

Project 3: Kernel Hacking

Määritelmä

Projektin tarkoituksena oli lisätä uusi järjestelmäkutsu xv6 käyttöjärjestelmään. Kyseinen järjestelmäkutsu, *getreadcount()* palauttaa tiedon siitä kuinka monta kertaa *read()* järjestelmäkutsua on kutsuttu käyttäjä prosessien toimesta siitä lähtien kun ydin on käynnistetty. Lisäksi *getreadcount()* järjestelmä kutsua oli mahdollista laajentaa toimimaan muidenkin järjestelmäkutsujen kanssa ja myös niin että laskurin voi resetoida.

Järjestelmäkutsun implementointi vaati monen xv6 järjestelmä tiedoston muokkausta: *syscall.h*, *syscall.c*, *user.h*, *usys.S*, *sysproc.c*. Lisäksi loin uuden ohjelman *test_count.c*, jota varten *Makefile*en tuli lisätä maininta.

test_count.c

Test_count.c on C kielinen ohjelman, jolla uutta järjestelmäkutsua voidaan testata. Ohjelmalle voi antaa 0-2 parametria, jos sille ei anneta yhtään parametria ohjelma kutsuu *getreadcount()* järjestelmäkutsua perusarvoilla, eli se palauttaa vain tällä hetkellä seurattavan järjestelmäkutsun laskurin arvon, joka seuraa oletuksena *read()* kutsuja, ja tulostaa kyseisen arvon. Jos parametreja on yksi ohjelma tarkistaa onko annettu parametri '-r' flagi, joka resatoi laskurin, jos ei niin ohjelma kutsuu *getreadcount()*:ia siten, että se vaihtaa seurattavaa järjestelmäkutsua vastaamaan annettua numeroarvoa. Jos ohjelmalle annetaan kaksi parametria, resetoidaan laskuri ja vaihdetaan seurattava järjestelmäkutsu. Tässä tapauksessa resetointi on oikeastaan turhaa, koska seurattavan kutsun vaihtaminen resatoi myös laskurin. Ohessa on kuva *test_count.c* tiedostosta.

Kuva: test_count.c

```
~/Ohjelmointi/C/CT30A3370/Projektit/kernel_hacking/xv6-public/test_count.c - Sublim
test_count.c x
1 // Calls getreadcount to get the number of
2 // times a tracked syscall has been called.
3 // Currently tracked syscall can be changed by
4 // giving the number of the new syscall to track.
5 // Argument -r will reset counter.
6 #include "types.h"
7 #include "stat.h"
8 #include "user.h"
9
10 int
11 main(int argc, char **argv)
12 {
13     int i = 0;
14     if (argc == 1) {
15         // Gets counter for currently tracked syscall
16         i = getreadcount(0, 0);
17     } else if (argc == 2) {
18         if (strcmp(argv[1], "-r") == 0) { // Resets counter
19             i = getreadcount(0, 1);
20         } else {
21             // Changes currently tracked syscall
22             i = getreadcount(atoi(argv[1]), 0);
23         }
24     } else if (argc == 3) {
25         if (strcmp(argv[2], "-r") == 0) { // Resets counter
26             i = getreadcount(atoi(argv[1]), 1);
27         } else {
28             // Changes currently tracked syscall
29             i = getreadcount(atoi(argv[1]), 0);
30         }
31     } else {
32         printf(1, "test_count: [syscalls[num]] [-r to reset counter]\n");
33         exit();
34     }
35
36     printf(1, "Count: %d\n", i);
37     exit();
38 }
```

Makefile

Jotta test_count.c käännetään makefilea ajettaessa tuli tiedostoon lisätä maininta test_count.c tiedostosta. Ohessa kuva.

