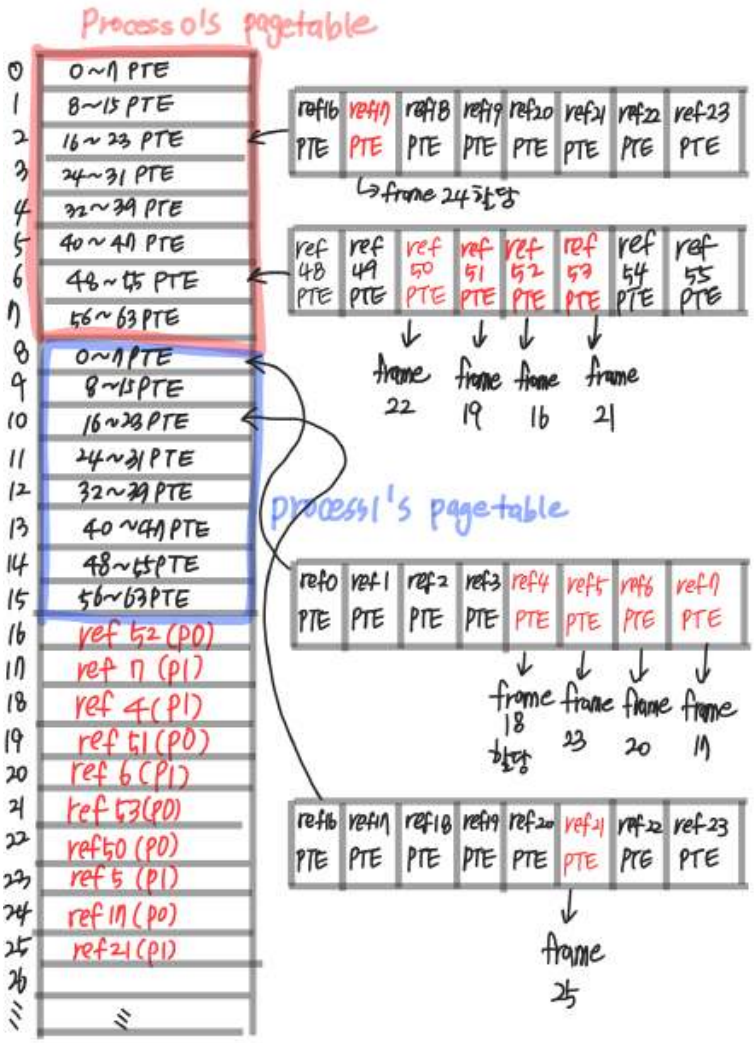
컴퓨터공학부 202011632 김수비

1. 과제 3-1

1) 자율진도표

단계	완료여부
메모리 개념 이해 및 추상화	O
fread() 함수를 사용해 이진 파일로부터 모든 프로세스 로드	O
for문을 이용한 프로세스별 PageTable 생성 및 초기화	O
list_for_each_entry사용 리스트 순회	O
프로세스 별 PageTable 값 할당	O
list_for_each_entry사용 출력	O
과제 3-1 완료(JOTA 확인)	O

2) 추상화



3) 프로그램 실행 흐름

①load(프로세스의 정보 Load)

②operation(실제 동작)

③print_info(수행 후 결과 출력)

④clear_process(사용한 메모리 동적할당 해제)

4) 전체 코드 설명

0~132: 리스트에 관한 내용으로 생략

①전역변수 및 구조체

```
133  typedef struct{
134      unsigned char frame; // 할당된 frame
135      unsigned char vflag; // vaild-invaild bit
136      unsigned char ref; // 참조 횟수
137      unsigned char pad; // padding
138
139  } pte; // 페이지테이블 엔트리(4B)
140
141  typedef struct{
142      int pid; // 프로세스 pid
143      int ref_len; // 페이지 개수
144      unsigned char *references; // 페이지
145      struct list_head job; // job이라는 리스트의 노드 생성
146  } process;
147
148  typedef struct {
149      unsigned char b[PAGESIZE]; // 페이지가 들어가있는 배열
150  } frame;
151
152  int pVisted_cnt[65]={0, }; // 프로세스별 방문 횟수 저장
153  int ref_index[11]; // 레퍼런스 주소 저장
154  int frame_cnt[11]; // 프로세스별 프레임 사용 개수
155  int pagefault_cnt[11]; // 프로세스별 페이지폴트 개수
156  int arr[11][65]; // 각 프로세스별 레퍼런스와 그때의 방문 횟수 저장 arr[0][0]: 0번 pid의 0번째 레퍼런스의 방문 횟수 =0
157  int frame_arr[11][65]; // 각 프로세스별 레퍼런스와 그때 할당된 frame 번호 저장
158  int allref_cnt;
159  int cnt=0; // 도는 레퍼런스의 개수 누적
160  int oom=1; // out of memory의 경우
161  int all_pf=0; // 전체 페이지 폴트
162  int all_ref=0; // 전체 레퍼런스
163  process p; // process 구조체 p
164  LIST_HEAD(job_q); // 리스트 헤드 선언
165  int frame_index=0; // 프레임 index
166  pte *PTE;
```

