Project 2 Report

추성재 (201911185)

1. 구현 방법

Inverted page table을 구현하기 위해 다음 함수들을 구현했다.

a. kalloc.c

searchidx()	virtual address와 pid를 이용해 inverted page table 내부에서 index를 찾는 함수이다. 1. 주어진 virtual address va를 이용해 vpn (va >> 12)를 구한다. 2. 구한 vpn을 단순한 hash function인 modulo hash function에 넣어 idx를 구한다. 3. 이후 idx를 시작으로 idx를 1씩 증가시켜가며 PID[idx]와 VPN[idx]의 값 중 pid와 vpn 모두와 동일한 값이 있는지 찾는다. 4. Pid와 vpn 모두 동일한 idx가 있다면, 이를 return한다. 5. 만약 동일한 idx가 없으면, searchidx_alloc()을 불러 새로운 idx를 할당한다. 3.번 과정이 hash collision을 handling하는 linear probing 과정이다.
searchidx_alloc()	virtual address와 pid를 이용해, inverted page table 내부에서 빈 공간을 찾아 해당 공간의 idx를 반환한다. 1. Searchidx()의 1에서 2번과 동일한 과정을 거친다 2. PID[idx]가 -1인 idx를 찾을 때까지 idx를 1씩 증가시킨다 3. 찾은 idx를 return한다 2.번 과정이 hash collision을 handling하는 linear probing 과정이다.
searchidx_kern()	virtual address가 주어지지 않는 상황(kalloc()에 v == -1 인 경우)에, inverted page table의 공간 중 아무 빈 공간의 idx를 반환하는 함수이다.
kfree()	넘겨받은 pid와 virtual address v를 이용해 inverted page table의 index를 searchidx를 이용해 찾고, PID[index], VPN[index], PTE_XV6[index]를 초기화 한다. 이후 index를 physical address로 바꾼 이후 memset()을 수행한다.
kalloc()	virtual address와 pid를 받은 후 inverted page table 속 page를 하나 할당하는 함수이다. 1. v 가 -1이 아니면, searchidx_alloc()을 이용해 index를 찾는다. v 가 -1이라면, pid를 0으로 만든 후, searchidx_kern()을 이용해 idx를 찾는다. 2. 구한 idx를 이용해 PID[idx], VPN[idx]를 설정한다. 이때, v = -1인 경우에는 VPN[idx]에 KERNBASE의 VPN을 넣는다. 3. 이후 idx를 physical address로 변환한 다음, P2V(physical address)를 반환한다.

b. vm.c

ittraverse()	주어진 virtual address와 pid를 바탕으로 pte의 주소를 반환한다. virtual address가 KERNBASE보다 크면 PTE_KERN[V2P(va)/PGSIZE]를 반환한다. virtual address가 user 영역의 주소라면, searchidx()를 이용해 idx를 구한 다음, PTE_XV6[idx]를 반환한다.
__virt_to_phys()	주어진 pid,와 virtual address를 ittraverse()에 넣은 후 pte를 구한다. 이후 pte 속 physical address에서 flag들을 제거한 후, virtual address offset을 더해 physical address를 구한다.
__get_flags()	주어진 pid와 virtual address를 ittraverse()에 넣은 후 pte를 구한다. 이후 pte 속 physical address에서 PTE_FLAGS macro를 이용해 flag를 구한다.

c. Hash collision 여부 출력

"ls" 명령어를 입력했을 때 PID 0, VA 0x0을 inverted page table에 넣을 시 hash collision과 linear probing 과정을 보여주는 print_hash_collision() 함수를 만들었다.

이 함수는 system call로 구현되었으며, system call 구현 방법은 mini-project 1에서 hostname을 바꾸는 system call 구현과 동일하게 구현했다.

Print_hash_collision() 함수에서 system call을 불렀을 때 호출되는 __print_hash_collision() 함수가 proc.c에 구현되어 있다. 이 함수는 searchidx_alloc()에 pid 0과 VA 0x0을 대입한다. 이때, searchidx_alloc()에는 print 옵션이 있어서, linear probing 과정을 과제 요구사항과 동일한 형식으로 출력한다.

2. usertest 실행

컴파일 후 부팅은 성공적으로 이루어졌다. 이후 usertest를 실행했다. Figure 1과 Figure 2는 usertest의 결과를 보여주는 이미지이다.

Usertest 중 sbrktest()를 수행하는 과정에서 kernel panic이 일어났고, 시스템을 강제 종료한 이후 다시 부팅한 후 ls를 실행했을 때, fs.img가 손상된 것처럼 이상하게 보였다. 이후 usertest.c에서 sbrktest()를 제외한 후 나머지 test를 실행했을 때는 모든 test들을 통과했다.

