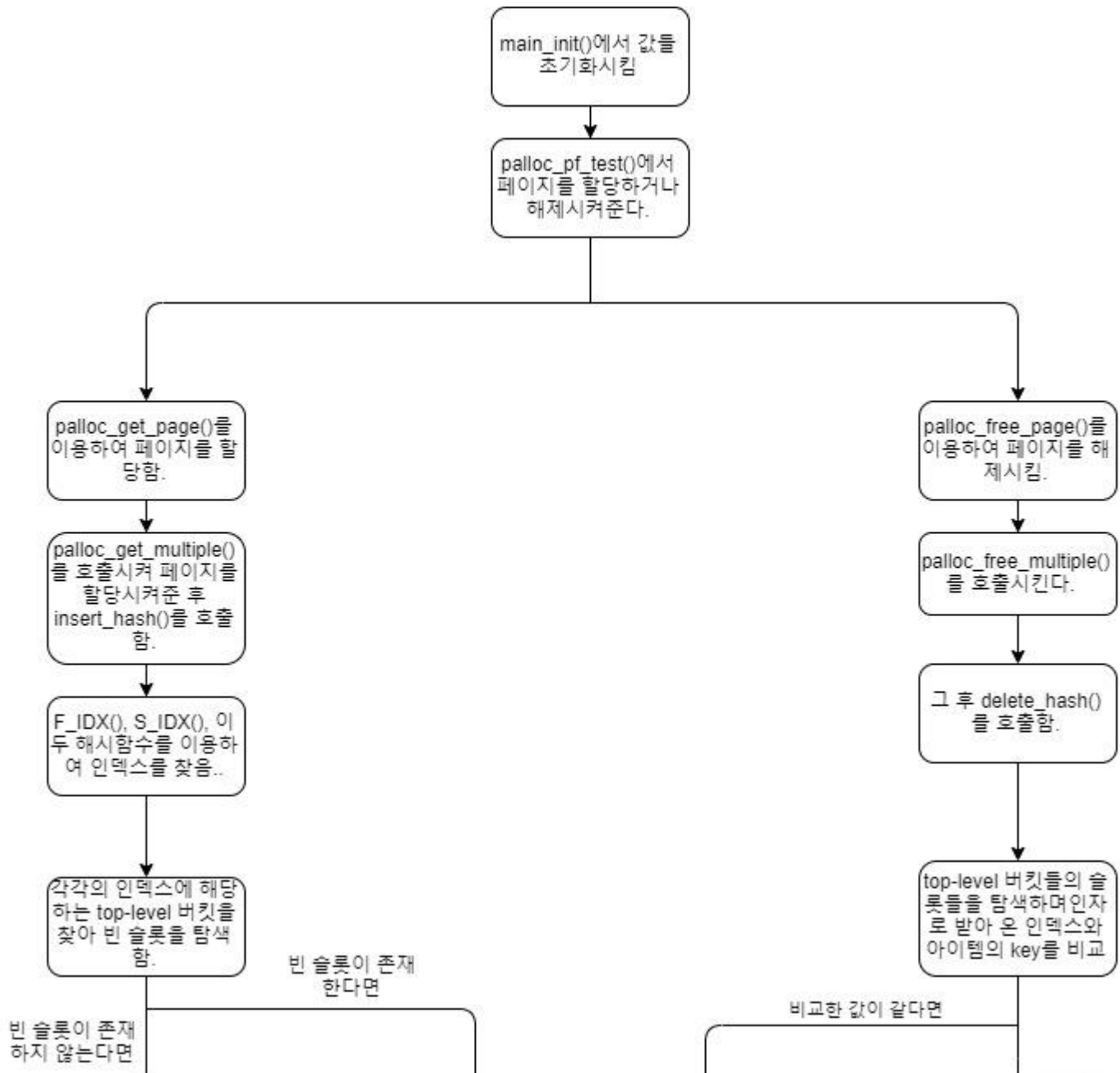
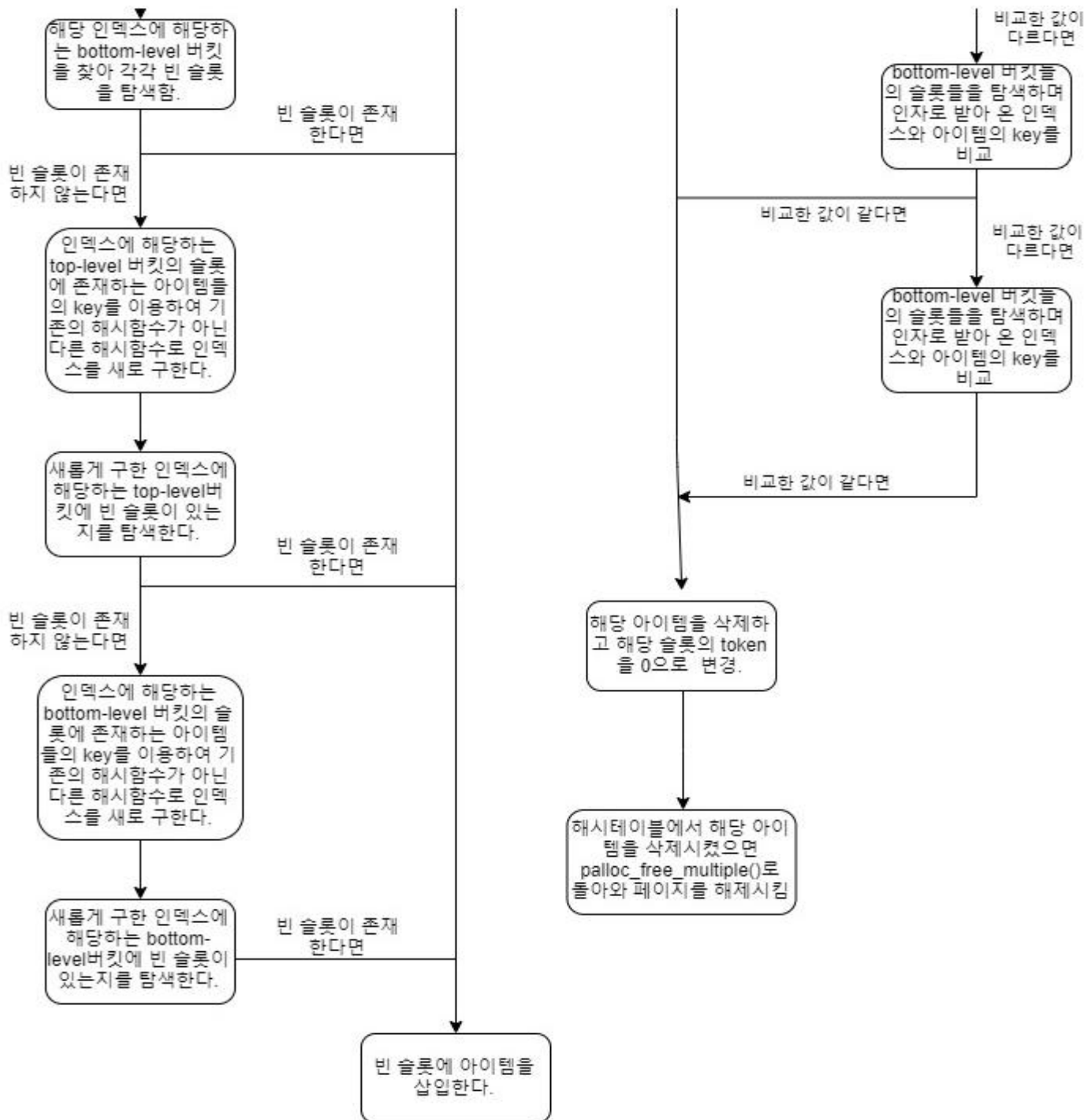


