운영체제 과제 2
보고서

컴퓨터인공지능학부 202323007 이진선

1. 코드 및 코드 설명

1) 2-1

```
process *data;
LIST_HEAD(Q_JOB);
void input(){
      while(1) {
            data = malloc(sizeof(*data));   //입력 -> 제대로 못받을시에 바로 동적할당 해제하고 while문에서 나감
            if(fread(&data->pid, sizeof(int),1 , stdin)!=1) {    free(data); break; }
               if(fread(&data->arrival_time, sizeof(int), 1, stdin)!=1) { free(data);    break; }
               if(fread(&data->code_bytes, sizeof(int), 1, stdin)!=1){ free(data);    break; }
            INIT_LIST_HEAD(&data->job);
            data->code = malloc(sizeof(code_tuple)*data->code_bytes/2);
                        code_tuple codeTuple;
            for(int i = 0;i<data->code_bytes/2; i++){
                  if(fread(&codeTuple, sizeof(code_tuple), 1, stdin)!=1) break;
               data->code[i] = codeTuple;     }
               list_add_tail(&data->job, &Q_JOB); }
}
void reverse_output() {//역순회를 통해 출력
               list_for_each_entry_reverse(data, &Q_JOB, job) {
                           fprintf(stdout, "PID: %03d₩tARRIVAL: %03d₩tCODESIZE: %03d₩n",data->pid, data->arrival_time,
data->code_bytes);
            for(int i = 0;i<data->code_bytes/2; i++){
                                       fprintf(stdout, "%d %d₩n", data->code[i].op, data->code[i].len); }}}
void over(){//종료 - 동적 할당 해제
               process *nxt;
               list_for_each_entry_safe(data, nxt, &Q_JOB, job){ list_del(&data->job); free(data->code);free(data); }}
int main(int argc, char* argv[]){
      input();     reverse_output();    over();
}
```

구조체 - Process 구조체는 pid, arrival_time, code_bytes, code_tuple, job, ready, wait로 이루어져있으며 code_tuple은 한 명령어의 operation code와 length를 저장한다.

Input - 프로세스를 읽어오는 함수, 메모리 할당 및 읽기, 코드 튜플을 저장, job queue에 프로세스를 삽입하는 함수이다.

Reverse_output() - Q_JOB를 리스트의 역순으로 출력한다. 각 프로세스의 정보와 code_tuple의 정보를 for문을 통해서 출력한다.

Over() - 종료 시 동적 할당을 해제하기 위해서 만든 함수으로 노드를 리스트 내에서 삭제하고, 동적할당한 code_tuple과 data를 해제한다.