Sbrktest()가 실패했다는 것은 현재 inverted page table에서는 process의 메모리 크기를 동적으로 할당하는데 문제가 있다는 것을 의미한다. 또한, sbrktest() 수행 과정 중 어느 과정도 수행하지 못하고 kernel panic이 일어났다는 것은 growproc() 함수가 제대로 실행되지 못한다는 것을 의미하나, 부팅이 정상적으로 이루어지는 것으로 보아, allocuvm()과 deallocuvm()은 성공적으로 동작하는 것으로 보인다. 현재로서는 searchidx() 과정에서 어떠한 index도 찾지 못했을 때 searchidx_alloc()을 불러서 새로운 index를 할당하는 과정이 문제인 것으로 보인다. 해당 과정이 논리적으로 보이지는 않지만, 여러 번의 경험 끝에 해당 과정을 넣어야만 시스템이 부팅했기 때문에 현재로서는 sbrktest()를 통과하지 못하는 이유를 찾을 수 없다.

3. memtest 실행

memtest는 "ls" 명령어를 실행했을 때 1-c에서 설명한 print_hash_collision()을 같이 실행해, 현재 inverted page table에 PID 0, VA 0x0을 넣을 때 hash collision 이유와, linear probing으로 새로운 idx를 찾는 과정을 출력하는 방식으로 수행했다. 결과는 Figure 3, 4, 5에 나와있다.

Figure 3에서는 ls를 실행했을 때 print_hash_collision()이 불리는 모습과, VPN이 다른 경우에 linear probing이 일어나는 모습을 보여준다. Figure 4에서는 PID가 다른 경우에 linear probing이 일어나는 모습을 보여준다. Figure 5에서는 linear probing 과정 끝에 최종적으로 비어있는 index를 찾은 모습을 볼 수 있다.

memtest 결과를 통해 현재 구현된 inverted page table은 hash collision이 일어났을 때 이를 linear probing을 이용해 대처할 수 있다는 것을 알 수 있다.