1. 개요

-가상메모리란 프로그램이 실행되기 위해서는 주기억장치로 들어가야 하는데, 실행될 프로그램이 주기억장치보다 크거나 여러 개인 경우에는 주기억장치 공간의 부족으로 인해 프로그램이 제대로 실행되지 못할 수 있다. 그래서 당장 실행에 필요한 부분만 주기억장치에 저장하고, 나머지는 보조기억장치에 두고 동작하도록 하여 이런 문제를 해결할 수 있는데, 이런 개념을 가상 메모리라 하며 운영체제에서 지원한다. 이러한 가상메모리에서 어떻게 Page Allocator를 이해해본다. 그리고 level hash의 삽입과 삭제, 이동기능을 구현한 다음, 이를 이용하여 역테이블페이지를 구현해본다.

2. 상세설계





- void init_hash_table(void); //해시테이블을 초기화하는 함수이다.
- void insert_hash(int idx, uint32_t addr); //해시테이블에 아이템을 삽입하는 함수이다. 삽입하려는 버킷이 모두 가득 차 있으면 해당 버킷의 아이템들을 이동시켜 빈 슬롯을 만든 후 그 곳에 아이템을 삽입한다.
- void delete_hash(int idx, uint32_t addr); //해시테이블에서 아이템을 삭제하는 함수이다.
- uint32_t *palloc_get_multiple (size_t page_cnt); //페이지를 할당하는 함수이다. 이 함수에서 페이지를 할당 한 후 해시테이블에 해당 정보를 삽입한다.
- void palloc_free_multiple (void *pages, size_t page_cnt); //페이지를 해제하는 함수이다. 이 함수에서는 해시테이블에서 아이템을 삭제한 후 해당 페이지를 해제한다.

3. 실행결과

1) Inverted Page Table 구현 직후 실행결과

```
Bochs x86 emulator, http://bochs.sourceforge.net/
=====
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
hash value deleted : idx : 76, key : 2, value : 202000
hash value deleted : idx : 13, key : 4, value : 204000
hash value deleted : idx : 45, key : 5, value : 205000
hash value inserted in top level : idx : 76, key : 2, value : 202000
hash value inserted in top level : idx : 45, key : 5, value : 205000
hash value inserted in top level : idx : 13, key : 4, value : 204000
one_page1 = c0002000
one_page2 = c0004000
two_page1 = c0005000
=====
hash value deleted : idx : 13, key : 4, value : 204000
Page fault : C0007000
hash value inserted in top level : idx : 109, key : 7, value : 207000
hash value inserted in top level : idx : 13, key : 4, value : 204000
one_page1 = c0002000
one_page2 = c0007000
three_page = c0004000
hash value deleted : idx : 76, key : 2, value : 202000
hash value deleted : idx : 13, key : 4, value : 204000
hash value deleted : idx : 45, key : 5, value : 205000
hash value deleted : idx : 109, key : 7, value : 207000
IPS: 14,050M  NUM  CAPS  SCRL  HD:0-M
```