2) 2-2

```
//초기 선언자
#define Context_switching_time 10
#define pid_idle 100
```

```
LIST_HEAD(Q_JOB);
LIST_HEAD(Q_READY);
LIST_HEAD(Q_WAIT);

cpu *CPU1;
cpu *CPU2;
int prev_pid1 = 100;
int prev_pid2 = 100;
int total_clock = 0;

void FINAL_REPORT(){
        //CPU util 은 "전체 시뮬레이션 시간 가운데, 실제로 사용자, 커널 작업(op = 0 또는 1)을 실행한 비율"
        //전체 시뮬레이션 시간 -  total_clock
        //실제로 사용자가 작업을 실행한 비율 - cpu_idle_time을 제외한 total_clock
        //비율 - 퍼센테이지로 표현
        double cpu1_util = 100.0 * (total_clock - CPU1->idle_time) / total_clock;
        double cpu2_util = 100.0 * (total_clock - CPU2->idle_time) / total_clock ;
        fprintf(stdout, "*** TOTAL CLOCKS: %04d CPU1 IDLE: %04d CPU2 IDLE: %04d CPU1 UTIL: %2.2f%% CPU2
UTIL: %2.2f%% TOTAL UTIL: %2.2f%%₩n",
                        total_clock, CPU1->idle_time, CPU2->idle_time,cpu1_util, cpu2_util, (cpu1_util+cpu2_util)/2.0);
}
int cpu_length(cpu *c){
        int length = 0;
        struct list_head *now;
        list_for_each(now, &c->ready) {length ++ ; }
        return length; }
void Dynamic_free(){ //--------동적할당 해제
        process *data;
        process *nxt;
        list_for_each_entry_safe(data, nxt, &Q_JOB, job){
                list_del(&data->job); free(data->code); free(data); }free(CPU1); free(CPU2); }
process *make_idle() {
        process *p_idle;
        p_idle = malloc(sizeof(*p_idle));
        p_idle->pid = 100;
        p_idle->arrival_time = 0;
        p_idle->code_bytes = 2;
        p_idle->pc = 0;
        p_idle->load = 0;
        p_idle->code = malloc(sizeof(code_tuple));
        p_idle->code[0].op = 0xFF;
        p_idle->code[0].len = 0;

        INIT_LIST_HEAD(&p_idle->job);
        INIT_LIST_HEAD(&p_idle->ready);
        INIT_LIST_HEAD(&p_idle->wait);
        return p_idle;
}
```

```
void set_idle_process(){ //-------idle 프로세스 세팅
		process *idle1 = make_idle();
		process *idle2 = make_idle();
		list_add_tail(&idle1->job, &Q_JOB);
		list_add_tail(&idle2->job, &Q_JOB); }
void set_cpu(){
		CPU1 = malloc(sizeof(cpu));
		CPU2 = malloc(sizeof(cpu));
		INIT_LIST_HEAD(&CPU1->ready);
		CPU1->cur_process = NULL;
		CPU1->idle_time = 0;
		CPU1->cs_time = 0;
		CPU1->io_time = 0;

		INIT_LIST_HEAD(&CPU2->ready);
		CPU2->cur_process = NULL;
		CPU2->idle_time = 0;
		CPU2->cs_time = 0;
		CPU2->io_time = 0; }
int empty_readyqueue(cpu *c){
		struct list_head *now;
		list_for_each(now, &c->ready) {
			process *p = list_entry(now, process, ready);
			if(p->pid != pid_idle) return 0;}
		return 1; }
int cpu_free(cpu *c) {
		if((c->cur_process == NULL || c->cur_process->pid == pid_idle)&& empty_readyqueue(c)
			&& c->cs_time == 0&& c->io_time == 0) return 1;
		else return 0;}
int user_process_over() {
		process *p ;
		list_for_each_entry(p, &Q_JOB, job) {
			if(p->pid == pid_idle) continue;
			if(p->pc*2< p->code_bytes) return 0;}
		return 1; }
int over(){
		//idle을 제외하고 ready큐가 비어있는지
		//cpu가 완전히 놀고 있는지
		//모든 user프로세스가 끝났는지
		if(!cpu_free(CPU1)) return 0;
		if(!cpu_free(CPU2)) return 0;
		if(!user_process_over()) return 0;
		return 1; }
void load_balance(){
		//job큐의 앞에서부터 arrival시간으로 인해 도달한 프로세스(무조건 time이 동일한 애만 해야한다.)만 ready
queue에 넣고.
		//cpu의 ready queue의 길이를 비교하고 짧은 cpu에 프로세스를 넣는다.
		//push완료 -> 다음 순서를 cur로 잡고 다음 순서를 돌아야함.
		struct list_head *now, *next;
```

```
        int a = 0;
        list_for_each_safe(now, next, &Q_JOB) {
                process *check_p = list_entry(now, process, job);
                if(check_p->arrival_time > total_clock) continue;
                if(check_p->load) continue;
                if(CPU1->cs_time>=1   && CPU2->cs_time>=1) {       continue;                }
                if(check_p->pid==100&& a==0){
                        check_p->load = 1;
                        list_add_tail(&check_p->ready, &CPU1->ready);
                        fprintf(stdout, "%04d CPU%d: Loaded PID: %03d₩tArrival: %03d₩tCodesize: %03d₩n"
                                ,total_clock, 1, check_p->pid, check_p->arrival_time, check_p->code_bytes );
                        a++;       continue; }
                if(check_p->pid==100&& a==1){
                        list_add_tail(&check_p->ready, &CPU2->ready);
                        check_p->load = 1;
                        fprintf(stdout, "%04d CPU%d: Loaded PID: %03d₩tArrival: %03d₩tCodesize: %03d₩n"
                                ,total_clock, 2, check_p->pid, check_p->arrival_time, check_p->code_bytes );
                        continue;  }
                if(cpu_length(CPU1)<=cpu_length(CPU2)){
                        list_add_tail(&check_p->ready, &CPU1->ready);
                        check_p->load = 1;
                        fprintf(stdout, "%04d CPU%d: Loaded PID: %03d₩tArrival: %03d₩tCodesize: %03d₩n"
                                ,total_clock, 1, check_p->pid, check_p->arrival_time, check_p->code_bytes );
                }
                else {
                        list_add_tail(&check_p->ready, &CPU2->ready);
                        check_p->load = 1;
                        fprintf(stdout, "%04d CPU%d: Loaded PID: %03d₩tArrival: %03d₩tCodesize: %03d₩n"
                                ,total_clock, 2, check_p->pid, check_p->arrival_time, check_p->code_bytes );
                }}}
void cpu_time(cpu *c, int num){
        //context switching 중일때
        //현재 실행 가능한 프로세스가 없을 때 <-> 현재 실행 가능한 프로세스가 있을 때
        int flag = 0;
        if(c->cur_process
                &&c->cur_process->pid == pid_idle
                &&c->cs_time == 0
                && !list_empty(&c->ready)) {
                list_add_tail(&c->cur_process->ready, &c->ready);
                c->cur_process = NULL; }
        if(!c->cur_process) { //현재 프로세스를 실행시키는 게 없을 때
                if (!empty_readyqueue(c)) {
                        process *p;
                        p = list_entry(c->ready.next, process, ready);
                        if(p->pid == pid_idle) list_move_tail(&p->ready, &c->ready); }
                c->cur_process = list_entry(c->ready.next , process, ready);
                list_del(&c->cur_process->ready);
                if(total_clock==0) {
                        if(num==1){prev_pid1 = c->cur_process->pid; }
```

```
                                        else {prev_pid2 = c->cur_process->pid; }                                }
                        else {
                                flag = 1;
                                c->cs_time = Context_switching_time;
                                c->idle_time += Context_switching_time;
                        }}

                if(c->cs_time == 0 ){
                        code_tuple *cur_tuple = &c->cur_process->code[c->cur_process->pc];
                        if(cur_tuple->op == 0) {
                                if(--cur_tuple->len == 0) c->cur_process->pc++; }
                        else if(cur_tuple->op == 1){
                                if(c->io_time == 0) {
                                        c->io_time = cur_tuple->len;
                                        flag = 1;
                                        c->idle_time += c->io_time;
                                        fprintf( stdout,"%04d CPU%d: OP_IO START len: %03d ends at: %04d₩n"
                                                , total_clock, num, cur_tuple->len, total_clock+cur_tuple->len ); }

                                c->io_time--;
                                if(c->io_time == 0) {c->cur_process->pc++;}
                        }}
                if(c->cur_process &&c->cur_process->pid ==pid_idle &&c->cs_time == 0 &&!flag) c->idle_time += 1;
                //프로세스 종료
                if(c->cur_process&&c->cur_process->pc*2 >= c->cur_process->code_bytes) {
                        if(c->cur_process->pid!=100) {    c->cur_process = NULL; }
                        else {
                                list_add_tail(&c->cur_process->ready, &c->ready);
                                c->cur_process = NULL; }
                        c->io_time = 0 ; }
}
void switching(cpu *c, int num) {
                if(c->cs_time > 0) { //context switching중일 때에 cs_time감소, idle타임 증가
                        c->cs_time --;
                        if(c->cs_time == 0 ) {
                                int prev_pid = num == 1 ? prev_pid1 : prev_pid2;
                                fprintf(stdout, "%04d CPU%d: Switched₩tfrom: %03d₩tto: %03d₩n",
                                        total_clock, num,prev_pid,c->cur_process->pid);
                                if(num==1){prev_pid1 = c->cur_process->pid; }
                                        else {prev_pid2 = c->cur_process->pid; }
                        }                       }}
void simulation(){
                switching(CPU1, 1);
                switching(CPU2, 2);
                load_balance();
                cpu_time(CPU1, 1);
                cpu_time(CPU2, 2);
                total_clock++;
}
```

```
int main(int argc, char* argv[]){
        input();
        set_idle_process();
        set_cpu();
        load_balance();
        while(1){
                simulation();
                if(over()) break;
        }
        FINAL_REPORT();
        Dynamic_free();
        return 0;
}
```

