

Λειτουργικά Συστήματα 2023-2024


1^η Δραστηριότητα

Τσάμπρας Κωνσταντίνος

up1083865

➔ Άσκηση 1η

Στο παρακάτω αρχείο (user.h) βλέπουμε τους ορισμούς των κλήσεων του συστήματος που έχει στην διάθεσή του ο χρήστης:



The screenshot shows a terminal window with the title 'ultron@debian: ~'. The editor is GNU nano 7.2, editing the file 'user.h'. The code defines several system call functions. At the bottom, there is a table of nano editor shortcuts.

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

// system calls
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*);

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Lin
```

Ίσως το πιο απλό παράδειγμα χρήσης system calls από τα προγράμματα χρήση είναι το αρχείο echo.c το οποίο χρησιμοποιεί την printf(). Αυτή με την σειρά της καλεί την write() κλήση συστήματος για να τυπώσει τους χαρακτήρες στην οθόνη.



The screenshot shows a terminal window with the title 'ultron@debian: ~'. The editor is GNU nano 7.2, editing the file 'echo.c'. The code includes 'types.h', 'stat.h', and 'user.h', and contains a main function that uses printf and exit. At the bottom, there is a table of nano editor shortcuts.

```
GNU nano 7.2 echo.c
#include "types.h"
#include "stat.h"
#include "user.h"

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

    for(i = 1; i < argc; i++)
        printf(1, "%s%s", argv[i], i+1 < argc ? " " : "\n");
    exit();
}

[ Read 13 lines ]

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

```
GNU nano 7.2 printf.c
#include <stdarg.h>

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

static void
putc(int fd, char c)
{
    write(fd, &c, 1);
}
```

➔ Άσκηση 2η

Δημιουργούμε ένα αρχείο (fib.c) στον φάκελο user όπου θα έχει την απλή λειτουργία να τυπώνει όσους αριθμούς Fibonacci του ζητήσουμε. Χρειάζεται να υλοποιήσουμε και την συνάρτηση getint() χρησιμοποιώντας την κλήση συστήματος read(), ώστε να διαβάζουμε τον ακέραιο που μας δόθηκε.

```
GNU nano 7.2 fib.c *
#include "types.h"
#include "stat.h"
#include "user.h"

int getint(){
    int num = 0;
    int is_negative = 0;
    char c;

    while(read(0, &c, 1)>0){
        if(c=='-'){
            is_negative=1;
        } else if (c>='0'&& c<='9'){
            num = num*10+ (c-'0');
        }else if (c =='\n' || c ==' ' ) {
            break;
        }else {
            printf(1, "Invalid input.");
            return -1;
        }
    }
}
```

```
GNU nano 7.2 fib.c *
    }else {
        printf(1, "Invalid input.");
        return -1;
    }
}

if (is_negative){
    num = -num;
}

return num;
}

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

Στην συνέχεια υλοποιούμε την main:

```
GNU nano 7.2                                fib.c *
int main(int argc, char *argv[])
{
    int num=0;
    printf(1, "\nHello this is my first file\n\n");
    printf(1, "Please provide the number of fibonacci numbers you

    num = getint();
    if(num<0){
        printf(1, "Invalid input.");
        return 0;
    }
    printf(1, "\nYou gave this number: %d\n", num);

    int fib1=0;
    int fib2=1;
    int current_fib=1;
    int i;
    printf(1, "The list of fibonacci numbers is this:\n\n");
    for(i=0; i<num; i++){
        printf(1, "%d\n", current_fib);

        int fib1=0;
        int fib2=1;
        int current_fib=1;
        int i;
        printf(1, "The list of fibonacci numbers is this:\n\n");
        for(i=0; i<num; i++){
            printf(1, "%d\n", current_fib);
            current_fib=fib1+fib2;
            fib1=fib2;
            fib2=current_fib;

        }
        printf(1, "\n\nDone\n");
        return 0;
    }
}
```

Αποτέλεσμα:

```
init: starting sh
$ fib

Hello this is my first file

Please provide the number of fibonacci numbers you wish to produce
5
$
You gave this number: 5
The list of fibonacci numbers is this:

