

NATALIA ΡΟΥΣΚΑ

1092581

20/10/2024

Δραστηριότητα 1: Εισαγωγή στο xv6 Kernel & Προσθήκη System Call

ΑΣΚΗΣΗ 1

Οι κλήσεις συστήματος που είναι διαθέσιμες στα προγράμματα χρήστη βρίσκονται στο header file *user.h* του φακέλου *include*.



```
GNU nano 7.2 user.h *
struct stat;
struct pstat;

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

// ulib.c
int stat(char*, struct stat*);
char* strcpy(char*, char*);
void *memmove(void*, void*, int);
char* strchr(const char*, char c);
int strcmp(const char*, const char*);
void printf(int, char*, ...);
char* gets(char*, int max);
uint strlen(char*);
void* memset(void*, int, uint);
void* malloc(uint);
void free(void*);
int atoi(const char*);
```

Από τον φάκελο *user* βλέπω τα προγράμματα χρήστη και επιλέγω το *kill.c*

```
natalia@debian:~/xv6/user$ ls
cat.c      grep.c    ln.c      mkdir.c   rm.c      stressfs.c  zombie.c
echo.c     init.c    lottery.c myprogram.c schedtest.c usertests.c
forktest.c kill.c    ls.c      _         program_getfavnum.c sh.c        wc.c
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ

1092581

20/10/2024

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

int
main(int argc, char **argv)
{
    int i;

    if(argc < 1){
        printf(2, "usage: kill pid...\n");
        exit();
    }
    for(i=1; i<argc; i++)
        kill(atoi(argv[i]));
    exit();
}
```

Αν ο αριθμός των ορισμάτων (argc) είναι μικρότερος από 1 τυπώνεται το μήνυμα *"usage: kill pid..."* στο standard error stream ώστε να καταλάβει ο χρήστης ότι πρέπει να περάσει ως όρισμα το process id και το πρόγραμμα τερματίζεται με κλήση της system call *exit()*. Αλλιώς για κάθε όρισμα στο argv (argument vector) καλείται η system call *kill(pid)* από το kernel, όπου pid ακέραιος. Η *atoi()* μετατρέπει το αριθμητικό string (όρισμα) σε integer. Η *kill(int pid)* βρίσκεται στο *kernel/proc.c* και σκοτώνει τη διεργασία με το δοσμένο pid. Το πρόγραμμα τερματίζει με την *exit()* όταν επιστρέψει σε user mode.

ΑΣΚΗΣΗ 2

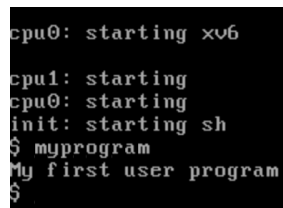
Δημιουργώ στο φάκελο *xv6/user* με την εντολή *touch myprogram.c* το παρακάτω πρόγραμμα χρήστη

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

int main(int argc, char* argv[])
{
    printf(1, "My first user program\n");
    exit();
}
```

Για να προστεθεί το πρόγραμμα στο image του xv6, προσθέτω στη λίστα *USER_PROGS* του *Makefile* το όνομα του προγράμματος χρήστη και με τις εντολές *make //compile ; make qemu //run on qemu* το λειτουργικό σύστημα xv6 τρέχει στον QEMU emulator και βλέπω το αποτέλεσμα του προγράμματος που δημιούργησα.

```
USER_PROGS=\
cat\
echo\
forktest\
schedtest\
grep\
init\
kill\
ln\
ls\
mkdir\
rm\
sh\
stressfs\
usertests\
wc\
zombie\
lotteryschedtest\
myprogram\
```



```
cpu0: starting xv6
cpu1: starting
cpu0: starting
init: starting sh
$ myprogram
My first user program
$
```

NATALIA ΡΟΥΣΚΑ

1092581

20/10/2024

ΑΣΚΗΣΗ 3

Με την εντολή `grep -r (global regular expression print)` μπορούμε να βρούμε την αναφορά της `uptime` στον τρέχοντα κατάλογο και σε όλους τους υποκαταλόγους.

```
natalia@debian:~/xv6$ grep -r "uptime" *
include/syscall.h:#define SYS_uptime 14
include/user.h:int uptime(void);
kernel/syscall.c:extern int sys_uptime(void);
kernel/syscall.c:[SYS_uptime] sys_uptime,
kernel/sysproc.c:sys_uptime(void)
ulib/usys.S:SYSCALL(uptime)
```

Η υλοποίηση της `uptime` βρίσκεται στο αρχείο `sysproc.c` του φακέλου `xv6/kernel`.