2-1과 동일한 함수에 대해서는 코드 주요코드에서 제외

Input()

- 프로세스 구조체를 동적할당한 뒤에 pid, arrival_time, code_bytes를 읽는다. 만약 입력이 제대로 되지 않을 경우에 바로 프로세스를 동적 할당 해제를 한 뒤에 while문을 나간다.

- job, ready, wait를 init_list_head함수를 통해서 초기화한 뒤, list_add_tail(&data->job, &Q_JOB);을 통해서 job queue에 할당하여 load_balance()함수를 통해서 스케줄링한다.

Make_idle()

- cpu 2개에 idle프로세스를 각각 할당하기 위해서 idle 프로세스를 생성하는 함수를 제작하였다.

- idle_process임을 알 수 있도록 pid=100, code[0].op = 0xff를 무조건 할당을 하여주고, 그 외의 프로세스의 생성을 위한 초기화를 진행한다. 생성한 idle_process를 반환하고 set_idle_process를 통해서 cpu의 job queue의 마지막에 할당한다.

Set_idle_process()

- 생성한 idle_process를 jobqueue에 넣어 load_balance()를 통해서 각 cpu의 job queue에 idle_process를 넣는다.

- process *idle1 = make_idle(); 를 통해서 idle_process를 생성하고, list_add_tail(&idle1->job, &Q_JOB); 에 넣어둔다.

Set_cpu()

- 두개의 cpu를 생성하기 위해서 cpu를 생성하는 코드를 정리해둔 함수이다.

