

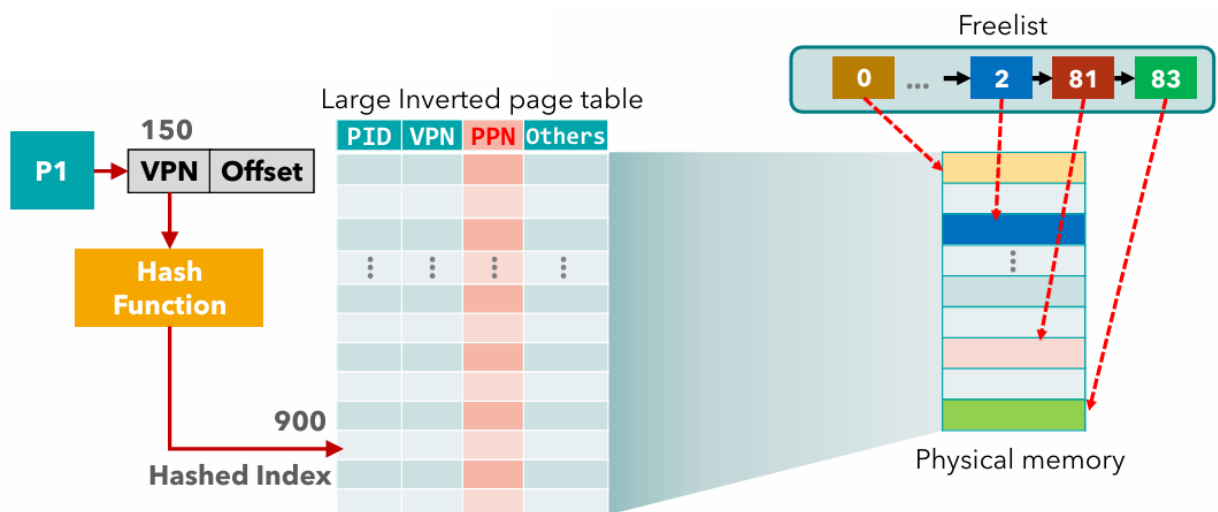
Contents Table

Description of implementation

1. hash()
2. kalloc()
3. kfree()
4. ittraverse()
5. deallocvm()

Result Analysis

Description of implementation



1. hash()

```
uint hashed = (pid+1)*vpn;
return hashed % MAXENTRY;
```

hash 함수는 단순히 구성하였다. 해당 hash function을 사용하면 collision으로 인한 속도 저하가 크지 않은 것을 확인할 수 있다. 만약 collision이 과도하게 발생한다면, 이로 인해 운영체제의 속도가 매우 느려질 수 있다.

2. kalloc()

```
if ((uint)v != -1) {
    initial_idx = hash(pid, (uint)v>>12) % MAXENTRY;
    idx = initial_idx;
}
```

```

else {
    initial_idx = hash(pid, (uint)r>>12) % MAXENTRY;
    idx = initial_idx;
}

```

먼저 hash 함수를 통해서 index를 찾는다. 이 때 kalloc()의 파라미터는 virtual address이므로 shift 연산을 통해 구한 VPN 값을 hash 함수의 인자로 넣어준다.

```

do {
    if ((uint)v != -1) {
        if ((PID[idx] == -1) || ((PID[idx] == pid) && (VPN[idx] ==
((uint)v>>12)))) {
            break;
        }
    }
    else {
        if ((PID[idx] == -1) || ((PID[idx] == 0) && (VPN[idx] ==
((uint)r>>12)))) {
            break;
        }
    }
    idx = (idx + 1) % MAXENTRY;
} while (idx != initial_idx);

```

do-while 문을 통해서 hash collision이 일어났을 때 linear probing을 진행한다. 만약 비어있는 index를 찾거나, 기존에 위치해 있던 index를 찾으면 loop를 탈출한다.

```

if (idx == initial_idx && (PID[idx] == pid || PID[idx] == -1)) {
}
else if (idx == initial_idx) {
    return 0;
} else {}

```

만약 index를 찾지 못했다면, 0을 return 해준다.

```

if ((uint)v == -1) {
    VPN[idx] = (uint)r>>12;
} else {
    VPN[idx] = (uint)v>>12;
}

PID[idx] = pid;
PPN[idx] = (uint)V2P(r);

```

이후 찾은 index를 바탕으로 해당 inverted page table entry의 VPN과 PPN, 그리고 PID를 업데이트 해준다. 이 때 전달받은 virtual address가 -1이라면, kernel에 의해 호출된 것 이므로 physical address를 바탕으로 virtual address를 결정해 준다. 위 코드에서는 free list에서 가져온(이 부분은 기존 스켈레톤 코드에 작성 되어있던 부분이었기 때문에 보고서에서는 생략하였다) r(P2V, 즉 r은 이미 physical address

가 변환된 상태이다)을 바탕으로 VPN을 초기화 하였다. virtual address가 -1이 아닌 경우에는 virtual address를 바탕으로 VPN을 초기화 하였다. 이 외에 PPN은 r을 physical address로 바꾸어서, PID는 전달받은 pid 값으로 초기화 시켰다.

