컴퓨터학부 20142385 엄재식

1. 과제 개요
2. heap, stack 할당, 반환
3. Virtual memory -> Real memory 매핑
4. 상세설계
5. palloc.c
   1. palloc\_get\_multiple

**while loop로 kpage를 순회하여 비어있는 kpage를 찾음**

* + - type이 free이고 nalloc이 page\_cnt와 같으면 while 종료
    - while을 반복할 때마다 page\_idx를 nalloc 만큼 증가시켜서 다음 할당될 주소공간이 어디인지 알아냄
    - while을 마칠 때까지 kpage를 찾지 못하면 kpage 제일 마지막 자리에 넣음

**case heap**

* + - page\_idx를 이용해 pages를 구함
    - vaddr, nalloc, type, pid 값 넣음
    - page\_cnt만큼 memset

**case stack**

* + - pages는 VKERNEL\_STACK\_ADDR으로 함
    - vaddr, nalloc, type, pid 값 넣음
    - page\_cnt만큼 memset
  1. palloc\_free\_multiple

**while loop로 kpage를 순회하여 vaddr과 pages가 일치하는 kpage를 찾음**

* + - nalloc값과 page\_cnt 값이 같아야함
    - type을 FREE\_\_와 or연산 해줌
  1. va\_to\_ra

**while loop로 kpage를 순회함**

* + - va가 kpage->vaddr과 같고 타입이 heap이면 해당 kpage의 heap주소 값을 리턴
    - va가 400000~420000 사이에 있고 kpage의 타입이 stack이면 해당 kpage의 stack주소 값을 리턴
  1. ra\_to\_va

**ra 값을 이용해 주소의 idx값을 찾는다**

* + - idx = (int)(((uint32\_t)ra - RKERNEL\_HEAP\_START) / PAGE\_SIZE);

**while loop로 kpage를 순회하며 nalloc값을 더해 그 값과 idx값이 같을 때 while 종료**

* + - 해당 kpage의 타입이 stack이면 VKERNEL\_STACK\_ADDR 리턴
    - while을 돌때 kpage의 타입이 heap일 때마다 변수 하나의 값을 nalloc만큼 증가시켜줌
    - 그 변수를 이용해 kpage 타입이 heap일 때 주소 값을 구해 리턴

1. child\_stack\_reset

부모 프로세스의 page directory 리셋

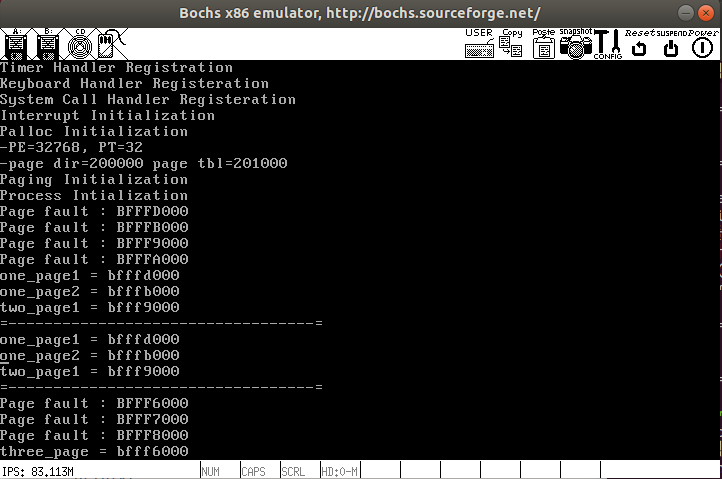
* + - ra\_to\_va(PID0\_PAGE\_DIR)[256] = 0;