- cpu1 = malloc(sizeof(cpu))를 통해서 cpu를 동적할당을 하면서 cpu를 초기화한다.

Cpu_length(cpu *c)

- cpu_length는 cpu의 ready queue내에 있는 프로세스(실행을 기다리고 있는 프로세스)의 개수를 반환하는 함수

- list_for_each(now, &c->ready)를 통해서 ready queue를 순회하면서 length를 1씩 증가시켜 length

를 반환한다.

Dynamic_free()

- 시뮬레이션이 모두 종료된 후에 동적 할당한 메모리를 해제하기 위해서 만든 함수

- list_for_each_safe를 통해서 Q_JOB을 순회하면서 각 프로세스의 동적할당한 메모리를 해제한다. Free(data->code), free(data)

- 2개의 CPU를 동적할당 했으므로 free(CPU1), free(CPU2)를 통해서 메모리를 해제했다.

Empty_readyqueue(cpu *c)

- 현재 cpu의 readyqueue가 비어있는지 확인하는 함수이다.

- readyqueue에는 무조건 idle_process가 존재한다고 했으므로 idle_process만 남아있는지를 확인하면 비어있는지를 확인할 수 있다.

- list_for_each(now, &c->ready)를 통해서 readyqueue를 순회하면서 pid ==100인 프로세스만 제외하고 판별한다. (p->pid!=pid_idle) return 0;를 통해서 만약 pid가 100이 아니라면 바로 0을 반환한다.

Cpu_free(cpu *c)

- cpu가 완전히 비어있는상태인지 확인하는 함수이다.

- Empty_readyqueue와 다르다. Cpu가 일을 하지 않고 있는 상태여야하므로 다음과 같은 조건들을 사용하여 파악한다.

- 현재 process가 idle_process이거나 processrk 존재하지 않는 경우(c->cur_process == NULL || cur_process->pid == pid_idle), ready queue에 idle_process만 존재할 경우 (empty_readyqueue(c)==1), context switching을 하지 않고 있는 상황 (c->cs_time == 0), I/O 작업을 하지 않고 있는 상황(c->io_time == 0)

User_process_over()

- 실행시켜야하는 모든 프로세스가 종료되었는지 확인하는 함수이다.

- list_for_each_entry 를 활용하여 Q_JOB 를 순회하면서 idle을 제외하고 프로세스가 존재하는지를 확인한다. 만약 덜 실행된 프로세스가 하나라도 있으면 0을 반환한다.

Over()

- 시뮬레이션을 종료하는 요건이 충족되었는지 확인하는 함수

- cpu_free(CPU1) ,cpu_free(CPU2)를 통해서 cpu가 모두 할 일이 끝난 상태인지 확인하고, user_process_over()를 통해서 실행시켜야하는 프로세스가 모두 종료되었는지를 확인한다.

Load_balance()

- 프로세스가 도착했는지 여부를 확인하고 도착했다면 프로세스를 cpu에 분배하는 함수이다.

- list_for_each_safe 를 통해서 Q_JOB을 돌면서 프로세스를 확인한다. arrival_time > total_clock 일 경

우, 아직 도착하기 전이므로 다음 프로세스로 넘어간다. check_p ->load를 통해서 load가 이미 된 경우에는 재차 load하지 않도록 표시한다. 두 프로세스 모두 context_switching 중일 경우에는 load_balance가 불가능하므로 넘어간다. pid_idle은 하나만 할당되어있어야하므로 따로 할당한다. cpu_length(cpu1) <= cpu_length(cpu2) 일 경우에 cpu1에 프로세스를 load한다.

Cpu_time(cpu *c, int num)

- cpu가 현재 실행하고 있는 것이 무엇인지 확인하고 이를 count한다.

- idle_time, cs_time, io_time, pc를 통해서 분류한다.

- idle상황일 경우에는 idle_time을 증가시키는데, total_clock마다 1씩 증가시킬 경우, 오류가 날 경우가 많기 때문에 idle_process가 아닌 idle상황일 경우(io_time, cs_time)은 미리 더한다.

- idle_process를 실행시키고 있었는데 ready_queue에 프로세스가 있을 경우에 idle_process를 다시 readyqueue에 넣고, 현재의 process를 null로 만든다.

- 현재의 프로세스가 null이지만 idle_process 이외에 프로세스가 존재할 경우에는 idle_process를 맨 뒤에 넣고, 새로운 프로세스를 cur_process에 할당한다. 이때, context switching이 발생하므로 cs_time과 idle_time에 값을 넣는다.