1
1
2
3
5

Done
$ _
```

→ Άσκηση 3η

Με χρήση της εντολής `grep` στο directory του kernel μπορούμε να βρούμε όλες τις αναφορές του `uptime`:

```
root@debian:/xv6# cd kernel
root@debian:/xv6/kernel# grep -r 'uptime'
syscall.c:extern int sys_uptime(void);
syscall.c:[SYS_uptime] sys_uptime,
sysproc.c:sys_uptime(void)
root@debian:/xv6/kernel#
```

```
GNU nano 7.2                                syscall.c
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_getpinfo(void);
```

```
ultron@debian: ~
GNU nano 7.2                                syscall.c
[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,
```

Παρακάτω έχουμε την υλοποίηση της:

```
int
sys_uptime(void)
{
    uint xticks;
    acquire(&tickslock);
    xticks = ticks;
    release(&tickslock);
    return xticks;
}
```

Βλέπουμε ότι αρχικά «δεσμεύει» την μεταβλητή `tickslock` ώστε να αποτρέψει άλλες διεργασίες από το να έχουν ταυτόχρονη πρόσβαση στην ίδια μεταβλητή. Στην συνέχεια αποθηκεύει την τιμή της μεταβλητής, «αποδεσμεύει» την `tickslock` και επιστρέφει την τιμή. Η τιμή που επιστρέφεται δεν είναι βέβαια σε δευτερόλεπτα αλλά σε πλήθος `timing interrupts`, στην περίπτωση του `xv6` έχουμε 100 `timing interrupts` κάθε

δευτερόλεπτο, άρα αν θέλουμε να βρούμε το uptime σε δευτερόλεπτα, απλά διαιρούμε τον αριθμό που μας επιστρέφει η uptime δια 100.

➔ Άσκηση 4η

Την συνάρτηση υλοποίησης των system calls εντοπίζουμε στο αρχείο syscall.c.

Αφού οριστούν οι system call functions και το array που αντιστοιχίζει τους δείκτες αυτών των συναρτήσεων σε αριθμούς από 1-21, ορίζεται η συνάρτηση syscall:

```
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;
    }
}
```

Αρχικά παίρνει τον αναγνωριστικό αριθμό (num) της συνάρτησης syscall που επιθυμούμε να καλέσουμε τον οποίο (σύμφωνα με την διαδικασία κλήσης συστήματος) έχουμε αποθηκεύσει στον καταχωρητή eax.

Στην συνέχεια ελέγχει αν ο αριθμός βρίσκεται στα αναμενόμενα όρια (από 1 έως το μήκος του array των system calls) και αν έχει δηλωθεί σωστά ο δείκτης στην αντίστοιχη συνάρτηση (π.χ. δεν είναι NULL pointer).

Αν ο έλεγχος επιτύχει, καλείται η system call με αναγνωριστικό αριθμό num. Το αποτέλεσμα της κλήσης αποθηκεύεται πάλι στον καταχωρητή eax σύμφωνα με το πρωτόκολλο.

Αν ο έλεγχος αποτύχει τυπώνεται αντίστοιχο μήνυμα και επιστρέφεται -1.

➔ Άσκηση 5η

1^η αλλαγή: Προσθήκη στο αρχείο syscall.h

```
GNU nano 7.2                                syscall.h *
```

```
#define SYS_mkdir 20
#define SYS_close 21
#define SYS_getpinfo 24

#define SYS_getfavnum 22
```

2^η αλλαγή: Προσθήκη στο defs.h κάτω από το proc.c

```
GNU nano 7.2                                defs.h *
```

```
//PAGEBREAK: 16
// proc.c
struct proc*  copyproc(struct proc*);
void          exit(void);
int          fork(void);
int          growproc(int);
int          kill(int);
void         pinit(void);
void         procdump(void);
void         scheduler(void) __attribute__((noreturn));
void         sched(void);
void         sleep(void*, struct spinlock*);
void         userinit(void);
int          wait(void);
void         wakeup(void*);
void         yield(void);

int          getfavnum(void);
```

3^η αλλαγή: Προσθήκη στο user.h

```
GNU nano 7.2                                user.h
```

```
int dup(int);
int getpid(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
int getpinfo(struct pstat*);

int getfavnum(void);
```

4^η αλλαγή: Προσθήκη κλήσης στο sysproc.c

```
GNU nano 7.2 sysproc.c *
}

// return how many clock tick interrupts have occurred
// since start.
int
sys_uptime(void)
{
    uint xticks;
    acquire(&tickslock);
    xticks = ticks;
    release(&tickslock);
    return xticks;
}

