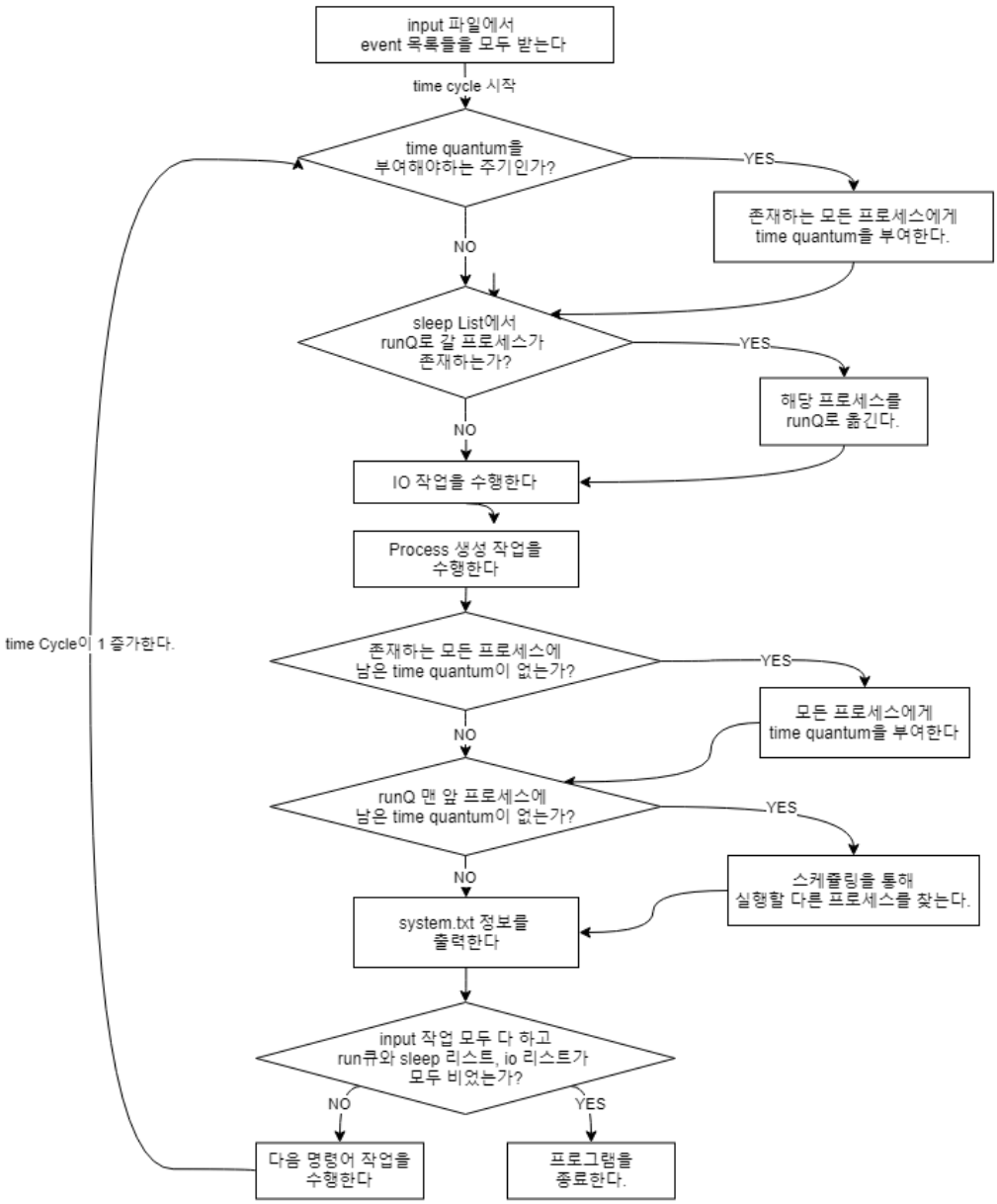
# OS Assignment 3


### 2016147571 김조현


## 프로그램 동작과정

이번 과제는 RoundRobin 스케쥴러와 Buddy 시스템을 사용하는 시뮬레이터를 구현하는 것이었다. 전체 동작과정을 나타낸 대략적인 순서도는 아래와 같다.



