운영체제 과제 3
보고서

컴퓨터인공지능학부 202323007 이진선

1. 코드 및 코드 설명

1) 3-1

3-1의 코드 함수는 크게 5가지로 나뉜다. load_process()은 프로세스 정보를 입력받고 각 프로세스에 PAGE TABLE 프레임을 할당한다. simulation()은 프로세스 각각 한 번에 한 가상 페이지를 참조하면서 demand paging 을 수행한다. print_result()은 프로세스마다의 결과를 출력한다. mem_clear()은 동적 할당된 메모리를 해제한다.

```
6   #define PTE_SIZE (4)
7   #define PAGE_INVALID (0)
8   #define PAGE_VALID (1)
9   #define MAX_REPERENCES (256)
10
11  typedef struct {
12      unsigned char frame; //frame number
13      unsigned char vflag;  //valid flag
14      unsigned char ref;  //reference counter
15      unsigned char pad;  //padding
16  } pte;
17
18  typedef struct {
19      int pid;
20      int ref_len;
21      unsigned char *references; //참조할 페이지 인덱스 목록
22  } process_raw;
23
24  typedef struct {
25      process_raw raw;
26      int next_ref; //다음 참고 index
27      int page_fault; //page_fault count
28      int first_frame; //시작 프레임 번호호
29  } pcb;
```

```
32  static int PAGESIZE;
33  static int VAS_PAGES;
34  static int PAS_FRAMES;
35  static int PAS_SIZE;
36  static int VAS_SIZE;
37  static int PAGETABLE_FRAMES;
38
39  static unsigned char *pas = NULL;  //physical memory space
40  static pcb process[MAX_REPERENCES];
41  static int proccess_cnt = 0;
42  static int nxt_free_frame = 0;
43  int cur_process = 0;
```

```
45  void load_setting() { //pagesize, pas_frames, vas_pages input
46      if(fread(&PAGESIZE, sizeof(int), 1, stdin)!=1) { return; }
47      if(fread(&PAS_FRAMES, sizeof(int), 1, stdin)!=1 ) { return; }
48      if(fread(&VAS_PAGES, sizeof(int), 1, stdin)!=1) { return; }
49
50      PAS_SIZE = PAGESIZE*PAS_FRAMES;
51      VAS_SIZE = PAGESIZE*VAS_PAGES;
52      PAGETABLE_FRAMES = VAS_PAGES*PTE_SIZE/PAGESIZE;
53
54      pas = malloc(PAS_SIZE);
55      memset(pas, 0, PAS_SIZE);
56  }
```

Load_setting()은 시스템 설정을 읽어오는 함수이다. 페이지 크기, 물리 프레임 수, 가상 페이지 수 순서대로 파일을 읽어오고 이에 따른 세팅에 대한 연산을 진행한다. 더하여 전체 물리 메모리를 나타내기 위한 동적 할당을 진행한다.

```
58  int set_frame() { //아직 할당되지 않은 프레임번호를 전달
59      if(nxt_free_frame>=PAS_FRAMES) return -1;  //프레임이 모두 사용되었을 경우 -> -1반환환
60      return nxt_free_frame++;
61  }
```

Set_frame은 아직 할당되지 않은 프레임번호를 전달하는 함수로 프레임이 모두 사용되었을 경우에는 "OUT OF MEMORY"가 표시되어야하므로 -1값을 반환하도록 한다.

```
63  static pte *frame_move(int i) { //i번째 프레임을 pte* 형태로
64      return (pte *)(pas + i*PAGESIZE);
65  }
```