2) main_init()에서 첫 번째 while(1)문 제거 후 실행결과

```
Bochs x86 emulator, http://bochs.sourceforge.net/
=====
hash value deleted : idx : 13, key : 4, value : 204000
Page fault : C0007000
hash value inserted in top level : idx : 109, key : 7, value : 207000
hash value inserted in top level : idx : 13, key : 4, value : 204000
one_page1 = c0002000
one_page2 = c0007000
three_page = c0004000
hash value deleted : idx : 76, key : 2, value : 202000
hash value deleted : idx : 13, key : 4, value : 204000
hash value deleted : idx : 45, key : 5, value : 205000
hash value deleted : idx : 109, key : 7, value : 207000
Testing semaphores...hash value inserted in top level : idx : 76, key : 2, value : 202000
hash value inserted in top level : idx : 13, key : 4, value : 204000
Page fault : C0000000
hash value inserted in top level : idx : 45, key : 5, value : 205000
Page fault : C0001000
hash value inserted in top level : idx : 77, key : 6, value : 206000
hash value deleted : idx : 13, key : 4, value : 204000
hash value deleted : idx : 45, key : 5, value : 205000
hash value deleted : idx : 77, key : 6, value : 206000
done.
===== initialization complete =====
IPS: 13,798M  NUM  CAPS  SCRL  HD:0-M
```

3) main_init()에서 두 번째 while(1)문 제거 후 실행결과

```
Bochs x86 emulator, http://bochs.sourceforge.net/
hash value deleted : idx : 77, key : 6, value : 206000
done.
===== initialization complete =====
hash value inserted in top level : idx :109, key : 7, value : 207000
hash value inserted in top level : idx :199, key : 2, value : 202000
hash value inserted in top level : idx : 13, key : 4, value : 204000
hash value inserted in top level : idx : 45, key : 5, value : 205000
hash value inserted in top level : idx : 77, key : 6, value : 206000
Page fault : C0008000
hash value inserted in top level : idx : 14, key : 8, value : 208000
Page fault : C0009000
hash value inserted in top level : idx : 46, key : 9, value : 209000
Page fault : C000A000
hash value inserted in top level : idx : 78, key :10, value : 20a000
Page fault : C000B000
hash value inserted in top level : idx :110, key :11, value : 20b000
Page fault : C000C000
hash value inserted in top level : idx : 15, key :12, value : 20c000
Page fault : C000D000
hash value inserted in top level : idx : 47, key :13, value : 20d000
Page fault : C000E000
hash value inserted in top level : idx : 79, key :14, value : 20e000
id :
```

IPS: 11.685M	NUM	CAPS	SCRL	HD:0-M						
--------------	-----	------	------	--------	--	--	--	--	--	--

4) while(1)문 제거 및 id, password 입력 후 uname 커맨드 입력 결과

```
Bochs x86 emulator, http://bochs.sourceforge.net/
Page fault : C000C000
hash value inserted in top level : idx : 15, key :12, value : 20c000
Page fault : C000D000
hash value inserted in top level : idx : 47, key :13, value : 20d000
Page fault : C000E000
hash value inserted in top level : idx : 79, key :14, value : 20e000
id : ssuos
password : oslab
> uname
Page fault : C000F000
hash value inserted in top level : idx :111, key :15, value : 20f000
Page fault : C0010000
hash value inserted in top level : idx : 16, key :16, value : 210000
Page fault : C0011000
hash value inserted in top level : idx : 48, key :17, value : 211000
Page fault : C0012000
hash value inserted in top level : idx : 80, key :18, value : 212000
SSUOS 0.1.02
made by OSLAB
modified by You
hash value deleted : idx : 16, key :16, value : 210000
hash value deleted : idx : 48, key :17, value : 211000
hash value deleted : idx : 80, key :18, value : 212000
>
```

IPS: 9.883M	NUM	CAPS	SCRL	HD:0-M						
-------------	-----	------	------	--------	--	--	--	--	--	--

```
Bochs x86 emulator, http://bochs.sourceforge.net/
USER Copy Paste Snapshot CONFIG Reset Suspend Power
Page fault : C000D000
hash value inserted in top level : idx : 47, key :13, value : 20d000
Page fault : C000E000
hash value inserted in top level : idx : 79, key :14, value : 20e000
id : ssuos
password : oslab
> ps
Page fault : C000F000
hash value inserted in top level : idx :111, key :15, value : 20f000
Page fault : C0010000
hash value inserted in top level : idx : 16, key :16, value : 210000
Page fault : C0011000
hash value inserted in top level : idx : 48, key :17, value : 211000
Page fault : C0012000
hash value inserted in top level : idx : 80, key :18, value : 212000
pid 0 ppid 0 state 1 prio 0 using time 105 sched time 0, pd = 200000
pid 2 ppid 0 state 1 prio 200 using time 0 sched time 0, pd = 202000
pid 3 ppid 0 state 1 prio 200 using time 0 sched time 0, pd = 208000
pid 4 ppid 0 state 1 prio 100 using time 4570 sched time 0, pd = 20c000
pid 5 ppid 4 state 1 prio 100 using time 20 sched time 0, pd = 210000
hash value deleted : idx : 16, key :16, value : 210000
hash value deleted : idx : 48, key :17, value : 211000
hash value deleted : idx : 80, key :18, value : 212000
>
```


4. 소스코드

```
-hashing.h
#ifndef __HASHING_H__
#define __HASHING_H__

#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <type.h>
#include <proc/proc.h>
#include <ssulib.h>

#define SLOT_NUM 4 // The number of slots in a bucket
#define CAPACITY 256 // level hash table capacity

typedef struct entry{
    uint32_t key;
    uint32_t value;
} entry;

typedef struct level_bucket
{
    uint8_t token[SLOT_NUM];
    entry slot[SLOT_NUM];
} level_bucket;

typedef struct level_hash {
    level_bucket top_buckets[CAPACITY];
    level_bucket bottom_buckets[CAPACITY / 2];
} level_hash;

level_hash hash_table;

void init_hash_table(void);
uint32_t F_IDX(uint32_t addr, uint32_t capacity); // Get first index to use at table
uint32_t S_IDX(uint32_t addr, uint32_t capacity); // Get second index to use at table
void insert_hash(int idx, uint32_t addr); //해시테이블에서 아이템을 삽입하는 함수를 선언했다.
void delete_hash(int idx, uint32_t addr); //해시테이블에서 아이템을 삭제하는 함수를 선언했다.

#endif
```

