운영체제 Lab1 보고서

-스케줄러 제작-

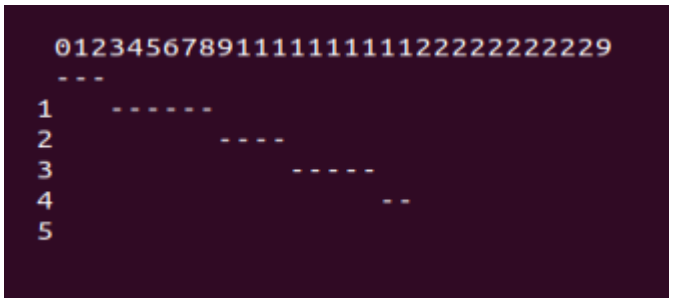
32181854 박준영

1. 실행결과

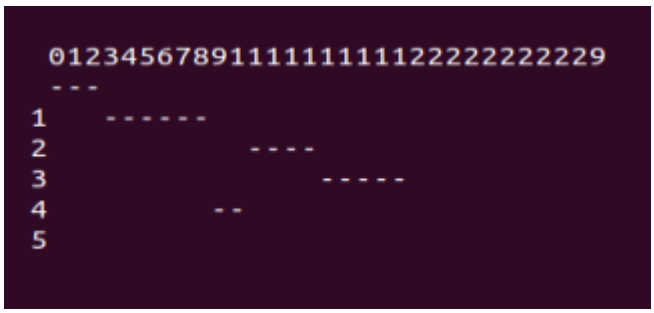
Job 초기화(24P)

```
oslab@oslab-DKU:~/Desktop/2021_DKU_OS/lab1_sched$ ./lab1_sched
How many processes do you create? > 5
arrived_time, service_time of process 0 >> 0 3
arrived_time, service_time of process 1 >> 2 6
arrived_time, service_time of process 2 >> 4 4
arrived_time, service_time of process 3 >> 6 5
arrived_time, service_time of process 4 >> 8 2
```

(1) FCFS



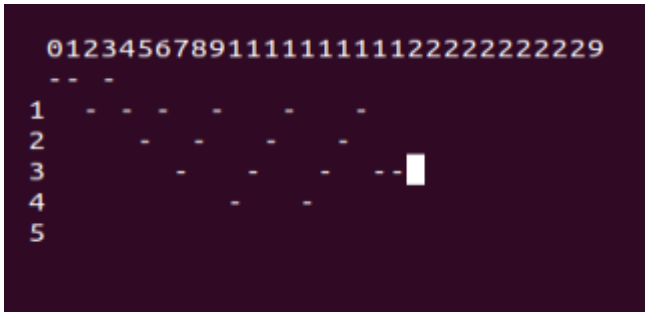
(2) SPN



(3) RR

```
oslab@oslab-DKU:~/Desktop/2021_DKU_OS/lab1_sched$ ./lab1_sched
How many processes do you create? > 5
arrived_time, service_time of process 0 >> 0 3
arrived_time, service_time of process 1 >> 2 6
arrived_time, service_time of process 2 >> 4 4
arrived_time, service_time of process 3 >> 6 5
arrived_time, service_time of process 4 >> 8 2
time quantum >> 1
```

Time quantum은 1

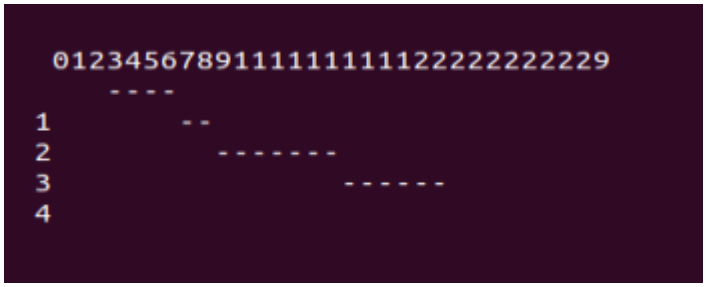


*새로운 워크로드

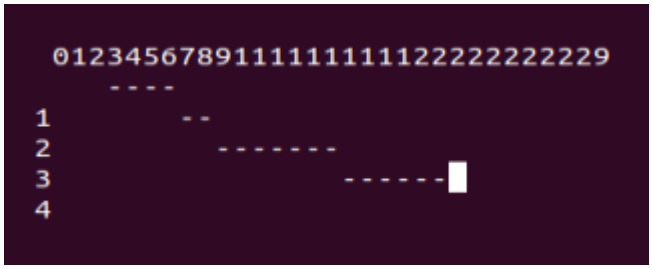
```
oslab@oslab-DKU:~/Desktop/2021_DKU_OS/lab1_sched$ ./lab1_sched
How many processes do you create? > 4
arrived_time, service_time of process 0 >> 3 4
arrived_time, service_time of process 1 >> 6 2
arrived_time, service_time of process 2 >> 8 7
arrived_time, service_time of process 3 >> 11 6
```