전역변수에 대한 설명은 주석을 참고

②load() 함수

```
void load(frame* PAS){
    process *cur_pro;
     while(1) // bin에서 읽어옴
    {
        cur_pro = malloc(sizeof(process));
        if(fread(&p, sizeof(int)*2, 1, stdin) == 1)// pid, ref_len 를 읽어옴
        {
            allref_cnt += p.ref_len; // 모든 레퍼런스의 개수를 저장
            //p의 정보를 cur로 옮겨줌(cur정보 초기화)
            cur_pro-> pid = p.pid;
            cur_pro-> ref_len = p.ref_len;
            cur_pro->references = malloc(sizeof(unsigned char) *cur_pro->ref_len); // cur의 레퍼런스 선언
            INIT_LIST_HEAD(&cur_pro->job); // cur의 job리스트 초기화
            list_add_tail(&cur_pro->job, &job_q); // job_q 리스트와 cur의 job 연결
            for(int i=0; i<cur_pro->ref_len; i++)
            {
                fread(&cur_pro->references[i], sizeof(char), 1, stdin); // cur의 references 읽어옴

            }
            // 페이지 테이블 만들기
            for(int i=0; i<8; i++) // 페이지테이블은 8칸으로 나누어짐
            {
                pte *PTE = (pte *)PAS[frame_index].b; // PTE생성, PAS의 frame에 넣음

                for(int j=0; j<8; j++) // 1칸에는 8개의 page가 저장
                {
                    // 각 page의 pte를 초기화
                    PTE[j].frame =0; // j번 페이지의 frame 정보
                    PTE[j].vflag =PAGE_INVALID;
                    PTE[j].ref=0;
                }
                frame_index++;
            }
        }
        else{ // bin 전부 돌면 나감
            break;
```

PAS를 매개변수로 받아온 후, 프로세스의 정보를 읽어와 프로세스 별로 동적할당 후 프로세스의 레퍼런스 길이에 따라 레퍼런스 동적할당 후 리스트에 저장 PAS를 통해 페이지 테이블을 만들어 프레임에 삽입

③operation() 함수

```
void operation(frame* PAS){
    process *cur_pro;
    while(oom)
    {
        if(cnt == allref_cnt){
            break;
        }

        // PAS 채우기
        list_for_each_entry(cur_pro, &job_q, job) // job_q리스트 순회
        {
            // 레퍼런스 다 돌면 순회 종료
            if(ref_index[cur_pro->pid] == cur_pro->ref_len) // 몇번째 레퍼런스인지(1,2,3...) = 길이 -> 종료
                continue;
            cnt++;

            // 현재 레퍼런스 페이지테이블에 할당해줌
                int cur_ref = cur_pro->references[ref_index[cur_pro->pid]]; // 현재 탐색 중인 레퍼런스 정보(52, 7...)
                int cur_frame = (cur_pro->pid)*8+cur_ref/8; // 레퍼런스가 들어있는 pagetable의 frame 번호
                int cur_pte = cur_ref%8; // 현재 pte값
                PTE = (pte *)PAS[cur_frame].b; // 현재 frame에 PTE 넣음
                if(PTE[cur_pte].vflag == PAGE_INVALID){// 할당되지 않은 페이지일 때(page fault 발생)
                    if(frame_index == MAX_REFERENCES) // 더이상 할당해 줄 프레임이 없을 때
                    {
                        printf("Out of memory!!\n");
                        oom = 0;
                        break;
                    }
                    else{
                        all_pf++;
                        pagefault_cnt[cur_pro->pid]++; // 프로세스 별 페이지 폴트 개수
                        PTE[cur_pte].frame = frame_index; // 남은 frame을 할당
                        frame_arr[cur_pro->pid][cur_ref] = frame_index; // 프로세스의 레퍼런스에 할당된 frame 정보 저장
                        PTE[cur_pte].vflag = PAGE_VALID; // 페이지 할당됐다고 표시
                        PTE[cur_pte].ref++; // 방문+1
                        frame_index++; // 프레임 하나 사용
                        frame_cnt[cur_pro->pid]++; // 프로세스 별 할당한 프레임 개수
                        arr[cur_pro->pid][cur_ref]++; // 프로세스의 레퍼런스 방문횟수 +1
                    }
                }
                else{ // 할당된 페이지
                    PTE[cur_pte].ref++; // 방문+1
                    //printf("Frame %03d\n", PTE[cur_pte].frame);
                    arr[cur_pro->pid][cur_ref]++; // 프로세스의 레퍼런스 방문횟수 +1
                }
                pVisted_cnt[cur_pro->pid]++; // 프로세스 방문 횟수 +1
                ref_index[cur_pro->pid]++; // 현재 레퍼런스의 작업 위치
                all_ref++; // 모든 프로세스 방문횟수 +1
```

