컴퓨터학부 20202920 조민혁

1.개요

운영체제 커널의 메모리 할당 및 해제 방법을 이해하기 위해서 ssusbrk()라는 시스템콜을 생성하여, 첫번째 인자로 양수가 들어온 경우에는 메모리를 지연할당하고, 음수로 들어온 경우에는 지연해제 후 현재 메모리의 상황을 출력하고 프로세스를 종료하는 기능을 하게끔 하였습니다. 지연해제의 경우에는 두번째 인자로 tick값을 입력받아서 해당하는 tick값만큼의 시간이 지난 후에 메모리가 해제된 시간을 출력하게끔 설정하여 지연해제의 개념을 적용해볼 수 있습니다. 또한 memstat시스템 콜을 구현하여, 현재 할당된 가상 메모리와 물리 메모리의 정보를 출력하는 기능을 수행합니다. 위의 2개의 시스템 콜을 사용하여 3개의 test프로그램을 수행하여 원하는 결과값이 출력되게끔 하는 프로그램입니다.

2.상세설계

2-1.구현한 함수의 프로토타입

int procMemstat(void);

위의 함수는 memstat시스템 콜에서 호출하기 위해서 구현한 함수입니다. 현재 프로세스의 페이지디렉토리를 구한 후에 가상 메모리 페이지 수와 물리 메모리 페이지 수를 구합니다. 현재 프로세스의 크기를 통한 계산으로 vp를 구할 수 있고 이를 바탕으로 페이지 테이블을 순회하며, walkpgdir함수를 통해 가상 주소에 대한 pte의 가상주소를 리턴받음으로써 해당 pte가 PTE\_P 플래그를 갖고 있는지 확인하여 물리 메모리에 할당된 페이지를 구해서 물리 메모리 페이지의 개수를 확인하여 출력합니다. 그 후 아래의 print\_pde\_pte함수를 호출하여 현재 프로세스가 갖고 있는 현재 물리 메모리로 할당되어 있는 pde와 pte의 값을 화면에 출력합니다.

void print\_pde\_pte(pde\_t \*pgdir, uint sz);

위의 함수는 사용자 주소 공간에 해당하는 PDE의 최대 인덱스를 계산한 후에 사용자의 주소 공간에 해당하는 부분에 대해서만 PDE를 순환합니다. 이때 인자로 받은 pgdir 배열에 할당된 인덱스들 중에서 PTE\_P 플래그가 존재하면 해당 페이지디렉토리엔트리의 값을 출력하고, 해당 페이지 테이을 순환하여, PTE\_P 플래그와 PTE\_U 플래그가 활성화된 페이지 테이블 엔트리의 값을 화면에 출력합니다. 만약에 페이지 테이블 엔트리를 하나도 발견하지 못한 경우에는 no PTE를 출력합니다.

int ssusbrkAlloc(int pageSize);

위의 함수는 ssusbrk시스템 콜에서 사용되는 함수입니다. 이 함수를 사용하여 만약에 시스템콜의 첫번째 인자로 양수가 들어온 경우에 원하는 크기만큼의 메모리를 지연 할당하기 위해서 사용합니다. 우선 인자로 들어온 메모리의 크기가 0이 되는지와 페이지 크기의 배수인지 확인합니다. 그 후 성공적으로 할당하였다면, 할당한 메모리의 주소를 반환합니다. 이는 이전 브레이크 주소로 할당된 메모리의 주소가 되는 것입니다.

int allocuvm\_without\_alloc(pde\_t \*pgdir, uint oldsz, uint newsz);

위의 함수는 vm.c에 구현되어 있으며, PGROUNDUP함수를 사용하여 주소를 페이지 크기의 배수로 올려주어 이를 통해 시작 주소를 페이지 경계로 정렬하여 줍니다. 특정 가상 주소에 대해서 페이지 테이블 엔트리를 찾아 반환하고, 페이지 테이블이 존재하지 않는 경우에는 새로 할당하여 줍니다. 또한 이미 맵핑된 경우에는 건너뛰고 그렇지 않은 것들에 대해서는 PTE\_W 와 PTE\_U 플래그를 설정하여 줍니다. 함수의 과정이 전부 종료된 후에는 새로운 메모리 크기를 반환하여줍니다.

int ssusbrkDealloc(int pageSize, int delayTicks);

위의 함수는 ssusbrk시스템 콜에 첫번째 인자가 음수로 들어온 경우에 처리하는 함수로서 먼저 들어온 pageSzie에 대한 부호를 변경시켜줍니다. 그 후에 pageSize가 0이 아닌지 페이지 크기의 배수가 맞는지 확인하여줍니다. 또한 현재 프로세스의 메모리 크기보다 큰 경우에 에러 처리해줍니다. 해제할 페이지의 수와 해제할 메모리의 시작 주소를 계산하여 줍니다. 이를 통해서 현재 프로세스의 인자들을 입력받은 pageSize와 delayTicks로 갱신시켜줍니다. 그 후에 현재 날짜 및 시간 정보를 출력하여 줍니다.

2-2. 기구현된 함수와 수정한 내용

먼저 ssusbrk\_test1,2,3을 추가하기 위해서 Makefile을 수정하였습니다.

