Kernel vs User space

Tuesday, March 28, 2023 5:46 PM

14-2008: Vežbe 10 29.3.2023. (raf.edu.rs)

RADE SE SISTEMSKI POZIVI.

USER I KERNEL IMAJU STACKOVE MEMORIJE

USER SPACE

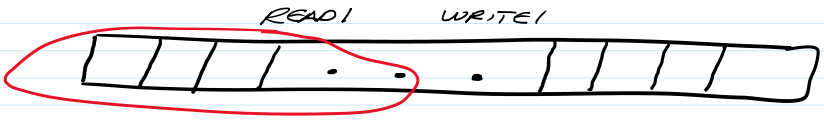
(UNUTAR NJEGA SE POKREĆU I IZVRŠAVAJU PROGRAMI IZ USER FOLDERA)

SISTEMSKI POZIV KREIRA SOFTVERSKI PREKID (INTERRUPT)

OVO JE ZIO (KROZ NJEGA SE PROBIJAMO POMOĆU SISTEMSKIH POZIVA, ONI OMOGUĆAVAJU USER PROGRAMIMA DA KORISTE KERNEL FUNKCIJE)

KERNEL IMA HANDLER PREKIDA KOJI IH PREPOZNAJE

KERNEL SPACE



XV6 IMA 2 REŽIMA RADA NORMALNI I ZAŠTIĆENI

FUNKCIJE UNUTAR KERNEL PROSTORA MOGU DA RADE KADA GOD

USER.H
USYS.S
SYSCALL.H

```
#include "kernel/syscall.h"
#include "kernel/traps.h"

#define SYSCALL(name) \
    .globl name; \
    name: \
        movl $SYS_ ## name, %eax; \
        int $T_SYSCALL; \
        ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
```

ASEMBLER U AT&T SINTAKSI

INTERRUPT

```
global open:
    movl 15, %eax
    INT T_SYS_CALL
    RET
```

MAKRO SISTEMSKOG POZIVA

SISTEMSKI PREKID IMA KOD 64

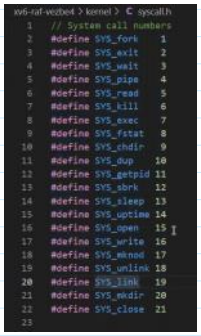
SISTEMSKI POZIV

KORISNIČKI PROGRAM POZOVE NEKU METODU (OPEN())

+ USYS.S NALAZI MAKRO ZA METODU I STAVLJA NA STEK

INT T_SYS_CALL

VECTOR.S → TRAPASM.S



DOLAZIMO DO TRAP.C I TRAPFRAME STRUKTURE

```
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(myproc()->killed)
      exit();
    myproc()->tf = tf;
    syscall();
    if(myproc()->killed)
      exit();
```

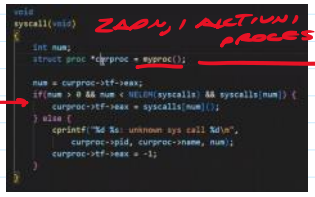
Syscall.c

Sysfile.c

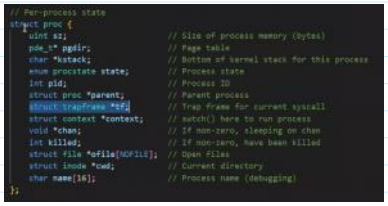
Syscall.c

↓

Sysfile.c



NUM CHECK



SYSCALLS NIZ

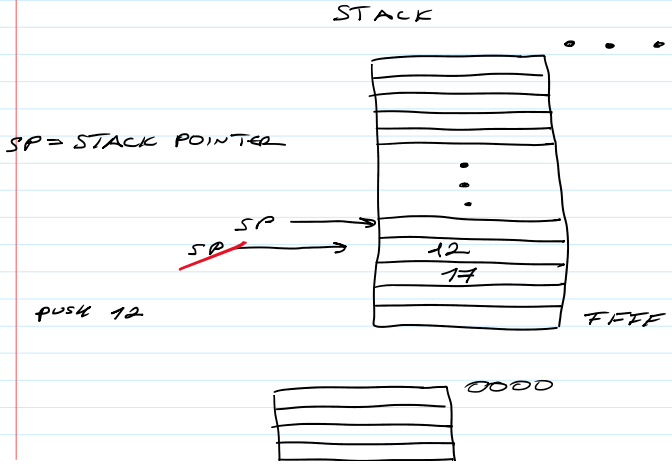
```
static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_getpid]  sys_getpid,
[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,
};
```

