Project04

Design

- `Initial Sharing`을 구현하기 위하여 `fork`로 자식프로세스를 생성할 때 주소공간을 복사하지 않고 같은 주소를 가리키게만 해야한다고 생각하였다.
  - 이를 위하여 `vm.c`의 `copyuvm` 함수를 수정해야한다.
- `Make a copy on write operation`을 구현하기 위하여 `page fault`를 발생시키기 위해서 `copyuvm`에서의 공유시에 그 페이지에 대한 쓰기 권한을 없애야 한다고 생각하였다.
  - 또한 `trap.c`에서는 `T_PGFLT`를 잡는 코드가 없으므로 `trap.c`에서 `T_PGFLT`를 잡게 만든 다음 `CoW_handler`로 넘어가야하게 해야한다고 생각하였다.
- 각 `page`에 대한 참조횟수를 저장하는 자료구조가 필요하다고 생각하였다.
  - 이를 `kalloc.c`에 배열로 저장하기로 하였다.
  - 또한 이 참조 횟수에 대한 관리는 `copyuvm`, `kalloc`, `kfree`, `Cow_handler`에서 해줘야한다.
    - 프로세스의 주소공간에 대한 할당 등의 관리는 위 네 개 함수에서만 일어난다.

Implement

1. Initial Sharing

`copyuvm` 수정

`Initial Sharing`을 구현하기 위하여 `xv6`에 기본적으로 구현되어있는 `copyuvm` 함수를 다음과 같이 수정하였다.

```
pde_t*
copyuvm(pde_t *pgdir, uint sz)
{
        pde_t *d;
        pte_t *pte;
        uint pa, i, flags;

        if((d = setupkvm()) == 0) return 0;

        for(i = 0; i < sz; i += PGSIZE) {
                if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
                        panic("copyuvm: pte should exist");
                if(!(*pte & PTE_P))
                        panic("copyuvm: page not present");

                *pte &= ~PTE_W; // Set to read-only for CoW operation
                pa = PTE_ADDR(*pte);
```

```
                flags = PTE_FLAGS(*pte);

                if(mappages(d, (void*)i, PGSIZE, pa, flags) < 0) {
                        goto bad;
                }
                incr_refc(pa);
        }
        lcr3(V2P(myproc()->pgdir));

        return d;

        bad:
        freevm(d);
        lcr3(V2P(myproc()->pgdir));
        return 0;
}
```

본래 `copyuvm` 함수는 `fork`시 호출되어 부모 프로세스의 주소공간을 새로 복사하여 자식 프로세스에게 할당하는 함수이지만, 새로 수정한 함수에서는 새로 복사하지 않고 `page table`에 `mappages`를 통하여 mapping 해주었다.

또한 자식 프로세스가 이미 존재하는 페이지를 가리키므로 `incr_refc` 함수를 호출하여 해당 페이지에 대한 참조 횟수를 증가시켰다.

그리고 `page table`이 변경되었기 때문에 `lcr3(V2P(myproc()->pgdir))`를 통해 마지막에 `TLB flush`를 해주었다.

`int count[]` 추가

```
int count[PHYSTOP/PGSIZE]; // number of references to each page
```

`kalloc.c`에 각 페이지에 대한 참조 횟수를 저장하는 배열인 `count`를 추가하였다.

`incr_refc` 추가

```
void incr_refc(uint pa) {
        acquire(&kmem.lock);
        count[pa/PGSIZE]++;
        release(&kmem.lock);
}
```

`decr_refc` 추가

```
void decr_refc(uint pa) {
        acquire(&kmem.lock);
```

```
        count[pa/PGSIZE]--;
        release(&kmem.lock);
}
```

`get_refc` 추가

```
int get_refc(uint pa) {
        acquire(&kmem.lock);
        int refc = count[pa/PGSIZE];
        release(&kmem.lock);
        return refc;
}
```

`incr_refc`, `decr_refc`, `get_refc`에서 `count`에 대한 경쟁이 일어날 수 있으므로 앞 뒤로 `lock`을 걸어주었다.

`kalloc` 함수 수정

```
char*
kalloc(void)
{
        struct run *r;
        if(kmem.use_lock)
                acquire(&kmem.lock);
        r = kmem.freelist;
        if(r) {
                kmem.freelist = r->next;
                count[V2P((char*)r)/PGSIZE] = 1; // 참조횟수를 1로 설정
        }
        if(kmem.use_lock)
        release(&kmem.lock);
        return (char*)r;
}
```

새로운 페이지를 할당하였을때, 참조 횟수를 1로 설정하는 부분을 추가하였다.