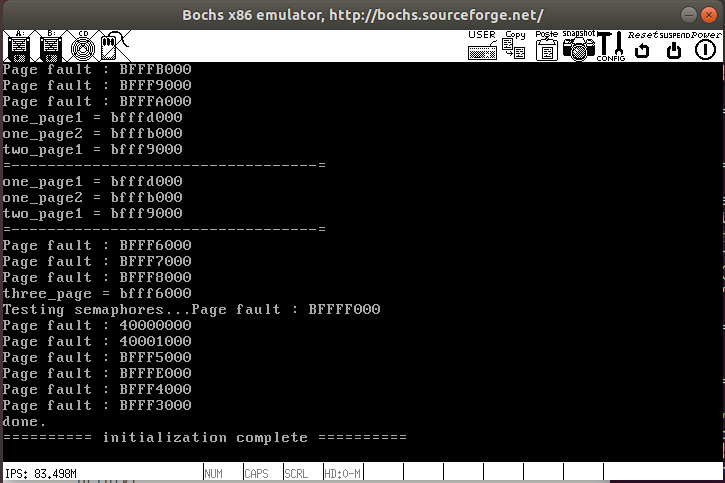
1. proc\_create

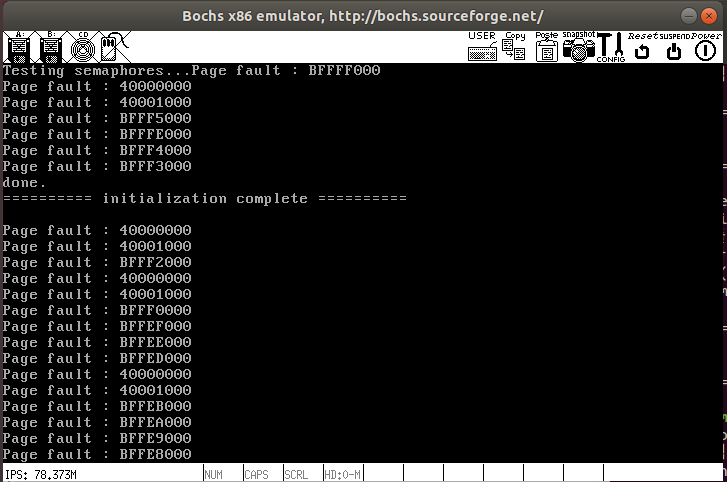
child\_stack\_reset 호출하여 부모 pd 리셋

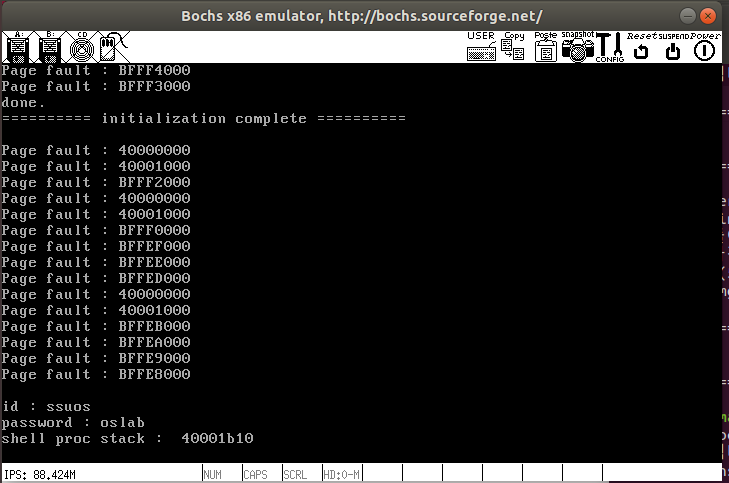
tmp 변수를 이용해 cur\_process->pid를 바꿔 자식 프로세스의 pid로 스택 생성

1. 실행결과









1. 소스코드
2. palloc.c

#include <mem/palloc.h>  
#include <bitmap.h>  
#include <type.h>  
#include <round.h>  
#include <mem/mm.h>  
#include <synch.h>  
#include <device/console.h>  
#include <mem/paging.h>  
#include <proc/proc.h>  
  
/\* Page allocator.  Hands out memory in page-size (or  
   page-multiple) chunks.    
   \*/  
  