프로세스를 읽어오며 저장했던 모든 레퍼런스를 수행할 때까지 리스트를 순회(while-> list_for_each_entry() 사용) 해당 프로세스의 모든 레퍼런스 수행 완료 시 continue, 현재 레퍼런스를 가지고 프레임의 번호와 pte의 값을 구한 후 구한 값으로 pte변수의 프레임 삽입

현재 레퍼런스가 invalid할 경우 pagefault로 간주하고 out of memory 체크 후 프레임 할당 및 관련 정보 업데이트

현재 레퍼런스가 valid할 경우 관련 정보 업데이트

④print_info()함수

```
void print_info(frame* PAS){
    process *cur_pro;

    list_for_each_entry(cur_pro, &job_q, job)
    {

        int cur_ref = cur_pro->references[ref_index[cur_pro->pid]]; // 현재 탐색 중인 레퍼런스
        int cur_frame = cur_ref/8; // 레퍼런스가 들어있는 pagetable의 frame 번호
        int cur_pte = cur_ref%8; // 현재 pte값
        PTE = (pte *)PAS[cur_frame].b; // 현재 frame에 PTE 넣음

        printf("** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n", cur_pro->pid, frame_cnt[cur_pro->pid]+8, pagefault_cnt[cur_pro->pid], pVisted_cnt[cur_pro->pid]);


        for(int j=0; j<64; j++)
        {
            if(arr[cur_pro->pid][j] != 0)
            {
                printf("%03d -> %03d REF=%03d\n", j, frame_arr[cur_pro->pid][j], arr[cur_pro->pid][j]);
            }
        }
    }
    printf("Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n", frame_index, all_pf, all_ref);
}
```

리스트를 순회하며 프로세스별 정보 출력 후 모든 프로세스의 정보를 취합하여 출력

⑤clear_process()함수

```
294  void clear_process(frame* PAS){
295      process *ptr1,*ptr2;
296
297      list_for_each_entry_safe(ptr1, ptr2, &job_q, job){
298          list_del(&ptr1->job); // 노드 동적할당 해제
299          free(ptr1->references);
300          free(ptr1);
301      }
302      free(PAS);
303  }
```

리스트를 순회하며 동적할당을 해제

⑥demand_paging()함수

```
304  void demand_paging(frame* PAS){
305      load(PAS);
306      operation(PAS);
307      print_info(PAS);
308      clear_process(PAS);
309  }
```

모든 함수들을 묶어서 실행하기 위한 함수

⑦main()함수

```
175  int main(int argc,char* argv[]){
176      frame* PAS= (frame*)malloc(PAS_SIZE); // PAS 선언
177      demand_paging(PAS);
178  }
```

demand_paging() 함수 실행

4) 결과

- jota 제출

Submission of 운영체제 과제 3-1 by os202011632

View source
Resubmit

Compilation Warnings

```
oshw31c.c: In function 'main':
oshw31c.c:294:13: warning: unused variable 'cur_pte' [-Wunused-variable]
  294 |         int cur_pte = cur_ref%8; // 현재 pte값
      |             ^~~~~~~
oshw31c.c:166:30: warning: unused variable 'next' [-Wunused-variable]
  166 |     process *cur_pro, *ptr, *next;// 현재 process 가르킴
      |                              ^~~~
oshw31c.c:166:24: warning: unused variable 'ptr' [-Wunused-variable]
  166 |     process *cur_pro, *ptr, *next;// 현재 process 가르킴
      |                        ^~~
oshw31c.c:187:17: warning: ignoring return value of 'fread', declared with attribute warn_unused_result [-Wunused-result]
  187 |                 fread(&cur_pro->references[i], sizeof(char), 1, stdin); // cur의 references 읽어옴
      |                 ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Execution Results

✔ ✔ ✔ ✔ ✔

> **Test case #1:** AC [0.004s,524.00 KB] (2/2)
> **Test case #2:** AC [0.004s,524.00 KB] (2/2)
> **Test case #3:** AC [0.004s,524.00 KB] (2/2)
> **Test case #4:** AC [0.004s,524.00 KB] (2/2)
> **Test case #5:** AC [0.004s,524.00 KB] (2/2)

Resources: 0.019s, 524.00 KB
Final score: 10/10 (10.0/10 points)

- test3.bin