-hashing.c

1) init_hash_table() 수정

void init_hash_table(void)//해시테이블의 값들을 초기화시켜주는 함수

```
{
    for(int i=0;i<CAPACITY;i++){
        if(i/2==0){
            for(int j=0;j<SLOT_NUM;j++){
                hash_table.bottom_buckets[i/2].token[j]=0; //bottom_level 버킷들의 각 슬롯의
token을 0으로 초기화
                hash_table.bottom_buckets[i/2].slot[j].key=0; //bottom_level 버킷들의 각 슬롯의
key을 0으로 초기화
                hash_table.bottom_buckets[i/2].slot[j].value=0; //bottom_level 버킷들의 각 슬롯의
value를 0으로 초기화
            }
        }
        for(int j=0;j<SLOT_NUM;j++){
            hash_table.top_buckets[i].token[j] = 0; //top_level 버킷들의 각 슬롯의 token을 0으로 초기화
            hash_table.top_buckets[i].slot[j].key = 0; //top_level 버킷들의 각 슬롯의 key을 0으로 초기화
            hash_table.top_buckets[i].slot[j].value = 0; //top_level 버킷들의 각 슬롯의 value를 0으로 초
기화
        }
    }
}
```

2) insert_hash() 추가

void insert_hash(int idx, uint32_t addr){

```
    uint32_t Fpos, Spos;
    int tmpPos;
    int check=0;
    Fpos = F_IDX(RH_TO_VH(addr), CAPACITY);
    Spos = S_IDX(RH_TO_VH(addr), CAPACITY);
    //첫번째 해시함수와 두번째 해시함수의 인덱스에 해당하는 top-level 버킷들의 슬롯을 각각 번갈아 확인하며 빈
슬롯을 찾는다.
    for(int i=0;i<SLOT_NUM;i++){
        if(hash_table.top_buckets[Fpos].token[i]==0){ //빈 슬롯이 있다면 아이템을 삽입한다.
            hash_table.top_buckets[Fpos].slot[i].key=idx;
            hash_table.top_buckets[Fpos].slot[i].value = addr;
            hash_table.top_buckets[Fpos].token[i]=1;
            printf("hash value inserted in top level : idx :%3d, key :%2d, value
:%7x\n",Fpos,hash_table.top_buckets[Fpos].slot[i].key,hash_table.top_buckets[Fpos].slot[i].value);
            check=1;
            break;
        }
    }
```

```

else if(hash_table.top_buckets[Spos].token[i]==0){ //빈 슬롯이 있다면 아이템을 삽입한다.
    hash_table.top_buckets[Spos].slot[i].key=idx;
    hash_table.top_buckets[Spos].slot[i].value = addr;
    hash_table.top_buckets[Spos].token[i]=1;
    printk("hash value inserted in top level : idx :%3d, key :%2d, value
:%7x\n",Spos,hash_table.top_buckets[Spos].slot[i].key,hash_table.top_buckets[Spos].slot[i].value);
    check=1;
    break;
}
}

```

```

if(!check){//top-level 버킷들을 다 탐색했는데 빈 슬롯이 없을 때는
    for(int i=0;i<SLOT_NUM;i++){// 첫번째 해시함수와 두번째 해시함수의 인덱스에 해당하는 bottom-level
버킷들의 슬롯을 각각 번갈아 확인하며 빈 슬롯을 찾는다.
        if(hash_table.bottom_buckets[Fpos/2].token[i]==0){
            hash_table.bottom_buckets[Fpos/2].slot[i].key=idx;
            hash_table.bottom_buckets[Fpos/2].slot[i].value = addr;
            hash_table.bottom_buckets[Fpos/2].token[i]=1;
            printk("hash value inserted in bottom level : idx :%3d, key :%2d, value
:%7x\n",Fpos/2,hash_table.bottom_buckets[Fpos/2].slot[i].key,hash_table.bottom_buckets[Fpos/2].slot[i].value);
            check=1;
            break;
        }

        else if(hash_table.bottom_buckets[Spos/2].token[i]==0){
            hash_table.bottom_buckets[Spos/2].slot[i].key=idx;
            hash_table.bottom_buckets[Spos/2].slot[i].value = addr;
            hash_table.bottom_buckets[Spos/2].token[i]=1;
            printk("hash value inserted in bottom level : idx :%3d, key :%2d, value
:%7x\n",Spos/2,hash_table.bottom_buckets[Spos/2].slot[i].key,hash_table.bottom_buckets[Spos/2].slot[i].value);
            check=1;
            break;
        }
    }
}
}

```

//top-level과 bottom-level의 버킷들에 빈 슬롯이 없을 때는 이미 삽입된 아이템 중 하나를 다른 Top-level/bottom-level로 이동 후 아이템 삽입한다.

```

if(!check){//toplevel(Fpos) 버킷아이템 중 하나를 다른 toplevel 버킷으로 이동 후 아이템을 삽입한다.
    for(int i=0;i<SLOT_NUM;i++){
        //toplevel(Fpos)안에 있는 아이템들이 F_IDX해시함수를 쓴 결과라고 가정하고 S_IDX해시함수를

