20192392 컴퓨터학부 김한승

1. 개요

본 과제를 통해서 xv6 운영체제에 새로운 시스템 호출을 추가하는 것을 수행하였습니다.

“alarm\_test [seconds]” / “datetest” 사용을 위하여 새로운 시스템 호출을 추가하였습니다.

1.user.h : xv6의 시스템 호출 정의에 추가로 int date(struct rtcdate\*) 와 int alarm(int seconds)을 추가하였습니다.

2.usys.S : xv6의 시스템 호출 리스트에 추가로 SYSCALL(date) 와 SYSCALL(alarm)을 추가하였습니다.

3.syscall.h : 시스템 호출 번호 매핑에 #define SYS\_DATE 22 와 #define SYS\_alarm 23을 추가하였습니다. 새로운 시스템 호출을 위해 22 / 23에 새로운 매핑을 추가한 것입니다.

4.syscall.c : 시스템 호출 인수를 구문 분석하는 함수 및 실제 시스템 호출 구현에 대한 포인터에 새로운 시스템 호출 함수 포인터를 추가하였습니다. [SYS\_date] sys\_date / [SYS\_alarm] sys\_alarm 와 extern int sys\_date(void); 와 extern int sys\_alarm(void);를 추가하였습니다.

5.sysproc.c : 시스템 호출 코드를 추가하였습니다. int sys\_date(void) 와 int sys\_alarm(void) 추가

6.proc.h : proc 구조체 내부 변수를 2개 추가하였습니다.(int alarmticks와 int alarm\_timer) 변수 추가

7.proc.c : static struct proc\* allocproc(void); proc 구조체 초기화 하는 함수내에 proc 구조체에 새로 추가한 변수 2개를 초기화 하는 코드 추가

8. trap.c : trap(struct trapframe \*tf) 함수내에 타이머 인터럽트 부분을 수정하였습니다.

9.datetest.c 구현 : user.h에 정의된 int date(struct rtcdate\*)를 호출합니다. struct rtcdate 변수에 date() 함수를 통한 결과를 출력합니다.

10.alarm\_test.c 구현 : 교수님께서 설계과제2 명세서로 제공해주신 코드(수정 불가) 이므로 설명 생략

11.Makefile 수정 : Makefile 내에 -rtc base=localtime 옵션을 추가하여서, datetest 와 alarm\_test 명령어 사용시 대한민국 표준 시간으로 시각이 출력 될 수 있도록 하였습니다.

date 시스템 콜과 alarm 시스템 콜 추가를 통한 “datetest” / “alarm\_test [seconds]” 명령어 사용을 구현하였고, 이를 통해 xv6 커널에 대한 이해와 시스템 호출 과정에 대한 이해와 커널 모드 / 유저 모드에 차이에 대해서 생각할 수 있었습니다.

2. 상세 설계

개요에서 나열한 10개의 파일 수정을 중심으로 하여 설계에 대한 설명하겠습니다.

datetest 명령어 / alarm\_test 명령어 모두 유저 프로그램에서 시작합니다.

각 유저 프로그램 내에서 date() / alarm() user.h에 선언 되어있는 시스템 콜 함수를 호출하고,

해당 시스템 콜 함수 호출 시, 소프트웨어 인터럽트(trap)가 발생합니다.

소프트웨어 인터럽트 발생을 통해서 커널 모드로 진입하고 trap(struct trapframe\*) 함수를 통해서

syscall() 함수를 호출합니다. 이때, 어떠한 시스템 콜 핸들러를 실행할지는 (syscall.h에 매핑 된 번호 와 syscall.c 시스템 콜 핸들러 테이블)을 통해 결정합니다.

sysproc.c에 구현 되어있는 시스템 콜 핸들러를 호출한 이후, 커널 모드를 종료하고 유저 모드로 전환 후 시스템 콜을 호출한 유저 프로그램에서 시스템 콜 결과를 출력합니다.

2-1. datetest

datetest.c 유저 프로그램 내부에서 “user.h”에 정의된 int date(struct rtcdate\*) 시스템 콜 함수를 호출한다.