I번째 프레임이 시작되는 주소를 pte* 형태로 반환한다.

```c
int load_process() { //프로세스 정보를 받아오는 함수
    while(proccess_cnt<MAX_REPERENCES){
        int pid, ref_len;
        if(fread(&pid, sizeof(int), 1, stdin)!=1) break;
        if(fread(&ref_len, sizeof(int), 1, stdin)!=1) break;

        pcb *p = &process[cur_process];
        cur_process++;
        proccess_cnt++;
        p->raw.pid = pid;
        p->raw.ref_len = ref_len;
        p->raw.references = malloc(ref_len);
        if(fread(p->raw.references, 1, ref_len, stdin)!=ref_len) {
            return -1;
        }

        //시작프레임을 set_frame()으로 할당하기
        int base = set_frame();
        if(base == -1) {return -1; }
        for(int i = 1; i<PAGETABLE_FRAMES; ++i) { //나머지 PAGETABLE_FRAMES만큼 할당
            if(set_frame()==-1) {return -1; }
        }
        p->first_frame = base;
        memset(frame_move(base), 0, VAS_PAGES*sizeof(pte));

        p->next_ref = 0;
        p->page_fault = 0;
    }
    return 1;
}
```

Load_process는 프로세스 정보를 받아오는 함수로, 각 프로세스 마다 Pagetable_frames만큼을 할당한다. 프로세스 마다 RAW.REFERENCE를 메모리에 저장하고, 연속된 프레임을 할당하여 준다.

```c
int simulation() { //프로세스마다 순서에 따라서 가상 페이지를 참조하도록 하는 함수
    int check = proccess_cnt;

    while(check > 0 ){
        check = 0;
        for(int i = 0; i< proccess_cnt; ++i) {
            pcb *p = &process[i];
            if(p->next_ref >= p->raw.ref_len) continue;
            check ++ ;

            unsigned char page_check = p->raw.references[p->next_ref];
            pte *table = frame_move(p->first_frame);

            //페이지테이블에서 유효하지 않을 경우
            //새 프레임 할당 & PAGE_FAULT 증가가
            if(table[page_check].vflag == PAGE_INVALID) {
                int new_frame = set_frame();
                if(new_frame == -1) return -1;

                table[page_check].frame = new_frame;
                table[page_check].vflag = PAGE_VALID;
                table[page_check].ref = 1;
                p->page_fault ++;
            }else {
                table[page_check].ref++;
            }
            p->next_ref++;
        }
    }
    return 1;
}
```

Simulation() 은 프로세스마다 순서에 따라서 가상 페이지를 참조하도록 하는 함수로, pid 순서대로 번갈아가면서 페이지에 접근한다. 참조한 가상 페이지가 없을 경우에 페이지테이블에서 유효하지 않을 경우로 new_frame = set_frame()을 통해서 새 프레임을 할당하고, p->page_fault++; 를 통해서 page_fault를 1씩 증가시킨다.

만약 유효할 경우에는 table[page_check].ref++; 를 통해서 reference만 증가시킨다.

```
130  void print_result() { //프로세스마다 출력 및 총 합계 출력
131      int total_pageFault = 0;
132      int total_ref = 0;
133      int total_frame = nxt_free_frame;
134
135      for(int i = 0; i<proccess_cnt; ++i) {
136          pcb *p = &process[i];
137          fprintf(stdout, "** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n",
138              p->raw.pid, p->page_fault+PAGETABLE_FRAMES, p->page_fault, p->next_ref);
139
140          pte *table = frame_move(p->first_frame);
141          for(int j =0; j<VAS_PAGES; ++j) {
142              if(table[j].vflag == PAGE_VALID) {
143                  fprintf(stdout, "%03d -> %03d REF=%03d\n", j, table[j].frame, table[j].ref);
144              }
145          }
146
147          total_pageFault += p-> page_fault;
148          total_ref += p->next_ref;
149      }
150      fprintf(stdout, "Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n",
151          total_frame, total_pageFault, total_ref
152      );
153  }
```

Print_result()는 프로세스마다 출력 및 총 합계를 출력하는 함수로 프로세스를 process_cnt만큼 돌면서 각각 프로세스마다 결과를 모두 출력한다.

```
155  void mem_clear() { //메모리 해제
156      for(int i = 0; i< proccess_cnt; ++i) {
157          free(process[i].raw.references);
158      }
159      free(pas);
160  }
```