```


이용해서 새로운 인덱스를 구해본다.

```
tmpPos =
```

```
S_IDX(RH_TO_VH(hash_table.top_buckets[Fpos].slot[i].value),CAPACITY); //S_IDX해시함수를 이용해서 새로운 인덱스  
구하고
```

```
if(tmpPos == Fpos) //현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른 슬롯으  
로 넘어간다.
```

```
continue;
```

```
for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 top-level버킷에 빈 슬롯이 있는지 탐색한다.
```

```
if(hash_table.top_buckets[tmpPos].token[j]==0){ //빈 슬롯이 있다면
```

```
hash_table.top_buckets[tmpPos].slot[j].key=hash_table.top_buckets[Fpos].slot[i].key;
```

```
hash_table.top_buckets[tmpPos].slot[j].value =
```

```
hash_table.top_buckets[Fpos].slot[i].value;
```

```
hash_table.top_buckets[tmpPos].token[j]=1; //아이템이동성공
```

```
printf("hash value inserted in the another top level : idx :%3d, key  
:%2d,
```

```
:%7x\n",tmpPos,hash_table.top_buckets[tmpPos].slot[j].key,hash_table.top_buckets[tmpPos].slot[j].value);
```

```
//비어진 기존슬롯에 아이템을 삽입한다.
```

```
hash_table.top_buckets[Fpos].slot[i].key=idx;
```

```
hash_table.top_buckets[Fpos].slot[i].value = addr;
```

```
hash_table.top_buckets[Fpos].token[i]=1;
```

```
printf("hash value inserted in top level : idx :%3d, key :%2d, value  
:%7x\n",Fpos,hash_table.top_buckets[Fpos].slot[i].key,hash_table.top_buckets[Fpos].slot[i].value);
```

```
return;
```

```
}
```

```
}
```

```
}
```

```
//새로운 인덱스의 top-level버킷에 빈 슬롯이 존재하지 않았다면
```

```
for(int i=0;i<SLOT_NUM;i++){
```

```
//toplevel(Fpos)안에 있는 아이템들이 S_IDX해시함수를 쓴 결과라고 가정하고 F_IDX해시함수를  
이용해서 새로운 인덱스를 구해본다.
```

```
tmpPos = F_IDX(RH_TO_VH(hash_table.top_buckets[Fpos].slot[i].value),CAPACITY); //새로  
운 인덱스 구하고
```

```
if(tmpPos == Fpos) //현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른 슬롯으  
로 넘어간다.
```

```
continue;
```

```
for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 top-level버킷에 빈 슬롯이 있는지 탐색한다.
```

```
if(hash_table.top_buckets[tmpPos].token[j]==0){ //빈 슬롯이 있다면 아이템을 이동한
```

다.

```
hash_table.top_buckets[tmpPos].slot[j].key=hash_table.top_buckets[Fpos].slot[i].key;
hash_table.top_buckets[tmpPos].slot[j].value
hash_table.top_buckets[Fpos].slot[i].value;
hash_table.top_buckets[tmpPos].token[j]=1;//아이템이동성공
printf("hash value inserted in the another top level : idx :%3d, key
:%2d,
:%7x\n",tmpPos,hash_table.top_buckets[tmpPos].slot[j].key,hash_table.top_buckets[tmpPos].slot[j].value);
```

```
//비어진 기존슬롯에 아이템을 삽입한다.
hash_table.top_buckets[Fpos].slot[i].key=idx;
hash_table.top_buckets[Fpos].slot[i].value = addr;
hash_table.top_buckets[Fpos].token[i]=1;
printf("hash value inserted in top level : idx :%3d, key :%2d, value
:%7x\n",Fpos,hash_table.top_buckets[Fpos].slot[i].key,hash_table.top_buckets[Fpos].slot[i].value);
//check_m=1;
return;
}
```

```
}
```

```
//toplevel(Spos) 버킷아이템 중 하나를 다른 toplevel 버킷으로 이동 후 아이템을 삽입한다.
for(int i=0;i<SLOT_NUM;i++){
//toplevel(Spos)안에 있는 아이템들이 F_IDX해시함수를 쓴 결과라고 가정하고 S_IDX해시함수를
이용해서 새로운 인덱스를 구해본다.
tmpPos = S_IDX(RH_TO_VH(hash_table.top_buckets[Spos].slot[i].value),CAPACITY);//새로
운 인덱스 구하고
if(tmpPos == Spos)//현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른 슬롯으
로 넘어간다.
```

```
continue;
for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 top-level버킷에 빈 슬롯이 있는지 탐색한다.
if(hash_table.top_buckets[tmpPos].token[j]==0){//빈 슬롯이 있다면 아이템을 이동한
다.
```

```
hash_table.top_buckets[tmpPos].slot[j].key=hash_table.top_buckets[Spos].slot[i].key;
hash_table.top_buckets[tmpPos].slot[j].value
hash_table.top_buckets[Spos].slot[i].value;
hash_table.top_buckets[tmpPos].token[j]=1;//아이템이동성공
printf("hash value inserted in the another top level : idx :%3d, key
:%2d,
:%7x\n",tmpPos,hash_table.top_buckets[tmpPos].slot[j].key,hash_table.top_buckets[tmpPos].slot[j].value);
```

```

        //비어진 기존슬롯에 아이템을 삽입한다.
        hash_table.top_buckets[Spos].slot[i].key=idx;
        hash_table.top_buckets[Spos].slot[i].value = addr;
        hash_table.top_buckets[Spos].token[i]=1;
        printf("hash value inserted in top level : idx :%3d, key :%2d, value
:%7x\n",Spos,hash_table.top_buckets[Spos].slot[i].key,hash_table.top_buckets[Spos].slot[i].value);
        return;
    }
}

//새로운 인덱스의 top-level버킷에 빈 슬롯이 존재하지 않았다면
for(int i=0;i<SLOT_NUM;i++){

    //toplevel(Spos)안에 있는 아이템들이 S_IDX해시함수를 쓴 결과라고 가정하고 F_IDX해시함수를
이용해서 인덱스를 구한다.
    tmpPos = F_IDX(RH_TO_VH(hash_table.top_buckets[Spos].slot[i].value),CAPACITY);//새로
운 인덱스 구하고

    if(tmpPos==Spos)//현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른 슬롯으로
넘어간다.

        continue;
    for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 top-level버킷에 빈 슬롯이 있는지 탐색한다.
        if(hash_table.top_buckets[tmpPos].token[j]==0){//빈 슬롯이 있다면 아이템을 이동한
다.

hash_table.top_buckets[tmpPos].slot[j].key=hash_table.top_buckets[Spos].slot[i].key;
hash_table.top_buckets[tmpPos].slot[j].value =
hash_table.top_buckets[Spos].slot[i].value;
hash_table.top_buckets[tmpPos].token[j]=1;//아이템이동성공
        printf("hash value inserted in the another top level : idx :%3d, key
:%2d,
:%7x\n",tmpPos,hash_table.top_buckets[tmpPos].slot[j].key,hash_table.top_buckets[tmpPos].slot[j].value);

        //비어진 기존슬롯에 아이템을 삽입한다.
        hash_table.top_buckets[Spos].slot[i].key=idx;
        hash_table.top_buckets[Spos].slot[i].value = addr;
        hash_table.top_buckets[Spos].token[i]=1;
        printf("hash value inserted in top level : idx :%3d, key :%2d, value
:%7x\n",Spos,hash_table.top_buckets[Spos].slot[i].key,hash_table.top_buckets[Spos].slot[i].value);
        return;
    }
}