# 함수 및 class 설명

메모리를 구현할 때 pate table을 pt라는 클래스를 만들어서 pt 클래스의 변수를 생성하여 그 변수가 페이지 테이블의 역할을 하도록 하였다. pt 클래스의 변수를 프로세스의 갯수만큼 만들었고, 그 프로세스의 갯수만큼의 변수 전체가 page table의 역할을 하도록 하였다. 또한 vm 클래스를 만들어서 virtual memory 를 구현할때 각 frame을 vm 클래스로 채워넣어 사용하였다.

pm 클래스는 physical memory 부분을 구현할때 사용하였고, buddy 클래스는 buddy 시스템을 구현할 때 buddy 클래스를 가진 변수 하나를 만들어서 사용하였다.

또한 task 클래스를 만들어서 각 프로세스별로 남은 해야할 작업들을 task 클래스의 변수로 저장해둘 수 있었다. event 클래스는 처음에 input 파일에서 각 줄을 event 클래스에 저장하였고, 이중에 프로세스 생성을 통해 생성된 event 클래스의 변수는 후에 프로세스의 역할을 하게 되었다.

## buddy_rel 함수

buddy_rel 함수는, 이 함수를 호출한 프로세스의 메모리 정보를 physical memory와 virtual memory에서 제거해주는 함수이다. 수도코드는 아래와 같다

> releaseIndex = -1
>
> releaseIndex = physical memory에서 지워야 하는 부분의 index를 저장
>
> if(releaseIndex >= 0) : 지울 부분이 존재한다면
>
> > 지울 physical memory의 부분의 aid와 pid부분을 모두 초기화시키고 empty 변수를 true로 만들어준다.
> >
> > if(physical memory에 들어있는 프로세스 개수가 1보다 크다면):
> >
> > > if(가장 왼쪽 것이 지워졌을때)
> > >
> > > > 바로 오른쪽 pm을 봤을때 크기가 같고 empty 하면 합친다.
> > >
> > > if(가장 오른쪽 것이 지워졌을때)
> > >
> > > > 바로 왼쪽 pm을 봤을때 크기가 같고 empty 하면 합친다
> > >
> > > else (중간 어딘가가 지워졌을때)
> > >
> > > > 양쪽을 봤을때 크기가 같고 empty하면서 binary Index의 뒤에서 두번째 값이 같은 경우에 합친다.
>
> virtual memory에서 buddy_rel을 요청한 프로세스 부분의 valid값을 0으로 설정한다
>
> > 만약 LRU에 의해 physical memory에 공간이 부족해서 빠진 경우가 아니라면 aId(allocation ID)도 -1로 설정해준다.

위에서 binary Index는 buddy system을 구현하기 위해 사용하였다. buddy system을 구현할때, 공간이 나눠지는 것을 binary tree 와 같이 이해할 수 있는데, 이번 과제에서 구현할 때 physical memory에 들어있는 프로세스들을 pm이라는 클래스의 변수로 만들어서 넣어두었기 때문에, 그안의 string 변수인 bI(binary Index)를 통해 이진 트리를 표현하였다.

예를들어 physical memory의 크기가 128이라고 한다면, 그 크기 128짜리 블록의 bI는 0이 된다. 거기에 64만큼의 자리가 할당된다면, 그 64만큼의 자리의 bI는 00, 그리고 그 오른쪽의 크기 64만큼의 자리의 bI는 01이 된다. bI가 01인 블록이 반으로 쪼개지면 그 쪼개진 블록중 왼쪽것의 bI는 010, 오른쪽의 bI는 011로 설정되게 해두었다. 이렇게 함으로써, 내 옆의 블록이 크기가 같을때 합쳐질 수 있는 경우인지를 bI 값의 오른쪽에서 두번째 숫자가 같은지

를 확인함으로서 판별할 수 있게 되었다.

buddy_rel 함수에 byLRU변수가 들어간 이유는 다음과 같다. buddy_rel 함수를 호출하게 되는 경우는 첫번째로 프로세스가 종료되어 메모리를 모두 release하려 하는 경우나, 두번째로 다른 프로세스가 들어올 자리를 마련하기 위해 release 하려는 경우 이렇게 두가지로 볼 수 있다. 두번째 경우에 byLRU 변수는 true가 되게 되고, 이때는 virtual memory에서까지 지워져서는 안되기 때문에 byLRU 변수를 사용하였다.