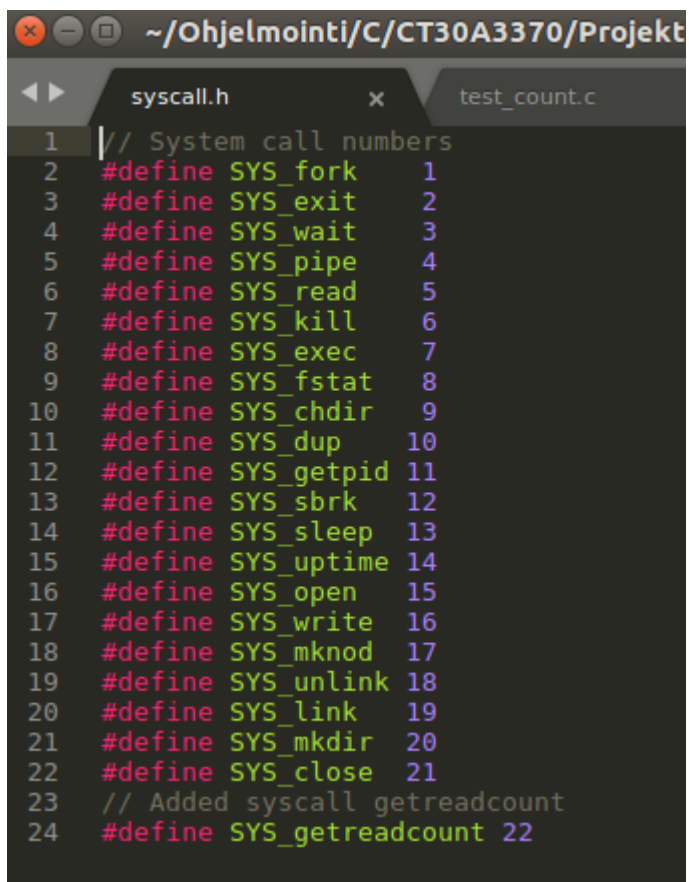
Kuva: Makefile

```
247 # after running make dist, probably want to
248 # rename it to rev0 or rev1 or so on and then
249 # check in that version.
250
251 EXTRA=\
252     mkfs.c ulib.c user.h cat.c echo.c forkttest.c grep.c kill.c\
253     ln.c ls.c mkdir.c rm.c stressfs.c test_count.c usertests.c wc.c zombie.c\
254     printf.c umalloc.c\
255     README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
256     .gdbinit.tmpl gdbutil\
257
258 dist:
259     rm -rf dist
260     mkdir dist
```

syscall.h, usys.S, user.h

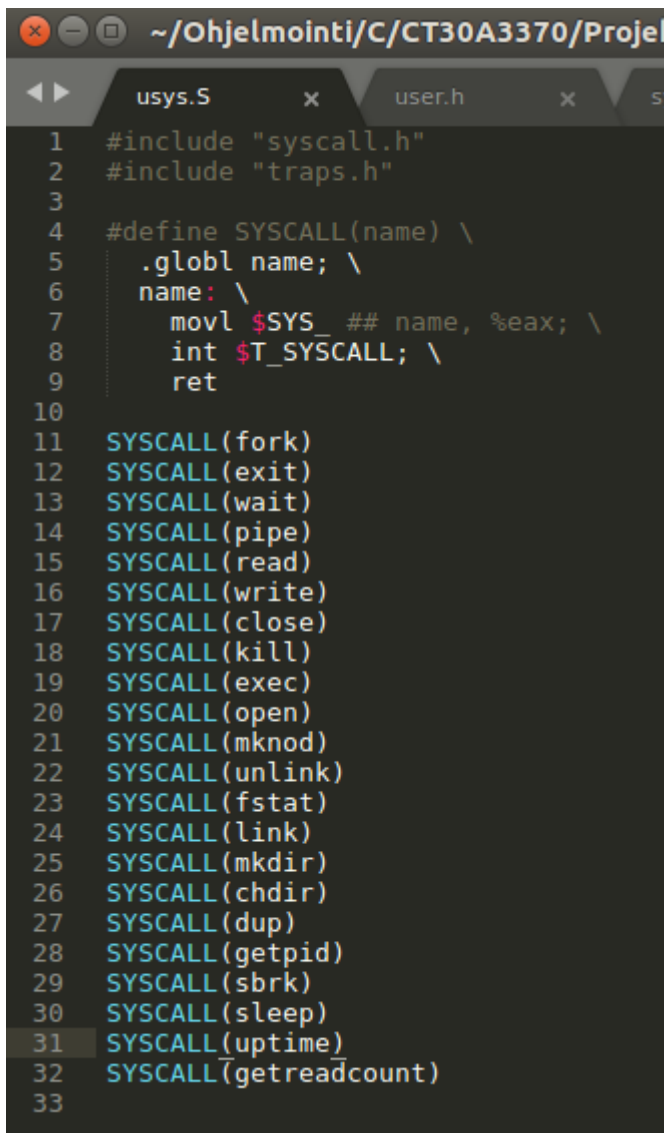
Jokaisesta järjestelmäkutsusta tulee olla maininnat näissä tiedostoissa, joten lisäsin ne maininnat *getreadcount()* kutsusta. Syscall.h tiedostossa on määritelty, millä numerolla järjestelmäkutsut tunnistetaan oletuksena ne ovat 1-21 ja kun lisäsin uuden annoin sille numeron 22. Usys.S tiedostossa on määritelty makro SYSCALL(name) ja lisäksi siellä on maininnat jokaisesta järjestelmäkutsusta, joten lisäsin maininnan uudelle kutsulle. User.h tiedostossa on julistettu järjestelmäkutsut, eli määritelty minkä tyyppin se palauttaa ja millä parametreilla sitä kutsutaan. Koska halusin että uuden kutsun parametrarit ovat kaksi kokonaislukua lisäsin user.h tiedostoon rivin *int getreadcount(int, int);*. Ohessa on kuvat näistä kolmesta tiedostosta, joissa näkyvät tehdyt lisäykset.