```
ubuntu@202011632:~/hw3$ gcc -o os3-1 os3-1.c
ubuntu@202011632:~/hw3$ cat test3.bin | ./os3-1
** 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
```

5) 과제 수행 시 어려웠던 점

(1) 메모리에 대한 정확한 개념을 이해하고 구현하는 부분에서 어려움을 겪었다. 머릿속에만 있는 내용을 정확히 정리를 위해 그림을 그리며 추상화를 하였다. pagetable은 연속된 8개의 프레임으로 이루어져있고, 64PTE를 가지기 때문에 1개의 프레임에는 8개의 PTE가 들어간다는 것을 알았다. 그렇게 만들어진 pagetable은 다시 PAS에 저장되며 한 개의 pagetable은 PAS에서 8칸의 프레임을 차지한다는 사실을 정리할 수 있었다. 이렇게 정리를 끝낸 후 구현을 시작했다.

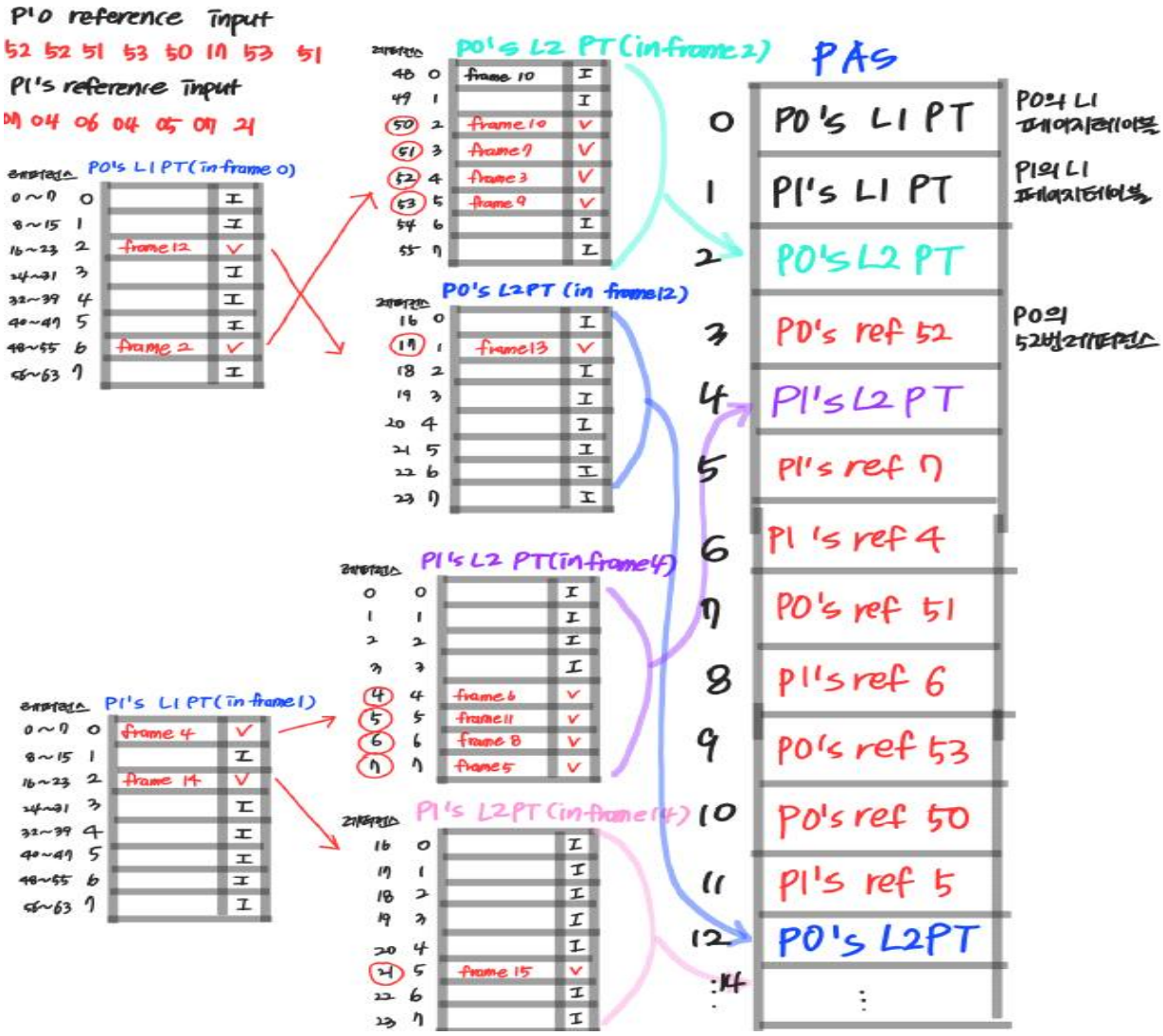
(2) test3-4.bin 파일로 테스트를 할 때 pacing 단계에서 0, 1, 2, 3 프로세스의 리스트를 차례로 순회하다가 1번 프로세스의 레퍼런스가 먼저 끝나 0번에서 2번으로 건너뛰어 순회하는 과정이 제대로 구현되지 않았다. 이에 조건문과 리스트에 대한 여러 자료를 찾아보았고 실수를 찾을 수 있었다. 한 프로세스의 레퍼런스가 모두 끝나면 break를 사용해주었는데 여기서 연결이 아예 끊긴 것이다. 조건문에 대한 여러 자료를 검색하며 잘못된 것을 깨닫고 break 대신 contine로 수정하여 1번 프로세스를 건너뛰고 다음 프로세스를 순회할 수 있도록 수정하였다.

2. 과제 3-2

1) 자율진도표

단계	완료여부
2-level Hierarchical Page Table에 대한 이해 및 추상화	O
프로세스 로드와 동시에 level1 PageTable 생성 및 초기화	O
리스트 순회 (1): Level1 PageTable: InValid 경우 설계	O
리스트 순회 (2): Level1 PageTable: Valid, Level2 PageTable: InValid 경우 설계	O
리스트 순회 (3): Level1 PageTable: Valid, Level2 PageTable: Valid 경우 설계	O
out of memory 상황 추가	O
list_for_each_entry() 이용 순회하며 결과 출력	O
과제 3-2 완료(JOTA 확인)	O

2) 추상화



3) 프로그램 실행 흐름

①load(프로세스의 정보 Load)

②operation(실제 동작)

③print_info(수행 후 결과 출력)

④clear_process(사용한 메모리 동적할당 해제)

4) 전체 코드 설명

0~129: 리스트에 관한 내용으로 생략

①전역변수 및 구조체