### buddy_all 함수

프로세스를 메모리에 할당할 때 사용한 buddy_all 함수의 수도코드는 다음과 같다

> placeToGo = false
>
> physical memory에서 현재 프로세스를 할당 할 수 있는 블록이 발견되면 placeToGo 는 True가 된다.
>
> if(placeToFo == false) : 할당할 수 있는 공간이 없는 경우에는
>
> > buddy_rel 함수를 사용하여 LRU 리스트에서 가장 앞에 있는 프로세스를 할당 해제해 주었다.
>
> 갈 수 있는 자리가 무조건 생겼으니 toGoI 변수에 넣고자 하는 사이즈보다 크지만, 남은 사이즈 중에서 가장 작은 사이즈의 블록을 선택해서 그 인덱스를넣어준다.
>
> toGoI 인덱스의 블록을 현재 함수를 호출한 프로세스의 크기보다 작지 않은 한 최대한 크기를 2로 나누며 줄여준다. 동시에 늘어난 블록들도 초기화 시켜준다.
>
> toGoI 인덱스에 프로세스를 할당해준다
>
> if(byAccess == false) : Access 명령어로 이 함수를 호출한것이 아니라면,
>
> > 이 프로세스의 virtual memory 맨뒤에 현재 할당하고자 하는 사이즈만큼 aid를 할당하고 valid값을 1로 만들어준다
>
> else : Access 명령어를 통해 이 함수를 호출한 것이라면,
>
> > 이 프로세스의 virtual memory에서 일치하는 aid 부분을 찾아서 valid 값을 1로 만들어준다.

buddy_all 함수에서 byAccess라는 변수를 받은 이유는 다음과 같다. buddy_all 함수를 호출하게 되는 경우에는 두가지가 있는데, 이것은 각 프로세스가 실행해야할 명령어 들 중, allocate 명령을 받았을때와 access 명령을 받았을 때이다. access 명령을 받아서 buddy_all 함수를 호출한 경우에는 byAccess 변수를 true로 바꾸어 넣어주었다. 이 차이점을 둔 이유는, allocate 명령을 받았을 때에는 해당 프로세스에 allocation id(aId)를 1씩 더해가면서 virtual memory에 계속 써주면 되지만, access 명령을 받았을 때에는 해당 프로세스의 aId를 찾아서 그부분의 valid값을 1로 만들어주는 조금 다른 작업을 수행해야하기 때문이었다.

## main 함수

### input 파일 받기

인풋 파일에서 전체 정보를 각 변수를 만들어서 저장해 두었고, 두번째 줄부터 한줄씩 eventQ라는 event 클래스의 변수를 담은 queue에 저장해두었다. 후에 여기서 하나씩 빼면서 각 이벤트들의 동작을 수행하였다.

## cycle distribution

일정 시간마다 존재하는 모든 프로세스에게 timequantum을 부여해야 했다. 그래서 처음에는 현재 시간을 타임퀀텀을 주는 주기로 나누어서 나머지가 0일때마다 부여하도록 코드를 짜두었다. 그러나 후에 그 어떤 프로세스도 남은 타임 퀀텀이 없는 경우가 생기고, 그때 타임퀀텀을 주는 주기가 재시작 되어야 한다는 것을 알게되었고,

```
if((nowCycle-modifyCycle) % bonusqt == 0){}
```

그 이후로 타임퀀텀을 부여할지 말지 선택하는 if문을 위와같이 변경하였다.

```
modifyCycle = nowCycle % bonusqt;
```

이때 modifyCycle는 처음에 0으로 설정해두고 모든 프로세스가 남은 시간이 없어서 타임퀀텀을 부여하고 주기가 재시작 되었을때의 nowCycle값을 타임퀀텀을 주는 주기로 나누었을때의 나머지를 계속 더했다. 이렇게 설정해 두었을때, 예를들어 주기가 25라면 0초, 25초, 50초 이렇게 타임퀀텀을 부여하다가, 53초에 모든 프로세스가 남은 시간이 없었다면, 그 이후에는 53초에 부여하고, 78초에 부여하고, 103초에 부여하게 되었다.