```

```

    }
    //bottomlevel(Fpos/2) 버킷아이템 중 하나를 다른 toplevel 버킷으로 이동 후 아이টে을 삽입한다.
    for(int i=0;i<SLOT_NUM;i++){
        //bottomlevel(Fpos/2)안에 있는 아이টে들이 F_IDX해시함수를 쓴 결과라고 가정하고 S_IDX해시
함수를 이용해서 새로운 인덱스를 구한다.
        tmpPos =
S_IDX(RH_TO_VH(hash_table.bottom_buckets[Fpos/2].slot[i].value),CAPACITY);//새로운 인덱스 구하고
        if((tmpPos/2) == (Fpos/2))//현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른
슬롯으로 넘어간다.
            continue;
        for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 bottom-level버킷에 빈 슬롯이 있는지 탐색한다.
            if(hash_table.bottom_buckets[tmpPos/2].token[j]==0){//빈 슬롯이 있다면 아이টে을 이
동한다.

hash_table.bottom_buckets[tmpPos/2].slot[j].key=hash_table.bottom_buckets[Fpos/2].slot[i].key;
                hash_table.bottom_buckets[tmpPos/2].slot[j].value =
hash_table.bottom_buckets[Fpos/2].slot[i].value;
                hash_table.bottom_buckets[tmpPos/2].token[j]=1;//아이টে이동성공
                printk("hash value inserted in the another bottom level : idx :%3d, key
:%2d, value
:%7x\n",tmpPos/2,hash_table.bottom_buckets[tmpPos/2].slot[j].key,hash_table.bottom_buckets[tmpPos/2].slot[j].val
ue);

                //비어진 기존슬롯에 아이টে을 삽입한다.
                hash_table.bottom_buckets[Fpos/2].slot[i].key=idx;
                hash_table.bottom_buckets[Fpos/2].slot[i].value = addr;
                hash_table.bottom_buckets[Fpos/2].token[i]=1;
                printk("hash value inserted in bottom level : idx :%3d, key :%2d, value
:%7x\n", Fpos/2, hash_table.bottom_buckets[Fpos/2].slot[i].key,hash_table.bottom_buckets[Fpos/2].slot[i].value);
                return;
            }
        }
    }

    //새로운 인덱스의 bottom-level버킷에 빈 슬롯이 존재하지 않았다면
    for(int i=0;i<SLOT_NUM;i++){
        //bottomlevel(Fpos/2)안에 있는 아이টে들이 S_IDX해시함수를 쓴 결과라고 가정하고 F_IDX해시
함수를 이용해서 인덱스를 구한다.
        tmpPos =
F_IDX(RH_TO_VH(hash_table.bottom_buckets[Fpos/2].slot[i].value),CAPACITY);//새로운 인덱스 구하고
        if((tmpPos/2)==(Fpos/2))//현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른
슬롯으로 넘어간다.
            continue;
        for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 bottom-level버킷에 빈 슬롯이 있는지 탐색한다.