Mem_clear를 통해서 main이 종료되기 전에 할당되었던 동적 메모리를 해제한다.

```
162  int main(int argc, char *argv[]) {
163      //입력 값 받기
164      load_setting();
165
166      //메모리 부족을 확인하고 -1이면 OUT OF MEMORY가 출력되도록 작성
167      if(load_process()==-1||simulation()==-1) fprintf(stdout, "Out of memory!!\n");
168      //출력
169      print_result();
170
171      //memory 정리
172      mem_clear();
173
174      return 0;
175  }
```

If(load_process()==-1||simulation()==-1)을 통해서 메모리가 부족해서 종료하는 상황에는

load_process() 혹은 simulation()의 반환값이 -1로 들어오게 되고, out of memory가 출력되도록 한다.

2) 3-2

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

#define PTE_SIZE (4)
#define PAGE_INVALID (0)
#define PAGE_VALID (1)
#define MAX_REPERENCES (256)

typedef struct {
    unsigned char frame; //frame number
    unsigned char vflag;  //valid flag
    unsigned char ref;  //reference counter
    unsigned char pad;  //padding
} pte;

typedef struct {
    int pid;
    int ref_len;
    unsigned char *references; //참조할 페이지 인덱스 목록
} process_raw;

typedef struct {
    process_raw raw;
    int next_ref; //다음 참고 index
    int page_fault;  //page_fault count
    int L1_Frame; //L1이 저장되어있는 프레임 번호
} pcb;


static int PAGESIZE;
static int VAS_PAGES;
static int PAS_FRAMES;
static int PAS_SIZE;
static int VAS_SIZE;
static int PAGETABLE_FRAMES;

static int L2_Frame; //L2 페이지 테이블 프레임의 갯수수
static unsigned char *pas = NULL;  //physical memory space
static pcb process[MAX_REPERENCES];
static int proccess_cnt = 0;
static int nxt_free_frame = 0;
static int PT_entry;
```

```c
void load_setting() {
    if(fread(&PAGESIZE, sizeof(int), 1, stdin)!=1) { return; }
    if(fread(&PAS_FRAMES, sizeof(int), 1, stdin)!=1 ) { return; }
    if(fread(&VAS_PAGES, sizeof(int), 1, stdin)!=1) { return; }

    PAS_SIZE = PAGESIZE*PAS_FRAMES;
    VAS_SIZE = PAGESIZE*VAS_PAGES;
    PAGETABLE_FRAMES = VAS_PAGES*PTE_SIZE/PAGESIZE;
    PT_entry = PAGESIZE/PTE_SIZE;  //한 페이지에 가능한 pte갯수
    L2_Frame = VAS_PAGES/PT_entry;

    pas = malloc(PAS_SIZE);
    memset(pas, 0, PAS_SIZE);
}

int set_frame() {
    if(nxt_free_frame>=PAS_FRAMES) return -1;
    return nxt_free_frame++;
}

static pte *frame_move(int i) {
    return (pte *)(pas + i*PAGESIZE);
}
```

```c
int cur_process = 0;
int load_process() {
    while(proccess_cnt<MAX_REPERENCES){
        int pid, ref_len;
        if(fread(&pid, sizeof(int), 1, stdin)!=1) break;
        if(fread(&ref_len, sizeof(int), 1, stdin)!=1) break;

        pcb *p = &process[cur_process];
        cur_process++;
        proccess_cnt++;
        p->raw.pid = pid;
        p->raw.ref_len = ref_len;
        p->raw.references = malloc(ref_len);
        if(fread(p->raw.references, 1, ref_len, stdin)!=ref_len) {
            return -1;
        }

        p->L1_Frame = set_frame();
        if(p->L1_Frame == -1) {return -1; }
        memset(frame_move(p->L1_Frame), 0, PT_entry*sizeof(pte));

        p->next_ref = 0;
        p->page_fault = 0;
    }
    return 1;
}
```

L1 은 무조건 존재하니까 프로세스를 로드하는 순간에 L1 PT 를 위해서 프레임 하나를 할당한다. L2 PT 은 on_demand 로 할당하도록 한다. L2 는 필요할 때에 사용되므로 동적할당으로 작성한다.