```
131  typedef struct{
132      unsigned char frame; // 할당된 frame
133      unsigned char vflag; // vaild-invaild bit
134      unsigned char ref; // 참조 횟수
135      unsigned char pad; // padding
136
137  } pte; // 페이지테이블 엔트리(4B)
138
139  typedef struct{
140      int pid; // 프로세스 pid
141      int ref_len; // 페이지 개수
142      unsigned char *references; // 페이지
143      struct list_head job; // job이라는 리스트의 노드 생성
144  } process;
145
146  typedef struct {
147      unsigned char b[PAGESIZE]; // 페이지가 들어가있는 배열
148  } frame;
149
150  process p; // process 구조체 p
151  LIST_HEAD(job_q); // 리스트 헤드 선언
152  int frame_index; // 프레임 index
153  int cnt=0; // 도는 레퍼런스의 개수 누적
154  int oom=1; // out of memory의 경우
155  int all_pf=0; // 전체 페이지 폴트
156  int all_ref=0; // 전체 레퍼런스
157  int pVisted_cnt[65]={0, }; // 프로세스별 방문 횟수 저장
158  int ref_index[11]; // 레퍼런스 주소 저장
159  int frame_cnt[11]; // 프로세스별 프레임 사용 개수
160  int pagefault_cnt[11]; // 프로세스별 페이지폴트 개수
161  int arr[11][65]; // 각 프로세스별 레퍼런스와 그때의 방문 횟수 저장 arr[0][0]: 0번 pid의 0번째 레퍼런스의 방문 횟수 =0
162  int frame_arr[11][65]; // 각 프로세스별 레퍼런스와 그때 할당된 frame 번호 저장
163  int allref_cnt;
164
165  void load(frame* PAS);//프로세스의 정보를 읽어오는 함수
166  void operation(frame* PAS);//실제 2-Level Hierarchical Paging을 수행하는 함수
167  void print_info(frame* PAS);// 동작 후 정보 출력 함수
168  void clear_process(frame* PAS);// 동적할당 해제 함수
169  void demand_paging(frame* PAS);// 동작함수들을 묶어 실행하는 함수
```

변수 및 구조체에 대한 설명은 주석을 참조

①load()함수

