운영체제 Project 2

202111057 문서흔

I. 구현 (수정 및 추가)

1. kalloc.c

① uint hash

```
75 uint hash(int pid, char *vpn) {
76   //TODO: Implement your own hash function
77   uint hash = pid;
78   hash ^= ((uint)vpn >> 3) * 31;
79   hash = ((hash >> 16) ^ hash) * 0x45d9f3b;
80   hash = (hash >> 16) ^ hash % NPID;
81   return hash;
82 }
```

해시 충돌을 최소화하기 위해 PID와 VPN을 조합하고 XOR 연산 및 시프트 연산을 통해 해시 값 계산. 해시 함수는 NPID 값을 사용하여 전체 해시 테이블 크기 내에서 해시 값을 유지하도록 함.

- hash ^= ((uint)vpn >> 3) * 31;: VPN의 일부를 PID와 XOR하여 해시 값 변형
- hash = ((hash >> 16) ^ hash) * 0x45d9f3b;: 추가적인 시프트 연산과 XOR 연산.
- hash = (hash >> 16) ^ hash % NPID;: 최종적으로 NPID 값에 대해 modular 연산-> 전체 해시 테이블 크기 내에 해시 값 유지

② void kfree

```
84  void kfree(int pid, char *v){ // v is virtual address
85
86    uint kv, idx;
87    struct run *r;
89    //1. Find the corresponding physical address for given pid and VA
90    //To do 1), find the corresponding entry of inverted page table and read the PPN[idx]
91    idx = hash(pid, v); // (added)
92    while (PID[idx] != pid || VPN[idx] != (uint)v) {
93      idx = (idx + 1) % 40000; // Linear probing to handle collisions // (added)
94      if (idx == hash(pid, v)) { // Infinite loop check //debugging
95        cprintf("Error: Infinite loop in kfree due to collision.\n");
96        return;
97      }
98    }
100   //2. Initialize the PID[idx], VPN[idx], PPN[idx] and PTE_XV6[idx]
101   PID[idx] = -1; // (added)
102   VPN[idx] = 0; // (added)
103   PPN[idx] = 0; // (added)
104   PTE_XV6[idx] = 0; // (added)
105   //3. For memset(), convert the physical address for free to kernel's virtual address by
106   memset(P2V(kv), 1, PGSIZE); // (added)
107   //4. Insert the free page into kmem.freelist
108   // memset(kv, 1, PGSIZE); //TODO: You must perform memset for P2V(physical address);
109   if(kmem.use_lock)
110     acquire(&kmem.lock);
111   r = (struct run*)P2V(kv); // (modified)
112   r->next = kmem.freelist;
113   kmem.freelist = r;
114   if(kmem.use_lock)
115     release(&kmem.lock);
116
117   cprintf("kfree: Freed memory for PID %d, VA %x\n", pid, v); //debugging
118 }
```

메모리 페이지를 해제. 주어진 PID와 가상 주소를 통해 inverted page table에서 해당 entry를 찾고, 물리 메모리 페이지를 해제하고 freelist에 추가. 이를 위해 hash 함수를

사용하여 페이지를 검색하며, 충돌 발생 시에는 linear probing을 사용. 무한 루프를 방지하기 위해 원래의 해시 값으로 되돌아오는 경우 에러를 출력하도록 함.

- idx = hash(pid, v);: 주어진 PID와 가상 주소를 이용해 해시 값을 계산하여 inverted page table에서 검색 시작 위치를 결정한다.
- while (PID[idx] != pid || VPN[idx] != (uint)v): PID와 VPN이 일치하는 entry를 찾을 때까지 linear probing을 수행한다. 이는 충돌이 발생할 경우에도 올바른 entry를 찾기 위한 과정이다.
- PID[idx] = -1; VPN[idx] = 0; PPN[idx] = 0; PTE_XV6[idx] = 0;: 페이지가 해제될 때 해당 entry를 초기화하여 이후 재사용 가능하도록 한다.
- memset(P2V(kv), 1, PGSIZE);: 해제된 페이지의 내용을 초기화하여 이전 데이터가 남아 있는 것을 방지한다.

③ char* kalloc

```
201   while (PID[idx] != -1) {
202     cprintf("Hash collision for idx %d: PID %
203     idx = (idx + 1) % 40000; // Linear probin
204   }
205
206   if (v == (char*)-1) { // Kernel caller case
207     v = P2V(V2P(r)); }
208
209   PID[idx] = pid; // (added)
210   VPN[idx] = (uint)v; // (added)
211   PPN[idx] = V2P(r); // (added)
```

새로운 페이지를 할당. 메모리의 freelist에서 사용 가능한 페이지를 가져오며, inverted page table에서 해시 값을 이용해 적절한 위치에 페이지 정보를 기록. 할당 과정에서 hash 함수를 통해 PID와 가상 주소를 해시하여 해당 페이지를 저장할 위치를 찾으며, 이미 사용 중인 위치라면 linear probing을 통해 빈 entry를 찾는다.