`kfree` 함수 수정

```
void
kfree(char *v)
{
        struct run *r;
        if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
                panic("kfree");
                // Fill with junk to catch dangling refs.
        if(kmem.use_lock)
```

```
                acquire(&kmem.lock);
        int refc = count[V2P(v)/PGSIZE];
        r = (struct run*)v;
        if(refc > 0) {
                count[V2P(v)/PGSIZE]--;
        }
        if(refc == 0) {
                memset(v, 1, PGSIZE);
                r->next = kmem.freelist;
                kmem.freelist = r;
        }
        if(kmem.use_lock)
        release(&kmem.lock);
}
```

만약 `kfree`를 호출하였을때 참조횟수가 0보다 크다면 참조횟수를 감소시키고, 감소시켰을때 참조횟수가 0이라면 `freelist`로 반환하도록 하였다.

2. Make a copy

`T_PGFLT` 추가

```
void
trap(struct trapframe *tf)
{
...
        case T_PGFLT:
                CoW_handler();
                break;
...
}
```

`page fault`가 일어나면 `CoW_handler`로 넘어갈 수 있도록 `trap`에 `T_PGFLT`를 잡는 부분을 추가하였다.

`CoW_handler` 구현

```
void CoW_handler(void) {
        uint va = rcr2();
        pte_t *pte = walkpgdir(myproc()->pgdir, (void *) va, 0);
        if(!pte) {
                cprintf("CoW_handler: pte should exist\n");
                exit();
        }
        uint pa = PTE_ADDR(*pte);
        uint flags = PTE_FLAGS(*pte);
        char *mem;
        int refc = get_refc(pa);
```

```
        if(refc > 1) {
                if((mem = kalloc()) == 0) return;
        memmove(mem, (char*)P2V(pa), PGSIZE);
        *pte = V2P(mem) | flags | PTE_P | PTE_U | PTE_W;
        decr_refc(pa);
        } else if(refc == 1) {
                *pte |= PTE_W;
        }
        lcr3(V2P(myproc()->pgdir));
}
```

`page fault`가 일어났을 때 해당 페이지를 `rcr2()`함수를 통하여 `va`에 저장하고, 해당 주소에 있는 데이터를 새로운 공간에 복사하고 저장한다. 또한 원래 페이지에 대해서 참조 횟수를 1 줄여준다.

만약 page fault가 발생한 페이지가 존재하지 않는다면 에러메세지를 출력하고 종료한다.

만약 현재 프로세스의 참조 횟수가 1임에도 불구하고 `page fault`가 일어났다면, 이는 반드시 쓰기 권한이 없는 테이블에 쓰기를 시도한 것이므로 해당 페이지에 쓰기 권한을 준다.

마지막으로 Page Table이 변경되었으므로 TLB를 flush한다.

Result

테스트를 위하여 다음 4개의 함수를 구현하였다:

`countfp(void)`

```
int
countfp(void) {
        int cnt = 0;
        for (int i = 0; i < PHYSTOP/PGSIZE; i++) {
                if (count[i] == 0) {
                        cnt++;
                }
        }
        return cnt;
}
```

`count[i] == 0`이라면 해당 페이지는 참조되지 않았다는 뜻이므로 해당 `index`의 갯수를 센다.

`countvp(void)`

```
int countvp(void) {
        struct proc *p = myproc();
        int cnt = 0;
        for (uint va = 0; va < p->sz; va += PGSIZE) {
```

```
                if (walkpgdir(p->pgdir, (void *)va, 0)) cnt++;
        }
        return cnt;
}
```

`walkpgdir`가 0을 반환한다면 현재 페이지가 `mapping` 되어있지 않다는 뜻이므로 0이 아닌 page를 모두 센다.

`countpp(void)`

```
int countpp(void) {
        struct proc *p = myproc();
        pde_t *pgdir = p->pgdir;
        int cnt = 0;
        pte_t *pte;
        for (uint va = 0; va < p->sz; va += PGSIZE) {
                if ((pte = walkpgdir(pgdir, (void *)va, 0)) && (*pte & PTE_P))
cnt++;
        }
        return cnt;
}
```

`countvp`와 같지만 `pte`가 `present`한 경우에만 `cnt++`를 한다.

`countptp(void)`

```
int countptp(void) {
        struct proc *p = myproc();
        int cnt = 1;
        pde_t *pgdir = p->pgdir;
        for (int i = 0; i < NPTENTRIES; i++) {
        if (pgdir[i] & PTE_P) cnt++;
        }
        return cnt;
}
```

페이지 디렉토리의 각 `index`에 대해서 `present` 한 것의 갯수를 세서 반환하도록 만들었다.

테스트 결과

```
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
init: starting sh
$ 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

$
```

문제 없이 잘 작동하는 것을 볼 수 있다.

Trouble Shooting

- 처음 `copyuvm`함수를 수정할 때 `TLB flush`를 해주지 않아서 계속 `trap 6`번이 떴다.
  - 마지막에 `TLB flush` 한 줄 추가하니까 잘 돌아가게 되었다.