/\* page struct \*/  
struct kpage{  
    uint32\_t type;  
    uint32\_t \*vaddr;  
    uint32\_t nalloc;  
    pid\_t pid;  
};  
  
  
static struct kpage \*kpage\_list;  
static uint32\_t page\_alloc\_index;  
  
/\* Initializes the page allocator. \*/  
    void  
init\_palloc (void)   
{  
    /\* Calculate the space needed for the kpage list \*/  
    size\_t pool\_size = sizeof(struct kpage) \* PAGE\_POOL\_SIZE;  
  
    /\* kpage list alloc \*/  
    kpage\_list = (struct kpage \*)(KERNEL\_ADDR);  
  
    /\* initialize \*/  
    memset((void\*)kpage\_list, 0, pool\_size);  
    page\_alloc\_index = 0;  
}  
  
/\* Obtains and returns a group of PAGE\_CNT contiguous free pages.  
   \*/  
    uint32\_t \*  
palloc\_get\_multiple (uint32\_t page\_type, size\_t page\_cnt)  
{  
    void \*pages = NULL;  
    struct kpage \*kpage = kpage\_list;  
    size\_t page\_idx = 0;  
    int i = 0,j = 0;  
      
    if (page\_cnt == 0)  
        return NULL;  
      
    //비어있는 kpage 찾기  
    while (j < page\_alloc\_index) {  
        kpage = (struct kpage \*)((uint32\_t)kpage\_list + sizeof(struct kpage) \* i);              
                  
        if ((kpage->type&FREE\_\_) == FREE\_\_)  
            if (kpage->nalloc == page\_cnt && (kpage->type&page\_type) == page\_type)  
                break;  
  
        if ((kpage->type&page\_type) == page\_type)  
            page\_idx += kpage->nalloc;  
  
        j += kpage->nalloc;  
        i++;  
    }  
  
    if (page\_alloc\_index == 0) {  
        kpage = (struct kpage \*)((uint32\_t)kpage\_list);  
        page\_alloc\_index += page\_cnt;  
    }  
    else if (j >= page\_alloc\_index) {  
        kpage = (struct kpage \*)((uint32\_t)kpage\_list + sizeof(struct kpage) \* i);  
        page\_alloc\_index += page\_cnt;  
    }  
          
      
    page\_idx += page\_cnt;  
  
    switch(page\_type){  
        case HEAP\_\_: //(1)              
            pages = (void\*)(VKERNEL\_HEAP\_START - (page\_idx) \* PAGE\_SIZE);  
            kpage->vaddr = pages;  
            kpage->nalloc = page\_cnt;  
            kpage->type = HEAP\_\_;  
            kpage->pid = cur\_process->pid;  
            //printk("Palloc kpage[%d] heap -> ", i);  
            //printk("%x , ra -> ", pages);  
            //printk("%x\n", va\_to\_ra(pages));  
            memset((void \*)pages, 0, page\_cnt \* PAGE\_SIZE);  
            break;  
        case STACK\_\_:  
            pages = (void\*)(VKERNEL\_STACK\_ADDR);  
            kpage->vaddr = pages;  
            kpage->nalloc = page\_cnt;  
            kpage->type = STACK\_\_;  
            kpage->pid = cur\_process->pid;  
            //printk("Palloc kpage[%d] stack -> ", i);  
            //printk("%x , ra -> ", pages);  
            //printk("%x\n", va\_to\_ra(pages));  
            memset((void \*)pages - page\_cnt \* PAGE\_SIZE, 0, page\_cnt \* PAGE\_SIZE);        //(2)  
            break;  
        default:  
            return NULL;  
    }  
  
    return (uint32\_t\*)pages;   
}  
  
/\* Obtains a single free page and returns its address.  
   \*/  
    uint32\_t \*  
palloc\_get\_page (uint32\_t page\_type)   
{  
    return palloc\_get\_multiple (page\_type, 1);  
}  
  