- 현재 cpu작업, io작업을 진행할 경우에는 cpu작업일 경우에는 program counter(pc)를 1씩 증가시키고, io작업을 할 경우에는 idle상황이므로 io_time은 1씩 감소시키고 idle_time을 io_time만큼 증가시킨다. 조건식을 두지 않을 경우에는 idle_time에 io_time이 계속 더해지게 되므로 floag를 통해서 한번만 더하도록 한다.

- 현재 프로세스가 idle_process일 경우, idle_time을 1씩 증가시킨다.

- 프로세스가 종료될 수 있는 요건을 충족할 경우에는 프로세스를 null로 바꾼다.

Switching (cpu * c, int num)

- context switching이 진행될 경우에 cs_time을 1씩 감소하고 완료될 경우에 메시지를 출력한다.

Simulation()

- 시뮬레이션을 돌릴 때 필요한 함수를 담아둔다.

- switching(cpu, cpu number), load_balance(), cpu_time(cpu, cpu number), total_clock++

을 통해서 시뮬레이션을 순서대로 돌린다.

Final_report()

- 시뮬레이션이 끝나고 난 뒤 cpu의 util time과 total time, utile time을 통해서 이용률을 출력한다.

어려웠던 점 및 해결 방안

- cpu가 2개로 서로의 조건을 확인하고 결론을 도출해앟는 연산이 많아서, 비교하는 연산과 조건들을 설정하는데에 오래 걸렸다. Idle_time가 1씩 차이나서 이 오류를 잡는데에 시간이 오래 걸렸다. 이는 각 함수로 나눠서 각 함수가 실행될 때마다 함수마다의 initial과 idle_time을 출력하여 어디에서 문제가 발

생하는 것인지 파악하여 수정하였다.

- 조건들과 연산을 해야하는 것들이 많아서 하나의 코드 내에 뒀을 때에 작성하고 있던 코드가 어디서 비롯된 것인지 파악하기 어려운 문제점이 존재했다. 이는 특정한 기능을 하는 코드를 모두 함수로 분류하여 각각을 역할을 확실하게 분리하여 코드를 재사용했다.

2-3)

```
typedef struct {
    struct list_head ready;
    struct list_head wait;
    process *cur_process;
    int idle_time;
    int cs_time;
    int io_time;
    int mig_time;
} cpu;

int count_readyqueue(cpu *c){ //idle_process를 제외하고 readyqueue에 몇개의 프로세스가 있는지지
    struct list_head *now;
    int len = 0;
    list_for_each(now, &c->ready) {
        process *p = list_entry(now, process, ready);
        if(p->pid != pid_idle) len++;
    }
    return len;
}

int migration_condition(int num) { //migration을 할 조건인지 확인
    cpu *c, *from;
    if (num == 1) { c = CPU1; from = CPU2; }
    else { c = CPU2; from = CPU1; }
    if(cpu_free(c)&&c->mig_time == 0 &&
    (c->cur_process==NULL || c->cur_process->pid == pid_idle)&&
    (count_readyqueue(from)>=1)) return num;
    return 0;
}

void migration(int num) { //migration 처리
    struct list_head *now, *next;
    process *p;
    cpu *c, *from;
    if(num == 1) { c = CPU1; from = CPU2; }
    else { c = CPU2; from = CPU1; }

    if(c->mig_time > 0 ){
        c->mig_time --;
        if(c->mig_time == 0) {
            list_for_each_safe(now, next, &c->wait) {
```

```
                p = list_entry(now, process, wait);
                list_del(&p->wait);
                list_add_tail(&p->ready, &c->ready);
                fprintf(stdout, "%04d   CPU%d: Migrate : COMPLETED!₩tPID: %03d₩n", total_clock, num, p->pid);
            }
        }
    }
    else if(migration_condition(num)) {
        int move = count_readyqueue(from)/2;
        now = from->ready.prev;
        while(move-- > 0 && now != &from->ready){
            p = list_entry(now, process, ready);
            now = now->prev;
            if(p->pid == pid_idle) { move++; continue; }
            list_del(&p->ready);
            list_add_tail(&p->wait, &c->wait);
        }
        c->mig_time = 30;
    }

}
```

2-2 코드에서 migration에 관련된 코드를 추가한 코드이므로 추가한 코드에 대해서 설명한다.

Migration을 파악하기 위해서 과제에 자신의 migration이 진행되고 있지 않을 경우, 라는 조건이 있으므로 mig_time을 통해서 migration을 하고 있는중인지 아닌지를 확인한다.

Migration을 할 경우에 완료되기 전에 각각의 wait queue에 값을 넣어야하므로 cpu구조체 안에 struct list_head wait를 추가하였다.

Count_readyqueue(cpu *c)

- cpu의 ready queu에 idle_process를 제외하고 존재하는 프로세스의 수를 반환한다. 이는 migration 할 개수를 구하기 위해서 사용한다.