시스템 콜 호출로 인하여 소프트웨어 인터럽트(트랩)이 발생하고,

trap(struct trapframe\*); 함수에서 syscall();을 호출합니다.

trap() 함수인자인 trapframe 구조체 포인터를 통해서 trapframe 내에 시스템 콜 번호를 통해 (syscall.h 22번 SYS\_date) syscall.c에 시스템 호출 번호 22로 매핑된 sys\_date() (시스템 콜 핸들러)를 호출합니다.

sys\_date()함수는 sysproc.c내부에 구현 되어있고, 해당 시스템 콜 함수를 호출 한 이후, 커널 모드를 종료합니다.

커널 모드가 종료된 이후, 사용자 모드로 전환된 후에는 시스템 콜 함수 실행 결과를 통해 현재 시각을 출력합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

2-2. alarm\_test

alarm\_test.c 유저 프로그램 내부에서 “user.h”에 정의된 int alarm(int seconds) 시스템 콜 함수를 호출한다.

시스템 콜 호출로 인하여 소프트웨어 인터럽트(트랩)가 시스템 콜 호출이 발생하고,

trap(struct trapframe\*); 함수에서 syscall();을 호출합니다.

trap() 함수인자인 trapframe 구조체 포인터를 통해서 trapframe 내에 시스템 콜 번호를 통해 (syscall.h 23번 SYS\_alarm) syscall.c에 시스템 호출 번호 23로 매핑된 sys\_alarm() (시스템 콜 핸들러)를 호출합니다.

sys\_alarm()함수는 sysproc.c내부에 구현 되어있고, proc 구조체 내에 alarm\_timer 변수를 유저 프로그램 인자인

[seconds] \* 100(1sec == 100 tick)으로 초기화 합니다.

해당 시스템 콜 함수를 호출한 이후, 커널 모드를 종료합니다.

그 후, int date(struct rtcdate\*)를 호출합니다.

시스템 콜 호출로 인한 소프트웨어 인터럽트 및 시스템 콜 핸들러 함수 실행은 datetest때와 동일합니다.

그 결과를 통해 얻은 struct rtcdate 변수를 통해 현재 시각을 호출하고, while(1); 문을 통한 무한 루프에 들어간다.

유저 프로그램이 무한 루프에 빠진 동안 타이머 인터럽트가 발생될 때마다 trap(struct trapframe\*) 함수가 실행됩니다.

현재 실행중인 프로세스의 구조체가 null이 아니고, 유저 모드인 것을 확인하면 alarmticks를 증가시킵니다.

sys\_alarm() 호출로 인해 초기화 되었던 alarm\_timer와 alarmticks가 같아질 때까지 이를 반복합니다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

3. 실행 결과

3-1. datetest

텍스트, 폰트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

3-2. alarm\_test

3-2-1. alarm\_test 10

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

3-2-2. alarm\_test 50

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4. 소스 코드

4-1. user.h

int date(struct rtcdate\*);

int alarm(int seconds);

4-2. usys.S

SYSCALL(date)

SYSCALL(alarm)

4-3. syscall.h

#define SYS\_date   22

#define SYS\_alarm  23

4-4. syscall.c

extern int sys\_date(void);

extern int sys\_alarm(void);

static int (\*syscalls[])(void) = {

[SYS\_fork]    sys\_fork,

[SYS\_exit]    sys\_exit,

[SYS\_wait]    sys\_wait,

[SYS\_pipe]    sys\_pipe,

[SYS\_read]    sys\_read,

[SYS\_kill]    sys\_kill,

[SYS\_exec]    sys\_exec,

[SYS\_fstat]   sys\_fstat,

[SYS\_chdir]   sys\_chdir,

[SYS\_dup]     sys\_dup,

[SYS\_getpid]  sys\_getpid,

[SYS\_sbrk]    sys\_sbrk,

[SYS\_sleep]   sys\_sleep,

[SYS\_uptime]  sys\_uptime,

[SYS\_open]    sys\_open,

[SYS\_write]   sys\_write,

[SYS\_mknod]   sys\_mknod,

[SYS\_unlink]  sys\_unlink,

[SYS\_link]    sys\_link,

[SYS\_mkdir]   sys\_mkdir,

[SYS\_close]   sys\_close,

[SYS\_date]    sys\_date,

[SYS\_alarm]   sys\_alarm,

};