## check sleepL

sleepL 리스트에는 현재 sleep 중인 프로세스들의 목록이 들어가 있었다. 이 목록이 비어있지 않는 경우에는, 목록 안에 있는 프로세스들이 다시 run queue로 돌아가야 할 때인지를 판단하고 그런 경우에는 돌려보냈다. 판단하는 방법은 아래와 같다.

```
tempE.wakeUpTime = nowCycle+nowT.argument;//set wakeUpTime;
```

위와 같이 tempE(어떤 프로세스)가 sleep 하라는 task를 받았을때, 그 프로세스의 wakeUpTime 값을 현재 cycle 값에 sleep List에 머무를 시간을 더한 값으로 변경해 두었다. 이렇게 해두면 후에 sleepL안의 모든 프로세스의 wakeUpTime을 검사해서, 현재 cycle값과 같으면 빼는 방식으로 작동하게 할 수 있었다.

## input - IO & input - Process

input-IO의 경우에는 io List를 보고, 거기에 명령 받은 프로세스가 있는 경우에는 io List에 있던 그 프로세스를 runQueue로 옮겨주는 작업을 했다.

input-Process의 경우에는 프로세스를 새로 생성해 주었고, 그 생성된 프로세스를 runQueue에 넣어주었다.

## choose process to run

이때에 모든 프로세스가 남은 시간이 없는지를 확인하였다. run Queue, sleep list, io list에 존재하는 모든 프로세스가 남은 시간이 없다면, 모든 프로세스에게 타임 퀀텀을 부여해 주었다.

run Queue에 있는 첫번째 프로세스는 현재 실행되고 있는 프로세스이고, 두번째 부터가 runQueue에서 기다리고 있는 프로세스라고 할 수 있다. 이때 runQueue에 있는 첫번째 프로세스가 남은 시간이 없다면, 남은 시간이 있는 다른 프로세스가 나올 때까지, runQueue 맨 앞의 프로세스를 빼서 맨뒤로 넣는 작업을 반복했다. 이 과정은 현재 돌릴 수 있는 프로세스를 찾기위해 scheduling하는 작업이라고 할 수 있다.

또한, 현재 실행중이었던 프로세스가 Round Robin scheduler가 사용하는 timequantum을 모두 사용한 경우에는 run Queue의 맨 뒤로 가게 해두었다.

## print

현재 실행되고 있는 프로세스에 대한 정보를 system.txt에 출력하는 코드가 있다.

## process instruction

각 프로세스는 해야하는 명령어들이 task로 존재한다. 0부터 5까지의 option code로 명령어의 종류가 주어진다

#### 0 : memory allocation

현재 task의 opcode가 0인 경우에는 프로세스를 메모리에 할당해주어야 한다. 이때 buddy_all 함수를 사용하여 physical memory와 virtual memory에 프로세스를 할당해 주게된다. 할당할 때마다 aId는 0부터 1씩 증가하게 된다. 그리고 memory allocation작업을 해주었으니, 그 프로세스와 aid의 값은 LRU 리스트의 맨 뒤에 새로 생기거나 맨 뒤로 옮겨지게 된다.

#### 1. memory access

현재 task의 opcode가 1인 경우에는 해당하는 프로세스의 aId 값을 보고 메모리에 접근해야 한다. 이때 physical memory에 접근하고자 하는 정보가 없으면 접근할 수가 없기 때문에, virtual memory에서의 정보를 page table을 통해 가져와서 physical memory에 다시 할당해주어야 한다.

그래서 만약 access하고자 하는 메모리가 physical memory에 있는 경우에는 현재 프로세스의 aid값을 LRU 리스트의 맨 뒤로 옮겨주었고, physical memory에 access하고자 하는 메모리가 없는 경우에는 buddy_all 함수에 byAccess변수를 true로 설정하여 호출해서 physical memory에 메모리를 할당해주었다. 이때도 마찬가지로 프로세스의 aid값은 LRU리스트의 맨 뒤로 옮겨주었다.