Migration_condition(int num)

- cpu로 migration을 진행할 조건이 되는지를 확인한다. 조건들은 cpu가 idle_process상태로 readyqueue에 아무 프로세스도 존재하지 않고, 현재 cpu에서는 migration을 하지 않는 상태, 다른 cpu는 ready queue에 프로세스가 존재할 경우에 migration을 실행한다. num을 통해서 num == 1일 경우, cpu1으로 프로세스를 이동할 조건이 되는지, num == 2일 경우에는 cpu2으로 프로세스를 이동시킬 조건이 되는지를 파악한다.

Migration(int num)

-실질적으로 migration을 처리하는 함수이다.

- 다른 cpu로부터 /2만큼 ready큐에서 꺼내와 c의 wait큐에 넣는다.

- 이때 변수는 넣어지는 곳을 c, 어디로부터 오는 데이터인지를 from으로 설정하였다.

- mig_time이 지날 시에는 출력문구와 함께 wait큐에 있는 프로세스들을 ready큐에 넣는다.

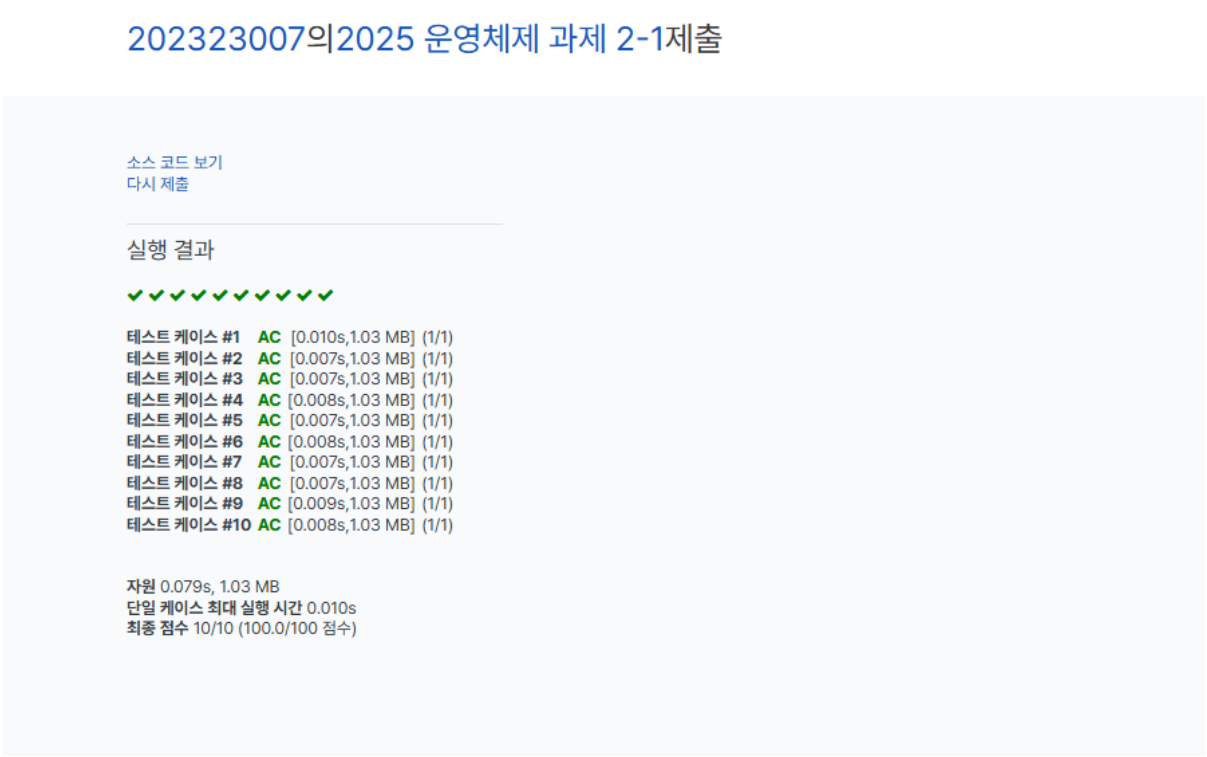
- 이때 idle_process는 이동시키지 않을 것이므로 예외처리를 통해서 이동시키지 않았다.

어려웠던 점 및 해결 방안

- 테스트 케이스의 경우에는 반례가 생기지 않아 어떠한 점에서 문제가 생기는 것인지파악하기 어려웠다. 이를 해결하기 위해서 문제가 발생할 만한 요소에 printf를 삽입하여 결과를 대조하는 방법을 선택하였다. 다만, 아쉽게도 3개의 테스트 케이스에 대해서는 성공하지 못했다.