```c
int set_frame() { //아직 할당되지 않은 프레임번호를 전달
    if(nxt_free_frame>=PAS_FRAMES) return -1;  //프레임이 모두 사용되었을 경우 -> -1반환
    return nxt_free_frame++;
}

static pte *frame_move(int i) { //i번째 프레임을 pte* 형태로
    return (pte *)(pas + i*PAGESIZE);
}
```

```c
int simulation() {
    int check = proccess_cnt;

    while(check > 0 ){
        check = 0;
        for(int i = 0; i< proccess_cnt; ++i) {
            pcb *p = &process[i];
            if(p->next_ref >= p->raw.ref_len) continue;
            check ++ ;

            unsigned char page_check = p->raw.references[p->next_ref];
            pte *L1 = frame_move(p->L1_Frame);
            unsigned char L1_index = page_check/PT_entry;
            unsigned char L2_index = page_check%PT_entry;

            if(L1[L1_index].vflag == PAGE_INVALID) {
                int new_frame = set_frame();
                if(new_frame == -1) return -1;

                L1[L1_index].frame = new_frame;
                L1[L1_index].vflag = PAGE_VALID;
                L1[L1_index].ref = 1;
                memset(frame_move(new_frame), 0, PT_entry*sizeof(pte));
                p->page_fault ++;
            }

            pte *L2 = frame_move(L1[L1_index].frame);

            if(L2[L2_index].vflag == PAGE_INVALID) {
                int frame = set_frame();
                if(frame == -1) {
                    return -1;
                }
                L2[L2_index].frame = frame;
                L2[L2_index].vflag = PAGE_VALID;
                L2[L2_index].ref = 1;
                p->page_fault++;
            }else {
                L2[L2_index].ref++;
            }
            p->next_ref++;
        }
    }
    return 1;
}
```

3-1 과 달리, L1 의 vflag 가 invalid 하면 L2 가 존재하지 않으므로 L2 프레임을 새로 할당하고, L2 가 invalid 할 경우에는 페이지 프레임을 새로 할당한다.

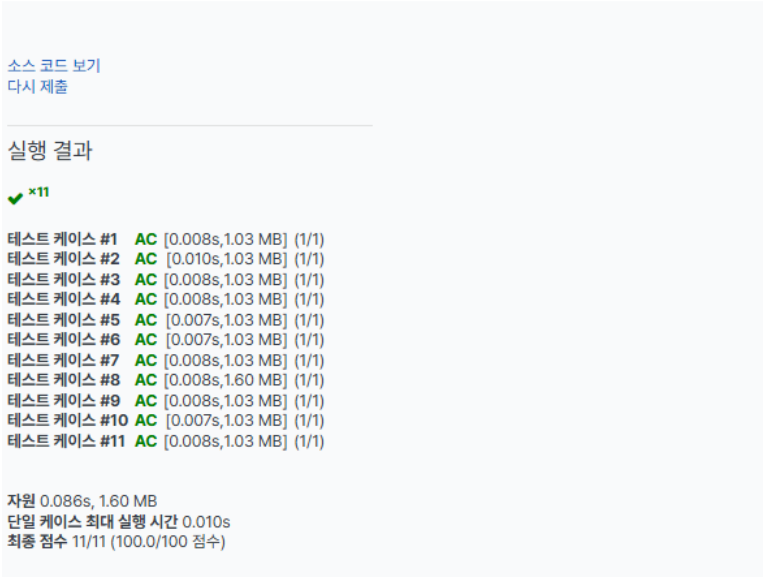
```
143  void print_result() {
144      int total_pageFault = 0;
145      int total_ref = 0;
146      int total_frame = nxt_free_frame;
147
148      for(int i = 0; i<proccess_cnt; ++i) {
149          pcb *p = &process[i];
150
151          fprintf(stdout, "** Process %03d: Allocated Frames=%03d PageFaults/References=%03d/%03d\n",
152              p->raw.pid, 1+p->page_fault, p->page_fault, p->next_ref);
153
154          pte *L1 = frame_move(p->L1_Frame);
155          for(int j =0; j<PT_entry; ++j) {
156              if(L1[j].vflag == PAGE_INVALID) continue;
157              fprintf(stdout, "(L1PT) %03d -> %03d\n", j, L1[j].frame);
158
159              pte *L2 = frame_move(L1[j].frame);
160              for(int k = 0; k<PT_entry; ++k) {
161                  if(L2[k].vflag == PAGE_VALID) {
162                      fprintf(stdout, "(L2PT) %03d -> %03d REF=%03d\n",
163                          j*PT_entry+k, L2[k].frame, L2[k].ref);
164                  }
165              }
166          }
167
168          total_pageFault += p-> page_fault;
169          total_ref += p->next_ref;
170      }
171      fprintf(stdout, "Total: Allocated Frames=%03d Page Faults/References=%03d/%03d\n",
172          total_frame, total_pageFault, total_ref
173      );
174  }
```

출력값도 L1 과 L2 를 분리하여 출력한다.