CFLAGS = -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer  
-Werror 플래그를 제거하여 관련된 컴파일 에러를 없앴습니다.

UPROGS에 아래와 같이

\_ssusbrk\_test1\

\_ssusbrk\_test2\

\_ssusbrk\_test3 를 추가하여 주었습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

또한 EXTRA에도

ssusbrk\_test1.c\

ssusbrk\_test2.c\

ssusbrk\_test3.c\ 를 추가하여주었습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음으로는 시스템 콜을 사용하기 위해서 수정한 파일들입니다.

먼저 usys.S입니다.

텍스트, 스크린샷, 폰트, 흑백이(가) 표시된 사진

자동 생성된 설명

기존에 있는 부분에서 SYSCALL(ssusbrk) 와 SYSCALL(memstat) 을 추가하여 시스템 콜을 사용할 수 있도록 하였습니다.

다음으로 user.h입니다.

…기존코드

int sleep(int);

int uptime(void);

int ssusbrk(int pageSize, int delayTicks);

int memstat(void);

…기존코드

기존 코드에 int ssusbrk(int pageSize, int delayTicks); 와 int memstat(void); 를 추가하였습니다.

다음으로 syscall.h입니다.

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

#define SYS\_ssusbrk 22

#define SYS\_memstat 23  
기존 코드에 위의 두 부분을 추가하였습니다.

다음으로 syscall.c입니다.

…기존코드

extern int sys\_ssusbrk(void);

extern int sys\_memstat(void);

…기존코드

[SYS\_ssusbrk] sys\_ssusbrk,

[SYS\_memstat] sys\_memstat,

…기존코드

로 extern int sys\_ssusbrk(void); 와 extern int sys\_memstat(void); ,

[SYS\_ssusbrk] sys\_ssusbrk, 와 [SYS\_memstat] sys\_memstat, 를 추가하였습니다.

이를 통해서 시스템 호출 번호와 시스템 호출 함수 간의 매핑을 정의하는 배열에 추가할 수 있습니다.

다음으로 sysproc.c입니다.

//ssusbrk함수의 구현부

int

sys\_ssusbrk(void){

int pageSize;

int delayTicks;

//잘못된 값이 들어온 경우에 대한 예외처리

if(argint(0,&pageSize) < 0){

return -1;

}

if(pageSize > 0){

//메모리를 지연할당하는 경우

argint(1,&delayTicks);

return ssusbrkAlloc(pageSize);

}else if(pageSize < 0){

//메모리를 지연해제하는 경우

if(argint(1,&delayTicks)<0)

return -1;

if(delayTicks <=0)

return -1;

return ssusbrkDealloc(pageSize,delayTicks);

}else{

return -1;

}

}

ssusbrk의 구현부로서 pageSzie가 첫번째 인자의 값으로써 0이 나온 경우에 -1을 리턴하고, 양수인 경우에는 ssusbrkAlloc함수를 호출하며 음수인 경우에는 ssusbrkDealloc을 호출하여 줍니다.

//memstat함수의 구현부

int

sys\_memstat(void){

return procMemstat();

}

시스템 콜인 memstat함수의 구현부로서 proc.c에 구현되어 있는 procMemstat함수를 호출하여 줍니다.

이렇게 sysproc.c에서 pro.c의 함수를 사용하기 위해서 defs.h에 추가한 함수들을 추가하여 주었습니다.

이때 vm.c에 있는 기존에 static이었던 함수들을 사용하기 위해서 static 키워드를 없애고 추가해주었습니다. 이를 위해서 mmu.h를 포함시켜야 했는데 중복적으로 선언될 수 있기 때문에 중복 처리 매크로를 mmu.h에 추가해주었습니다.

아래는 mmu.h에 있는 소스코드로서 최상단과 최하단에 중복선언을 방지하는 매크로를 설정해두었습니다.

#ifndef MMU\_H

#define MMU\_H

…기존코드

#endif // MMU\_H

pte\_t \*

walkpgdir(pde\_t \*pgdir, const void \*va, int alloc)  
  
와   
int

mappages(pde\_t \*pgdir, void \*va, uint size, uint pa, int perm)  
는 둘 다 static 키워드가 붙어있던 함수로서 vm.c에 구현되어 있던 함수입니다. vm.c에서는 이 두 함수의 static 키워드를 삭제하고,   
int

allocuvm\_without\_alloc(pde\_t \*pgdir, uint oldsz, uint newsz)

{

char \*a;

pte\_t \*pte;

if(newsz >= KERNBASE)

return 0;

if(newsz < oldsz)

return oldsz;

a = (char\*)PGROUNDUP(oldsz);

for(; (uint)a < newsz; a += PGSIZE){

if((pte = walkpgdir(pgdir, a, 1)) == 0)

return 0;

if(\*pte & PTE\_P)

continue; // 이미 매핑되어 있으면 넘어감

// 물리 메모리를 할당하지 않고 PTE 생성

\*pte = PTE\_W | PTE\_U; // 쓰기 및 사용자 접근 가능 설정

// PTE\_P 플래그는 설정하지 않음

}

return newsz;

}  
를 추가하였습니다.

defs.h에서는   
#include "mmu.h"

…기존코드

int ssusbrkAlloc(int pageSize);

int ssusbrkDealloc(int pageSize, int delayTicks);  
…기존코드

int allocuvm\_without\_alloc(pde\_t \*pgdir, uint oldsz, uint newsz);

pte\_t \* walkpgdir(pde\_t \*pgdir, const void \*va, int alloc);

int mappages(pde\_t \*pgdir, void \*va, uint size, uint pa, int perm);  
…기존코드

의 형태로서 프로그램내에서 사용하기 위한 함수들의 프로토타입을 추가하여 호출할 수 있도록 만들어주었습니다.

