WIKI for OS project04

컴퓨터소프트웨어학부

2022085069 손주은

Design

1. Common: 페이지 참조 횟수를 추적할 수 있는 데이터 구조를 추가한다.

- 참조 횟수 관리 배열

- free page 개수 관리 변수

2. Initial sharing: fork한 후에 페이지를 복사하는 게 아니라 공유되도록 한다.

- 복사가 이루어지는 부분을 찾아서 공유하도록 수정한다.

- 참조 횟수++;

3. Make a copy: write가 발생했을 때, 페이지 복사를 진행한다.

Handler step)

- 새로운 페이지를 할당한다. (페이지 참조 횟수 = 1)

- 페이지를 복사한다.

- 기존의 페이지 참조 횟수--;

Implementation

$ vim kalloc.c

① struct kmem

```
20 struct {
21   struct spinlock lock;
22   int use_lock;
23   struct run *freelist;
24   int fp;
25   uint refc[PHYSTOP >> PGSHIFT]; // 참조 횟수 관리 배열
26 } kmem;
```

- int fp: # of free pages

- uint refc[PHYSTOP >> PGSHIFT]: 참조 횟수 관리 배열

  → PHYSTOP: virtual address space의 크기, PGSHIFT: 12

  → virtual address space / PGSIZE(4096 = 2^12) 와 같은 의미

② kalloc()

```
88      acquire(&kmem.lock);
89    r = kmem.freelist;
90    if(r)
91      kmem.freelist = r->next;
92    if(kmem.use_lock)
93      release(&kmem.lock);
94    return (char*)r;
```

[수정 전]

```
142     acquire(&kmem.lock);
143   r = kmem.freelist;
144   if(r){
145       kmem.freelist = r->next;
146       kmem.fp--;
147       kmem.refc[V2P((char*)r) >> PGSHIFT] = 1;
148   }
149   if(kmem.use_lock)
150     release(&kmem.lock);
151
152   return (char*)r;
```

[수정 후]

- fp를 1만큼 감소시키고, 페이지 참조 횟수를 1로 설정한다.

③ kfree()

```
115   if(kmem.refc[V2P(v) >> PGSHIFT] > 0)
116       kmem.refc[V2P(v) >> PGSHIFT]--;
117       //decr_refc(V2P(v));
118
119   if(kmem.refc[V2P(v) >> PGSHIFT] == 0){
120       memset(v, 1, PGSIZE);
121
```

[수정 후]

- 해당 페이지 참조 횟수가 0보다 크다는 건 그 페이지를 공유하고 있는 프로세스가 1개 이상이라는 뜻이므로, 그냥 참조 횟수를 1감소시킨다.

- 만약 페이지 참조 횟수가 0이라면 더 이상 그 페이지를 가리키는 프로세스가 없다는 뜻이므로 실제로 kfree를 진행해준다.

④ 그 외 함수들

```
29 void incr_refc(uint pa)
30 {
31     acquire(&kmem.lock);
32     kmem.refc[pa>>PGSHIFT]++;
33     release(&kmem.lock);
34
35 }
```

[incr_refc()]

```
37 void decr_refc(uint pa)
38 {
39     acquire(&kmem.lock);
40     if(--kmem.refc[pa>>PGSHIFT] == 0){
41         kfree((char*)P2V(pa));
42     }
43     //if(kmem.refc[pa>>PGSHIFT]>0)
44         //kmem.refc[pa>>PGSHIFT]--;
45
46     release(&kmem.lock);
47
```

[decr_refc()]

```
50 int get_refc(uint pa)
51 {
52     int ref_count;
53     acquire(&kmem.lock);
54     ref_count= kmem.refc[pa>>PGSHIFT];
55
56     release(&kmem.lock);
57     return ref_count;
58 }
```

[get_refc()]

```
60 int countfp(void)
61 {
62     int ret;
63     acquire(&kmem.lock);
64     ret= kmem.fp;
65     release(&kmem.lock);
66     return ret;
67
68 }
```

[countfp()]

$ vim vm.c

① copyuvm()

```
330     pa = PTE_ADDR(*pte);
331     *pte &= ~PTE_W; // read-only 모드로 set
332     flags = PTE_FLAGS(*pte);
333     /*if((mem = kalloc()) == 0)
334       goto bad;
335     memmove(mem, (char*)P2V(pa), PGSIZE);
336     */
337     //*pte &= ~PTE_W; // read-only 모드로 set
338     if(mappages(d, (void*)i, PGSIZE, pa, flags) < 0) {
339       goto bad;
340     }
341     incr_refc(pa);
342
343   }
344
345   lcr3(V2P(pgdir));
346   return d;
```

- 주석 처리한 부분: 기존의 코드로, 실제로 메모리가 할당되고 복사가 이루어지는 코드이다.

