OS Assignment 3

2014121136 구한모

1. 작성한 프로그램의 동작과정과 구현 방법

Class process : 생성된 프로세스의 정보를 담는 객체

```
        int remainQuantum; // 남아있는 timequantum(cpu를 한번에 돌릴 수 있는 cycle)을 표현
        int remainFeed;// 남아있는 cpu cycle을 저장
        int pid; // process 의 pid를 저장
        string name; // process의 name을 저장
        int currState = 0; // 현재 process의 상태. 0이면 rrQueue에, 1이면 sleep, 2면 IOwait,
3이면 수행중인 process
        int sleepQuantum = 0; // process의 남아있는 sleep cycle
        int vmemSize; // process의 가상 메모리 크기
        int *instructions; // process가 수행해야하는 op들을  담아 놓은 배열
        int *instructionPara; // process가 수행해야 하는 op와 index를 같게 하여 인수를 담아
놓은 배열
        int opIndex = 0; //process가 cpu에서 수행될 때 시행해야하는 operation의 index
        int opNum; // process가 총 수행해야하는 op들의 개수
        int* pages;// process의 가상메모리(page). allocation ID를 각 page에 담아놓은 배열.
        int* pageTable; // 각 page들이 어떤 frame의 index에 해당되는지를 담아놓은 배열
        int* pageValid; // 각 page들의 validbit(1이면 valid, 0이면 not valid)들을 담아놓은
배열
        int allocationCount = 0; // 프로세스에 배당되는 가상메모리의 allocation 순서를
담아놓음.
```

Class tree : buddy system을 구현하기 위한 tree 자료구조.

각 노드들은 각각의 block의 의미를 띄며, leaf node들만이 실제 frame을 갖거나, 사용가능한 (available) block들을 의미한다.

```
void merge()
```

```
void mergeNode(node* n)
```

위 두 함수는, 물리메모리 내의 buddy block들이 비었을 때, 혹은 사용 가능해지면 합칠 수 있는 buddy block을 합치는 함수이다. Preorder 방식으로 재귀적으로 탐색하여 합칠 수 있는 조건의 node(block)들을 buddy system 내에서 전부 합치면서 올라온다.

```
node* findPlace(int pageNum)
```

```
node* find(node* n, int pageNum)
```

위 두 함수는, page개수가 들어오면 해당 page를 저장할 수 있는 최소의 2^n 크기의 block을 찾아주는 함수이다. 만약, 해당 page를 저장할 수 있는 2^n의 크기의 block이 최소사이즈가 아닌 경우, block이 page들을 저장할 수 있으면서 동시에 최소 사이즈를 만족하도록 쪼개고, 가장 왼쪽 블록에 저장할 수 있도록, 블록의 주소를 반환한다.

```
node* pidAlloc(int pid, int alloc)
```

```
node* findPidAlloc(node* n, int pid, int alloc)
```

위 두 함수는 특정 page들의 pid와 alloc을 받아 해당 page들에 연결된 frame들의 block의 주소를 반환하는 함수들이다. 실제 OS에서는 page table의 역할이지만, 해당 과제에서는 block들의 separator를 구현해야 하기에 tree로 물리메모리를 구현하였기에 물리메모리인 tree내의 함수를 이용하여 frame의 위치를 반환받았다.

```
void printSystemtxt(ofstream* system)
```

```
void printDFS(node* n, ofstream* system)
```

system.txt 에서 물리메모리 부분의 출력을 위해 만들어진 함수들이다.

Class node : 물리메모리인 tree내의 블락을 의미하는 node클래스들이다.

```
void split()
```

node의 block이 page들을 담기에 너무 큰 경우(최소size가 아닌 경우) 그 블록을 반으로 쪼개는 함수이다.

```
bool ifmerge()
```

buddy들이 서로 합쳐도 되는지(buddy가 유효한 방향으로 붙어있으며, 둘다 비어 있고 쪼개져 있지 않은 지)를 반환하는 함수이다.

다음은 멤버 함수가 아닌 일반 함수들이다.

```
void feedProcess(list<process>);
```
모든 process에 cpu cycle을 feed size만큼 주는 함수. Cycle 배분 주기이거나 round robin에 있는 프로세스들이 남아있는 time quantum이 없을 때 쓰임.
```
process* processMake(int, string);
```
새로운 프로세스를 만드는 함수. 새로 프로세스를 만든 뒤, cycle을 배분하고 round robin queue 뒤에 넣어준다.
```
void allocateMemory(process*, int);
```
process의 memory allocation 동작 수행 시 수행되는 함수. 프로세스와 필요한 page개수를 넘겨 받고 필요한만큼의 frame수를 확보하고 주소를 넘겨 받아 process에 할당하여준다. 또한 process의 page table에 해당 주소와 valid bit, page의 내용(allocation id)을 추가한다.
```
void accessMemory(process*, int);
```
process의 memory access 동작 수행 시 수행되는 함수. 프로세스와 해당 프로세스가 접근하고자 하는 allocation id를 넘겨받으면 해당 page가 valid인지 invalid인지를 판단하고, valid한 경우 access 에 추가하고 LRU목록을 업데이트하며, invalid하다면 LRU 함수를 통해 least recently used인 프레임들을 물리메모리에서 할당 해제하고 해당page들의 valid bit를 0으로 바꿔버린 후에 다시 access하려는 메모리를 가장 적절한 frame과 연결하여 물리메모리에 다시 채워 넣는다.(원래는 디스크에서 불러오는 것이 OS의 방식이지만, 가상시스템 한계 상 다시 allocation ID를 page들에 채워 넣는 것으로 한다.) 그리고 다시 LRU목록을 업데이트 한다.
```
void releaseMemory(process*, int);
```
process의 memory release 동작 수행시 수행되는 함수. 프로세스와 release하려는 페이지의 allocation ID를 넘겨받으면, 해당 page들이 valid하지 않으면 단순히 프로세스 내에서 page 내용을 없애버림으로써 가상 메모리에서만 지워버리며, 만약 page들이 valid하다면 page와 연결된
```
bool quantumCheck(process*);
```
process가 수행된 뒤 quantum이 남았는지를 check하는 함수. 만약 feed가 다 떨어졌거나 time quantum을 모두 소진했을 경우 true를 반환하고 이를 받으면 수행중인 프로세스가 IO Wait이나 Sleep에 걸리지 않았더라면 round robin Queue에 맨 뒤로 돌아가고 수행중인 프로세스는 다음