다음으로는 proc.h에서 rtcdate 구조체를 사용하는 부분이 있기에 date.h에 중복방지 매크로를 설정하였습니다.

#ifndef DATE\_H

#define DATE\_H

struct rtcdate {

uint second;

uint minute;

uint hour;

uint day;

uint month;

uint year;

};

#endif  
와 같은 형태로 date.h를 변경하였습니다.

다음으로는 proc.h입니다.

#include "date.h"

…기존코드

// Per-process state

struct proc {

uint sz; // Size of process memory (bytes)

pde\_t\* pgdir; // Page table

char \*kstack; // Bottom of kernel stack for this process

enum procstate state; // Process state

int pid; // Process ID

struct proc \*parent; // Parent process

struct trapframe \*tf; // Trap frame for current syscall

struct context \*context; // swtch() here to run process

void \*chan; // If non-zero, sleeping on chan

int killed; // If non-zero, have been killed

struct file \*ofile[NOFILE]; // Open files

struct inode \*cwd; // Current directory

char name[16]; // Process name (debugging)

// 지연 메모리 해제를 위한 변수들

int allowDelayTicks; // 적용시킨 tick수

int pending\_free\_ticks; // 남은 tick 수

int pending\_free\_pages; // 지연 해제할 페이지 수

uint pending\_free\_addr; // 해제할 메모리 시작 주소

struct rtcdate ssusbrk\_call\_time; // ssusbrk() 호출 시의 시간

};

…기존코드

로서 지연 메모리 해제를 위한 변수들을 추가하여서 ssusbrk시스템콜에 음수가 들어온 경우에 해당되는 지연 메모리 해제에 대한 출력과 관련된 부분들을 설정할 수 있게됩니다.

다음으로는 proc.c입니다.

…기존코드

void print\_pde\_pte(pde\_t \*pgdir, uint sz);

…기존코드

static struct proc\*

allocproc(void)

{

struct proc \*p;

char \*sp;

acquire(&ptable.lock);

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)

if(p->state == UNUSED)

goto found;

release(&ptable.lock);

return 0;

found:

p->state = EMBRYO;

p->pid = nextpid++;

p->allowDelayTicks = 0; // 초기 tick 수는 0

p->pending\_free\_ticks = 0; // 초기 남은 tick 수는 0

p->pending\_free\_pages = 0; // 초기 해제할 페이지 수는 0

p->pending\_free\_addr = 0; // 초기 해제 시작 주소는 0

memset(&p->ssusbrk\_call\_time, 0, sizeof(p->ssusbrk\_call\_time)); // 초기 시간은 0으로 설정

release(&ptable.lock);

// Allocate kernel stack.

if((p->kstack = kalloc()) == 0){

p->state = UNUSED;

return 0;

}

sp = p->kstack + KSTACKSIZE;

// Leave room for trap frame.

sp -= sizeof \*p->tf;

p->tf = (struct trapframe\*)sp;

// Set up new context to start executing at forkret,

// which returns to trapret.

sp -= 4;

\*(uint\*)sp = (uint)trapret;

sp -= sizeof \*p->context;

p->context = (struct context\*)sp;

memset(p->context, 0, sizeof \*p->context);

p->context->eip = (uint)forkret;

return p;

}

…기존코드

//지연할당과 관련된 부분입니다.

int ssusbrkAlloc(int pageSize)

{

//pageSize가 0이거나 페이지크기의 배수가 아닌 경우에는 return -1

if(pageSize == 0 || pageSize % PGSIZE != 0){

return -1;

}

struct proc \*curproc = myproc();

uint sz = curproc->sz;

uint newsz;

newsz = sz + pageSize;

if(newsz >= KERNBASE)

return -1; // 커널 영역을 침범한 경우에 대한 예외처리

if(allocuvm\_without\_alloc(curproc->pgdir, sz, newsz) == 0)

return -1; // 할당 실패

curproc->sz = newsz;

return sz; // 이전 브레이크 주소 반환

}

//지연해제와 관련된 부분입니다.

int ssusbrkDealloc(int pageSize, int delayTicks)

{

// pageSize가 0이거나 페이지 크기의 배수가 아닌 경우에는 return -1

//부호 변경

pageSize = -pageSize;

if(pageSize == 0 || pageSize% PGSIZE != 0){

return -1;

}

if(delayTicks <= 0)

return -1;

struct proc \*curproc = myproc();

uint sz = curproc->sz;