/\* Frees the PAGE\_CNT pages starting at PAGES. \*/  
    void  
palloc\_free\_multiple (void \*pages, size\_t page\_cnt)   
{  
  
    struct kpage \*kpage = kpage\_list;  
    int i = 0, j = 0;      
      
    if (pages == NULL || page\_cnt == 0)  
        return;  
    //page 같은 것 찾기  
    while (j < page\_alloc\_index) {      
        kpage = (struct kpage \*)((uint32\_t)kpage\_list + sizeof(struct kpage) \* i);  
  
        if (kpage->vaddr == pages && kpage->nalloc == page\_cnt) {  
            kpage->type = kpage->type|FREE\_\_;//FREE\_\_ 추가          
            break;  
        }          
          
        j += kpage->nalloc;  
        i++;  
    }  
  
}  
  
/\* Frees the page at PAGE. \*/  
    void  
palloc\_free\_page (void \*page)   
{  
    palloc\_free\_multiple (page, 1);  
}  
  
  
    uint32\_t \*  
va\_to\_ra (uint32\_t \*va){  
    struct kpage \*kpage = kpage\_list;  
    int i = 0, j = 0;   
    //printk("call va\_to\_ra, %x\n", va);  
    if ((uint32\_t)va >= 0 && (uint32\_t)va < RKERNEL\_HEAP\_START)  
        return va;  
    //kpage 순회  
    while (j < page\_alloc\_index) {  
        kpage = (struct kpage \*)((uint32\_t)kpage\_list + sizeof(struct kpage) \* i);  
          
        if (kpage->vaddr == va) {  
            //힙 리 턴  
            if ((kpage->type&HEAP\_\_) == HEAP\_\_)  
                return (uint32\_t \*)(j \* PAGE\_SIZE + RKERNEL\_HEAP\_START);  
        }  
        //스 택 리 턴  
        if (va <= VKERNEL\_STACK\_ADDR && va >= (VKERNEL\_STACK\_ADDR - 2 \* PAGE\_SIZE)) {  
            if ((kpage->type&STACK\_\_) == STACK\_\_) {  
                if (kpage->pid == cur\_process->pid)  
                    return (uint32\_t \*)(j \* PAGE\_SIZE + RKERNEL\_HEAP\_START + ((uint32\_t\*)VKERNEL\_STACK\_ADDR - va));  
            }  
        }  
        j += kpage->nalloc;  
        i++;  
    }  
      
    return NULL;  
  
}  
          
    uint32\_t \*  
ra\_to\_va (uint32\_t \*ra){  
    struct kpage \*kpage = kpage\_list;  
    int i = 0, j = 0, idx;      
    int s = 0, h = 0;  
    //printk("call ra\_to\_va, %x\n", ra);  
    //1:1 매칭  
    if ((uint32\_t)ra >= 0 && (uint32\_t)ra < RKERNEL\_HEAP\_START)  
        return ra;  
    //인덱스 찾기  
    idx = (int)(((uint32\_t)ra - RKERNEL\_HEAP\_START) / PAGE\_SIZE);  
    //인덱스의 kpage찾기  
    while (j < page\_alloc\_index) {  
        kpage = (struct kpage \*)((uint32\_t)kpage\_list + sizeof(struct kpage) \* i);  
          
        if (j == idx)  
            break;  
  
        if ((kpage->type&HEAP\_\_) == HEAP\_\_)  
            h += kpage->nalloc;  
          
        j += kpage->nalloc;  
        i++;  
    }  
  
    if (j == 0)  
        j += kpage->nalloc;  
  
    if ((kpage->type&STACK\_\_) == STACK\_\_)  
        return (uint32\_t \*)(VKERNEL\_STACK\_ADDR);  
    if ((kpage->type&HEAP\_\_) == HEAP\_\_)  
        return (uint32\_t \*)(VKERNEL\_HEAP\_START - (h + kpage->nalloc) \* PAGE\_SIZE);  
  
    return NULL;  
  