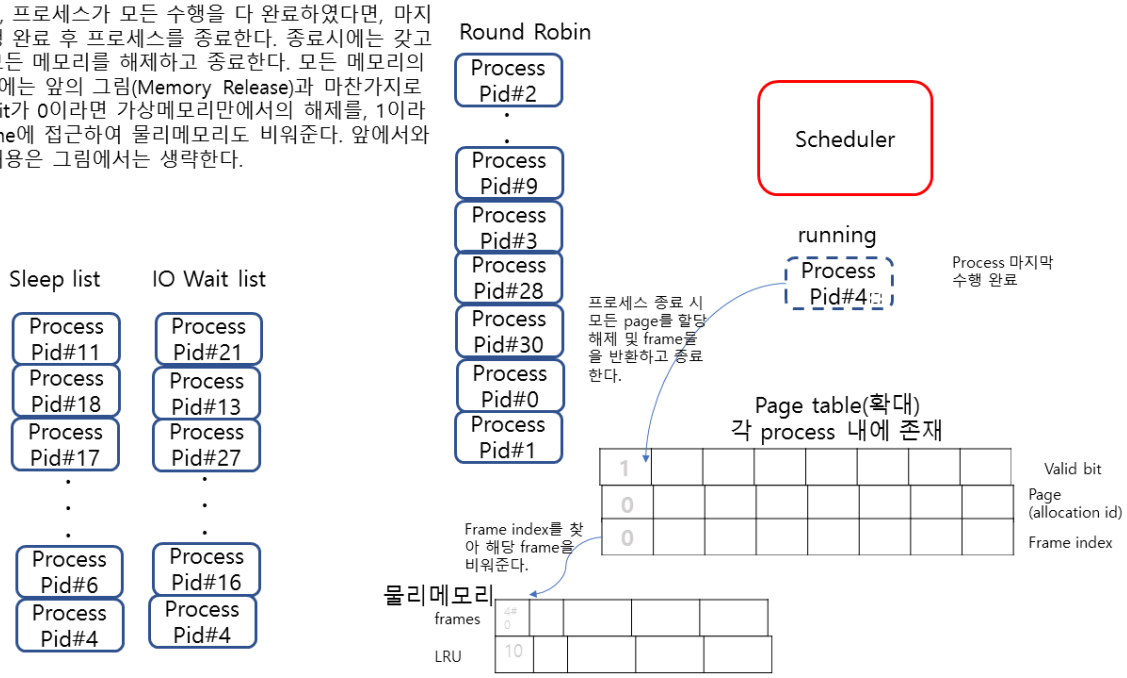
사이클에서 다시 구하게 된다.