- 이 부분을 지움으로써 공유하도록 만들고, 페이지 참조 횟수를 증가시킨다.

- 페이지 테이블 항목을 변경했으므로 TLB를 flush한다.

② CoW_handler()

```
394 void CoW_handler(void)
395 {
396     uint va = rcr2(); //page fault가 발생한 가상주소 저장
```

- va 변수에 page fault가 발생한 가상주소를 저장한다.

```
399     if(va >= KERNBASE || va >= myproc()->sz ){ // 범위 check
400         cprintf("Error: [CoW_handler] Invalid virtual address\n");
401         myproc()->killed=1;
402         return;
```

- va의 범위를 체크하고, 잘못된 범위라면 에러메시지를 출력하고 프로세스를 종료한다.

```
405     // page table entry
406     pte_t *pte = walkpgdir(myproc()->pgdir, (void*) va, 0);
```

- va에 해당하는 page table entry를 가져온다.

```
408     if(*pte & PTE_W){ // 읽기 전용인 경우에만 처리
409
410         panic("page fault already writeable\n");
411     }
```

- 읽기 전용이 아니라면 공유를 시키면 안 되기 때문에 검사한다.

```
413     uint pa = PTE_ADDR(*pte); // 상위 20비트 추출(offset 제외)
414     uint refc= get_refc(pa);
415     char* mem;
```

```
417     if(refc>1){
418             if((mem = kalloc())==0){
419
420                 cprintf("Error: [CoW_handler] Faild to allocate memory\n");
421                 myproc()->killed=1;
422                 return;
423             }
424
425             memmove(mem, (char*)P2V(pa), PGSIZE); // 페이지 복사
426
427             uint flags= PTE_FLAGS(*pte);
428             *pte= V2P(mem) | flags | PTE_W;
429             decr_refc(pa);
430         }
```

- 페이지 참조횟수가 1보다 크다면, 그 페이지와 구분되는 별도의 페이지를 복사해야 한다.

- mem 변수에 메모리를 할당하고, 그 공간에 복사를 해준다.

- 새롭게 할당된 페이지에 대해서 PTE_W 플래그를 추가해준다.

```
432     else if(refc==1) { // 마지막 프로세스인 경우, 그냥 읽기 전용 제한만 풀어줌.
433         *pte |= PTE_W;
434     }
435     else{
436         cprintf("ref: %d\n", refc);
437         panic("pagefault ref_count is wrong\n");
438
439     }
```

- 만약 마지막 남은 프로세스라면 읽기 제한을 풀어준다.

```
441     lcr3(V2P(myproc()->pgdir)); // TLB flush
442
443 }
```

- 페이지 테이블 항목이 변경됐으므로 TLB를 flush해준다.

③ system calls

```
445 int countvp(void)
446 {
447     struct proc *p= myproc();
448     uint sz= p->sz;
449     return (sz >> PGSHIFT); // size of usermemory / PGSIZE = # of pages
450 }
```

- virtual address space 크기를 PGSIZE만큼 나누어서 page 개수를 구하고, return 한다.

```
452 int countpp(void)
453 {
454     struct proc *p= myproc();
455     pde_t *pgdir = p->pgdir; // 현재 프로세스의 페이지 디렉토리
456     uint sz= p->sz; //logical address space의 크기
457     uint i;
458     int count=0;
459 
460     for(i=0; i < sz; i+= PGSIZE){ //현재 프로세스의 페이지 테이블 탐색
461         pte_t *pte = walkpgdir(pgdir, (void *)i, 0);
462         if(pte && (*pte & PTE_P)) count++; // 유효한 물리 주소가 할당된 경우
463     }
464     return count;
465 
466 }
```