}  
  
void palloc\_pf\_test(void)  
{  
    uint32\_t \*one\_page1 = palloc\_get\_page(HEAP\_\_);  
    uint32\_t \*one\_page2 = palloc\_get\_page(HEAP\_\_);  
    uint32\_t \*two\_page1 = palloc\_get\_multiple(HEAP\_\_,2);  
    uint32\_t \*three\_page;  
      
    printk("one\_page1 = %x\n", one\_page1);   
    printk("one\_page2 = %x\n", one\_page2);   
    printk("two\_page1 = %x\n", two\_page1);  
  
    printk("=----------------------------------=\n");  
    palloc\_free\_page(one\_page1);  
    palloc\_free\_page(one\_page2);  
    palloc\_free\_multiple(two\_page1,2);  
  
    one\_page1 = palloc\_get\_page(HEAP\_\_);  
    two\_page1 = palloc\_get\_multiple(HEAP\_\_,2);  
    one\_page2 = palloc\_get\_page(HEAP\_\_);  
  
    printk("one\_page1 = %x\n", one\_page1);  
    printk("one\_page2 = %x\n", one\_page2);  
    printk("two\_page1 = %x\n", two\_page1);  
  
    printk("=----------------------------------=\n");  
    three\_page = palloc\_get\_multiple(HEAP\_\_,3);  
  
    printk("three\_page = %x\n", three\_page);  
    palloc\_free\_page(one\_page1);  
    palloc\_free\_page(one\_page2);  
    palloc\_free\_multiple(two\_page1,2);  
    palloc\_free\_multiple(three\_page, 3);  
      
}

1. paging.c, child\_stack\_reset

void child\_stack\_reset(pid\_t pid){// (2)  
    //부모 page directory 리셋  
    if (pid == 0 && ra\_to\_va(PID0\_PAGE\_DIR)[0] != NULL)  
        ra\_to\_va(PID0\_PAGE\_DIR)[256] = 0;  
  
}

1. proc.c, proc\_create

pid\_t proc\_create(proc\_func func, struct proc\_option \*opt, void\* aux)  
{  
    struct process \*p;  
    pid\_t tmp;  
      
    int idx;  
  
    enum intr\_level old\_level = intr\_disable();  
  
    pid\_t pid = getValidPid(&idx);  
    p = &procs[idx];  
  
    p->pid = pid;  
    p->state = PROC\_RUN;  
  
    if(opt != NULL)  
        p->priority = opt->priority;     
    else  
        p->priority = (unsigned char)0;  
  
    p->time\_used = 0;  
    p->time\_sched= 0;  
    p->parent = cur\_process;  
    p->simple\_lock = 0;  
    p->child\_pid = -1;  
      
    **tmp = cur\_process->pid;  
    child\_stack\_reset(tmp);  
    //자식프로세스 스택추가  
    cur\_process->pid = p->pid;  
        int \*top = (int\*)palloc\_get\_multiple(STACK\_\_, 2);  
    cur\_process->pid = tmp;**    int stack = (int)(top-1);  
  
    \*(--top) = (int)aux;        //argument for func  
    \*(--top) = (int)proc\_end;    //return address from func  
    \*(--top) = (int)func;        //return address from proc\_start  
    \*(--top) = (int)proc\_start; //return address from switch\_process  
  
    \*(--top) = stack ; //ebp  
    \*(--top) = 1; //eax  
    \*(--top) = 2; //ebx  
    \*(--top) = 3; //ecx  
    \*(--top) = 4; //edx  
    \*(--top) = 5; //esi  
    \*(--top) = 6; //edi  
  
    p->stack = top;  
    p->pd = pd\_create(pid);  
    p->elem\_all.prev = NULL;  
    p->elem\_all.next = NULL;  
    p->elem\_stat.prev = NULL;  
    p->elem\_stat.next = NULL;  
  
    list\_push\_back(&p\_list, &p->elem\_all);  
    list\_push\_back(&r\_list, &p->elem\_stat);  
  
    intr\_set\_level (old\_level);  
    return p->pid;  
}