```
178  void load(frame* PAS){
179      process *cur_pro;// 현재 process 가르킴
180
181      while(1) // bin에서 읽어옴
182      {
183          cur_pro = malloc(sizeof(process));
184
185          if(fread(&p, sizeof(int)*2, 1, stdin) == 1)// pid, ref_len 를 읽어옴
186          {
187              allref_cnt += p.ref_len; // 모든 레퍼런스의 개수를 저장
188              //p의 정보를 cur로 옮겨줌(cur정보 초기화)
189              cur_pro-> pid = p.pid;
190              cur_pro-> ref_len = p.ref_len;
191              cur_pro->references = malloc(sizeof(unsigned char) *cur_pro->ref_len); // cur의 레퍼런스 선언
192              INIT_LIST_HEAD(&cur_pro->job); // cur의 job리스트 초기화
193              list_add_tail(&cur_pro->job, &job_q); // job_q 리스트와 cur의 job 연결
194
195              for(int i=0; i<cur_pro->ref_len; i++)
196              {
197                  fread(&cur_pro->references[i], sizeof(char), 1, stdin); // cur의 references 읽어옴
198              }
199
200              // LV1 페이지 테이블 만들기
201              pte *PTE = (pte *)PAS[frame_index].b;
202
203              for(int i=0; i<8; i++) // 페이지 테이블은 8칸으로 나누어짐
204              {
205                  // 각 프로세스의 Lv1 PT를 초기화(2개 청소)
206                  PTE[i].frame=0;
207                  PTE[i].vflag = PAGE_INVALID;
208              }
209              frame_index++; // 프로세스 당 Lv1 페이지 테이블을 위한 프레임 할당(1개)
210          }
211          else // bin 전부 돌면 나감
212          {
213              break;
```

PAS를 매개변수로 받아온 후, 프로세스의 정보를 읽어와 프로세스 별로 동적할당. 프로세스의 레퍼런스 길이에 따라 레퍼런스 동적할당 후 리스트에 저장. Level1 페이지 테이블을 만들어 프레임에 삽입

②operation(실제 동작)

```
217  void operation(frame* PAS){
218      process *cur_pro;
219      while(oom){
220          if(cnt == allref_cnt){ // 한 프로세스 당 모든 레퍼런스의 순회가 끝났을 때 while문을 나감
221              break;
222          }
223          // PAS 채우기
224          list_for_each_entry(cur_pro, &job_q, job){ // job_q리스트 순회
225              // 레퍼런스 다 돌면 순회 종료
226              if(ref_index[cur_pro->pid] == cur_pro->ref_len){ // 몇번째 레퍼런스인지(1,2,3...) = 길이 -> 종료
227                  continue; // 다음 리스트로 건너뜀
228              }
229              cnt++;
230              // 현재 레퍼런스 페이지테이블에 할당해줌
231                  int cur_ref = cur_pro->references[ref_index[cur_pro->pid]]; // 현재 탐색 중인 레퍼런스 정보(52, 7...)
232                  int lv1_frame = cur_ref/8; // 레퍼런스가 들어갈 lv1페이지 테이블 주소
233                  int lv2_frame = cur_ref%8; // lv2의 프레임(52의 lv2_frame=4)
234                  pte *lv1_pte = (pte *)PAS[cur_pro->pid].b; // lv1 페이지테이블의 프레임에 lv1_PTE를 넣음(52: 6번 frame)
```

프로세스를 읽어오며 저장했던 모든 레퍼런스를 수행할 때까지 리스트를 순회(while-> list_for_each_entry() 사용) 해당 프로세스의 모든 레퍼런스 수행 완료 시 continue, 레퍼런스의 정보를 통해 level1, level2 페이지 테이블의 주소를 구함

(1) level1 pagetable이 invalid 할 경우