Kuva: syscall.h



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

Kuva: usys.S



```
1  #include "syscall.h"
2  #include "traps.h"
3
4  #define SYSCALL(name) \
5      .globl name; \
6      name: \
7          movl $SYS_ ## name, %eax; \
8          int $T_SYSCALL; \
9          ret
10
11 SYSCALL(fork)
12 SYSCALL(exit)
13 SYSCALL(wait)
14 SYSCALL(pipe)
15 SYSCALL(read)
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
19 SYSCALL(exec)
20 SYSCALL(open)
21 SYSCALL(mknod)
22 SYSCALL(unlink)
23 SYSCALL(fstat)
24 SYSCALL(link)
25 SYSCALL(mkdir)
26 SYSCALL(chdir)
27 SYSCALL(dup)
28 SYSCALL(getpid)
29 SYSCALL(sbrk)
30 SYSCALL(sleep)
31 SYSCALL(uptime)
32 SYSCALL(getreadcount)
33
```

Kuva: user.h



```
1 struct stat;
2 struct rtcdate;
3
4 // system calls
5 int fork(void);
6 int exit(void) __attribute__((noreturn));
7 int wait(void);
8 int pipe(int*);
9 int write(int, const void*, int);
10 int read(int, void*, int);
11 int close(int);
12 int kill(int);
13 int exec(char*, char**);
14 int open(const char*, int);
15 int mknod(const char*, short, short);
16 int unlink(const char*);
17 int fstat(int fd, struct stat*);
18 int link(const char*, const char*);
19 int mkdir(const char*);
20 int chdir(const char*);
21 int dup(int);
22 int getpid(void);
23 char* sbrk(int);
24 int sleep(int);
25 int uptime(void);
26 // Added syscall getreadcount
27 int getreadcount(int, int);
28
29 // ulib.c
30 int stat(const char*, struct stat*);
31 char* strcpy(char*, const char*);
32 void *memmove(void*, const void*, int);
33 char* strchr(const char*, char c);
34 int strcmp(const char*, const char*);
35 void printf(int, const char*, ...);
36 char* gets(char*, int max);
37 uint strlen(const char*);
38 void* memset(void*, int, uint);
39 void* malloc(uint);
40 void free(void*);
41 int atoi(const char*);
42
```

sysproc.c

Sysproc.c tiedostossa on määritelty itse järjestelmäkutsu, eli `sys_getreadcount` funktio. Lisäsin tiedostoon myös kaksi globaalia muuttujaa `int callCount`, `callToFollow`, joita käytetään määrittämään seurattava järjestelmäkutsu, (oletus on 5, eli `read()`) ja kutsujen

määrä, eli laskuri. Funktio on varsin yksinkertainen se käyttää kahta lokaalia muuttujaa, *int reset*, *follow*, jotka määrittävät resetoitako laskuri ja mitä kutsua ruvetaan seuraamaan. Näihin muuttujiin saadaan arvot *argint* funktiolla, jolla saadaan välitettyä kokonaisluku parametreja järjestelmäkutsuihin, eli ne ottavat arvot, joilla järjestelmäkutsua kutsuttiin. Jos reset saa arvon 1, globaali laskuri resetoitetaan ja jos follow saa arvon, joka on suurempi kuin 0 ja eri kuin tällä hetkellä seurattavan järjestelmäkutsun numeroarvo, resetoitetaan laskuri ja vaihdetaan seurattavaa järjestelmäkutsua, muuttamalla globaalin *callToFollow* muuttujan arvoa. Lopuksi palautetaan laskurin arvo. Jotta saataisiin laskettua järjestelmäkutsuja, tulee muokata vielä yhtä tiedostoa. Ohessa on kuva sysproc.c tiedostosta.