// pageSize가 현재 프로세스의 메모리 크기보다 큰 경우 에러 처리

if(pageSize > sz)

return -1;

// 해제할 페이지 수 계산

int num\_pages = pageSize / PGSIZE;

// 해제할 메모리의 시작 주소 계산

int dealloc\_addr = sz - (curproc->pending\_free\_pages + num\_pages) \* PGSIZE;

// 이전에 해제 요청이 있는 경우 페이지 수 누적 및 tick 값 업데이트

curproc->allowDelayTicks = delayTicks;

curproc->pending\_free\_pages += num\_pages;

curproc->pending\_free\_ticks = delayTicks;

curproc->pending\_free\_addr = dealloc\_addr;

struct rtcdate curDateTime;

cmostime(&curDateTime);

curproc->ssusbrk\_call\_time = r;

// 출력 형식에 맞게 시간 출력

cprintf("Memory deallocation request(%d): %d-%d-%d %d:%d:%d\n",

curproc->allowDelayTicks,

curproc->ssusbrk\_call\_time.year,

curproc->ssusbrk\_call\_time.month,

curproc->ssusbrk\_call\_time.day,

curproc->ssusbrk\_call\_time.hour,

curproc->ssusbrk\_call\_time.minute,

curproc->ssusbrk\_call\_time.second);

// 성공적으로 처리되었으므로 이전 브레이크 주소 반환

return sz;

}

//memstat함수와 관련된 부분입니다.

int

procMemstat(void)

{

//현재 프로세스를 구함

struct proc \*curproc = myproc();

pde\_t \*pgdir = curproc->pgdir;

uint sz = curproc->sz;

int total\_vpages = (sz + PGSIZE - 1) / PGSIZE; // 가상 메모리 페이지 수

int total\_ppages = 0; // 물리 메모리 페이지 수

// 페이지 테이블을 순회하며 물리 메모리에 매핑된 페이지 수 계산

uint addr;

for(addr = 0; addr < sz; addr += PGSIZE){

pte\_t \*pte = walkpgdir(pgdir, (void\*)addr, 0);

if(pte && (\*pte & PTE\_P)){

total\_ppages++;

}

}

//vp 와 pp의 값을 출력합니다.

cprintf(" vp: %d, pp: %d\n", total\_vpages, total\_ppages);

// PDE와 PTE 값을 출력하는 함수 호출

print\_pde\_pte(pgdir, sz);

return 0;

}

void

print\_pde\_pte(pde\_t \*pgdir, uint sz)

{

int pde\_found = 0; // PDE가 발견되었는지 여부

// 사용자 주소 공간에 해당하는 PDE의 최대 인덱스 계산

uint max\_pdx = (sz - 1) >> PDXSHIFT;

// PDE 순회

for(uint i = 0; i <= max\_pdx; i++){

if(pgdir[i] & PTE\_P){

pde\_found = 1;

cprintf(" PDE - 0x%x\n", pgdir[i]);

pte\_t \*pgtab = (pte\_t\*)P2V(PTE\_ADDR(pgdir[i]));

int pte\_found = 0; // PTE가 발견되었는지 여부

if(pde\_found == 1){

cprintf(" PTE");

}

// 해당 페이지 테이블의 PTE 순회

for(uint j = 0; j < NPTENTRIES; j++){

uint va = (i << PDXSHIFT) | (j << PTXSHIFT);

if(va >= sz)

break; // 프로세스의 가상 메모리 크기를 넘으면 종료

if((pgtab[j] & PTE\_P)&& (pgtab[j] & PTE\_U)){

pte\_found = 1;

cprintf(" - 0x%x",pgtab[j]);

}

}

if(pte\_found == 0){

cprintf("no PTE");

}

cprintf("\n");

}

}

}

로 수정하였습니다. 기존의 allocproc함수에 proc.h에 추가하였던 메모리 지연해제와 관련된 부분들을 초기화해주며, 맨 위에 프로토타입을 선언하여주었습니다. 또한 ssusbrkAlloc, ssusbrkDealloc, procMemstat,print\_pde\_pte함수를 추가하여주었습니다. 새롭게 추가한 함수들은 위의 프로토타입 설명에 그 기능들을 기술해두었습니다.

마지막으로 trap.c의 수정내용입니다.

#include "date.h"

//ptable 가져오기

extern struct {

struct spinlock lock;

struct proc proc[NPROC];

} ptable;

를 추가하여 trap.c에서 ptable을 사용할 수 있도록 하였습니다.

if(tf->trapno == T\_PGFLT){

uint va = rcr2(); // 잘못된 부분의 가상 주소를 가져옴

struct proc \*curproc = myproc();

if(curproc == 0){

// 프로세스가 없는 경우에 대한 처리

panic("No process");

}

// 유효한 주소인지 확인

//현재 프로세스의 크기보다 큰 것은 아닌지 커널베이스를 넘어가는 것은 아닌지 확인

if(va >= curproc->sz || va >= KERNBASE){

cprintf("Memory is out of bound\n");

curproc->killed = 1;

return;

}

// 물리 메모리 할당 및 매핑

char \*phyMem = kalloc();