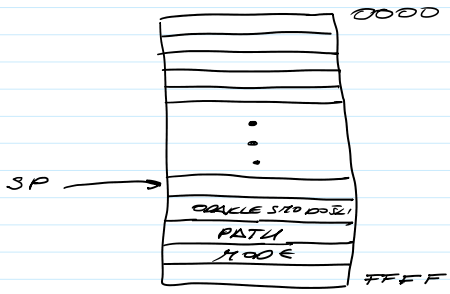
15.

UNUTAR SYSFILE.C SE NALAZE FUNKCIJE POPUT SYS_OPEN(void)

GDJE NEMA ARGUMENATA ZA FUNKCIJU, ALI PODACI KOJI NAM TREBAJU SE NALAZE NA STEKU, NJIH DOBIVAMO KORISTEĆI ARGINT I ARGSTR

KAKO TO FUNKCIONIŠE





OPEN (PATH, MODE)

U SYSFILE.C PIŠEMO FUNKCIJE SISTEMSKIH POZIVA

U SYSCALL.H DODAMO U DEFINE FUNKCIJU SA PREFIKSOM SYS_XYZ.... I BROJ KOJI IDE UEĆ POSTAVLJENIM REDOSLIJEDOM

ZATIM U SYSCALL.C FUNKCIJU IZ SYSFILE.C DODAJEMO KAO EXTERN DODATAK, I STAVLJAMO U MAP U SYSCALLS

```
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);

static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
```

I UNUTAR USER.H DEFINIŠEMO ŠTA NAŠA FUNKCIJA PRIMA

NA KRAJU U MAKEFILE-U DODAMO KORISNIČKI PROGRAM KOJI SMO NAPISALI U USER FOLDERU

EVO KAKO TO SVE IZGLEDA KADA RADI:

KORISNIČKI PROGRAM

```
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user.h"
#include "kernel/fs.h"

int
main(int argc, char *argv[])
{
  hello();
  exit();
}
```

OVO NAS IZ KORISNIČKOG PROGRAMA VODI OVDJE

USYS.S

```
#include "kernel/syscall.h"
#include "kernel/traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(hello)
```

OVDJE DOBIJA .ASM TIJELO SISTEMSKOG POZIVA

USER.H

```
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int hello(int n);
```

DEFINISANO IZGLEDA NAM JE FORMATA FUNKCIJA

SYSCALL.H

SYSCALL.C

```
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_hello  22
```

OVDJE POZIVU BROJ

ZATIM KROZ TRAP.C SE POZIVA SISTEMSKI, POZIV ODGOVARAJUĆEG BROJA

DODIJELJUJEMO

```
extern int sys_uptime(void);
extern int sys_hello(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_hello]   sys_hello,
};
```

SYSFILE.C

```
int sys_hello(void)
{
        int n;

        if(argint(0, &n) < 0) return -1;

        int i;
        for(i = 0; i < n; i++)
        {
                cprintf("Hello world\n");
        }

        return 0;
}
```

UNUTAR MAKEFILE SCRIPTE

\+

DODAMO KORISNIČKI PROGRAM

TO JE TO

FILE STRUKTURA

FILE.H

```
1  struct file {
2      enum { FD_NONE, FD_PIPE, FD_INODE } type;
3      int ref; // reference count
4      char readable;
5      char writable;
6      struct pipe *pipe;
7      struct inode *ip;
8      uint off;
9  };
```