3. kfree()

```
do {
    if ((PID[idx] == pid) && (VPN[idx] == ((uint)v>>12))) {
        break;
    }
    idx = (idx + 1) % MAXENTRY;
} while (idx != initial_idx);
```

kfree()에서는 pid와 virtual address를 바탕으로 inverted page table에서 index를 찾아야 한다. kalloc()에서와 비슷하게 do-while문을 이용해서 탐색한다.

```
kv = (uint)P2V(pa);

if (kv > 2415919104) {
    return;
}

memset((char*)kv, 1, PGSIZE); // 메모리 초기화
```

kfree()에서는 memset()을 위해서 physical address를 kernel virtual address(kv)로 변환해 주어야 한다. 이후 kv를 바탕으로 memset을 진행하면 된다. 이 때 kv에 trash value가 들어갈 수 있으므로 memset 전에 예외 처리를 진행해주어야 한다.

```
PID[idx] = -1;
VPN[idx] = 0;
PPN[idx] = 0;
PTE_XV6[idx] = 0;

if (kmem.use_lock && !holding(&kmem.lock))
    acquire(&kmem.lock);
r = (struct run*)kv;
r->next = kmem.freelist;
kmem.freelist = r;
if (kmem.use_lock && holding(&kmem.lock))
    release(&kmem.lock);
```

memset()을 진행한 이후에는 inverted table entry를 초기화 해준다. 이 초기화 값은 main.c를 참고하였다. 이후에는 free list에 free page를 넣으면 된다.

4. ittraverse()

```
idx = searchit(va, pid);
return &PTE_XV6[idx];
```

ittraverse()에서는 searchit() 함수만 추가해 주었다. 이 때 searchit() 함수는 kfree()에서 entry를 찾는 방식과 동일하게 entry를 찾아 인자로 받은 virtual address와 pid가 있는 index를 return 한다.

5. deallocvm()

```
for (uint idx=0; idx<MAXENTRY; idx++) {  
    uint va = VPN[idx]<<12;  
    if ((PID[idx] == pid) && ((newsz <= va) && (va < oldsz))) {  
        kfree(pid, (char*)(va));  
    }  
}  
  
return newsz;
```

deallocvm() 함수는 위와 같이 구성하였다. Inverted page table을 한 번 돌면서 범위 내에 있는 virtual address를 지워준다. 위와 같이 구성하면 inverted page table을 ‘한 번’만 돌아도 범위 내에 있는 virtual address를 모두 지워줄 수 있기 때문에 속도 향상에 효과적이다.

Result Analysis

usertests를 실행한 결과는 다음과 같다. 실행이 완료되면 ALL TESTS PASSED라는 문구가 뜬다.

```
❯ jangjiwon@DESKTOP-M4HGQUK: ~/OS/xv6
Bootng from Hard Disk,,xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
exec [/init]: 1
init: starting sh
exec [sh]: 2
$ usertests
exec [usertests]: 3
usertests starting
arg test passed
createdelete test
createdelete ok
linkunlink test
linkunlink ok
concreate test
concreate ok
fourfiles test
fourfiles ok
sharedfd test
sharedfd ok
bigwrite test
bigwrite ok
bss test
bss test ok
sbrk test
No free memory
allocuvm out of memory
No free memory
allocuvm out of memory
No free memory
allocuvm out of memory
No free memory
allocuvm out of memory
sbrk test OK
validate test
validate ok
open test
open test ok
small file test
creat small succeeded; ok
writes ok
open small succeeded ok
read succeeded ok
small file test ok
big files test
big files ok
many creates, followed by unlink test
many creates, followed by unlink; ok
openinput test
openinput test ok
exitinput test
exitinput test ok
lput test
lput test ok
pipe1 ok
preempt: kill... wait... preempt ok
exitwait ok
rmdot test
rmdot ok
fourteen test
fourteen ok
bigfile test
bigfile test ok
subdir test
subdir ok
linktest
linktest ok
unlinkread test
unlinkread ok
dir vs file
dir vs file OK
empty file name
empty file name OK
fork test
fork test OK
uio test
pid 487 usertests: trap 13 err 0 on cpu 0 eip 0x35f7 addr 0xcf9c--kill proc
uio test done
exec test
exec [echo]: 3
ALL TESTS PASSED
$
```