```
235                  if(lv1_pte[lv1_frame].vflag == PAGE_INVALID){ // Lv1 할당 x
236                      if(frame_index == MAX_REFERENCES){// 더이상 할당해 줄 프레임이 없을 때
237                          printf("Out of memory!!\n");
238                          oom = 0;
239                          break;}
240                      else{
241                          pagefault_cnt[cur_pro->pid]++; // 프로세스 별 페이지 폴트 개수+1
242                          all_pf++; // 전체 pagefault+1
243                          // Lv1 칸 정보 업데이트(6번 프레임)
244                          lv1_pte[lv1_frame].frame = frame_index; // 남은 frame을 할당
245                          lv1_pte[lv1_frame].vflag = PAGE_VALID; // 페이지 할당됐다고 표시
246                          pte *lv2_pte = (pte *)PAS[frame_index].b; // 2번 프레임 참소
247                          frame_index++; // 프레임 하나 사용
248                          frame_cnt[cur_pro->pid]++; // 프로세스 별 할당한 프레임 개수
249                          // Lv2 페이지테이블 만들기
250                          for(int i=0; i<8; i++){
251                              // lv2 페이지 테이블 초기화(2번 프레임 청소)
252                              lv2_pte[i].frame=0;
253                              lv2_pte[i].vflag = PAGE_INVALID;
254                          }
255                          if(frame_index == MAX_REFERENCES){ // 더이상 할당해 줄 프레임이 없을 때
256                              printf("Out of memory!!\n");
257                              oom = 0;
258                              break;
259                          }
260                          pagefault_cnt[cur_pro->pid]++; // 프로세스 별 페이지 폴트 개수+1
261                          all_pf++; // 전체 pagefault+1
262
263                          // Lv2칸 정보 업데이트(2번 프레임 값 할당)
264                          lv2_pte[lv2_frame].frame = frame_index;
265                          lv2_pte[lv2_frame].vflag = PAGE_VALID;
266                          lv2_pte[lv2_frame].ref++;
267                          frame_index++;
268                          frame_cnt[cur_pro->pid]++; // 프로세스 별 할당한 프레임 개수
```

레퍼런스를 페이지테이블에 할당해줄 때 level1 페이지 테이블이 invalid한 경우, pagefault를 증가해주고 페이지 테이블에 레퍼런스 할당, 할당해 줄 프레임이 없을 경우 out of memory처리. 레퍼런스 할당 후 그에 해당하는 level2 페이지 테이블 생성 및 관련 정보 업데이트

(2) Level1 페이지 테이블이 valid한 경우

```
269                 }
270             }
271             else{ // Lv1 할당 ○
272                 lv1_pte[lv1_frame].ref++; // 방문+1
273 
274                 pte *lv2_pte = (pte *)PAS[lv1_pte[lv1_frame].frame].b; // lv2의 PTE생성
275                 if(lv2_pte[lv2_frame].vflag == PAGE_INVALID){ // lv2 할당 x
276                     if(frame_index == MAX_REFERENCES){ // 더이상 할당해 줄 프레임이 없을 때
277                         printf("Out of memory!!\n");
278                         oom = 0;
279                         break;
280                     }
281                     pagefault_cnt[cur_pro->pid]++; // 프로세스 별 페이지 폴트 개수+1
282                     all_pf++; // 전체 pagefault+1
283 
284                     // Lv2에 값을 할당
285                     lv2_pte[lv2_frame].frame = frame_index;
286                     lv2_pte[lv2_frame].vflag = PAGE_VALID;
287                     lv2_pte[lv2_frame].ref++;
288 
289                     frame_index++;
290                     frame_cnt[cur_pro->pid]++; // 프로세스 별 할당한 프레임 개수
291                 }
292 
293                 else{ // lv2 할당 ○
294                     lv2_pte[lv2_frame].ref++; // 방문+1
295                 }
296             }
297             pVisted_cnt[cur_pro->pid]++; // 프로세스 방문 횟수 +1
298             ref_index[cur_pro->pid]++; // 현재 레퍼런스의 작업 위치
299             all_ref++; // 모든 프로세스 방문횟수 +1
```

- level2 페이지 테이블이 invalid: 페이지 폴트 증가. level2 페이지 테이블에 관련 정보를 업데이트, 할당해 줄 프레임이 없을 경우 out of memory 출력

- level2 페이지 테이블 valid: 관련 정보 업데이트

③print_info(수행 후 결과 출력)

```
303 void print_info(frame *PAS){
304     process *cur_pro;
305 
306 
307     list_for_each_entry(cur_pro, &job_q, job)
308         {
309             pte *p1 = (pte *)PAS[cur_pro->pid].b; // lv1 순회
310 
311             printf("** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n", cur_pro->pid, frame_cnt[cur_pro->pid]+1, pagefault_cnt[cur_pro->pid], pVisted_cnt[cur_pro->pid]);
312 
313             for(int i=0; i<8; i++)
314             {
315                 pte *p2 = (pte *)PAS[p1[i].frame].b;
316                 if(p1[i].vflag != 0) // lv1 페이지 테이블 valid일 때
317                 {
318                     printf("(L1PT) %03d -> %03d\n", i, p1[i].frame);
319 
320                     for(int j=0; j<8; j++) // 그때의 lv2 페이지 테이블 탐색
321                     {
322                         if(p2[j].vflag != 0)
323                         {
324                             printf("(L2PT) %03d -> %03d REF=%03d\n", i*8+j, p2[j].frame, p2[j].ref);
325                         }
326                     }
327                 }
328             }
329         }
330         printf("Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n", frame_index, all_pf, all_ref);
```

리스트를 순회하며 프로세스별 정보 출력 후 모든 프로세스의 정보를 취합하여 출력

④clear_process(사용한 메모리 동적할당 해제)