Kuva: sysproc.c

```
~/Ohjelmointi/C/CT30A3370/Projektit/kernel_hacking/xv6-public/s
sysproc.c x syscall.h x test_count.c x M
1  #include "types.h"
2  #include "x86.h"
3  #include "defs.h"
4  #include "date.h"
5  #include "param.h"
6  #include "memlayout.h"
7  #include "mmu.h"
8  #include "proc.h"
9
10 int callCount = 0; // Initialize syscall counter
11 int callToFollow = 5; // Default syscall, read, to count
12
13 // Return how many times a tracked syscall
14 // has been called
15 int
16 sys_getreadcount(void)
17 {
18     int reset, follow;
19     argint(0, &follow);
20     argint(1, &reset);
21
22     if(reset == 1) {
23         // Reset counter if argument given
24         callCount = 0;
25     }
26     if(follow > 0 && follow != callToFollow) {
27         // Change currently tracked syscall and reset counter
28         callToFollow = follow;
29         callCount = 0;
30     }
31     return callCount;
32 }
33
34 int
35 sys_fork(void)
36 {
37     return fork();
38 }
```


syscall.c

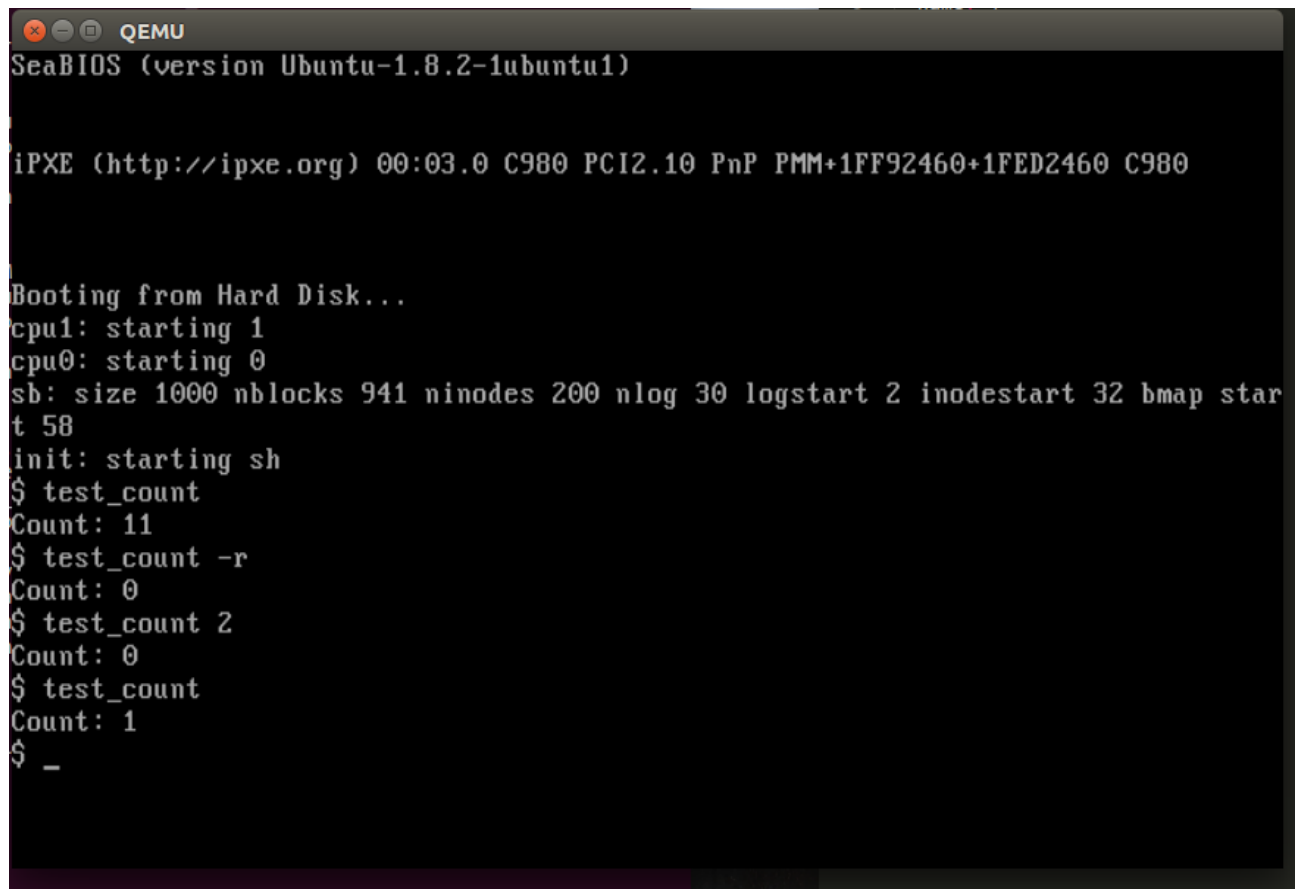
Tässä tiedostossa on maininnat, jokaisesta sysproc.c tai sysfile.c tiedostoissa määritellystä sys_järjestelmäkutsusta ja lisäksi taulukko, jossa on järjestelmäkutsujen numerot, joten lisäsin uuden järjestelmäkutsun molempiin. Lisäsin lisäksi maininnan sysproc.c tiedostossa määrittämistäni kahdesta globaalista muuttujasta: laskurista ja seurattavasta kutsusta. Tässä tiedostossa on myös funktio *syscall()*, jonka läpi jokainen järjestelmäkutsu kulkee, joten lisäsin kyseiseen funktioon if lauseen, joka tarkastaa onko kutsuttu järjestelmäkutsu se, jota seurataan tällä hetkellä ja jos on niin laskuria kasvatetaan yhdellä. Näin saadaan pidettyä yllä laskuria, jota *getreadcount()* järjestelmä kutsu voi käyttää ja palauttaa tiedon kutsuvalle ohjelmalle. Ohessa on kuva syscall.c tiedostosta ja uuden järjestelmäkutsun testaamisesta.