```
176
177  void mem_clear() {
178      for(int i = 0; i< proccess_cnt; ++i) {
179          free(process[i].raw.references);
180      }
181      free(pas);
182  }
183
184  int main(int argc, char *argv[]) {
185      //입력 값 받기
186      load_setting();
187      if(load_process()==-1||simulation()==-1) fprintf(stdout, "Out of memory!!\n");
188      //출력
189      print_result();
190
191      //memory 정리
192      mem_clear();
193
194      return 0;
195  }
```

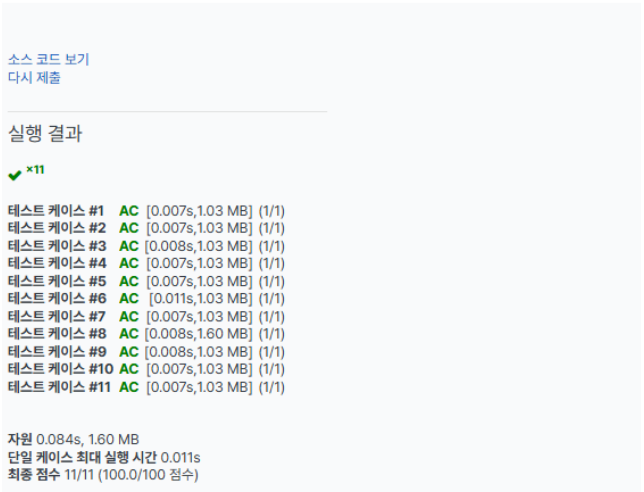
2. Litmus 제출 및 제출결과

202323007의2025 운영체제 과제 3-1제출



5 월 30 일 오전 5 시 59 분 제출

202323007의2025 운영체제 과제 3-2제출

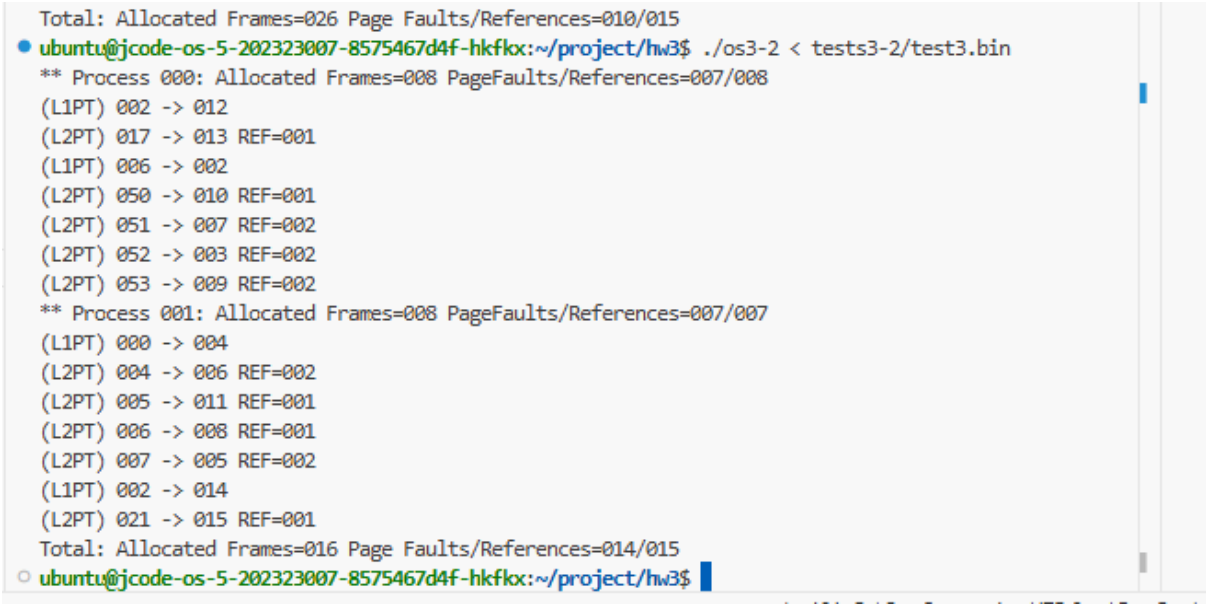


5 월 30 일 오전 6 시 1 분

3. 수행 결과

1) 3-1

2)3-2



4. 과제 수행 시 어려웠던 점 및 해결 방안

1) 과제 3-1