```
332  void clear_process(frame *PAS){
333      process *ptr1,*ptr2;
334
335      list_for_each_entry_safe(ptr1, ptr2, &job_q, job){
336          list_del(&ptr1->job); // 노드 동적할당 해제
337          free(ptr1->references);
338          free(ptr1);
339      }
340      free(PAS);
341  }
```

리스트를 순회하며 동적할당을 해제

⑥demand_paging()함수

```
342  void demand_paging(frame *PAS){
343      load(PAS);
344      operation(PAS);
345      print_info(PAS);
346      clear_process(PAS);
347  }
```

모든 함수들을 묶어서 실행하기 위한 함수

⑦main()함수

```
171  int main(int argc,char* argv[]){
172
173      frame* PAS= (frame*)malloc(PAS_SIZE); // PAS 선언
174
175      demand_paging(PAS);
```

demand_paging()함수를 실행

5) 결과

- jota 제출

Submission of 운영체제 과제 3-2 by os202011632

View source
Resubmit

Compilation Warnings

```
oshw32c.c: In function ‘main’ :
oshw32c.c:167:30: warning: unused variable ‘next’ [-Wunused-variable]
  167 |     process *cur_pro, *ptr, *next;// 현재 process 가르킴
      |                              ^~~~
oshw32c.c:167:24: warning: unused variable ‘ptr’ [-Wunused-variable]
  167 |     process *cur_pro, *ptr, *next;// 현재 process 가르킴
      |                        ^~~
oshw32c.c:188:17: warning: ignoring return value of ‘fread’ , declared with attribute warn_unused_result [-Wunused-result]
  188 |                 fread(&cur_pro->references[i], sizeof(char), 1, stdin); // cur의 references 읽어옴
      |                 ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Execution Results

✔✔✔✔✔

> Test case #1: AC [0.003s,524.00 KB] (2/2)
> Test case #2: AC [0.003s,524.00 KB] (2/2)
> Test case #3: AC [0.004s,524.00 KB] (2/2)
> Test case #4: AC [0.004s,524.00 KB] (2/2)
> Test case #5: AC [0.004s,524.00 KB] (2/2)

Resources: 0.018s, 524.00 KB
Final score: 10/10 (10.0/10 points)

- test3.bin

```
ubuntu@202011632:~/hw3$ gcc -o os3-2 os3-2.c
ubuntu@202011632:~/hw3$ cat test3.bin | ./os3-2
** 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
```

6) 과제 수행 시 어려웠던 점 및 해결 방안

(1) pte *lv1_pte = (pte *)PAS[cur_pro->pid].b;

level1 pagetable에 값들을 할당할 때 사용하는 포인터 lv1_pte의 접근 위치 설정에 어려움을 겪었다. 처음엔 레퍼런스가 들어갈 level1의 페이지 주소, 즉 레퍼런스/8로 접근하면 될 거라고 생각했지만, 그러면 level1 pagetable 내부주소가 아니라 PAS 전체의 주소로 접근하게 되었다. 이에 추상화를 통해 프로세스 전체에 접근해야한다는 사실을 알게 되었고 문제를 해결할 수 있었다.

(2) pte *lv2_pte = (pte *)PAS[lv1_pte[lv1_frame].frame].b;

(1)과 동일한 이유로, level2 pagetable을 만들어줄 때도 접근 레퍼런스%8로 접근하여 PF를 확인하기도 전에 공간을 할당해주는 오류를 범했는데, 개념 이해 후 위와 같이 고쳐주었다.

(3) 실제 레퍼런스에 프레임을 할당해주는 과정에서도 pagefault가 발생할수도 있다는 점을 간과하여 out of memory 발생 조건을 사전에 확인하지 않아 전체 pagefault가 맞지 않아 의도한 기능대로 동작하지 않는 현상이 있었다. 처음부터 레퍼런스가 프레임에 할당되기까지의 과정을 따라가다가 발견하여 수정하였다.