Kuva: syscall.c



```
103 extern int sys_wait(void);
104 extern int sys_write(void);
105 extern int sys_uptime(void);
106 // Added syscall getreadcount
107 extern int sys_getreadcount(void);
108
109 // Added variables for getreadcount
110 extern int callCount;
111 extern int callToFollow;
112
113 static int (*syscalls[])(void) = {
114 [SYS_fork]    sys_fork,
115 [SYS_exit]    sys_exit,
116 [SYS_wait]    sys_wait,
117 [SYS_pipe]    sys_pipe,
118 [SYS_read]    sys_read,
119 [SYS_kill]    sys_kill,
120 [SYS_exec]    sys_exec,
121 [SYS_fstat]   sys_fstat,
122 [SYS_chdir]   sys_chdir,
123 [SYS_dup]     sys_dup,
124 [SYS_getpid]  sys_getpid,
125 [SYS_sbrk]    sys_sbrk,
126 [SYS_sleep]   sys_sleep,
127 [SYS_uptime]  sys_uptime,
128 [SYS_open]    sys_open,
129 [SYS_write]   sys_write,
130 [SYS_mknod]   sys_mknod,
131 [SYS_unlink]  sys_unlink,
132 [SYS_link]    sys_link,
133 [SYS_mkdir]   sys_mkdir,
134 [SYS_close]   sys_close,
135 [SYS_getreadcount] sys_getreadcount,
136 };
137
138 void
139 syscall(void)
140 {
141     int num;
142     struct proc *curproc = myproc();
143
144     num = curproc->tf->eax;
145     if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
146         // Check if syscall being called is being tracked for getreadcount
147         if(num == callToFollow) {
148             // Increment counter for getreadcount
149             callCount++;
150         }
151         curproc->tf->eax = syscalls[num]();
152     } else {
153         cprintf("%d %s: unknown sys call %d\n",
154             curproc->pid, curproc->name, num);
155         curproc->tf->eax = -1;
156     }
157 }
158
```

Kuva: kutsun testaaminen



```
QEMU
SeaBIOS (version Ubuntu-1.8.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF92460+1FED2460 C980

Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ test_count
Count: 11
$ test_count -r
Count: 0
$ test_count 2
Count: 0
$ test_count
Count: 1
$ _
```