- 현재 프로세스의 페이지 테이블을 탐색하고(페이지마다), 유효한 물리 주소가 할당된 페이지 테이블 엔트리의 개수를 반환한다.

```
468 int countptp(void)
469 {
470     struct proc* p= myproc();
471     pde_t *pgdir = p->pgdir;
472     int count =0;
473     //int i,j;
474     int i;
475     //int j;
476     count++; // 페이지 디렉토리 자체는 하나의 페이지를 사용
477 
478     // 페이지 디렉토리 순회
479     for (i=0; i<NPDENTRIES; i++){
480         if(pgdir[i] & PTE_P){ // 페이지 디렉토리 엔트리가 유효한지 확인
481             count++; // 페이지 테이블 자체도 하나의 페이지 사용
482         }
483 
484     }
485     return count;
486 }
```

- 페이지 디렉토리와 페이지 테이블에 할당된 page의 개수 count

$ vim trap.c

```
 80   case T_PGFLT:
 81     CoW_handler();
 82     break;
```

- T_PGFLT exception 처리를 위해 handler를 추가해준다.

$ vim defs.h

```
71 void            incr_refc(uint pa);
72 void            decr_refc(uint pa);
73 int             get_refc(uint pa);
74 int             countfp(void);
```

[kalloc.c]

```
192 void           CoW_handler(void);
193 int            countvp(void);
194 int            countpp(void);
195 int            countptp(void);
```

[vm.c]

- 새롭게 구현한 함수와 시스템콜을 추가해준다.

$ vim sysproc.c

```
94 int
95 sys_countfp()
96 {
97     return countfp();
98 }
99
100 int
101 sys_countvp()
102 {
103     return countvp();
104 }
```

```
106 int
107 sys_countpp()
108 {
109     return countpp();
110 }
111
112 int
113 sys_countptp()
114 {
115     return countptp();
116 }
```

- wrapper function을 작성한다.

$ vim syscall.c

$ vim syscall.h

```
106 extern int sys_countfp(void);
107 extern int sys_countvp(void);
108 extern int sys_countpp(void);
109 extern int sys_countptp(void);
```

```
134 [SYS_countfp] sys_countfp,
135 [SYS_countvp] sys_countvp,
136 [SYS_countpp] sys_countpp,
137 [SYS_countptp] sys_countptp,
```

```
23 #define SYS_countfp 22
24 #define SYS_countvp 23
25 #define SYS_countpp 24
26 #define SYS_countptp 25
```

- sysproc.c에서 작성한 wrapper function을 syscall.c, syscall.h 파일에 각각 등록한다.

$ vim user.h

```
26 int countfp(void);
27 int countvp(void);
28 int countpp(void);
29 int countptp(void);
```

$ vim usys.S

```
32 SYSCALL(countfp)
33 SYSCALL(countvp)
34 SYSCALL(countpp)
35 SYSCALL(countptp)
```

- 새로 작성한 시스템콜을 등록한다.

$ vim Makefile

```
185     _test0\
186     _test1\
187     _test2\
188     _test3\
```

[UPROGS]

```
test0.c test1.c test2.c test3.c\
```

[EXTRA]

Result

```
$ test0
[Test 0] default
ptp: 66 66
[Test 0] pass

$ test1
[Test 1] initial sharing
[Test 1] pass

$ test2
[Test 2] Make a Copy
[Test 2] pass
```

```
$ test3
[Test 3] Make Copies
child [0]'s result: 1
child [1]'s result: 1
child [2]'s result: 1
child [3]'s result: 1
child [4]'s result: 1
child [5]'s result: 1
child [6]'s result: 1
child [7]'s result: 1
child [8]'s result: 1
child [9]'s result: 1
[Test 3] pass
```

Trouble shooting

1. Trap14

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
pid 1 init: trap 14 err 7 on cpu 0 eip 0x70 addr 0x2fcc--kill proc
lapicid 0: panic: init exiting
 80104024 80105cf5 80105a1e 0 0 0 0 0 0
```

- 실행하자마자 trap14 error가 떴다.

- trap.c에 핸들러를 추가하지 않아서 생기는 문제였다.

```
80    case T_PGFLT:
81      CoW_handler();
82      break;
```

- trap.c에 CoW_handler를 추가해주었다.

2. Fail to boot

```
Booting from Hard Disk..
```

- 아마도 kalloc, kfree 내부에서 참조 횟수 변수를 증가시킬 때, 함수를 써서 그런 것 같다.

- kalloc 내부에서 kmem.lock을 한 번 잡는데, incr_refc, decr_refc 내부에서도 kmem.lock을 잡으려고 시도하기 때문에 deadlock이 발생한 것 같다.