RAD SA FAJLOVIMA U FILE.C I SYSFILE.C

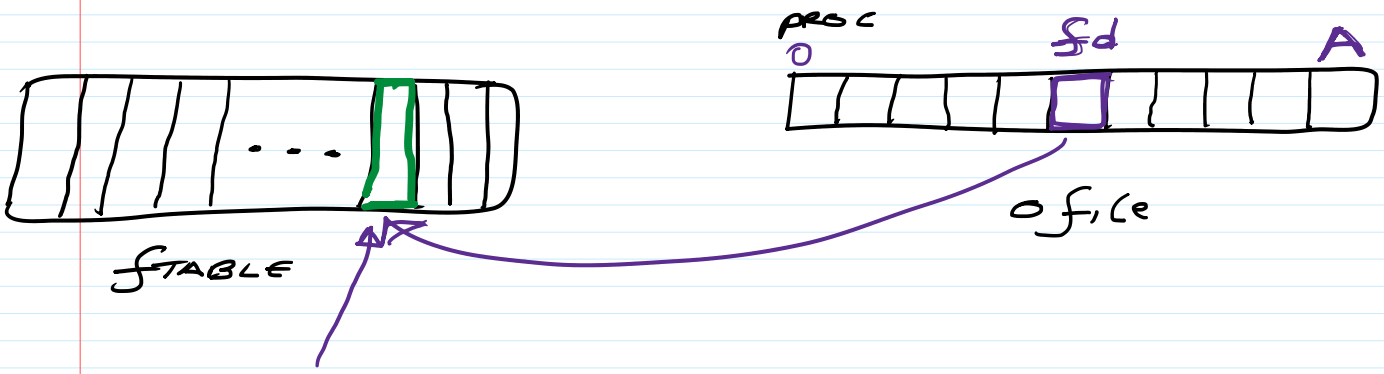
FCNTL.H

```
1  #define O_RDONLY  0x000
2  #define O_WRONLY  0x001
3  #define O_RDWR    0x002
4  #define O_CREATE  0x200
5
```

IMAMO FUNKCIJE OPEN, READ, WRITE

FTABLE

FTABLE



f → INODE = IP

f → . . . =

f → . . . =

RETURN fd ← file descriptor

```
sys_read(void)
{
    struct file *f;
    int n;
    char *p;

    if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)
        return -1;
    return fileread(f, p, n);
}
```

```
int
fileread(struct file *f, char *addr, int n)
{
    int r;

    if(f->readable == 0)
        return -1;
    if(f->type == FD_PIPE)
        return piperead(f->pipe, addr, n);
    if(f->type == FD_INODE){
        ilock(f->ip);
        if((r = readi(f->ip, addr, f->off, n)) > 0)
            f->off += r;
        iunlock(f->ip);
        return r;
    }
    panic("fileread");
}
```

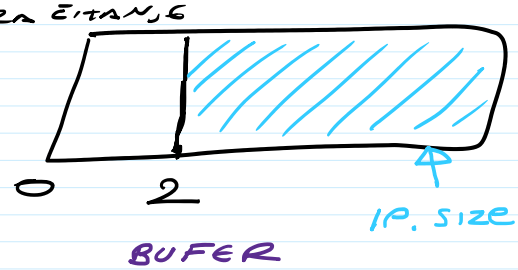
```
int
readi(struct inode *ip, char *dst, uint off, uint n)
{
    uint tot, m;
    struct buf *bp;

    if(ip->type == T_DEV){
        if(ip->major < 0 || ip->major >= NDEV || !devsw[ip->major].read)
            return -1;
        return devsw[ip->major].read(ip, dst, n);
    }

    if(off > ip->size || off + n < off)
        return -1;
    if(off + n > ip->size)
        n = ip->size - off;

    for(tot=0; tot<n; tot+=m, off+=m, dst+=m){
        bp = bread(ip->dev, bmap(ip, off/BSIZE));
        m = min(n - tot, BSIZE - off%BSIZE);
        memmove(dst, bp->data + off%BSIZE, m);
        brelse(bp);
    }
    return n;
}
```

OFF = OFFSET PROMIDENJIVA KOJA SE INKREMENTUJE ZA ČITANJE



BUFER

TOT → TOTAL NUMBER OF BYTES READ FROM FILE/INODE