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

5^η αλλαγή: usys.S

```
GNU nano 7.2 usys.S
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(getpinfo)

SYSCALL(getfavnum)
```

6^η αλλαγή: syscall.c

```
ultron@debian: ~
GNU nano 7.2 syscall.c *
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
extern int sys_getpinfo(void);

extern int sys_getfavnum(void);

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



```
Activities Terminal Oct 21 22:32
ultron@debian: ~
GNU nano 7.2 syscall.c *
[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_getfavnum] sys_getfavnum,
};
```

7^η αλλαγή: Γράφουμε την ίδια την συνάρτηση στο proc.c

```
Activities Terminal Oct 21 22:34
ultron@debian: ~
GNU nano 7.2 proc.c
else
    state = "???";
    cprintf("%d %s %s", p->pid, state, p->name);
    if(p->state == SLEEPING){
        getstackpcs((uintp*)p->context->ebp, pc);
        for(i=0; i<10 && pc[i] != 0; i++)
            cprintf(" %p", pc[i]);
    }
    cprintf("\n");
}

cprintf("\n\n");
}

int
getfavnum(void){
    return 420;
}

[ Wrote 478 lines ]
```

Έπειτα απλά δοκιμάζουμε να την καλέσουμε σε ένα πρόγραμμα.

```
GNU nano 7.2 favnum.c
#include "types.h"
#include "stat.h"
#include "user.h"

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

    printf(1, "Calling getfavnum: %d\n", getfavnum());
    exit();
}

init: starting sh
$ favnum
Calling getfavnum: 420
$
```

➔ Άσκηση 6η

Σύμφωνα με την παραπάνω διαδικασία δημιουργούμε το system call. Αυτή την φορά η υλοποίηση γίνεται με την εντολή `outw(0x604, 0x2000);`

```
GNU nano 7.2                                proc.c
    for(i=0; i<10 && pc[i] != 0; i++)
        cprintf(" %p", pc[i]);
    }
    cprintf("\n");
}

    cprintf("\n\n");
}

int
getfavnum(void){
    return 420;
}

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

Η κλήση γίνεται μέσω του προγράμματος χρήστη “halting” (halting.c):

```
GNU nano 7.2                                halting.c *
#include "types.h"
#include "stat.h"
#include "user.h"

int main(int argc, char *argv[]){
    char answer='n';

    printf(1, "\nAre you sure you want to halt? (y/n)\n");
    read(0, &answer, 1);
    if(answer=='y' || answer=='Y')
    {
        printf(1, "\n\nHalting!\n");
        halt();
        return -1;
    }
}
```

[Read 21 lines]

Αποτέλεσμα:

```
QEMU
Machine View
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
init: starting sh
$ halting

Are you sure you want to halt? (y/n)
```

```
$ halting
Are you sure you want to halt? (y/n)
y

Halting!
root@debian:/xv6#
```

➔ Άσκηση 7η

Αυτή η άσκηση χρειάστηκε αρκετή προσπάθεια κυρίως γιατί δεν ήμουν σίγουρος αν ο τρόπος (τον οποίο ακολούθησα τελικά) είναι «σωστός». Δεν είμαι σίγουρος καν αν θεωρείται system call. Η υλοποίηση πάντως γίνεται στην syscall():

```
GNU nano 7.2          syscall.c
};

void
syscall(void)
{
    static int calls[30]={0};
    int num;
    num = proc->tf->eax;
    if(num==25)
    {
        int call;
        argint(0, &call);
        calls[num]=calls[num]+1;
        proc->tf->eax = calls[call];
    }
    else{

        if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
```

```

else{

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