어려웠던 점: Demand paging 이 정확히 어떻게 구현되고 돌아가는지에 대한 이해가 부족해서 어떻게 코드를 구현해야하는 건지에 대해서 고민이 되었다. 특히, 구조체를 어떻게 구성해야 구현이 정확히 될지에 대해서 구상하는 게 어려웠던 것 같다.

해결 방안 : demand paging 이 무엇인지에 대해서 먼저 확실히 알아야한다고 생각했고, 코드를 구현하기 이전에 demand paging 에 대해서 확실히 정리한 뒤에 구조체를 구성하려고 하니, 구조체의 형태를 어떻게 구성해야하는지에 대해서 이해가 갔다. 구조체의 형태를 구상하고 나니 코드 구현 코드도 눈에 보이기 시작했다.

2) 과제 3-2

어려웠던 점: 1 개였던 level 이 2 개가 되었을 때, 기존에 가지고 있던 구조체에서 변경하여 활용하여야하는지에 대해서 고민을 많이 했다. 또한 과제의 내용을 잘못읽어 무조건 8 개씩으로만 쪼개지는 것으로 이해해서 코드를 잘못 구성하여 오류가 많이 났다.

해결 방안 : 오류 상황인 프로세스들을 살펴보고 차이점을 분석하였더니 8 이라고 고정하여 나누기와 나머지 연산을 했을 때, 문제가 발생한다는 점을 알게 되었고, 이를 활용하여 다른 테스트 케이스도 돌려보아서 어떠한 점에서 문제가 발생하는지에 대해서 중점적으로 살펴보고 위 문제가 어디 코드에서 발생한 것인지를 찾아냈다.

어디 코드에서 문제가 발생한 것인지에 대해서 찾고 나니, 해결할 수 있는 방법도 자연스럽게 찾게되어 해결할 수 있었다.