#### 2. memory release

현재 task의 opcode가 2인 경우에는 해당하는 프로세스의 aid값을 보고 그 aid값과 pid값을 가지는 메모리를 release 해주어야 한다. 그래서 buddy_rel 함수를 호출하여 physical memory와 virtual memory에 있던 할당되어 있던 값을 해제해 주었다. 그리고 LRU 리스트에서도 이 값을 없애주었다.

#### 3. non-memory instruction

현재 task의 opcode가 3인 경우에는 메모리관련 작업이 일어나지 않았다. 그렇기 때문에 그냥 1 cycle을 소모하였다

#### 4. go to sleepL

현재 task의 opcode가 4인 경우에는 프로세스를 sleep List로 보내야 했다. 이때, 이 명령이 마지막 task였다면 sleepList로 보내지 않고 그냥 프로세스를 종료시켰으며, 마지막 task가 아닌 경우에는 runQueue에서 빼서 sleep List안에 넣어주었다.

**5. go to ioL**

현재 task의 opcode가 5인 경우에는 프로세스를 io List로 보내야 했다. 4번 명령과 마찬가지로, 이때도 이 명령이 마지막 task, 였다면 io List로 보내지 않고 그냥 프로세스를 종료시켰으며, 마지막 명령이 아닌 경우에만 runQueue에서 빼서 sleep List 안에 넣어주었다.

**[pid].txt 출력**

위의 명령어 수행이 끝나고 나면 각 프로세스의 pid를 이름으로 가지고 있는 txt 파일에 수행한 일을 적어주었다.

**process 종료?**

위의 명령어 수행을 모두 하면서, 수행하고 있는 task가 그 프로세스의 마지막 task인 경우에 laftTask 변수의 값을 true로 만들도록 설정했다. 그리고 명령에 대한 작업이 모두 끝난 후, laftTask의 값이 true라면 이 프로세스를 run Queue의 맨앞에서 빼냈고, buddy_rel 함수를 호출하여 physical memory와 virtual memory 모두에서 지워냈으며, LRU List에서도 찾아서 지워주었다.

# 개발환경

```
zimkjh@ubuntu:~/Desktop/hw3/submit$ uname -a
Linux ubuntu 4.14.24-2016147571 #1 SMP Thu Mar 8 0
0:31:04 PST 2018 x86_64 x86_64 x86_64 GNU/Linux
```

linux 터미널에 uname -a를 입력했을때 출력 화면은 위와 같다. 또한 컴퓨터의 cpu와 ram은 아래와 같다.

| | |
|---|---|
| 프로세서: | Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz |
| 설치된 메모리(RAM): | 8.00GB |

# 문제점 및 해결방안

### 길이를 모르는 어레이의 생성

pm이라는 클래스를 만들어서 physical memory에 들어있는 블록들의 정보를 담고자 하였다. pm이라는 클래스는 갯수가 1개에서 프로세스의 갯수만큼 늘어날 수 있었기 때문에, 그 크기를 정적으로 할당하는 것은 적절하지 않다고 생각되어 동적할당을 하고자 하였다. 그래서 input 파일을 읽고나서 알게되는 프로세스의 갯수만큼을 배열의 크기로 할당하고 싶었으나, main함수 이전에 쓰여있는 buddy_rel 함수에 이 배열이 사용되었기 때문에 그러기 어려

웠다.

**해결방법**

```
class buddy{
    public:
        buddy(int temp){
            pmL = new pm[temp];
            //기타 코드들
        }
};
```

위와 같이 buddy 클래스에 생성자를 만들어두어 main 함수에서 생성자를 통해 buddy 클래스 변수 내의 pmL이라는 리스트의 개수를 설정해 줄 수 있었다.

코딩을 마치고 나니 이 문제에 대한 해결방법을 함수의 선언부와 구현부를 나누어서 main 함수 뒤에 buddy_all 함수의 구현부와 buddy_all 함수의 구현부를 두면 더 간단히 가능하지 않았을까 하는 생각도 들었다.

# 참고문헌

c++ list iterator 사용하기

http://printf.egloos.com/v/1901255