copy on write 과제

20181259 조수민

개발환경

2021 m1 맥북 프로 MacOs Monteray

i686-elf-gcc 사용

copy on write 구현 설명

1. 자식프로세스가 생성될 때 별도의 페이지 테이블을 할당하지 않고 부모 프로세스의 페이지 테이블을 그대로 참조하도록 한다.
2. 자식 프로세스를 read-only로 생성한다. 그러면 자식 프로세스가 변경을 시도할 때, pagefault가 발생한다.
3. 자식 프로세스의 수정으로 인해 pagefault가 발생했다면, 새로운 페이지(물리 메모리)를 참조하도록 변경한다.
4. 같은 페이지 테이블 주소를 사용하는 자식 프로세스의 존재 여부는 pgrefcount를 통해 알 수 있다. ref_count > 1이라면 같은 주소를 참조하는 자식 프로세스가 존재하는 것이다.

kalloc.c

```
uint num_free_pages; // number of free pages
uint pgrefcount[PHYSTOP >> PGSHIFT]; // page reference count

//레퍼런스 count를 반환해주는 함수
uint get_refcount(uint p) {
  if(p >= PHYSTOP || p < (uint)V2P(end))
    panic("get_refcount");

  uint refcount = pgrefcount[p >> PGSHIFT];
  return refcount;
}
```

```
//레퍼런스 count를 더해주는 함수
void inc_refcount(uint p) {
  if(p >= PHYSTOP || p < (uint)V2P(end))
    panic("inc_refcount");

  ++pgrefcount[p >> PGSHIFT];
}

//레퍼런스 count를 빼주는 함수
void dec_refcount(uint p){
  if(p >= PHYSTOP || p < (uint)V2P(end))
    panic("dec_refcount");

  --pgrefcount[p >> PGSHIFT];
}

//free_pages를 반환해주는 시스템콜
uint getNumFreePages(void) {
 if(kmem.use_lock)
    acquire(&kmem.lock);
  uint free_pages = num_free_pages;
  if(kmem.use_lock)
    release(&kmem.lock);
  return free_pages;
}


void
freerange(void *vstart, void *vend)
{
  char *p;
  p = (char*)PGROUNDUP((uint)vstart);
  for(; p + PGSIZE <= (char*)vend; p += PGSIZE){
    //페이지 레퍼런스 카운트를 0으로 초기화 해준다
    pgrefcount[V2P(p) >> PGSHIFT] = 0;          // initialse the reference count to 0
    kfree(p);
  }
}

void
kinit1(void *vstart, void *vend)
{
  initlock(&kmem.lock, "kmem");
  kmem.use_lock = 0;
  //free_pages를 0으로 초기화 해준다
  num_free_pages = 0;
  freerange(vstart, vend);
}

void
kfree(char *v)
{
  struct run *r;

  if((uint)v % PGSIZE || v < end || V2P(v) >= PHYSTOP)
```

```
    panic("kfree");

  if(kmem.use_lock)
    acquire(&kmem.lock);

  r = (struct run*)v;

  //레퍼런스 count가 1 이상일 때 레퍼런스 카운트를 감소 시킨다
  if(get_refcount(V2P(v)) > 0)
    --pgrefcount[V2P(v) >> PGSHIFT];

  //레퍼런스 count가 0일 때, 즉 페이지를 참조하는 프로세스가 없을 때
  if(get_refcount(V2P(v)) == 0){
    memset(v, 1, PGSIZE);
    //해당 주소값을 참조하는 프로세스가 없으므로 free한 페이지라는 뜻
    //그러므로 free_pages를 ++해준다
    ++num_free_pages;
    r->next = kmem.freelist;
    kmem.freelist = r;

  }
  if(kmem.use_lock)
    release(&kmem.lock);
}

char*
kalloc(void)
{
  struct run *r;

  if(kmem.use_lock)
    acquire(&kmem.lock);
  r = kmem.freelist;
  if(r){
    //새로운 프로세스에 페이지를 할당해주기 때문에 free_pages를 -- 해준다
    --num_free_pages;
    kmem.freelist = r->next;
    //페이지를 참조하는 프로세스가 생성됐으므로 pgrefcount를 1로 초기화
    pgrefcount[V2P((char*)r) >> PGSHIFT] = 1;
  }
  if(kmem.use_lock)
    release(&kmem.lock);
  return (char*)r;
}
```

uint num_free_pages - free pages의 수를 저장하는 변수
uint pgrefcount[PHYSTOP >> PGSHIFT] - 주소값에 대응하는 페이지를 참조하는 프로세스의 수를 저장하는 변수