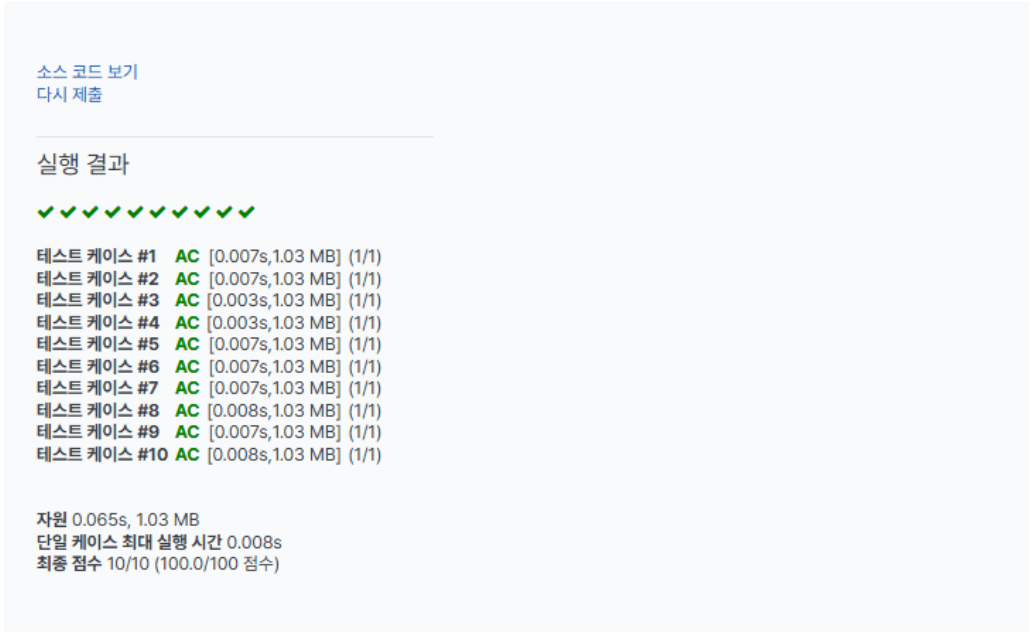
- migration을 어떤 식으로 나눠서 활용할지에 대해서 고민했다. 특히, wait queue를 각 cpu마다 사용해야하므로 어떠한 cpu로 프로세스를 이동시킬 건지가 중요해서 함수에 변수를 어떤 식으로 전달하여 cpu를 분류할 것인지가 어려웠다. 이 방식에 대해서는 최대한 다양하게(cpu를 함수로 직접 전달, 하나의 migration안에 cpu 두개를 확인하는 방식) 활용해보고 가장 코드가 간결하다고 생각하는 방식을 택하여 해결하였다.