*

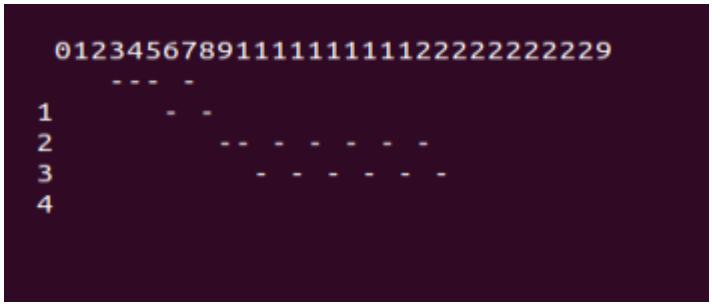
(1) FCFS



(2) SPN



(3) RR


01234567891111111111222222222 9
1
2
3
4

2. 코드설명

(0)

```
#include <stdbool.h>
#include <ncurses.h>
```

기본으로 양식에 추가되어 있는 헤더파일 이외에 bool 자료형을 사용하기 위한 〈stdbool.h〉와 콘솔창에서 커서 이동을 위한 〈ncurses.h〉 헤더파일을 추가하였다.

```
void Timeline(int length)
{
        int i, j;
        initscr();
        for(i = 0; i<30; i++){
                move(4, 5+i);
                printw("%d",i);
        }
        for(i = 1; i<=length; i++) {
                move(5+i,4);
                printw("%d",i);
        }
        refresh();
        endwin();
}
void MinSort(int* _times, int length)
{
        int i, j;
        int tmp;

        //sorting
        for(i = 0; i < length-1; i++) {
                for(j = i+1; j < length; j++) {
                        if(_times[i] >= _times[j]) {    //swap times[i] and times[j]
                                tmp = _times[j];
                                _times[j] = _times[i];
                                _times[i] = tmp;        }
                }}


}
```

1) Timeline 함수

콘솔 창에 pid 와 타임을 표시한다.

2) MinSort

시간이 담긴 배열의 주소를 받아 오름차순으로 정렬한다.

```
bool IsEmpty(List* sched)
{
        if(!(sched->head))
                return true;
        else    return false;
}
void Push(List* sched, Process* job)
{
        if(IsEmpty(sched))      //empty List
                sched->head = sched->tail = job;
        else {
                sched->tail->next = job;
                sched->tail = job;
        }
}
```

3) IsEmpty 함수

파라미터로 받은 리스트가 비어 있는지 확인한다.

4) Push 함수

파라미터로 받은 연결리스트(큐)에 매개변수 job 을 push 한다.

```
void Pop(List* sched)
{
        if(IsEmpty(sched)) {
                printf("Empty Queue\n");
                return;
        }
        sched->head = sched->head->next;
        if(IsEmpty(sched)) {
                sched->tail = NULL;
        }
}
void Delete(List* list, Process* job)
{
        //search
        Process* p = list->head;
        if(p->pid == job->pid) {        //head node
                list->head = list->head->next;
                return;
        }

        while(p)
        {
                if(p->next->pid == job->pid) {
                        p->next = job->next;
                        return;
                }
                else
                        p = p->next;
        }
}
void draw(int Y,int X, int start, int end)
{
        int width = end - start;
        int i;
        X += 5;

        initscr();

        for(i = 0; i<width; i++) {
                move(Y,X);
                printw("%c",'-');
                X++;
        }
        refresh();
        sleep(3);
        endwin();

}
```

5) Pop 함수

매개변수로 받은 연결리스트에서 head 를 pop 한다. (일반적인 큐의 pop)

6) Delete 함수

Pop 과 달리 특정 리스트 노드를 지운다.