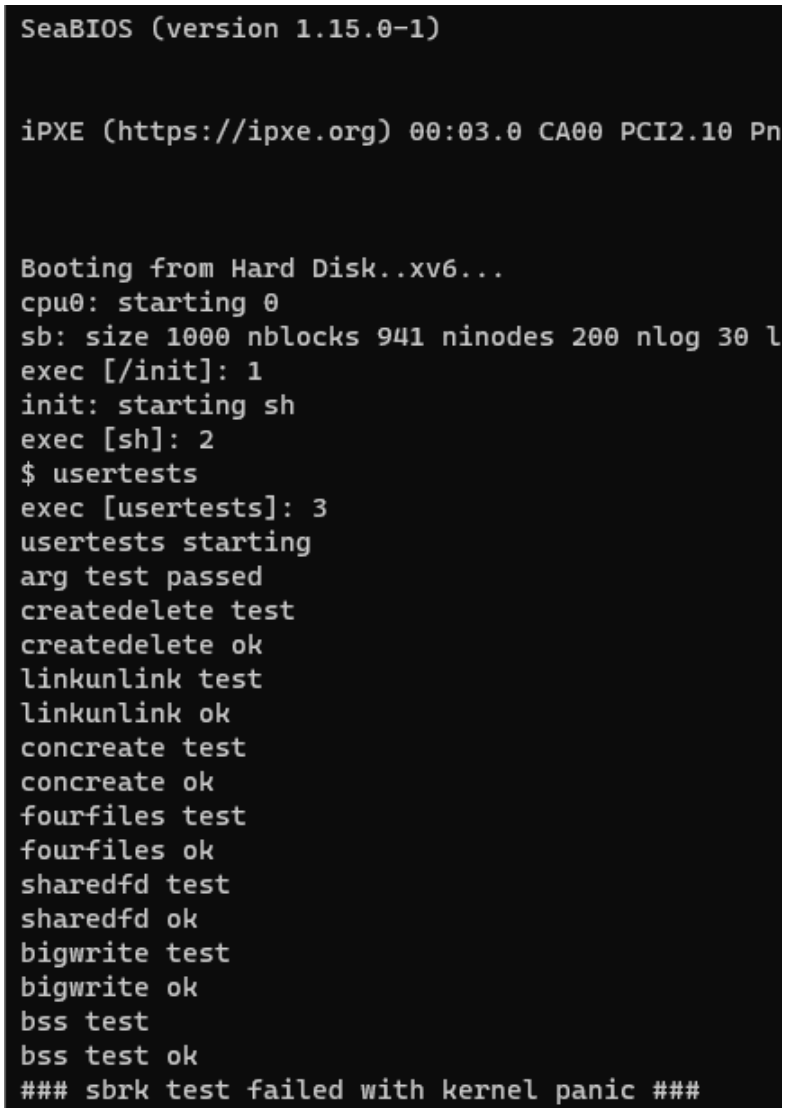

Figure 1 : usertest 1

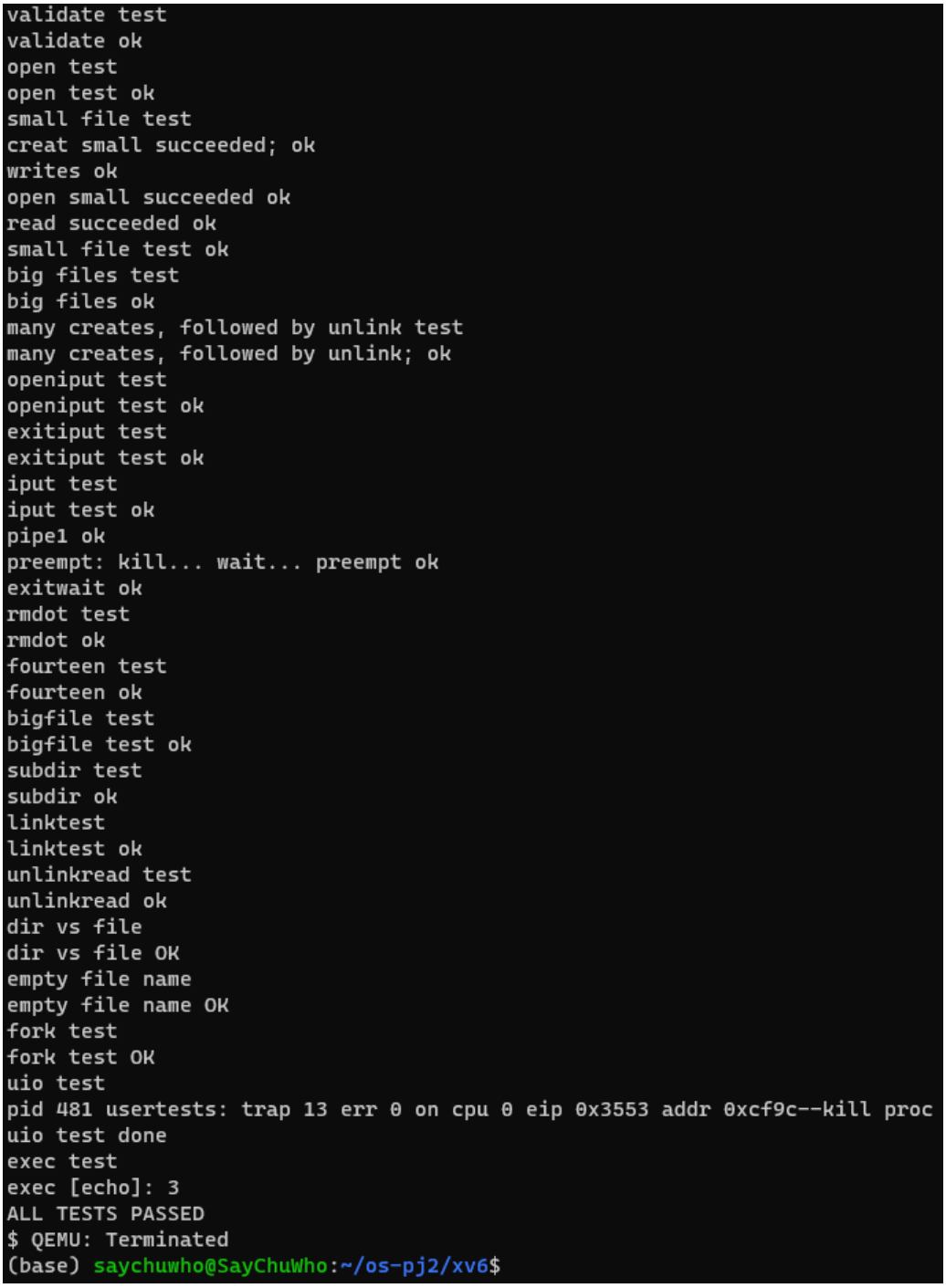

Figure 2 : usertest 2

```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting 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
$ ls
exec [ls]: 3
[Hash collision for idx: 501]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 502]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 503]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 504]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 505]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 506]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 507]  PID: 0, VA: 0x0, VA is different
```

Figure 3 : memtest 1

```
[Hash collision for idx: 696]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 697]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 698]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 699]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 700]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 701]  PID: 0, VA: 0x0, PID is different
[Hash collision for idx: 702]  PID: 0, VA: 0x0, PID is different
[Hash collision for idx: 703]  PID: 0, VA: 0x0, PID is different
[Hash collision for idx: 704]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 705]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 706]  PID: 0, VA: 0x0, VA is different
```

Figure 4 : memtest 2

```
[Hash collision for idx: 902]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 903]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 904]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 905]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 906]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 907]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 908]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 909]  PID: 0, VA: 0x0, VA is different
[Hash collision for idx: 910]  PID: 0, VA: 0x0, VA is different
[Completion idx: 911] PID: 0, VA: 0x0
```

Figure 5 : memtest 3