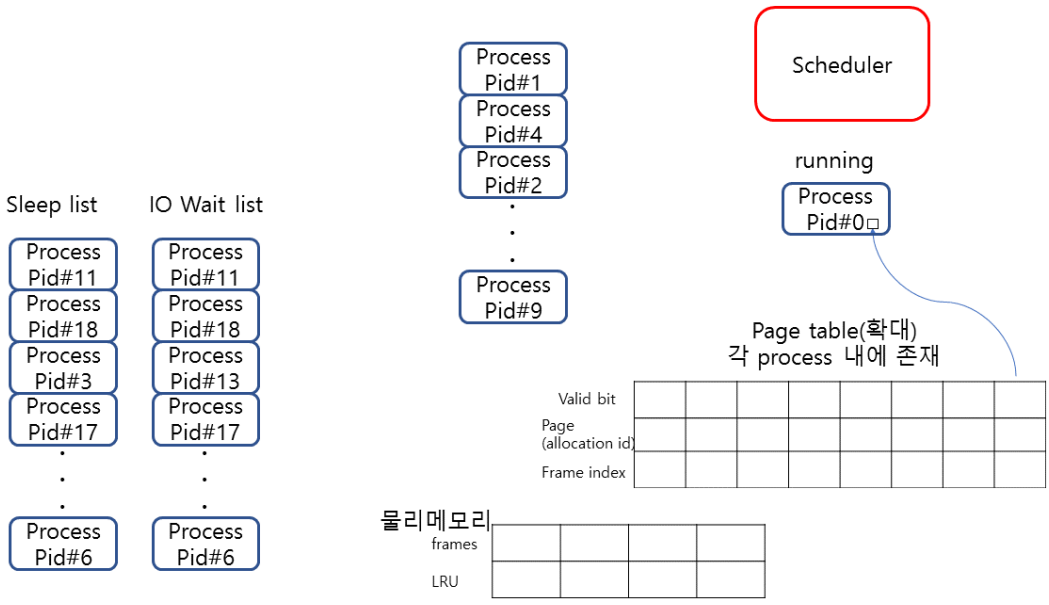
```
void LRU();
```

LRU 알고리즘에 의해서 frame에서 page를 내쫓아야 할 경우, 가장 예전에 쓰인 frame을 내쫓는 알고리즘. Frame은 초기값으로 되돌려주고, 내쫓김 당한 page의 valid bit는 해당 프로세스에 접근하여 0으로 바꾸어준다.

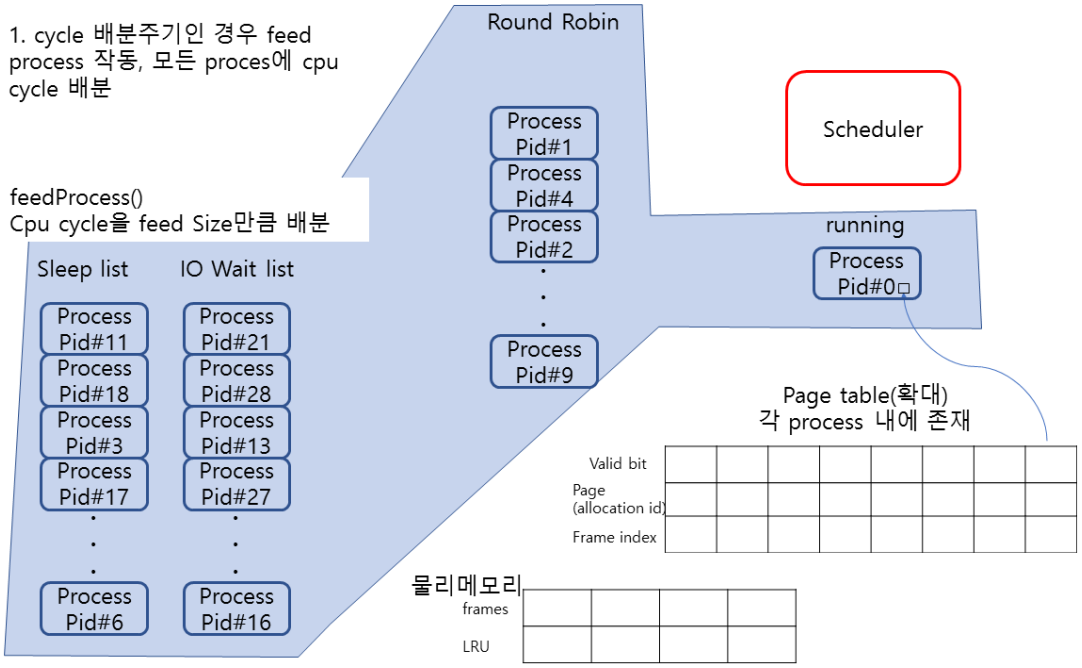
2. 작성한 프로그램의 동작 및 순서도

6. 만약, 프로세스가 모든 수행을 다 완료하였다면, 마지막 수행 완료 후 프로세스를 종료한다. 종료시에는 갖고 있던 모든 메모리를 해제하고 종료한다. 모든 메모리의 해제시에는 앞의 그림(Memory Release)과 마찬가지로 valid bit가 0이라면 가상메모리만에서의 해제를, 1이라면 frame에 접근하여 물리메모리도 비워준다. 앞에서와 같은 내용은 그림에서는 생략한다.