//물리 메모리가 부족하여 할당되지 않은 경우에 대한 예외처리

if(phyMem == 0){

cprintf("Out of physical memory.\n");

curproc->killed = 1;

return;

}

//물리메모리 영역을 0으로 초기화

memset(phyMem, 0, PGSIZE);

//주어진 가상 주소를 페이지 경계로 정렬

va = PGROUNDDOWN(va);

//가상주소와 새로운 물리주소를 매핑시켜줍니다.

if(mappages(curproc->pgdir, (char\*)va, PGSIZE, V2P(phyMem), PTE\_W|PTE\_U|PTE\_P) < 0){

kfree(phyMem);

cprintf("Mappages ERROR\n");

curproc->killed = 1;

return;

}

// 페이지 폴트 처리 완료

return;

}

void trap(struct trapframe \*tf) 함수 내부에 해당 부분을 추가함으로써 ssusbrk시스템 콜로 가상 메모리를 할당한 후 물리 메모리가 매핑되지 않았을 때 해당 부분이 trap에서 실행되게 됩니다. 먼저 문제가 발생한 가상 메모리 주소를 읽어 저장합니다. 현재 실행중인 프로세스가 없는 경우에 대한 예외처리를 하고 저장했던 가상 주소가 유효한 주소인지 확인합니다. 그 후에 새로운 물리메모리를 할당 받아, 할당 하였을 때 물리메로리가 부족하여 할당되지 않은 경우에 대한 예외처리를 진행합니다. 그 후에 새롭게 할당받은 물리메모리를 0으로 초기화하고, 가상 주소와 새로운 물리 주소를 매핑시켜줍니다.

switch(tf->trapno){

case T\_IRQ0 + IRQ\_TIMER:

if(cpuid() == 0){

acquire(&tickslock);

ticks++;

wakeup(&ticks);

release(&tickslock);

}

lapiceoi();

// 지연 메모리 해제 처리 추가

struct proc \*curproc = myproc();

if(curproc){

acquire(&ptable.lock);

if(curproc->pending\_free\_ticks > 0){

curproc->pending\_free\_ticks--;

if(curproc->pending\_free\_ticks == 0){

// 메모리 해제 수행

uint oldsz = curproc->sz;

uint newsz = curproc->pending\_free\_addr;

if(deallocuvm(curproc->pgdir, oldsz, newsz) == 0){

cprintf("Failed to deallocate memory\n");

curproc->killed = 1;

} else {

curproc->sz = newsz;

// 현재 시간 가져오기

struct rtcdate curDateTime;

cmostime(&curDateTime);

//메모리 해제 시간에 대한 정보 출력

cprintf("Memory deallocation execute: %d-%d-%d %d:%d:%d\n",

curDateTime.year, curDateTime.month, curDateTime.day,

curDateTime.hour, curDateTime.minute, curDateTime.second);

// memstat() 호출

procMemstat();

// 메모리 해제 후 프로세스 종료

curproc->killed = 1;

}

// 지연 해제 정보 초기화

curproc->pending\_free\_pages = 0;

curproc->pending\_free\_addr = 0;

curproc->allowDelayTicks = 0;

memset(&curproc->ssusbrk\_call\_time, 0, sizeof(curproc->ssusbrk\_call\_time));

}

}

release(&ptable.lock);

}

break;

tick을 증가시킬 때 현재 프로세스에 지연해제에 할당해두었던 tick을 감소시키고 tick이 0이 되는 경우에는 메모리 해제를 수행시켜줍니다. deallocuvm함수를 호출하여 메모리를 할당해제시켜주고 현재 날짜와 시간을 출력하며, procMemstat함수를 호출하여 메모리 해제를 한 후의 vp와pp, PDE와 PTE의 갓을 출력합니다. 메모리가 해제된 후에는 프로세스를 killed->1로 설정하여 프로세스가 종료될 수 있도록 하고, 지연해제관련 정보를 초기화시켜줍니다.

2-3.함수호출그래프

ssusbrk시스템콜의 함수호출 그래프

도표, 평면도, 기술 도면, 개략도이(가) 표시된 사진

자동 생성된 설명

memstat시스템콜의 함수호출그래프

도표, 평면도, 기술 도면, 개략도이(가) 표시된 사진

자동 생성된 설명

3.실행결과

ssusbrk\_test1 -> 2 -> 3번 순서대로 수행한 경우

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

ssusbrk\_test1 -> 3 -> 2번 순서대로 수행한 경우

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

ssusbrk\_test2 -> 3 -> 1번 순서대로 수행한 경우

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

ssusbrk\_test2->1->3의 순서로 실행

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

ssusbrk\_test3 ->1 -> 2번 순서대로 수행한 경우

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

ssusbrk\_test3->2->1번 순서대로 수행한 경우

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

4.소스코드

date.h

#ifndef DATE\_H

#define DATE\_H

struct rtcdate {

uint second;

uint minute;

uint hour;

uint day;

uint month;

uint year;

};

#endif

defs.h

#include "mmu.h"

…기존코드

int procMemstat(void);

int ssusbrkAlloc(int pageSize);

int ssusbrkDealloc(int pageSize, int delayTicks);