7) draw 함수

pid 가 가지고 있는 고유의 위치(position –후술)를 높이(y 표)으로 고정하고 x 좌표를 옮겨가며 cpu time (=end-start)만큼 '–' 표시를 그린다.

```c
#ifndef _LAB1_HEADER_H
#define _LAB1_HEADER_H
#include <sys/types.h>
#include <stdbool.h>

typedef struct _Process {
        int pid;
        struct _Process* next;
        int arrived_time;
        int service_time;
        int remained_time;

        //size_t start[100];
        //size_t end[100];
        int wait_time;
        int position;   //for presenting timeline, Y coordinate


}Process;
typedef struct _List {
        Process* head;
        Process* tail;
}List;
```

8) struct _Process (aka Process)

리스트의 노드에 해당

pid 는 main 에서 1,2,3,… 처럼 생성 순으로 자동 지정되며 position (timeline 에서 y 좌표값)을 결정함

연결리스트 구현을 위해 자기참조구조체 형태로 구현 (struct _Process* next)

9) struct _List (aka List)

연결리스트 체인에 해당

(1) FCFS

```c
void FCFS(Process* jobs, int length)
{
        int i, j;
        int times[length];
        Process* _jobs = (Process *)malloc(sizeof(Process) * length);

        for(i = 0; i<length; i++)
                _jobs[i] = jobs[i];     //copy the elements

        //sorting
        for(i=0; i<length; i++)
                times[i] = _jobs[i].arrived_time;


        MinSort(times, length);

        List* _scheduled = (List *)malloc(sizeof(List));
        _scheduled->head = NULL;
        _scheduled->tail = NULL;

        //scheduling
        for(i=0; i<length; i++)
                for(j = 0; j<length; j++)
                        if(_jobs[j].arrived_time == times[i]) {
                                Push(_scheduled,&(_jobs[i]));
                        }

        //executing (drawing timeline)
        Process* _current = _scheduled->head;
        int timer = 0;
        Process* p;
        while(!IsEmpty(_scheduled))
        {
                for(timer; timer <= _current->arrived_time-1; timer++);
                draw(_current->position, timer, timer, timer+_current->service_time);

                for(p=_current->next; p; p=p->next) {
                        p->wait_time += _current->service_time;
                }

                timer += _current->service_time;
                _current->remained_time = 0;
                _current = _current->next;
                Pop(_scheduled);        //current process complete
        }
}
```

-main에서 매개변수로 넘긴 원본 jobs을 훼손하지 않기 위해 지역변수 _jobs 생성 후 복사 (_jobs는 이하 jobs라고 지칭)

-jobs의 arrived_time을 times배열에 옮겨 담아 오름차순 정렬

-정렬된 times 배열의 element값과 jobs들의 arrived_time을 비교하여 동일한 시간을 가진 process를 scheduled 연결리스트(큐)에 push => 결과적으로 arrived_time 순서대로 sort 된 연결리스트 생성

-scheduled에서 순서대로 pop하면서 draw. 이 때 timeline에서 x좌표를 결정하는 timer에 유의해야 함.

(2) SPN

```
void SPN(Process* jobs, int length)
{
        int i, j;
        int count;
        int times[length];
        Process* _jobs = (Process *)malloc(sizeof(Process) * length);

        for(i = 0; i<length; i++)
                _jobs[i] = jobs[i];     //copy the elements

        //initial sorting
        for(i = 0; i<length; i++)
                times[i] = _jobs[i].service_time;

        MinSort(times, length);

        Process* _scheduled;

        List* _temp = (List *)malloc(sizeof(List));
        _temp->head = NULL;
        _temp->tail = NULL;

        //initially, push jobs in temp list as sorted_times (guarantee that this order is the shortest
job order)
        for(i=0; i<length; i++)
                for(j = 0; j<length; j++)
                        if(_jobs[j].service_time == times[i])
                                Push(_temp, &(_jobs[j]));

        //drawing timeline
        Process* _current;
        int timer = 0;
        Process* p = _temp->head;
        while(!IsEmpty(_temp)){
                _scheduled = NULL;
                p = _temp->head;

                while(!_scheduled)      //scheduling
                {
                        if(!p) {
                                timer++;
                                p = _temp->head;
                        }
                        if(p->arrived_time <= timer) {
                                _scheduled = p;
                                Delete(_temp, p);
                        } else
                                p = p->next;

                }


                draw(_scheduled->position, timer, timer, timer+_scheduled->service_time);

                timer += _scheduled->service_time;
        }
}
```