4-5. sysproc.c

int

sys\_date(void)

{

  struct rtcdate \*date;

  if(argptr(0, (void\*)&date, sizeof(struct rtcdate\*)) < 0)

    return -1;

  cmostime(date);

  return 0;

}

int sys\_alarm(void)

{

  int timer\_seconds;

  if(argint(0, &timer\_seconds) < 0)

    return -1;

  myproc()->alarm\_timer = timer\_seconds \* 100; // 1초당 100 tick

  return 0;

}

4-6. proc.h

// 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)

  // alarm 관련 필드

  int alarmticks;              // 현재 축적된 틱의 수

  int alarm\_timer;             // alarmtimer를 종료시킬 틱의 수

};

4-7. proc.c

//PAGEBREAK: 32

// Look in the process table for an UNUSED proc.

// If found, change state to EMBRYO and initialize

// state required to run in the kernel.

// Otherwise return 0.

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++;

  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;

  // alarm 관련 변수들 초기화

  p->alarmticks = 0;

  p->alarm\_timer = 0;

  return p;

}

4-8. trap.c

//PAGEBREAK: 41

void

trap(struct trapframe \*tf)

{

  if(tf->trapno == T\_SYSCALL){

    if(myproc()->killed)

      exit();

    myproc()->tf = tf;

    syscall();

    if(myproc()->killed)

      exit();

    return;

  }

  switch(tf->trapno){

  case T\_IRQ0 + IRQ\_TIMER: // 타이머 인터럽트가 발생 했을때 실행될 코드

    if(cpuid() == 0){

      acquire(&tickslock);

      ticks++;

      wakeup(&ticks);

      release(&tickslock);

    }

  // Alarm 구현

  if(myproc() != 0 && (tf->cs & 3) == 3) // 현재 실행중인 프로세스를 나타내는 포인터가 null이 아니고, cs필드의 최하위 2비트 11인지 확인

  {

    myproc()->alarmticks++; // 알람 틱 증가

    if(myproc()->alarm\_timer == myproc()->alarmticks) // 알람 틱과 종료 될 알람틱이 같은지 비교

    {

      struct rtcdate r;

      cmostime(&r);

      myproc()->alarm\_timer = 0;

      cprintf("SSU\_Alarm!\n");

      cprintf("Current time : %d-%d-%d %d:%d:%d\n",r.year ,r.month, r.day, r.hour, r.minute, r.second);

      myproc()->killed = 1; // 프로세스 종료시키기 위하여 killed 1로 설정

    }

  }

    lapiceoi();

    break;

  case T\_IRQ0 + IRQ\_IDE:

4-9. datetest.c

#include "types.h"

#include "user.h"

#include "date.h"

int

main (int argc, char \*argv[])

{

    struct rtcdate r;

    if(date(&r))

    {

        printf(2, "date failed\n");

        exit();

    }

    printf(1, "Current time : %d-%d-%d %d:%d:%d\n", r.year, r.month, r.day, r.hour, r.minute, r.second);

    exit();

}

4-10. alarm\_test.c(수정 불가)

#include "types.h"

#include "user.h"

#include "date.h"

int main(int argc, char \*argv[])

{

    int seconds;

    struct rtcdate r;

    if(argc <= 1)

        exit();

    seconds = atoi(argv[1]);

    alarm(seconds);

    date(&r);

    printf(1, "SSU\_Alarm Start\n");

    printf(1, "Current time : %d-%d-%d %d:%d:%d\n", r.year, r.month, r.day, r.hour, r.minute, r.second);

    while(1)

        ;

    exit();

}

4-11. Makefile

UPROGS=\

    \_cat\

    \_echo\

    \_forktest\

    \_grep\

    \_init\

    \_kill\

    \_ln\

    \_ls\

    \_mkdir\

    \_rm\

    \_sh\

    \_stressfs\

    \_usertests\

    \_wc\

    \_zombie\

    \_datetest\

    \_alarm\_test\

QEMUOPTS = -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp $(CPUS) -m 512 $(QEMUEXTRA) -rtc base=localtime

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 datetest.c alarm\_test.c\

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

    .gdbinit.tmpl gdbutil\