…기존코드

int allocuvm\_without\_alloc(pde\_t \*pgdir, uint oldsz, uint newsz);

pte\_t \* walkpgdir(pde\_t \*pgdir, const void \*va, int alloc);

int mappages(pde\_t \*pgdir, void \*va, uint size, uint pa, int perm);

…기존코드

Makefile

…기존코드

CFLAGS = -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer

…기존코드

UPROGS=\

\_cat\

\_echo\

\_forktest\

\_grep\

\_init\

\_kill\

\_ln\

\_ls\

\_mkdir\

\_rm\

\_sh\

\_stressfs\

\_usertests\

\_wc\

\_zombie\

\_ssusbrk\_test1\

\_ssusbrk\_test2\

\_ssusbrk\_test3

…기존코드

EXTRA=\

mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\

ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\

printf.c umalloc.c\

README dot-bochsrc \*.pl toc.\* runoff runoff1 runoff.list\

.gdbinit.tmpl gdbutil\

ssusbrk\_test1.c\

ssusbrk\_test2.c\

ssusbrk\_test3.c\

…기존코드

mmu.h

…기존코드

#ifndef MMU\_H

#define MMU\_H

…기존코드

#endif // MMU\_H

proc.c

…기존코드

void print\_pde\_pte(pde\_t \*pgdir, uint sz);

…기존코드

static struct proc\*

allocproc(void)

{

struct proc \*p;

char \*sp;

acquire(&ptable.lock);

for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)

if(p->state == UNUSED)

goto found;

release(&ptable.lock);

return 0;

found:

p->state = EMBRYO;

p->pid = nextpid++;

p->allowDelayTicks = 0; // 초기 tick 수는 0

p->pending\_free\_ticks = 0; // 초기 남은 tick 수는 0

p->pending\_free\_pages = 0; // 초기 해제할 페이지 수는 0

p->pending\_free\_addr = 0; // 초기 해제 시작 주소는 0

memset(&p->ssusbrk\_call\_time, 0, sizeof(p->ssusbrk\_call\_time)); // 초기 시간은 0으로 설정

release(&ptable.lock);

// Allocate kernel stack.

if((p->kstack = kalloc()) == 0){

p->state = UNUSED;

return 0;

}

sp = p->kstack + KSTACKSIZE;

// Leave room for trap frame.

sp -= sizeof \*p->tf;

p->tf = (struct trapframe\*)sp;

// Set up new context to start executing at forkret,

// which returns to trapret.

sp -= 4;

\*(uint\*)sp = (uint)trapret;

sp -= sizeof \*p->context;

p->context = (struct context\*)sp;

memset(p->context, 0, sizeof \*p->context);

p->context->eip = (uint)forkret;

return p;

}

…기존코드

//지연할당과 관련된 부분입니다.

int ssusbrkAlloc(int pageSize)

{

//pageSize가 0이거나 페이지크기의 배수가 아닌 경우에는 return -1

if(pageSize == 0 || pageSize % PGSIZE != 0){

return -1;

}

struct proc \*curproc = myproc();

uint sz = curproc->sz;

uint newsz;

newsz = sz + pageSize;

if(newsz >= KERNBASE)

return -1; // 커널 영역을 침범한 경우에 대한 예외처리

if(allocuvm\_without\_alloc(curproc->pgdir, sz, newsz) == 0)

return -1; // 할당 실패

curproc->sz = newsz;

return sz; // 이전 브레이크 주소 반환

}

//지연해제와 관련된 부분입니다.

int ssusbrkDealloc(int pageSize, int delayTicks)

{

// pageSize가 0이거나 페이지 크기의 배수가 아닌 경우에는 return -1

//부호 변경

pageSize = -pageSize;

if(pageSize == 0 || pageSize% PGSIZE != 0){

return -1;

}

if(delayTicks <= 0)

return -1;

struct proc \*curproc = myproc();

uint sz = curproc->sz;

// pageSize가 현재 프로세스의 메모리 크기보다 큰 경우 에러 처리

if(pageSize > sz)

return -1;

// 해제할 페이지 수 계산

int num\_pages = pageSize / PGSIZE;

// 해제할 메모리의 시작 주소 계산

int dealloc\_addr = sz - (curproc->pending\_free\_pages + num\_pages) \* PGSIZE;

// 이전에 해제 요청이 있는 경우 페이지 수 누적 및 tick 값 업데이트

curproc->allowDelayTicks = delayTicks;

curproc->pending\_free\_pages += num\_pages;

curproc->pending\_free\_ticks = delayTicks;

curproc->pending\_free\_addr = dealloc\_addr;

struct rtcdate curDateTime;

cmostime(&curDateTime);

curproc->ssusbrk\_call\_time = r;

// 출력 형식에 맞게 시간 출력

cprintf("Memory deallocation request(%d): %d-%d-%d %d:%d:%d\n",

curproc->allowDelayTicks,

curproc->ssusbrk\_call\_time.year,

curproc->ssusbrk\_call\_time.month,

curproc->ssusbrk\_call\_time.day,

curproc->ssusbrk\_call\_time.hour,

curproc->ssusbrk\_call\_time.minute,

curproc->ssusbrk\_call\_time.second);