-_jobs로 옮기는 과정은 FCFS와 동일

-times배열에 service_time을 담은 후 오름차순 정렬 => 이를 기준으로 temp 연결리스트 생성 (service time 순으로 정렬된 jobs 리스트)

-temp 리스트에서 해당 timer의 시간에서 arrived된 job을 scheduled 포인터로 가리킴 (자료형은 Process* 임에 주의, FCFS와 다름) => 이렇게 하면 temp에서 선택된 첫 번째 노드는 동일한 arrived time 우선순위를 가진 노드(프로세스)들 중 가장 service time이 short함을 보장할 수 있음

-이런 방식으로 계속 선택하면서 temp가 empty할 때까지 반복하며 draw

(3) RR

```
void RR(Process* jobs, int length, int q)
{
        int i, j;
        int _times[length];
        Process _jobs[length];
        for(i = 0; i<length; i++)
                _jobs[i] = jobs[i];

        for(i = 0; i<length; i++)
                _times[i] = _jobs[i].arrived_time;

        MinSort(_times, length);

        Process* temp = (Process *)malloc(sizeof(Process) * length);
        //jobs sorting
        for(i = 0; i<length; i++)
                for(j = 0; j<length; j++)
                        if(_jobs[j].arrived_time == _times[i])
                                temp[i] = _jobs[j];
        for(i = 0; i<length; i++)
                _jobs[i] = temp[i];

        free(temp);


        List* _scheduled = (List *)malloc(sizeof(List));
        _scheduled->head = NULL;
        _scheduled->tail = NULL;

        Push(_scheduled, &(_jobs[0]));

        int timer = _scheduled->head->arrived_time;
        int proc_index = 1;
        Process* current = _scheduled->head;

        while(!IsEmpty(_scheduled))
        {
                draw(current->position, timer, timer, timer+q);
                current->remained_time -= q;
                timer += q;


                for(i=proc_index; i < length; i++)
                        if(_jobs[i].arrived_time <= timer) {
                                Push(_scheduled, &(_jobs[i]));
                                proc_index++;
                        }
                if(current->remained_time > 0)
                        Push(_scheduled, current);

                Pop(_scheduled);

                current = _scheduled->head;

        }
}
```

-times 배열에 arrived time을 넣은 후 오름차순 정렬

-이를 이용해 jobs 또한 arrived time기준으로 오름차순 정렬

-scheduled->head는 jobs[0] (가장 빨리 arrived된 job)으로 두고 timer도 jobs.arrived_time으로 초기화한다.

-proc_index는 각 jobs들이 최초로 스케쥴링 될 시 +1 된다.

-current를 head로 초기화 하고 while문으로 들어간다.

-초기화 한 process를 time quantum (이하 q)만큼 draw한다. 여기서 가장 중요한 것은 remained time과 timer를 잘 계산해야한다는 것이다.

-한 번 실행을 마치고 ready상태로 돌아온 job은 pop되었다가 remained time이 남아있을 경우 다시 큐에 push 된다.

스케쥴링 되지 않은 나머지 job들도 timer가 증가함에 따라 arrived time에 맞춰 push와 pop을 반복한다.

3. Discussion

어려운 과제였지만 해 낼 수 있을 거라고 생각했는데, 연구실에서의 첫 학기를 맞이하면서 논문 발표일자와 마감일이 겹쳐 정말 힘들었다. 아쉬움이 많이 남는다. 우선 이런 아쉬움은 뒤로하고 많이 부족하지만 이번 과제를 하면서 알았던 점들을 풀어보려한다.

우선 C와 C++의 차이에 대해서 명확하게 알 수 있었다. C는 OOP가 아니기 때문에 c++이나 java, python 등에서 제공하는 class 기능을 사용할 수 없다. 하지만 c++에서 class를 처음 배울 때 struct와 무엇이 다른지 의문을 가졌었다. 사실 c언어를 처음 배울 때와 시스템프로그래밍, 운영체제 시간이 아니면 c를 사용할 일이 거의 없기 때문에 struct와 class는 '얼추 비슷한 것~' 정도로 기억하고 있었다. 그런데 이번에 process나 list struct를 만들어보면서 그 차이점을 확실하게 알았다. 수 많은 오류들이 알려준 결과이다... struct는 class와 다르게 멤버 함수를 선언할 수 없다. 만약 포함관계를 설명하고 싶다면 함수 포인터를 내부에 선언한 후 main문에서 초기화 해 주어야 한다. 또한 구조체 내에서 초기화를 하면 오류가 발생한다. class에서는 오히려 constructor를 이용해서 모든 변수들을 초기화 한 후 객체를 생성하는 것이 기본인데 struct는 많이 다른 특징을 보였다.