```
115   if(kmem.refc[V2P(v) >> PGSHIFT] > 0)
116       kmem.refc[V2P(v) >> PGSHIFT]--;
117       //decr_refc(V2P(v));
```

[kfree()]

```
144   if(r){
145       kmem.freelist = r->next;
146       kmem.fp--;
147       kmem.refc[V2P((char*)r) >> PGSHIFT] = 1;
148   }
```

[kalloc()]

- 함수를 쓰지 않고 직접 접근해서 값을 감소시키거나 1을 할당해주었다.

3. copyuvm()에서의 오류

3-1) 연속으로 test case를 시도하면 처음 테스트만 통과하고 나머지는 실패하는 문제

```
init: starting sh
$ test0
[Test 0] default
ptp: 66 66
[Test 0] pass

$ test1
Error: [CoW_handler] Invalid virtual address
$ test2
Error: [CoW_handler] Invalid virtual address
$ test3
Error: [CoW_handler] Invalid virtual address
$
```

```
init: starting sh
$ test1
[Test 1] initial sharing
[Test 1] pass
```

: test0을 실행한 후에 test1을 시도하면 에러가 떴는데, shell을 시작하고 바로 test1을 돌려보면 pass 되었다.

3-2) make a copy가 제대로 되지 않는 모습

```
init: starting sh
$ test2
[Test 2] Make a Copy
[Test 2] fail
```

```
init: starting sh
$ test3
[Test 3] Make Copies
child [0]'s result: 0
[Test 3] fail

child [1]'s result: 1
child [2]'s result: 1
child [3]'s result: 1
child [4]'s result: 1
child [5]'s result: 1
child [6]'s result: 1
child [7]'s result: 1
child [8]'s result: 1
child [9]'s result: 1
[Test 3] pass
```

: test2는 fail이 떴고, test3는 child[0]만 fail이고 나머지는 또 pass 되었다.

```
331     //*pte &= ~PTE_W; // read-only 모드로 set
332     flags = PTE_FLAGS(*pte);
333     /*if((mem = kalloc()) == 0)
334       goto bad;
335     memmove(mem, (char*)P2V(pa), PGSIZE);
336     */
337     *pte &= ~PTE_W; // read-only 모드로 set
```

[전]

```
331     *pte &= ~PTE_W; // read-only 모드로 set
332     flags = PTE_FLAGS(*pte);
333     /*if((mem = kalloc()) == 0)
334       goto bad;
335     memmove(mem, (char*)P2V(pa), PGSIZE);
336     */
337     //*pte &= ~PTE_W; // read-only 모드로 set
```

[후]

- mappages()에서 flags변수를 사용하기 때문에 flags를 설정하기 전에 read-only 모드로 set 해주어야 한다.

- 수정 전, 후 사진에서 line 331과 line 337번의 주석처리 된 부분이 서로 반대다.

- 이걸 수정했더니 위의 두 가지 문제 상황이 모두 해결되었다.

4. trivial

4-1) kmem 수정

- 처음에는 참조 횟수를 관리하는 별도의 데이터 구조로써 아예 새로운 page_info 구조체를 만들었다.

- 그 구조체 안에는 int type의 ref_count 변수 하나만 존재했다.

- 그러나 참조 횟수를 증감하는 함수에서 locking을 사용해야 해서 또 별도의 lock이 필요했다.

- 또, free page 개수를 return하는 함수도 필요했기 때문에, 그냥 별도의 구조체를 없애고 기존의 keme 구조체 안에 다 집어넣었다.

- 그래서 kmem 안에 참조 횟수 배열과, free page 개수 변수가 추가되었다.

4-2) myproc()->killed =1

```
$ test1
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
Error: [CoW_handler] Invalid virtual address
```

[전]

```
$ test1
Error: [CoW_handler] Invalid virtual address
$
```

[후]

- 이거는 디버깅하는 과정에서 불편을 줄이기 위해 추가했다.

- copyuvm()에서 오류를 발견하기 전에, 디버깅 구문에 걸려서 error message가 계속 출력되었다.

```
399     if(va >= KERNBASE || va >= myproc()->sz ){ // 범위 check
400         cprintf("Error: [CoW_handler] Invalid virtual address\n");
401         myproc()->killed=1;
402         return;
```

- CoW_handler 내부에서 에러 상황에 걸린 경우, 프로세스를 중단시킨다.