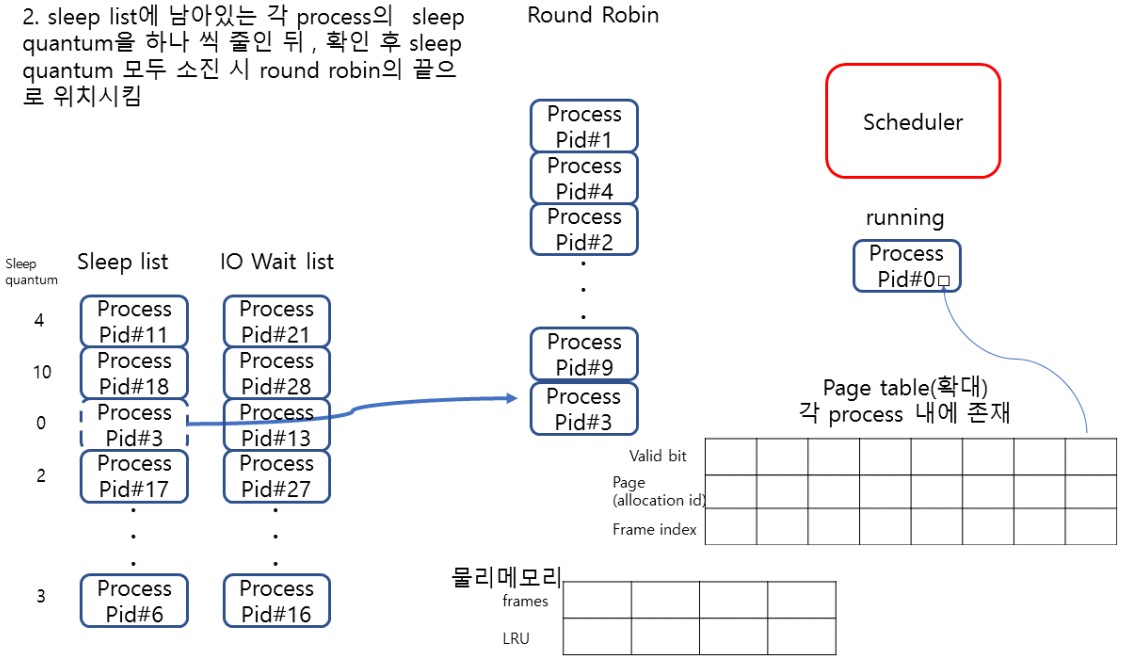




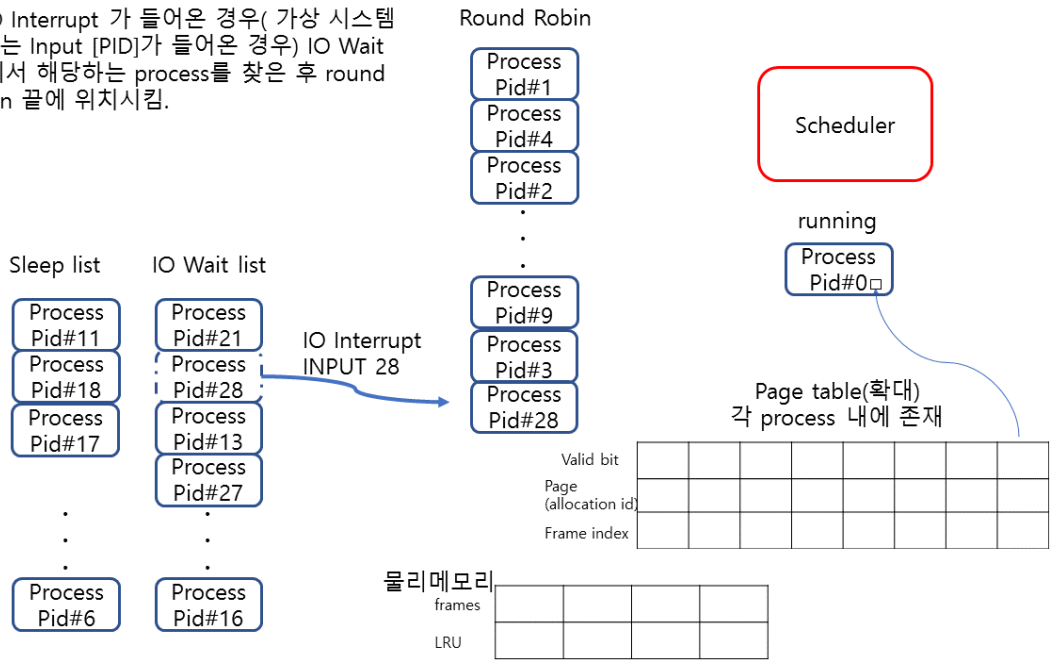
1. cycle 배분주기인 경우 feed process 작동, 모든 proces에 cpu cycle 배분



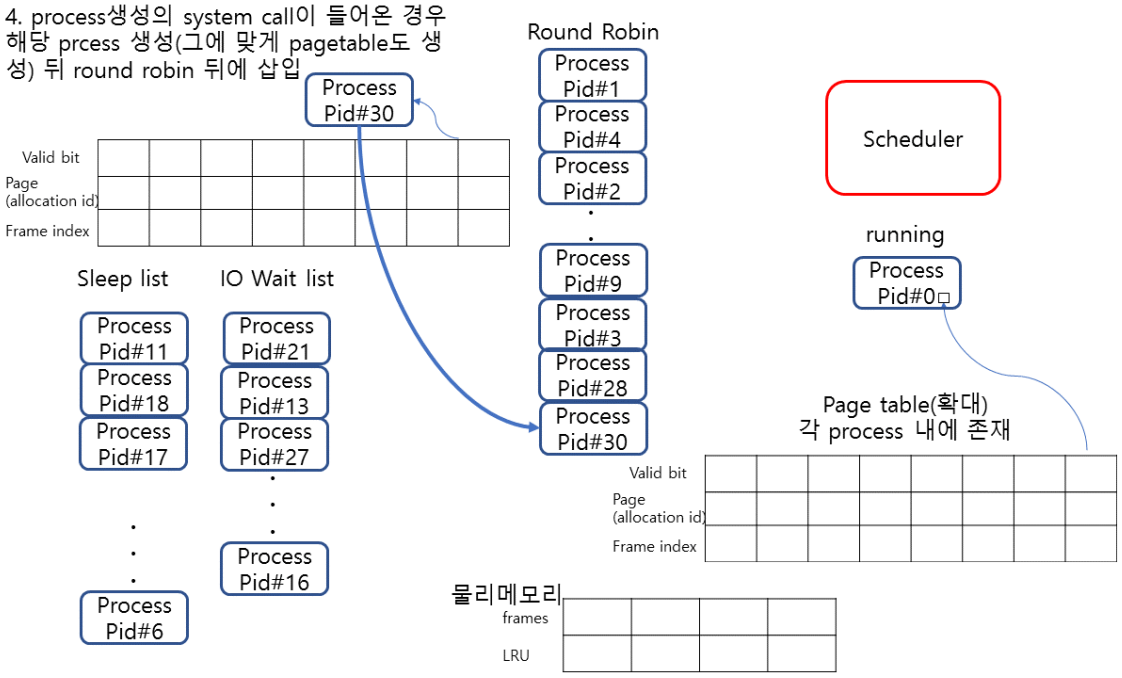
2. sleep list에 남아있는 각 process의 sleep quantum을 하나 씩 줄인 뒤 , 확인 후 sleep quantum 모두 소진 시 round robin의 끝으로 위치시킴