다음으로 어려웠던 것은 콘솔창에서의 커서 제어였다. 처음에는 이런 기능이 있는지 조차 생각하지 못했다. 그래서 각 Process 별로 start, end, wait time를 배열로 저장해서 스케쥴링이 모두 끝난 후 같은 인덱스끼리 계산해서 출력하려 했다. 하지만 이는 매우 비효율적인 방법이라는 생각이 계속 들었다. 그러다 문득 c언어를 처음 배울 때 'c를 잘하고 싶다면 테트리스 게임을 직접 만들어보라.'는 선배의 조언이 생각났다. C로 게임을 만들 수 있다면 당연히 커서 이동도 가능할 것이라고 생각하여 헤더파일을 찾았다. 처음에는 Windows.h라는 헤더파일을 발견했다. 커서 이동방법에 대해서 1주일 동안 고민하다 발견한 라이브러리라서 매우 기뻤지만 호기 넘치게 적용한 것과는 정 반대의 결과가 나왔다. 이 헤더파일은 windows 운영체제에서만 사용할 수 있다는 것이다. 그래서 리눅스에서 적용 가능한 커서 이동 라이브러리가 분명 있을 것이라고 생각하고 열심히 찾은 결과 ncurses.h라는 라이브러리를 찾았다.

라이브러리를 찾은 기쁨은 잠시였다. ncurses.h는 -lncurses 옵션을 붙여 컴파일 해야 하는데 Makefile을 제대로 다뤄본적이 없기에 혹여 잘못 건들였다가 백업도 못한 상태에서 모든 것을 초기화 해야하는 것이 아닌지 걱정했다. 또한 엎친데 덮친격으로 ncurses 헤더파일을 다운로드 하기 위해 명령어를 입력했는데, vm 우분투에서 생성된 ip가 갑자기 표시되지 않으면서 네트워크에 오류가 발생했다. 백업도 못한 상태였기에 매우 조심스럽게 네트워크를 수정했다. 결국은 Net으로 연결된 네트워크를 삭제하고 어댑터에서 바로 받아서 쓰는 것으로 설정을 변경하여 해결했다. 네트워크는 잘 모르는 분야였는데 며칠동안 찾아보면서 새로운 것을 공부할 수 있는 계기가 되었다.

네트워크를 돌려놓고 Makefile을 수정했다. Makefile 작성 문법을 간단하게 공부했다. 어셈블리어의 형식을 따르고 있다는 느낌을 받았다. 어느 부분에 옵션을 붙여야할 지 몰라서 계속 오류를 내며 시도한 결과 새로운 FLAGS를 추가해서 옵션으로 붙여넣어 컴파일 하는데 성공했다.

마지막으로 C언어의 네이밍 규칙에 대해서도 공부할 수 있는 계기가 되었다. 아직 완전히 숙지하지 못해서 중구난방으로 사용하고 있지만, 학점이 부족해서 소프트웨어공학 수업을 들을 수 없기에 이런 네이밍 규칙은 잘 알지 못했다. 윈도우에서 헤더파일을 살펴보면 __HEADER__ 의 형식으로 써진 것들도 많고 변수 이름 앞에 _ 를 붙여쓰거나 어떤 경우는 대문자로, 어떤 경우는 소문자로만, 어떤 경우에는 단어를 나눠서 적는 규칙이 있고 이 규칙들이 매우 다양하다는 것 또한 알았다. 앞으로 다음 과제들을 하면서 이런 규칙들을 더 연습해 볼 것이다.

여러 일정이 겹쳐 전부 끝내지도 못하고 결국 지각 제출까지 해 버려서 많이 아쉬운 운영체제 첫 과제였다. 다음 Lab2과제는 더 시간을 많이 투자하고 고민해서 더 좋은 결과를 낼 것이라는 다짐을 한다.