```
// return how many clock tick interrupts have occurred
// since start.
int
sys_uptime(void)
{
    uint xticks;

    acquire(&tickslock);
    xticks = ticks;
    release(&tickslock);
    return xticks;
}
```

Με την `acquire(&tickslock)` κλειδώνω την πρόσβαση στη μεταβλητή `ticks` ώστε να μη μπορεί να χρησιμοποιηθεί από άλλες διεργασίες. Η `xticks` παίρνει την τιμή της global μεταβλητής `ticks` που αυξάνεται κάθε φορά που το ρολόι προκαλεί διακοπή. Οπότε μετράει πόσες διακοπές ρολογιού έχουν γίνει από τη στιγμή που ξεκίνησε το σύστημα. Με την `release(&tickslock)` απελευθερώνω τη πρόσβαση στη μεταβλητή `ticks` και από άλλες διεργασίες.

ΑΣΚΗΣΗ 4

Ένα πρόγραμμα χρήστη καλεί κάποια system call, ο ορισμός της οποίας βρίσκεται στο `user.h`

Έπειτα εκτελείται ο assembly κώδικας που βρίσκεται στο `ulib/usys.S`

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

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

Ο αριθμός που αντιστοιχεί στην συγκεκριμένη system call αποθηκεύεται στο καταχωρητή `eax`. Οι αριθμοί `SYS_name` ορίζονται για κάθε system call στο `/include/syscall.h`

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ

1092581

20/10/2024

Καλείται διακοπή trap στον επεξεργαστή και ο έλεγχος περνάει στο kernel. Καλείται η συνάρτηση `syscall()` που υλοποιείται στο `kernel/syscall.c`

```
void
syscall(void)
{
    int num;

    num = proc->tf->eax;
    if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
        proc->tf->eax = syscalls[num]();
    } else {
        cprintf("%d %s: unknown sys call %d\n",
            proc->pid, proc->name, num);
        proc->tf->eax = -1;
    }
}

int sys_settickets(void) {
    return -1;
}
```

Το περιεχόμενο του καταχωρητή `eax` περνάει στη μεταβλητή `num`. Αν ο αριθμός είναι έγκυρος καλείται η `syscalls[num]()`, η οποία ανατρέχει στον πίνακα με τους δείκτες σε συναρτήσεις που φαίνεται παρακάτω και καλείται η αντίστοιχη συνάρτηση π.χ. `extern int sys_kill(void)`; που υλοποιείται στο `sysproc.c`

Οι συναρτήσεις `sys_name` υλοποιούνται στα αρχεία `sysproc.c` ή `sysfile.c`

```
static int (*syscalls[])(void) = {
[SYS_getfavnum] sys_getfavnum,
[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_getpinfo] sys_getpinfo,
[SYS_settickets] sys_settickets
};
```

ΑΣΚΗΣΗ 5

- Αντιστοίχιση της `getfavnum()` με τον αριθμό 28 στο `syscall.h`

```
#define SYS_getfavnum 28
```

- Στο `syscall.c` προσθέτουμε τον ορισμό της και στον πίνακα από function pointers το ζεύγος `[SYS_getfavnum] sys_getfavnum`

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ

1092581

20/10/2024

```
static int (*syscalls[])(void) = {
[SYS_getfavnum] sys_getfavnum,
[SYS_fork]      sys_fork,
[SYS_exit]     sys_exit,
[SYS_wait]     sys_wait,
[SYS_pipe]     sys_pipe,
[SYS_read]     sys_read,
[SYS_kill]     sys_kill,
extern int sys_getfavnum(void);
```

- Στο sysproc.c υλοποιούμε την συνάρτηση

```
int
sys_getfavnum(void)
{
    return 10;
}
```

- Δημιουργώ το πρόγραμμα χρήστη program_getfavnum.c που καλεί τη system call getfavnum() και το προσθέτω στη λίστα *USER_PROGS* του *Makefile*. Προσθέτω τον ορισμό της system call για να είναι διαθέσιμη στον χρήστη στο user.h

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

int main(void)
{
    printf(1, "My favourite number is %d\n", getfavnum());
    exit();
}

// system calls
int getfavnum(void);
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
```

- Προσθέτω SYSCALL(getfavnum) στο usys.S

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

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

Με τις εντολές *make //compile ; make qemu //run on qemu* το λειτουργικό σύστημα xv6 τρέχει στον QEMU emulator και βλέπω το αποτέλεσμα του προγράμματος που δημιούργησα.

```
$ program_getfav
My favourite number is 10
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ

1092581

20/10/2024

ΑΣΚΗΣΗ 6

- Αντιστοίχιση της `halt()` με τον αριθμό 29 στο `syscall.h`
- Στο `syscall.c` προσθέτουμε τον ορισμό της και στον πίνακα από function pointers το ζεύγος `[SYS_halt] sys_halt`
- Στο `sysproc.c` υλοποιούμε την συνάρτηση. Με την `outw()` γράφω στην διεύθυνση `0x604` την τιμή `0x2000` για να κλείσει ο emulator.

```
int
sys_halt(void)
{
    outw(0x604, 0x2000);
    return 0;
}
```

- Δημιουργώ το πρόγραμμα χρήστη `shutdown.c` που καλεί τη system call `halt()` και το προσθέτω στη λίστα `USER_PROGS` του `Makefile`. Προσθέτω τον ορισμό της system call για να είναι διαθέσιμη στον χρήστη στο user.

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

```
int main(void)
{
    halt();
    exit();
}
```

- Προσθέτω `SYSCALL(halt)` στο `usys.S`

Με την εντολή `shutdown` ο `qemu emulator` κλείνει

```
SeaBIOS (version 1.16.2-debian-1.16.2-1)
Booting from Hard Disk...
acpi: cpu#0 apicid 0
acpi: cpu#1 apicid 1
acpi: ioapic#0 @fec00000 id=0 base=0

cpu0: starting xv6
cpu1: starting
cpu0: starting
sinit: starting sh
$shutdown
```