```

if(hash_table.bottom_buckets[tmpPos/2].token[j]==0){ //빈 슬롯이 있다면 아이템을 이동한다.

```
hash_table.bottom_buckets[tmpPos/2].slot[j].key=hash_table.bottom_buckets[Fpos/2].slot[i].key;
hash_table.bottom_buckets[tmpPos/2].slot[j].value =
hash_table.bottom_buckets[Fpos/2].slot[i].value;
hash_table.bottom_buckets[tmpPos/2].token[j]=1;//아이템이동성공
printf("hash value inserted in the another bottom level : idx :%3d, key
:%2d,
:%7x\n",tmpPos/2,hash_table.bottom_buckets[tmpPos/2].slot[j].key,hash_table.bottom_buckets[tmpPos/2].slot[j].value);
```

```
//비어진 기존슬롯에 아이템을 삽입한다.
hash_table.bottom_buckets[Fpos/2].slot[i].key=idx;
hash_table.bottom_buckets[Fpos/2].slot[i].value = addr;
hash_table.bottom_buckets[Fpos/2].token[i]=1;
printf("hash value inserted in bottom level : idx :%3d, key :%2d, value
:%7x\n", Fpos/2, hash_table.bottom_buckets[Fpos/2].slot[i].key,hash_table.bottom_buckets[Fpos/2].slot[i].value);
return;
```

```
}
```

```
//bottomlevel(Spos/2) 버킷아이템 중 하나를 다른 toplevel 버킷으로 이동 후 아이템을 삽입한다.
for(int i=0;i<SLOT_NUM;i++){
//bottomlevel(Spos)안에 있는 아이템들이 F_IDX해시함수를 쓴 결과라고 가정하고 S_IDX해시함수를 이용해서 인덱스를 구한다.
tmpPos =
S_IDX(RH_TO_VH(hash_table.bottom_buckets[Spos/2].slot[i].value),CAPACITY);//새로운 인덱스 구하고
if((tmpPos/2)==(Spos/2))//현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른 슬롯으로 넘어간다.
```

```
continue;
for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 bottom-level버킷에 빈 슬롯이 있는지 탐색한다.
if(hash_table.bottom_buckets[tmpPos/2].token[j]==0){ //빈 슬롯이 있다면 아이템을 이동한다.
```

```
hash_table.bottom_buckets[tmpPos/2].slot[j].key=hash_table.bottom_buckets[Spos/2].slot[i].key;
hash_table.bottom_buckets[tmpPos/2].slot[j].value =
hash_table.bottom_buckets[Spos/2].slot[i].value;
hash_table.bottom_buckets[tmpPos/2].token[j]=1;//아이템이동성공
printf("hash value inserted in the another bottom level : idx :%3d, key
:%2d,
:%7x\n",tmpPos/2,hash_table.bottom_buckets[tmpPos/2].slot[j].key,hash_table.bottom_buckets[tmpPos/2].slot[j].value);
```

```

        //비어진 기존슬롯에 아이템을 삽입한다.
        hash_table.bottom_buckets[Spos/2].slot[i].key=idx;
        hash_table.bottom_buckets[Spos/2].slot[i].value = addr;
        hash_table.bottom_buckets[Spos/2].token[i]=1;
        printf("hash value inserted in bottom level : idx :%3d, key :%2d, value
:%7x\n", Spos/2, hash_table.bottom_buckets[Spos/2].slot[i].key,hash_table.bottom_buckets[Spos/2].slot[i].value);
        return;
    }
}

//새로운 인덱스의 bottom-level버킷에 빈 슬롯이 존재하지 않았다면
for(int i=0;i<SLOT_NUM;i++){
    //bottomlevel(Spos/2)안에 있는 아이템들이 S_IDX해시함수를 쓴 결과라고 가정하고 F_IDX해시
함수를 이용해서 인덱스를 구한다.
    tmpPos =
F_IDX(RH_TO_VH(hash_table.bottom_buckets[Spos/2].slot[i].value),CAPACITY);//새로운 인덱스 구하고
    if((tmpPos/2)==(Spos/2))//현재 인덱스와 새롭게 구한 인덱스가 같다면 이동이 안되므로 다른
슬롯으로 넘어간다.
        continue;
    for(int j=0;j<SLOT_NUM;j++){ //그 인덱스의 bottom-level버킷에 빈 슬롯이 있는지 탐색한다.
        if(hash_table.bottom_buckets[tmpPos/2].token[j]==0){ //빈 슬롯이 있다면 아이템을
이동한다.

hash_table.bottom_buckets[tmpPos/2].slot[j].key=hash_table.bottom_buckets[Spos/2].slot[i].key;
        hash_table.bottom_buckets[tmpPos/2].slot[j].value =
hash_table.bottom_buckets[Spos/2].slot[i].value;
        hash_table.bottom_buckets[tmpPos/2].token[j]=1;//아이템이동성공
        printf("hash value inserted in the another bottom level : idx :%3d, key
:%2d,
:%7x\n",tmpPos/2,hash_table.bottom_buckets[tmpPos/2].slot[j].key,hash_table.bottom_buckets[tmpPos/2].slot[j].val
ue);

        //비어진 기존슬롯에 아이템을 삽입한다.
        hash_table.bottom_buckets[Spos/2].slot[i].key=idx;
        hash_table.bottom_buckets[Spos/2].slot[i].value = addr;
        hash_table.bottom_buckets[Spos/2].token[i]=1;
        printf("hash value inserted in bottom level : idx :%3d, key :%2d, value
:%7x\n", Spos/2, hash_table.bottom_buckets[Spos/2].slot[i].key,hash_table.bottom_buckets[Spos/2].slot[i].value);
        return;
    }
}
}