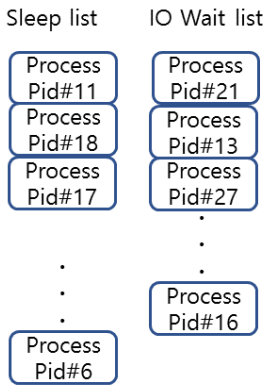
3. IO Interrupt 가 들어온 경우(가상 시스템에서는 Input [PID]가 들어온 경우) IO Wait list에서 해당하는 process를 찾은 후 round Robin 끝에 위치시킴.

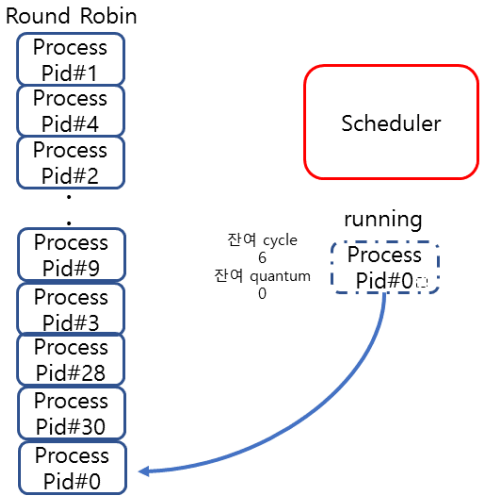


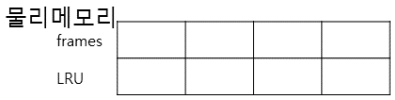
4. process생성의 system call이 들어온 경우 해당 prcess 생성(그에 맞게 pagetable도 생성) 뒤 round robin 뒤에 삽입



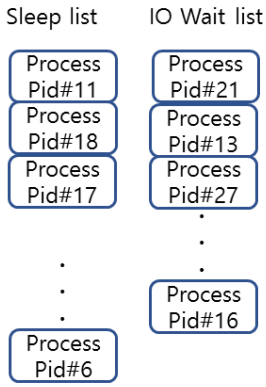
5. 스케줄러가 현재 running proces의 time quantum 및 잔여 cycle 확인 하여 둘 중 하나라도 모자랄 경우 round Robin 뒤에 위치시킴