```

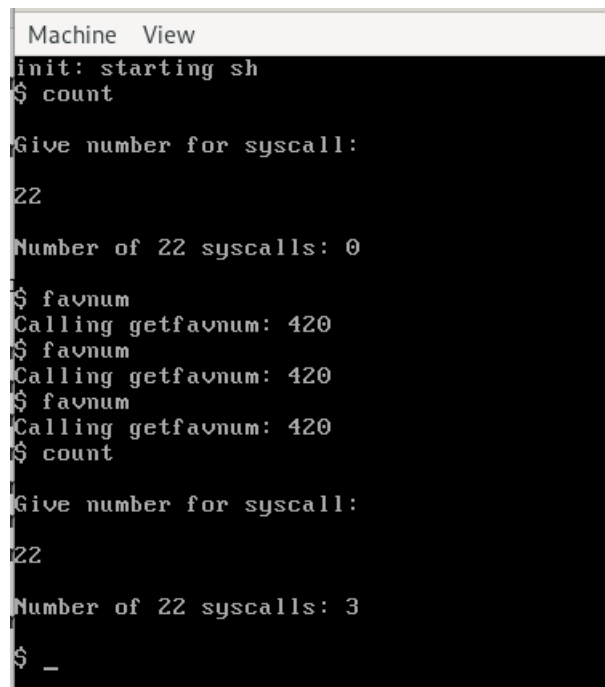
Όπως φαίνεται παραπάνω, χρησιμοποιούμε μία static array από integers στο επίπεδο της syscall() όπου αποθηκεύουμε τον αριθμό κλήσεων της κάθε syscall στην θέση που υποδεικνύει ο κωδικός της μέσα στο array. Όταν γίνεται η κλήση syscall με αριθμό 25, δεν πηγαίνουμε στην «κανονική ροή» ενός syscall, αλλά επιστρέφουμε κατευθείαν στον καταχωρητή την τιμή που πρέπει.

Στην «κανονική ροή» που προανέφερα χρειάστηκε προφανώς να προσθέσουμε το `calls[num]=calls[num]+1` για να κρατάμε τον αριθμό των κλήσεων της κάθε syscall.

Ο λόγος που χρησιμοποιούμε static για το array είναι προφανώς για να μην γίνονται reset τα δεδομένα κάθε φορά που καλείται η συνάρτηση.

Ως αποτέλεσμα έχουμε το παρακάτω:

```
#define SYS_getfavnum 22
```



```

Machine View
init: starting sh
$ count

Give number for syscall:
22

Number of 22 syscalls: 0

$ favnum
Calling getfavnum: 420
$ favnum
Calling getfavnum: 420
$ favnum
Calling getfavnum: 420
$ count

Give number for syscall:
22

Number of 22 syscalls: 3

$ _

```

```
#define SYS_write 16
```

```
$ favnum
Calling getfavnum: 420
$ favnum
Calling getfavnum: 420
$ favnum
Calling getfavnum: 420
$ count

Give number for syscall:
22

Number of 22 syscalls: 3

$ count

Give number for syscall:
16

Number of 16 syscalls: 234

$
```

➔ Άσκηση 8η

Με την πλέον γνωστή διαδικασία δημιουργούμε ένα νέο system call και παρακάτω έχουμε την υλοποίηση της συνάρτησης killrandom():

```
GNU nano 7.2      proc.c

int killrandom(void){
    static unsigned long int next =1;

    int rand(void)
    {
        next = next*1103515245+12345;
        return (unsigned int)(next/65536)%32768;
    }

    void srand(unsigned int seed)
    {
        next=seed;
    }

    struct proc *p;
    int pids[NPROC];
    int counter=0;
    acquire(&ptable.lock);
```

```
GNU nano 7.2      proc.c

struct proc *p;
int pids[NPROC];
int counter=0;
acquire(&ptable.lock);
for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
    pids[counter++]=p->pid;
}
release(&ptable.lock);

while(kill(pids[rand()%counter])===-1);

return 0;
```

Ορίζουμε μια ψευδοτυχαία συνάρτηση και έπειτα διατρέχουμε την λίστα με τα processes και αποθηκεύουμε τα ids τους σε ένα array.

Στην συνέχεια προσπαθούμε να «σκοτώσουμε» μια τυχαία διεργασία επιλέγοντας ένα τυχαίο pid και στέλνοντας το στην kill(). Αν για κάποιο λόγο δεν δουλέψει την πρώτη φορά (πχ το process διαγράφηκε από το table) διαλέγουμε ένα άλλο τυχαίο pid και επαναλαμβάνουμε.