2. Litmus 제출 및 제출결과



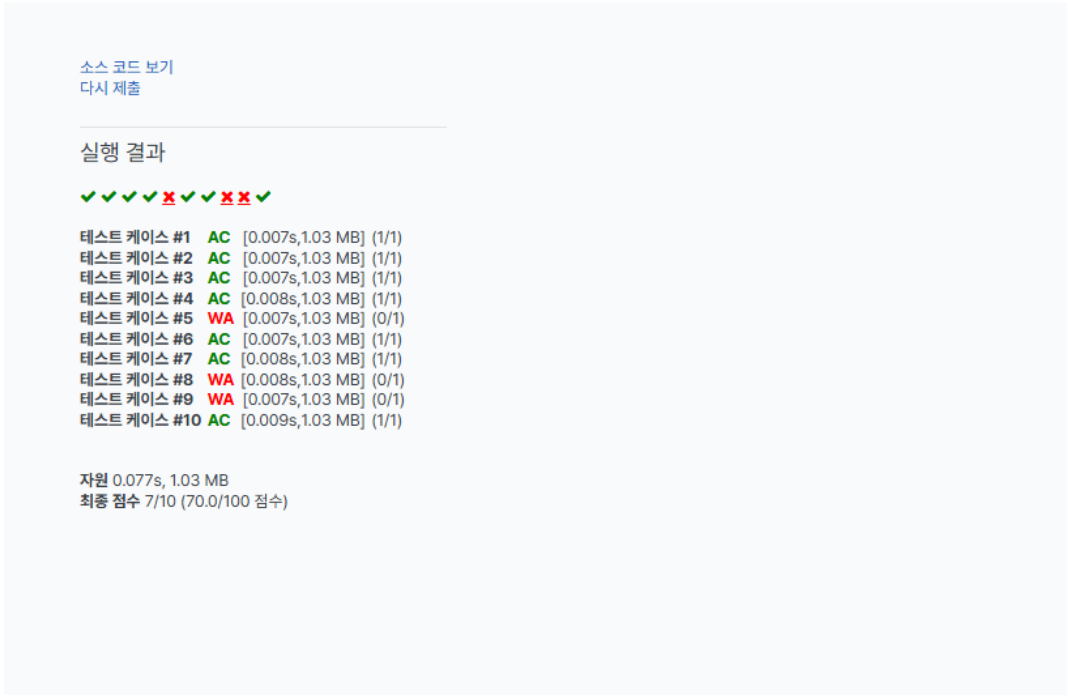
5 월 2 일 오후 5 시 36 분 제출

202323007의2025 운영체제 과제 2-2제출



5 월 2 일 오전 11 시 19 분

202323007의2025 운영체제 과제 2-3제출



5 월 2 일 오후 9 시 23 분

3. 수행 결과

1) 2-1

```
ubuntu@jcode-os-5-202323007-8575467d4f-hkfkx:~/project/hw2$ ./os2-1 < test2-1/test1.bin
PID: 002        ARRIVAL: 016    CODESIZE: 002
0 167
PID: 001        ARRIVAL: 003    CODESIZE: 006
0 2
1 72
0 5
PID: 000        ARRIVAL: 000    CODESIZE: 004
0 4
1 255
```

2)2-2

```
ubuntu@jcode-os-5-202323007-8575467d4f-hkfkx:~/project/hw2$ ./os2-2 < test2-2/test1.bin
0000 CPU1: Loaded PID: 000      Arrival: 000    Codesize: 004
0000 CPU1: Loaded PID: 100      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 100      Arrival: 000    Codesize: 002
0003 CPU2: Loaded PID: 001      Arrival: 003    Codesize: 006
0004 CPU1: OP_IO START len: 255 ends at: 0259
0013 CPU2: Switched     from: 100       to: 001
0015 CPU2: OP_IO START len: 072 ends at: 0087
0016 CPU1: Loaded PID: 002      Arrival: 016    Codesize: 002
0102 CPU2: Switched     from: 001       to: 100
0269 CPU1: Switched     from: 000       to: 002
*** TOTAL CLOCKS: 0436 CPU1 IDLE: 0265 CPU2 IDLE: 0429 CPU1 UTIL: 39.22% CPU2 UTIL: 1.61% TOTAL UTIL: 20.41%
ubuntu@jcode-os-5-202323007-8575467d4f-hkfkx:~/project/hw2$
```

3) 2-3

```
ubuntu@jcode-os-5-202323007-8575467d4f-hkfkx:~/project/hw2$ ./os2-3 < test2-3/test3.bin
0000 CPU1: Loaded PID: 001      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 002      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 003      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 004      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 005      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 006      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 007      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 008      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 009      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 010      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 011      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 012      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 014      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 015      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 016      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 017      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 018      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 019      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 020      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 021      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 022      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 023      Arrival: 000    Codesize: 002
0000 CPU1: Loaded PID: 100      Arrival: 000    Codesize: 002
0000 CPU2: Loaded PID: 100      Arrival: 000    Codesize: 002
0012 CPU2: Switched     from: 002       to: 004
0025 CPU2: Switched     from: 004       to: 006
0033 CPU1: Switched     from: 001       to: 003
0036 CPU2: Switched     from: 006       to: 008
0050 CPU2: Switched     from: 008       to: 010
0062 CPU2: Switched     from: 010       to: 012
0076 CPU2: Switched     from: 012       to: 015
0080 CPU1: Switched     from: 003       to: 005
0089 CPU2: Switched     from: 015       to: 017
0102 CPU2: Switched     from: 017       to: 019
0110 CPU1: Switched     from: 005       to: 007
0114 CPU2: Switched     from: 019       to: 021
0127 CPU2: Switched     from: 021       to: 023
0140 CPU1: Switched     from: 007       to: 009
0144 CPU2: Switched     from: 023       to: 100
0164  CPU2: Migrate : COMPLETED!        PID: 022
0164  CPU2: Migrate : COMPLETED!        PID: 020
0164  CPU2: Migrate : COMPLETED!        PID: 018
0174 CPU2: Switched     from: 100       to: 022
0187 CPU1: Switched     from: 009       to: 011
0199 CPU1: Switched     from: 011       to: 014
0212 CPU1: Switched     from: 014       to: 016
0229 CPU1: Switched     from: 016       to: 100
0234 CPU2: Switched     from: 022       to: 020
0249  CPU1: Migrate : COMPLETED!        PID: 018
0255 CPU2: Switched     from: 020       to: 100
0259 CPU1: Switched     from: 100       to: 018
*** TOTAL CLOCKS: 0278 CPU1 IDLE: 0110 CPU2 IDLE: 0183 CPU1 UTIL: 60.43% CPU2 UTIL: 34.17% TOTAL UTIL: 47.30%
ubuntu@jcode-os-5-202323007-8575467d4f-hkfkx:~/project/hw2$
```