- while (PID[idx] != -1): 해당 위치가 이미 사용 중이라면, linear probing을 통해 empty entry를 찾는다.
- PID[idx] = pid; VPN[idx] = (uint)v; PPN[idx] = V2P(r);: 페이지가 할당된 후 inverted page table에 해당 페이지의 PID, VPN, PPN 정보를 기록

2. vm.c

① static pte_t * ittraverse

```
56   if ((uint)va >= KERNBASE) {
57     return &PTE_KERN[(uint)V2P(va)/PGSIZE];
58   }
59   else{
60   //Implement 3) in here
61     idx = hash(pid, (char *)va); // (added)
62     while (PID[idx] != pid || VPN[idx] != (uint)va) { // (added)
63       idx = (idx + 1) % 40000;} //(added)
```

ittraverse 함수는 PTE 리턴. 이 함수는 커널 주소와 사용자 주소를 구분하여 처리하며, 커널 주소의 경우 PTE_KERN을 사용하고, 사용자 주소의 경우 hash 함수를 이용해 inverted page table에서 해당 entry를 찾는다. 충돌이 발생할 경우 linear probing을 사용하여 올바른 entry를 찾도록 구현.

- if ((uint)va >= KERNBASE): 가상 주소가 커널 주소 공간에 있는지 확인하여, 커널 PTE 리턴
- idx = hash(pid, (char *)va);: 사용자 주소의 경우 hash 함수를 통해 inverted page

table에서 해당 entry 찾는다

- while (PID[idx] != pid || VPN[idx] != (uint)va): PID와 VPN이 일치하는 entry를 찾을 때까지 linear probing을 수행하여 정확한 PTE 리턴

② deallocuvm 함수 수정

```
56  if ((uint)va >= KERNBASE) {
57    return &PTE_KERN[(uint)V2P(va)/PGSIZE];
58  }
59  else{
60  //Implement 3) in here
61    idx = hash(pid, (char *)va); // (added)
62    while (PID[idx] != pid || VPN[idx] != (uint)va) { // (added)
63      idx = (idx + 1) % 40000;} //(added)
64    return &PTE_XV6[idx];
```

가상 메모리 영역을 해제. 기존의 PTE를 탐색하여, 해당 가상 주소가 페이지로 할당되어 있다면 kfree 함수를 호출하여 해당 페이지 해제.

- pte_t *pte = ittraverse(myproc()->pid, pgdir, v, 0);: 주어진 가상 주소에 해당하는 PTE를 찾기 위해 ittraverse 함수 호출
- if(pte && (*pte & PTE_P)): 페이지가 실제로 할당되어 있는지를 확인하고, 할당된 경우 kfree 함수를 통해 메모리 해제

II. 결과

1. Hash collision

```
Hash collision for idx 15759: PID 0, VA 8dffe000

Hash collision for idx 15759: PID 0, VA 8dffe000
Hash collision for idx 15760: PID 0, VA 8dffd000

Hash collision for idx 15759: PID 0, VA 8dffe000
Hash collision for idx 15760: PID 0, VA 8dffd000
Hash collision for idx 15761: PID 0, VA 8dffc000

Hash collision for idx 15759: PID 0, VA 8dffe000
Hash collision for idx 15760: PID 0, VA 8dffd000
Hash collision for idx 15761: PID 0, VA 8dffc000
Hash collision for idx 15762: PID 0, VA 8dffb000

Hash collision for idx 15759: PID 0, VA 8dffe000
Hash collision for idx 15760: PID 0, VA 8dffd000
Hash collision for idx 15761: PID 0, VA 8dffc000
Hash collision for idx 15762: PID 0, VA 8dffb000
Hash collision for idx 15763: PID 0, VA 8dffa000

Hash collision for idx 15759: PID 0, VA 8dffe000
Hash collision for idx 15760: PID 0, VA 8dffd000
Hash collision for idx 15761: PID 0, VA 8dffc000
```

메모리 할당 및 해제 과정에서 해시 충돌이 빈번하게 발생.특히 idx 15759부터 시작하는 여러 PID와 VA에 대해 지속적으로 충돌이 발생하였다. Linear probing을 통해 해시 충돌을 줄이고자 하였지만, hash function의 성능 개선이 필요해 보였다.

2. Usertests

```
seoheun@DESKTOP-SEOHEUN:~/os-pj2/xv6$ make qemu-nox>usertests -j
boot block is 451 bytes (max 510)
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0359973 s, 142 MB/s
1+0 records in
1+0 records out
512 bytes copied, 0.00015642 s, 3.3 MB/s
419+1 records in
419+1 records out
214688 bytes (215 kB, 210 KiB) copied, 0.000800361 s, 268 MB/s
```