// 성공적으로 처리되었으므로 이전 브레이크 주소 반환

return sz;

}

//memstat함수와 관련된 부분입니다.

int

procMemstat(void)

{

//현재 프로세스를 구함

struct proc \*curproc = myproc();

pde\_t \*pgdir = curproc->pgdir;

uint sz = curproc->sz;

int total\_vpages = (sz + PGSIZE - 1) / PGSIZE; // 가상 메모리 페이지 수

int total\_ppages = 0; // 물리 메모리 페이지 수

// 페이지 테이블을 순회하며 물리 메모리에 매핑된 페이지 수 계산

uint addr;

for(addr = 0; addr < sz; addr += PGSIZE){

pte\_t \*pte = walkpgdir(pgdir, (void\*)addr, 0);

if(pte && (\*pte & PTE\_P)){

total\_ppages++;

}

}

//vp 와 pp의 값을 출력합니다.

cprintf(" vp: %d, pp: %d\n", total\_vpages, total\_ppages);

// PDE와 PTE 값을 출력하는 함수 호출

print\_pde\_pte(pgdir, sz);

return 0;

}

void

print\_pde\_pte(pde\_t \*pgdir, uint sz)

{

int pde\_found = 0; // PDE가 발견되었는지 여부

// 사용자 주소 공간에 해당하는 PDE의 최대 인덱스 계산

uint max\_pdx = (sz - 1) >> PDXSHIFT;

// PDE 순회

for(uint i = 0; i <= max\_pdx; i++){

if(pgdir[i] & PTE\_P){

pde\_found = 1;

cprintf(" PDE - 0x%x\n", pgdir[i]);

pte\_t \*pgtab = (pte\_t\*)P2V(PTE\_ADDR(pgdir[i]));

int pte\_found = 0; // PTE가 발견되었는지 여부

if(pde\_found == 1){

cprintf(" PTE");

}

// 해당 페이지 테이블의 PTE 순회

for(uint j = 0; j < NPTENTRIES; j++){

uint va = (i << PDXSHIFT) | (j << PTXSHIFT);

if(va >= sz)

break; // 프로세스의 가상 메모리 크기를 넘으면 종료

if((pgtab[j] & PTE\_P)&& (pgtab[j] & PTE\_U)){

pte\_found = 1;

cprintf(" - 0x%x",pgtab[j]);

}

}

if(pte\_found == 0){

cprintf("no PTE");

}

cprintf("\n");

}

}

}

proc.h

#include "date.h"

…기존코드

struct proc {

uint sz; // Size of process memory (bytes)

pde\_t\* pgdir; // Page table

char \*kstack; // Bottom of kernel stack for this process

enum procstate state; // Process state

int pid; // Process ID

struct proc \*parent; // Parent process

struct trapframe \*tf; // Trap frame for current syscall

struct context \*context; // swtch() here to run process

void \*chan; // If non-zero, sleeping on chan

int killed; // If non-zero, have been killed

struct file \*ofile[NOFILE]; // Open files

struct inode \*cwd; // Current directory

char name[16]; // Process name (debugging)

// 지연 메모리 해제를 위한 변수들

int allowDelayTicks; // 적용시킨 tick수

int pending\_free\_ticks; // 남은 tick 수

int pending\_free\_pages; // 지연 해제할 페이지 수

uint pending\_free\_addr; // 해제할 메모리 시작 주소

struct rtcdate ssusbrk\_call\_time; // ssusbrk() 호출 시의 시간

};

…기존코드

ssusbrk\_test1,2,3의 소스코드는 교수님이 올려주신 코드와 변경사항이 없기에 word내용에 추가하지 않았습니다.

syscall.c

…기존코드

extern int sys\_ssusbrk(void);

extern int sys\_memstat(void);

…기존코드

[SYS\_ssusbrk] sys\_ssusbrk,

[SYS\_memstat] sys\_memstat,

…기존코드

syscall.h

…기존코드

#define SYS\_ssusbrk 22

#define SYS\_memstat 23

sysproc.c

…기존코드

//ssusbrk함수의 구현부

int

sys\_ssusbrk(void){

int pageSize;

int delayTicks;

//잘못된 값이 들어온 경우에 대한 예외처리

if(argint(0,&pageSize) < 0){

return -1;

}

if(pageSize > 0){

//메모리를 지연할당하는 경우

argint(1,&delayTicks);

return ssusbrkAlloc(pageSize);

}else if(pageSize < 0){

//메모리를 지연해제하는 경우

if(argint(1,&delayTicks)<0)

return -1;

if(delayTicks <=0)

return -1;

return ssusbrkDealloc(pageSize,delayTicks);

}else{

return -1;

}

}

//memstat함수의 구현부

int

sys\_memstat(void){

return procMemstat();

}

trap.c

…기존코드

#include "date.h"

//ptable 가져오기

extern struct {

struct spinlock lock;

struct proc proc[NPROC];

} ptable;

…기존코드

void

trap(struct trapframe \*tf)

{

if(tf->trapno == T\_PGFLT){

uint va = rcr2(); // 잘못된 부분의 가상 주소를 가져옴

struct proc \*curproc = myproc();

if(curproc == 0){

// 프로세스가 없는 경우에 대한 처리

panic("No process");