getNumFreePages(void) - free_pages를 반환해주는 시스템콜

inc_refcount(uint p) - pgrecount를 ++해주는 함수

dec_refcount(uint p) - pgrecount를 —해주는 함수

get_refcount(uint p) - pgrecount를 반환해주는 함수

vm.c

```
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");

    // read-only로 만듬
    *pte &= ~PTE_W;
    pa = PTE_ADDR(*pte);
    flags = PTE_FLAGS(*pte);

    //새로운 메모리를 할당해주는 부분이므로 주석 처리
    // if((mem = kalloc()) == 0)
      // goto bad;
    // memmove(mem, (char*)P2V(pa), PGSIZE);

    if(mappages(d, (void*)i, PGSIZE, pa, flags) < 0)
      goto bad;

    //페이지를 참조하는 프로세스가 생성됐으므로 pgrefcount를 증가
    inc_refcount(pa);
  }
  lcr3(V2P(pgdir));
  return d;

bad:
  freevm(d);
  lcr3(V2P(pgdir));
  return 0;
}

// read-only인 프로세스가 변경을 시도해서 T_PGFLT가 발생했을 때
// 이를 처리해주는 pagefault 핸들러
void pagefault()
{
```

```
    uint va = rcr2();
    if(va < 0){
      panic("pagefault: rcr2 < 0");
      return;
    }
    //현재 프로세스의 페이지 테이블 엔트리 주소를 받아온다
    pte_t *pte = walkpgdir(myproc()->pgdir, (void*)va, 0);

    //페이지 테이블 엔트리를 이용해 물리 주소를 저장한다
    uint pa = PTE_ADDR(*pte);
    //pgrefcount를 저장한다
    uint refCount = get_refcount(pa);
    char *mem;

    //refcount가 1보다 클 때, 즉 부모 프로세스가 아닌 다른 프로세스가 페이지를 참조하고 있는 경우
    if(refCount > 1) {
      if((mem = kalloc()) == 0) {
        panic("allocate failed\n");
        return;
      }
      //mem에 공간을 할당받고, memmove를 통해 데이터를 옮긴다
      memmove(mem, (char *)P2V(pa), PGSIZE);
      //페이지 테이블이 가리키는 물리 주소값을 교체한다
      *pte = V2P(mem) | PTE_P | PTE_U | PTE_W;

      //새로운 페이지를 할당했으므로 refcount를 감소시킨다
      dec_refcount(pa);
    }
    else if(refCount == 1)
      *pte |= PTE_W;

    lcr3(V2P(myproc()->pgdir));
}
```

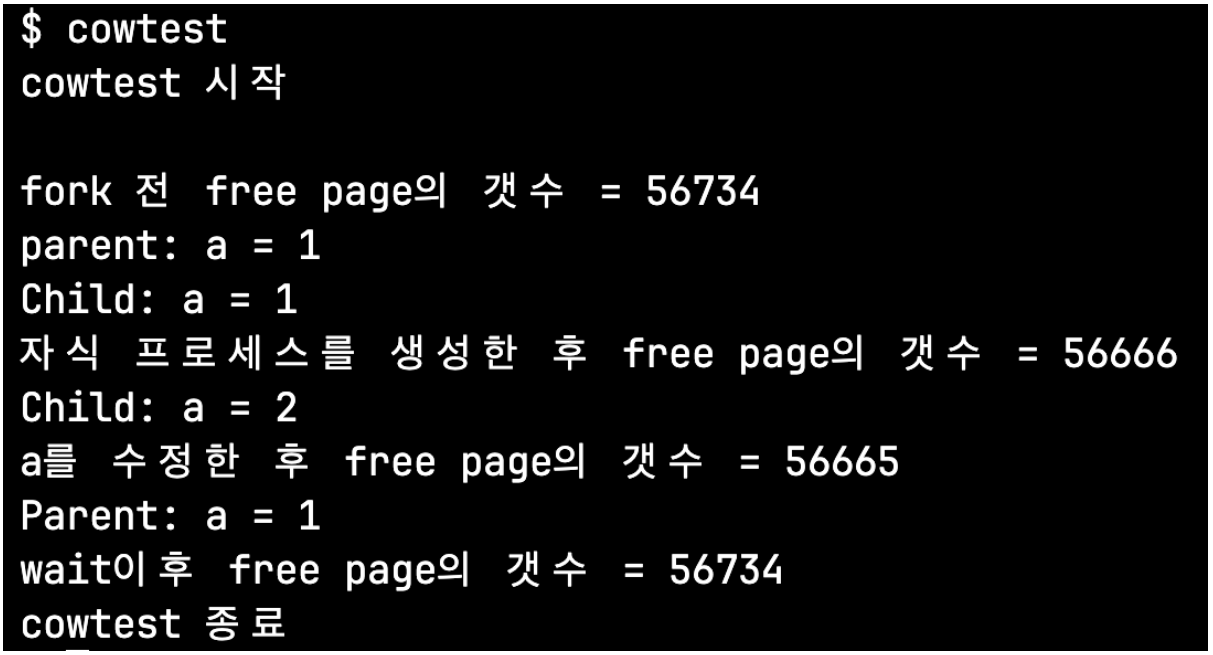
trap.c

```
void
trap(struct trapframe *tf)
{
  switch(tf->trapno){
  // 페이지폴트 발생시 pagefault가 처리
  case T_PGFLT:
    pagefault();
    break;
  }
}
```

mmu.h

```
//12비트를 밀어주기 위한 매크로
#define PGSHIFT         12
```

test 결과



자식 프로세스에서 변수를 수정한 이후 free page 갯수를 확인해보면 1이 적다.

새로운 페이지를 할당했다는 의미이다.

cowtest.c (테스트 코드)

```
#include "types.h"
#include "stat.h"
#include "user.h"

//부모프로세스와 자식프로세스는 전역변수 a를 공유한다
int a = 1;

void test()
{
    printf(1,"fork 전 free page의 갯수 = %d\n",getNumFreePages());

    printf(1,"parent: a = %d\n",a);
    int pid = fork();
```

```
    if(pid==0)
    {
        printf(1,"Child: a = %d\n",a);
        printf(1,"자식 프로세스를 생성한 후 free page의 갯수 = %d\n",getNumFreePages());
        a = 2;
        printf(1,"Child: a = %d\n",a);
        printf(1,"a를 수정한 후 free page의 갯수 = %d\n",getNumFreePages());
        exit();
    }
    wait();
    printf(1,"Parent: a = %d\n",a);
    printf(1,"wait이후 free page의 갯수 = %d\n",getNumFreePages());
    return ;
}

int main(void)
{
    printf(1,"cowtest 시작\n\n");
    test();
    printf(1,"cowtest 종료\n");

    exit();
}
```