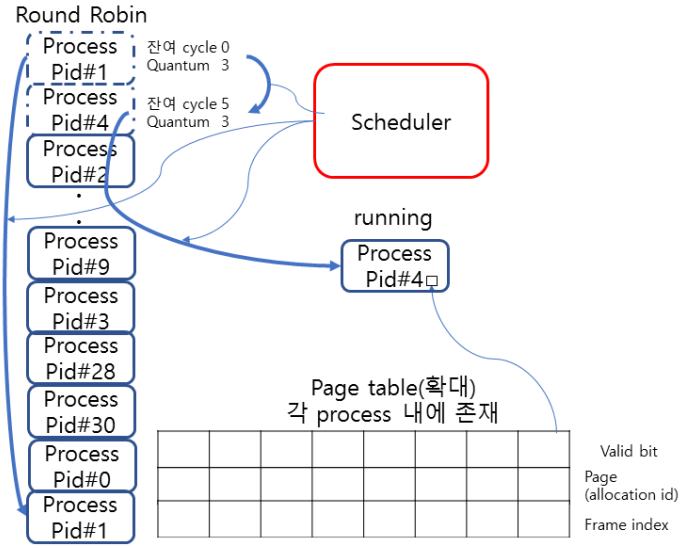


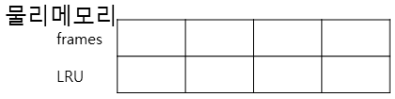




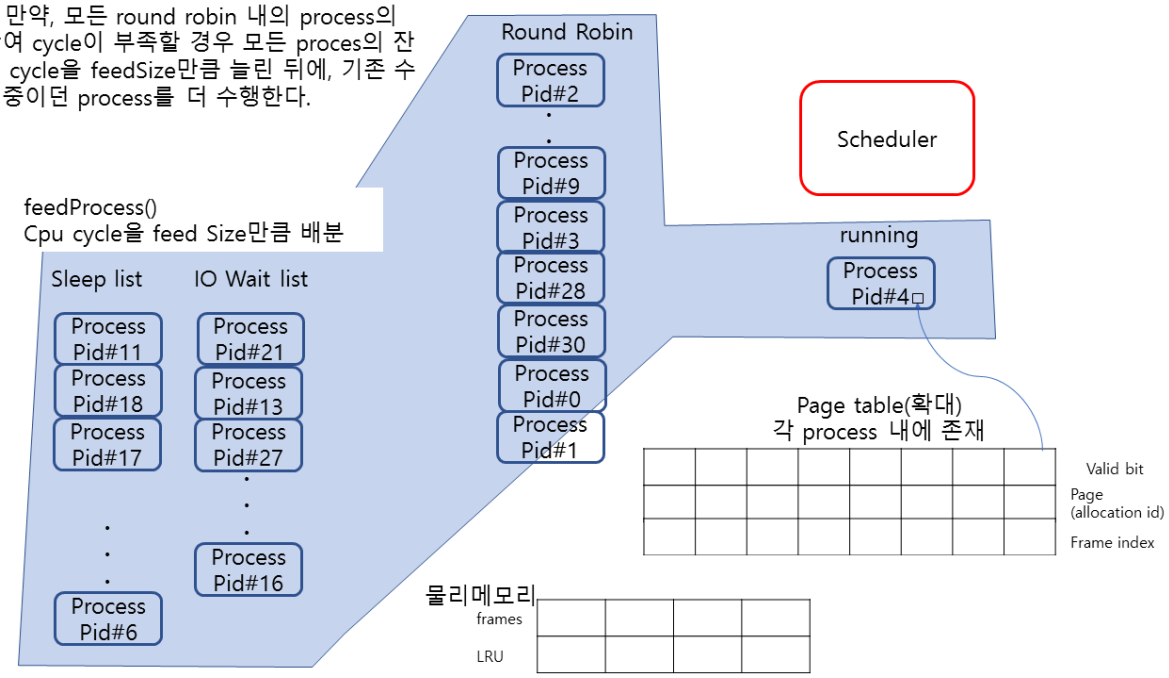
5. 현재 수행중인proces가 없을 경우 round robin queue에서 잔여 cycle이 남아있는 process를 앞에서부터 찾는다. 만약 접근한 process가 잔여 cycle이 없는 경우 Round robin의 가장 뒤에 위치하게 되고, 잔여 cycle이 있는 경우 running process로 수행할 준비를 한다.