```

```

    }
}

```

3) delete_hash() 추가

```

void delete_hash(int idx, uint32_t addr){
    //삭제 연산 시 해시 함수에 대응하는 위치의 top level 버킷 2개와 top level 버킷에 대응하는 각각의 bottom
level 버킷, 총 4개의 버킷을 탐색하여 아이템을 찾아 삭제
    uint32_t Fpos, Spos;
    int check=0;
    Fpos = F_IDX(RH_TO_VH(addr), CAPACITY);
    Spos = S_IDX(RH_TO_VH(addr), CAPACITY);

    for(int i=0;i<SLOT_NUM;i++){
        //인덱스가 Fpos인 top-level버킷을 탐색하며 인자로 받은 인덱스와 각각의 슬롯의 key값이 같은지 확인
한다.
        if(hash_table.top_buckets[Fpos].slot[i].key==idx){//같다면 해당 아이템을 삭제한다.
            printk("hash value deleted : idx :%3d, key :%2d, value :%7x\n",
Fpos,hash_table.top_buckets[Fpos].slot[i].key,hash_table.top_buckets[Fpos].slot[i].value);
            hash_table.top_buckets[Fpos].token[i]=0;
            hash_table.top_buckets[Fpos].slot[i].key=0;
            hash_table.top_buckets[Fpos].slot[i].value=0;
            check = 1;
            break;
        }
        //인덱스가 Spos인 top-level버킷을 탐색하며 인자로 받은 인덱스와 각각의 슬롯의 key값이 같은지 확인
한다.
        else if(hash_table.top_buckets[Spos].slot[i].key==idx){//같다면 해당 아이템을 삭제한다.
            printk("hash value deleted : idx :%3d, key :%2d, value :%7x\n",
Spos,hash_table.top_buckets[Spos].slot[i].key,hash_table.top_buckets[Spos].slot[i].value);
            hash_table.top_buckets[Spos].token[i]=0;
            hash_table.top_buckets[Spos].slot[i].key=0;
            hash_table.top_buckets[Spos].slot[i].value=0;
            check = 1;
            break;
        }
    }
}

if(!check){//top-level 버킷에는 찾고자하는 아이템이 존재하지 않을 때
    for(int i=0;i<SLOT_NUM;i++){
        //인덱스가 Fpos/2인 bottom-level버킷을 탐색하며 인자로 받은 인덱스와 각각의 슬롯의 key값
이 같은지 확인한다.
        if(hash_table.bottom_buckets[Fpos/2].slot[i].key==idx){//같다면 해당 아이템을 삭제한다.
            printk("hash value deleted : idx :%3d, key :%2d, value

```



```

:%7x\n",Fpos/2,hash_table.bottom_buckets[Fpos/2].slot[i].key, hash_table.bottom_buckets[Fpos/2].slot[i].value);
        hash_table.bottom_buckets[Fpos/2].token[i]=0;
        hash_table.bottom_buckets[Fpos/2].slot[i].key=0;
        hash_table.bottom_buckets[Fpos/2].slot[i].value=0;
        check = 1;
        break;
    }
    //인덱스가 Spos/2인 bottom-level버킷을 탐색하며 인자로 받은 인덱스와 각각의 슬롯의 key값
이 같은지 확인한다.
    else if(hash_table.bottom_buckets[Spos/2].slot[i].key==idx){//같다면 해당 아이템을 삭제한다.
        printk("hash value deleted : idx :%3d, key :%2d, value
:%7x\n",Spos/2,hash_table.bottom_buckets[Spos/2].slot[i].key, hash_table.bottom_buckets[Spos/2].slot[i].value);
        hash_table.bottom_buckets[Spos/2].token[i]=0;
        hash_table.bottom_buckets[Spos/2].slot[i].key=0;
        hash_table.bottom_buckets[Spos/2].slot[i].value=0;
        check = 1;
        break;
    }
}
}
}
}
}

```

-palloc.c

4) palloc_get_multiple() 수정

/* Obtains and returns a group of PAGE_CNT contiguous free pages.

*/

uint32_t *palloc_get_multiple (size_t page_cnt)

```

{
    void *pages = NULL;
    struct khpage *khpage = freelist.list;
    struct khpage *prepage = freelist.list;
    size_t page_idx;

    if (page_cnt == 0)
        return NULL;

    while(khpage != NULL){
        if(khpage->nalloc == page_cnt){
            page_idx = ((uint32_t)khpage - (uint32_t)khpage_list)/sizeof(struct khpage);
            pages = (void*)(VKERNEL_HEAP_START + page_idx * PAGE_SIZE);

            if(prepage == khpage){
                freelist.list = khpage->next;
                freelist.nfree--;
            }
        }
        prepage = khpage;
        khpage = khpage->next;
    }
}

```

```

        break;
    }else{
        prepage->next = khpage->next;
        freelist.nfree--;
        break;
    }

}

prepage = khpage;
khpage = khpage->next;
}

if(pages == NULL){
    pages = (void*)(VKERNEL_HEAP_START + page_alloc_index * PAGE_SIZE);
    page_idx = page_alloc_index;
    page_alloc_index += page_cnt;
}

if (pages != NULL)
{
    memset (pages, 0, PAGE_SIZE * page_cnt);
}

//page_idx가 page index == key
//page의 실제주소는 VH_TO_RH(pages)

insert_hash(page_idx, VH_TO_RH(pages)); //hash table에 아이টে을 삽입하는 함수 추가

return (uint32_t*)pages;
}

```

5) palloc_free_multiple() 수정

/* Frees the PAGE_CNT pages starting at PAGES. */

void palloc_free_multiple (void *pages, size_t page_cnt)

```

{
    struct khpage *khpage = freelist.list;
    size_t page_idx = (((uint32_t)pages - VKERNEL_HEAP_START) / PAGE_SIZE);

    if (pages == NULL || page_cnt == 0)
        return;
}

```

delete_hash(page_idx, VH_TO_RH(pages)); //해시테이블에서 아이템을 삭제하는 함수 추가

if(khpage == NULL){

freelist.list = khpage_list + page_idx;

freelist.list->nalloc = page_cnt;

freelist.list->next = NULL;

}

else{

while(khpage->next != NULL){

khpage = khpage->next;

}

khpage->next = khpage_list + page_idx;

khpage->next->nalloc = page_cnt;

khpage->next->next = NULL;

}

freelist.nfree++;

}