}

// 유효한 주소인지 확인

//현재 프로세스의 크기보다 큰 것은 아닌지 커널베이스를 넘어가는 것은 아닌지 확인

if(va >= curproc->sz || va >= KERNBASE){

cprintf("Memory is out of bound\n");

curproc->killed = 1;

return;

}

// 물리 메모리 할당 및 매핑

char \*phyMem = kalloc();

//물리 메모리가 부족하여 할당되지 않은 경우에 대한 예외처리

if(phyMem == 0){

cprintf("Out of physical memory.\n");

curproc->killed = 1;

return;

}

//물리메모리 영역을 0으로 초기화

memset(phyMem, 0, PGSIZE);

//주어진 가상 주소를 페이지 경계로 정렬

va = PGROUNDDOWN(va);

//가상주소와 새로운 물리주소를 매핑시켜줍니다.

if(mappages(curproc->pgdir, (char\*)va, PGSIZE, V2P(phyMem), PTE\_W|PTE\_U|PTE\_P) < 0){

kfree(phyMem);

cprintf("Mappages ERROR\n");

curproc->killed = 1;

return;

}

// 페이지 폴트 처리 완료

return;

}

…기존코드

// 지연 메모리 해제 처리 추가

struct proc \*curproc = myproc();

if(curproc){

acquire(&ptable.lock);

if(curproc->pending\_free\_ticks > 0){

curproc->pending\_free\_ticks--;

if(curproc->pending\_free\_ticks == 0){

// 메모리 해제 수행

uint oldsz = curproc->sz;

uint newsz = curproc->pending\_free\_addr;

if(deallocuvm(curproc->pgdir, oldsz, newsz) == 0){

cprintf("Memory Deallocation fault\n");

curproc->killed = 1;

} else {

curproc->sz = newsz;

// 현재 시간 가져오기

struct rtcdate curDateTime;

cmostime(&curDateTime);

//메모리 해제 시간에 대한 정보 출력

cprintf("Memory deallocation execute: %d-%d-%d %d:%d:%d\n",

curDateTime.year, curDateTime.month, curDateTime.day,

curDateTime.hour, curDateTime.minute, curDateTime.second);

// memstat() 호출

procMemstat();

// 메모리 해제 후 프로세스 종료

curproc->killed = 1;

}

// 지연 해제 정보 초기화

curproc->pending\_free\_pages = 0;

curproc->pending\_free\_addr = 0;

curproc->allowDelayTicks = 0;

memset(&curproc->ssusbrk\_call\_time, 0, sizeof(curproc->ssusbrk\_call\_time));

}

}

release(&ptable.lock);

}

break;

…기존코드

user.h

…기존코드

int ssusbrk(int pageSize, int delayTicks);

int memstat(void);

…기존코드

usys.S

…기존코드

SYSCALL(ssusbrk)

SYSCALL(memstat)

vm.c

…기존코드

pte\_t \*

walkpgdir(pde\_t \*pgdir, const void \*va, int alloc)

{

pde\_t \*pde;

pte\_t \*pgtab;

pde = &pgdir[PDX(va)];

if(\*pde & PTE\_P){

pgtab = (pte\_t\*)P2V(PTE\_ADDR(\*pde));

} else {

if(!alloc || (pgtab = (pte\_t\*)kalloc()) == 0)

return 0;

// Make sure all those PTE\_P bits are zero.

memset(pgtab, 0, PGSIZE);

// The permissions here are overly generous, but they can

// be further restricted by the permissions in the page table

// entries, if necessary.

\*pde = V2P(pgtab) | PTE\_P | PTE\_W | PTE\_U;

}

return &pgtab[PTX(va)];

}

…기존코드

int

mappages(pde\_t \*pgdir, void \*va, uint size, uint pa, int perm)

{

char \*a, \*last;

pte\_t \*pte;

a = (char\*)PGROUNDDOWN((uint)va);

last = (char\*)PGROUNDDOWN(((uint)va) + size - 1);

for(;;){

if((pte = walkpgdir(pgdir, a, 1)) == 0)

return -1;

if(\*pte & PTE\_P)

panic("remap");

\*pte = pa | perm | PTE\_P;

if(a == last)

break;

a += PGSIZE;

pa += PGSIZE;

}

return 0;

}

…기존코드

int

allocuvm\_without\_alloc(pde\_t \*pgdir, uint oldsz, uint newsz)

{

char \*a;

pte\_t \*pte;

if(newsz >= KERNBASE)

return 0;

if(newsz < oldsz)

return oldsz;

a = (char\*)PGROUNDUP(oldsz);

for(; (uint)a < newsz; a += PGSIZE){

if((pte = walkpgdir(pgdir, a, 1)) == 0)

return 0;

if(\*pte & PTE\_P)

continue; // 이미 매핑되어 있으면 넘어감

// 물리 메모리를 할당하지 않고 PTE 생성

\*pte = PTE\_W | PTE\_U; // 쓰기 및 사용자 접근 가능 설정

// PTE\_P 플래그는 설정하지 않음

}

return newsz;

}

…기존코드