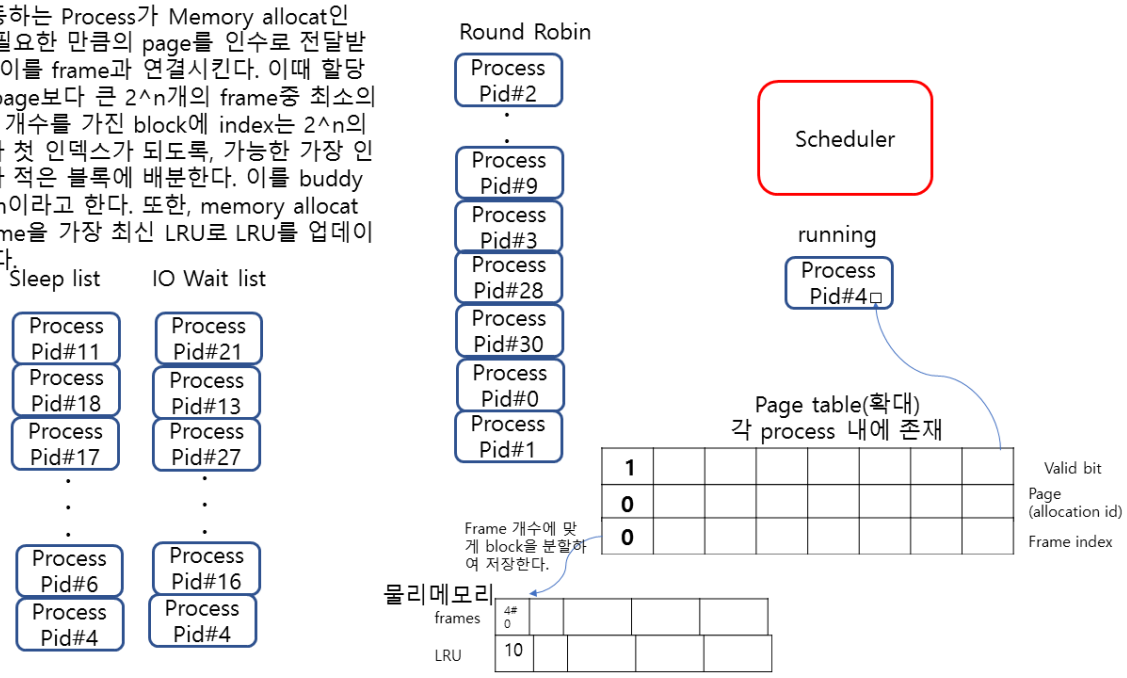




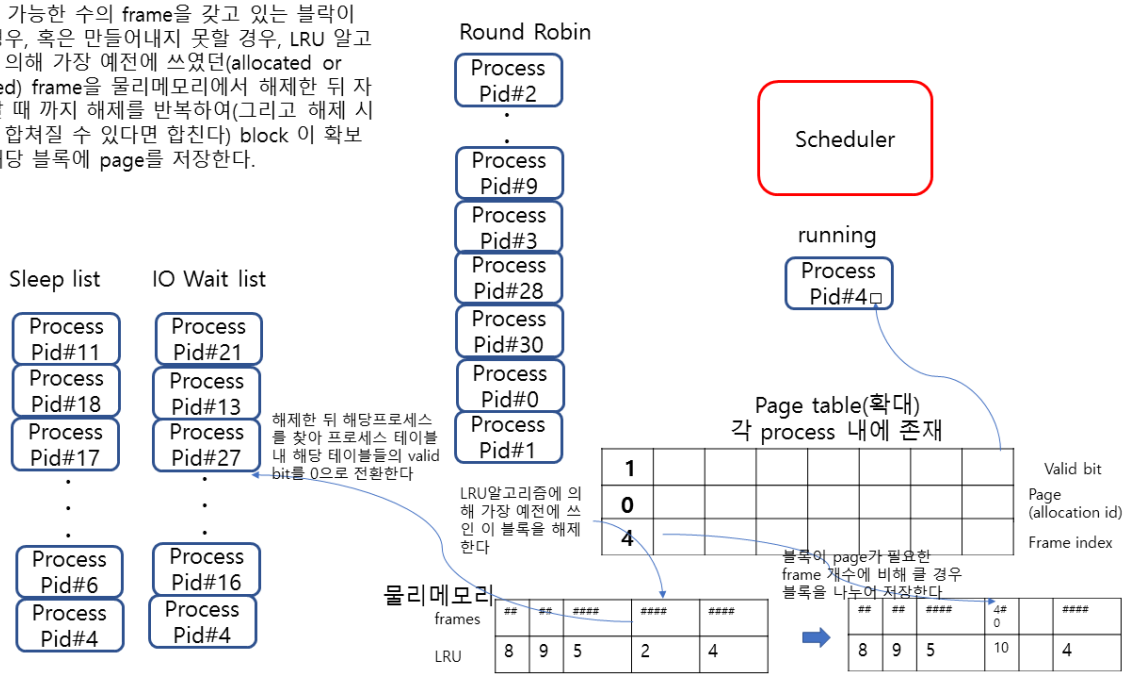
5. 만약, 모든 round robin 내의 process의 잔여 cycle이 부족할 경우 모든 proces의 잔여 cycle을 feedSize만큼 늘린 뒤에, 기존 수행중이던 process를 더 수행한다.



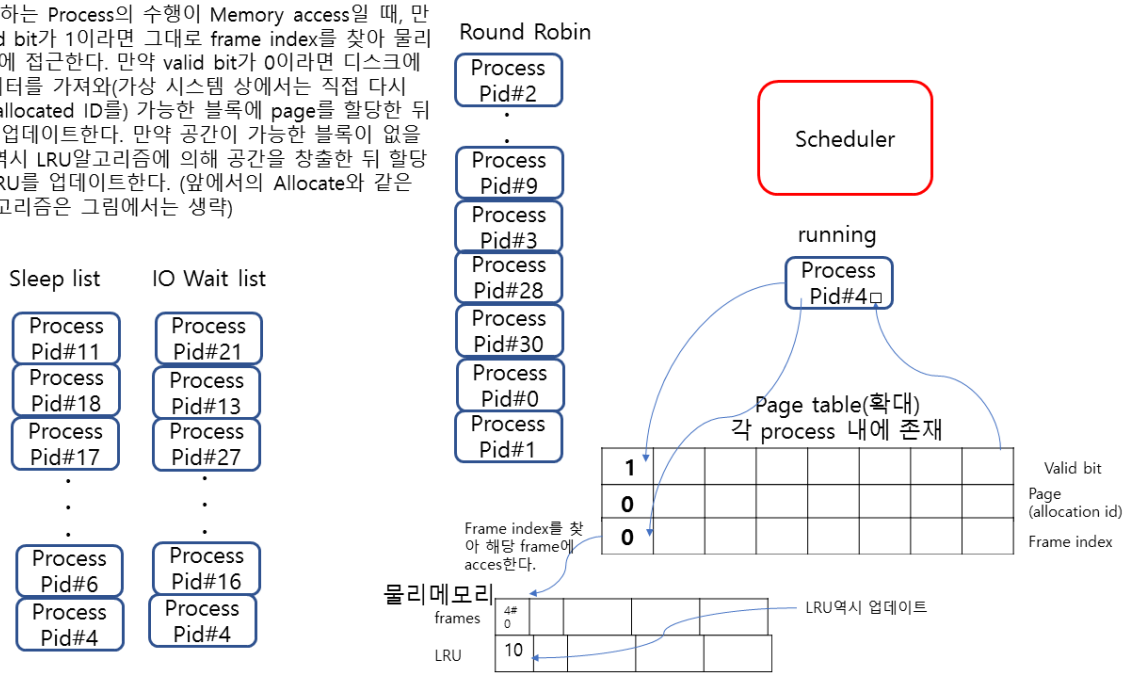
6. 작동하는 Process가 Memory allocat인 경우 필요한 만큼의 page를 인수로 전달받은 뒤 이를 frame과 연결시킨다. 이때 할당받은 page보다 큰 2^n개의 frame중 최소의 frame 개수를 가진 block에 index는 2^n의 배수가 첫 인덱스가 되도록, 가능한 가장 인덱스가 적은 블록에 배분한다. 이를 buddy system이라고 한다. 또한, memory allocat된 frame을 가장 최신 LRU로 LRU를 업데이트 한다.



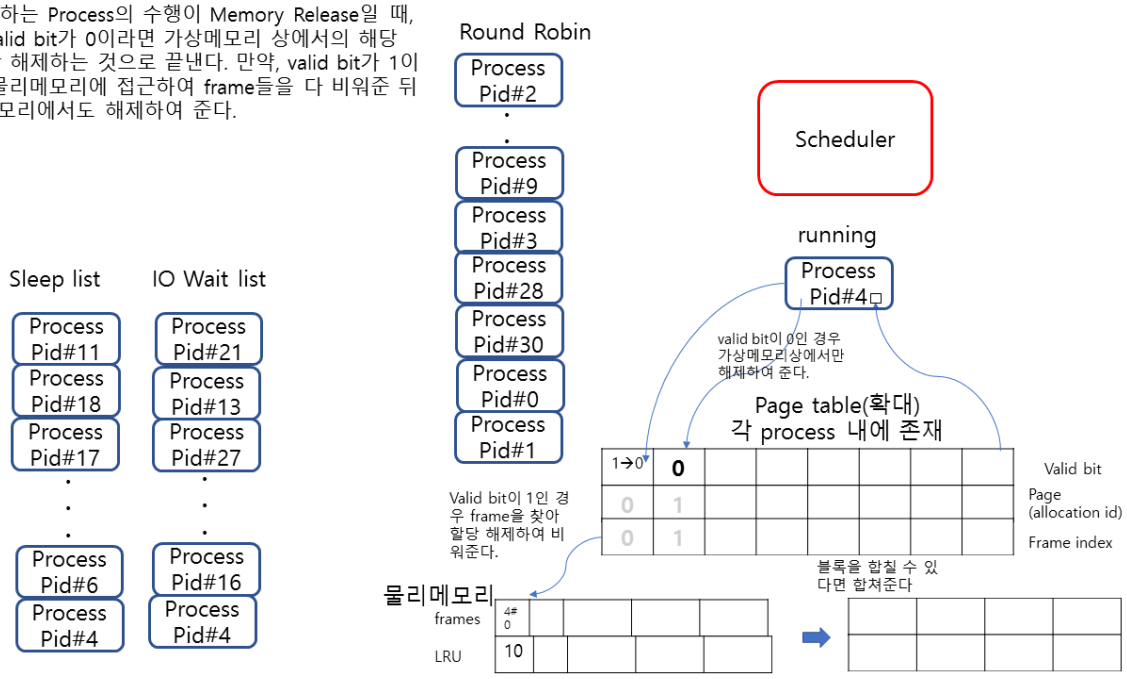
6. 만약 가능한 수의 frame을 갖고 있는 블락이 없을 경우, 혹은 만들어내지 못할 경우, LRU 알고리즘에 의해 가장 예전에 쓰였던(allocated or accessed) frame을 물리메모리에서 해제한 뒤 자리가 날 때 까지 해제를 반복하여(그리고 해제 시 블록이 합쳐질 수 있다면 합친다) block 이 확보되면 해당 블록에 page를 저장한다.



6. 작동하는 Process의 수행이 Memory access일 때, 만약 valid bit가 1이라면 그대로 frame index를 찾아 물리메모리에 접근한다. 만약 valid bit가 0이라면 디스크에서 데이터를 가져와(가상 시스템 상에서는 직접 다시 pid와 allocated ID를) 가능한 블록에 page를 할당한 뒤 LRU를 업데이트한다. 만약 공간이 가능한 블록이 없을 경우, 역시 LRU알고리즘에 의해 공간을 창출한 뒤 할당하고 LRU를 업데이트한다. (앞에서의 Allocate와 같은 LRU알고리즘은 그림에서는 생략)



6. 작동하는 Process의 수행이 Memory Release일 때, 만약 valid bit가 0이라면 가상메모리 상에서의 해당 page만 해제하는 것으로 끝낸다. 만약, valid bit가 1이라면, 물리메모리에 접근하여 frame들을 다 비워준 뒤 가상메모리에서도 해제하여 준다.



3. Uname -a 개발환경

```
hanmo@ubuntu:~/eclipse-workspace/os_assignment3$ uname -a
Linux ubuntu 4.14.24 #1 SMP Sun Mar 18 01:17:08 PDT 2018 x86_64 x86_64 x86_64 GN
U/Linux
```

Windows 버전

Windows 8.1 K

© 2013 Microsoft Corporation. All rights reserved.

새로운 Windows 버전의 추가 기능 가져오기

시스템

제조업체:	Samsung Electronics
프로세서:	Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz 2.30 GHz
설치된 메모리(RAM):	8.00GB
시스템 종류:	64비트 운영 체제, x64 기반 프로세서
펜 및 터치:	5개의 터치 포인트를 사용할 수 있는 제한된 터치식 지원

Samsung Electronics 지원

웹 사이트:	온라인 지원

컴퓨터 이름, 도메인 및 작업 그룹 설정

컴퓨터 이름:	hanmopc
전체 컴퓨터 이름:	hanmopc
컴퓨터 설명:	
작업 그룹:	WORKGROUP

4. 참고문헌 :

A. http://jdm.kr/blog/170

B. http://itguru.tistory.com/215

C. http://dream-cy.tistory.com/5

D. https://msdn.microsoft.com/ko-kr/library/802d66bt.aspx

E. http://initial4-blog.blogspot.com/2012/05/c-study-45-part.html

F. http://openness.tistory.com/entry/gdb%EB%A5%BC-%EC%9D%B4%EC%9A%A9%ED%95%9C-core-%ED%8C%8C%EC%9D%BC%EB%B6%84%EC%84%9D

G. https://stackoverflow.com/questions/29378849/remove-object-from-c-list?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa