

Laboratorul 3

Implementare funcții sistem

1 Compilare kernel

Kernelul OpenBSD se găsește în `/bsd`, iar, în principiu, codul sursă din care este compilat se găsește în `/sys`. Verificați existența celui director prin comanda:

```
# cd /sys
```

Dacă el nu există, codul sursă se poate descărca sub forma fișierului `sys.tar.gz`, de pildă, de la adresa

<https://cdn.openbsd.org/pub/OpenBSD/6.9/>

(atenție, versiunea trebuie să coincidă cu cea instalată!), iar apoi se poate salva (copia) în directorul `/tmp`. După executarea comenzilor următoare:

```
# cd /usr/src
# tar xzf /tmp/sys.tar.gz
```

codul sursă se va găsi într-adevăr în `/sys`.

Pentru a compila un kernel nou, executați următoarele comenzi (unde `$(machine)` este arhitectura pe care lucrați, cum ar fi `amd64`):

```
# cd /sys/arch/$(machine)/compile/GENERIC.MP
# make obj
# make config
# make
```

Înainte de a instala un kernel nou, este bine să faceți o copie a originalului pentru a putea reveni în cazul în care noul kernel are un defect la pornirea sistemului de operare:

```
# cp /bsd /bsd.1
```

După acest pas de siguranță, executați:

```
# make install
```

iar noul kernel va fi încărcat implicit la repornirea calculatorului cu comanda:

```
# reboot
```

2 Adăugarea unei noi funcții sistem

În OpenBSD, funcțiile de sistem sunt definite în `/sys/kern/syscalls.master`. Din acest fișier se vor genera fișiere C care definesc structurile de date, variabilele și funcțiile necesare.

De exemplu pentru funcțiile cunoscute `read(2)` și `write(2)` veți găsi în acest fișier următoarele intrări:

```
3  STD  { ssize_t sys_read(int fd, void *buf, size_t nbyte); }
4  STD  { ssize_t sys_write(int fd, const void *buf, \
                                size_t nbyte); }
```

Primul câmp reprezintă numărul de identificare a funcției de sistem, al doilea tipul (în general noi vom folosi tot timpul funcții de sistem standard STD) iar ultimul câmp este definiția C a funcției prefixată cu `sys_`.

2.1 Declararea

Adăugarea unei noi intrări se face la sfârșitul fișierului `/sys/kern/syscalls.master`. ID-ul pentru funcția noastră va fi valoarea imediat următoare celei a ultimei funcții existente. De exemplu, dacă ultimul ID este 330, noi vom folosi 331 pentru noua intrare.

```
331  STD  { int sys_khello(const char *msg); }
```

După modificarea fișierului `/sys/kern/syscalls.master` regenerați fișierele C aferente prin comanda

```
# cd /sys/kern && make syscalls
```

Fișierele generate sunt în directorul `/sys/kern` și `/sys/sys`. Modificările principale pe care le realizează comanda de mai sus sunt:

- `/sys/kern/syscalls.c` – adăugarea denumirii funcției în tabela `syscallnames`
- `/sys/sys/syscallargs.h` – definiția structurii ce va conține argumentele funcției sistem

```
struct sys_khello_args {
    syscallarg(const char *) msg;
};
```
- `/sys/sys/syscall.h` – definirea noului ID

```
#define SYS_khello 331
```

Declarația funcției are loc tot în `/sys/sys/syscallargs.h`

```
int sys_khello(struct proc *p, void *v, register_t *retval);
```

iar argumentele reale (din perspectiva kernelului) sunt

- `struct proc *p` – procesul care apelează
- `void *v` – pointer către structura `sys_khello_args`
- `register_t *retval` – pointer către rezultatul (ieșirea) funcției

2.2 Definirea

Funcția de sistem se definește de regulă în `/sys/kern/sys_generic.c`.

```
/*  
 * Hello system call  
 */  
int  
sys_khello(struct proc *p, void *v, register_t *retval)  
{  
    return 0;  
}
```

Atenție, această funcție întoarce un `int` care conține un cod de eroare de tipul `errno` folosit mai departe de kernel. Valoarea pe care o întoarce în userland este diferită și trebuie pusă în argumentul `retval`.

Următorul pas este să citim argumentele de la intrare de la adresa indicată de `v`. Pentru asta trebuie să folosim structura `sys_khello_args` definită mai devreme.

```
struct sys_khello_args *uap = v;
```

Conținutul structurii este comentat pentru ușurința programatorului, dar poate fi omis în funcție de gust.

Pentru a citi un argument se folosește macroul `SCARG`

```
#if _BYTE_ORDER == _BIG_ENDIAN  
#define SCARG(p, k) ((p)->k.be.datum)  
#elif _BYTE_ORDER == _LITTLE_ENDIAN  
#define SCARG(p, k) ((p)->k.le.datum)  
#else  
#error "what byte order is this machine?"  
#endif
```

care se ocupă cu încărcarea din registrul care conține adresa la care se află structura cu argumentele trimise de utilizator (vezi Cursul 2). De exemplu, pentru a obține argumentul `msg` al noului nostru syscall `khello` folosim `SCARG(uap, msg)`.

| Funcție | Apel | Descriere |
|----------------------------|---|--------------------------------|
| <code>copyin(9)</code> | <code>copyin(ubuf, kbuf, len)</code> | copiază buffer user → kernel |
| <code>copyout(9)</code> | <code>copyout(kbuf, ubuf, len)</code> | copiază buffer kernel → user |
| <code>kcopy(9)</code> | <code>kcopy(srckbuf, dstkbuf, len)</code> | copiază buffer kernel → kernel |
| <code>copyinstr(9)</code> | <code>copyinstr(ubuf, kbuf, len, &done)</code> | copiază string user → kernel |
| <code>copyoutstr(9)</code> | <code>copyoutstr(kbuf, ubuf, len, &done)</code> | copiază string kernel → user |
| <code>copystr(9)</code> | <code>copystr(kbuf, kbuf, len, &done)</code> | copiază string kernel → kernel |

Tabela 1: Funcții de copiere pentru kernel

Când avem de a face cu buffere primite din userland trebuie să le mutăm din spațiul de adresare specific procesului `p` în spațiul de adresare al kernelului. Pentru asta se pun la dispoziție seria de funcții din Tabela 1. Mecanismul trebuie folosit în cadrul funcției de sistem `khello` pentru a copia primii 100 bytes din mesajul `msg` în spațiul kernelului

```
copyinstr(SCARG(uap, msg), kmsg, 100, NULL);
```

Partea de verificare a apelului este intenționat omisă pentru simplitate. În mod normal ea trebuie să existe, dar nu face obiectul laboratorului.

3 Funcții utilitare în kernel

Atenție, în kernel nu există biblioteca C standard sau alte funcții utilitare cu care suntem obișnuiți când scriem programe în userland. Cu toate acestea, în kernelul de OpenBSD, există funcții similare cu cele din standardul de C.

De exemplu, avem funcția `printf(9)`, care se comportă similar cu funcția standard `printf(3)`. Diferența este că mesajul `printf(9)` apare în consola principală (`ttyC0`) (accesibilă prin `Ctrl+Alt+F1`; apăsați `Ctrl+Alt+F5` pentru a reveni la `ttyC4` care este folosit de mediul grafic), precum și în logul `/var/log/messages`.

Pentru alocarea și eliberarea memoriei folosiți `malloc(9)` și `free(9)`. `malloc(9)` primește două argumente în plus: tipul memoriei alocat și cum să se efectueze alocarea. De exemplu, pentru a alocă 120 bytes pentru un buffer temporar (folosit doar local în funcție) se folosește apelul

```
buf = malloc(120, M_TEMP, M_WAITOK);
```

ultimul argument anunțând că apelantul poate aștepta până se găsește memorie disponibilă. Când operațiile asupra lui `buf` s-au încheiat, acesta trebuie eliberat:

```
free(buf, M_TEMP, 120);
```

4 Apelare din userland

Cel mai rapid mod de a apela o nouă funcție de sistem creată este cu ajutorul lui `syscall(2)`. Această funcție de sistem apelează o altă funcție de sistem

cu ID-ul din primul argument. Restul argumentelor primite sunt pasate mai departe. De exemplu, pentru a apela noua funcție `khello` cu ID-ul 331 am folosi

```
syscall(331, "foo");
```

iar pentru a scrie "Hello!" pe ecran cu ajutorul funcției cunoscute `write(2)` am apela astfel

```
syscall(4, 1, "Hello!", 6);
```

unde 4 este ID-ul lui `write(2)`, conform fișierului `/sys/kern/syscalls.master`.

Apelarea elegantă cu numele funcției de sistem se face prin declararea funcției în mai multe fișiere de tip `include` din sistem. Acest pas este lăsat drept exercițiu pentru acasă.

5 Sarcini de laborator

1. Compilați un kernel nou.
2. Adăugați o funcție de sistem nouă simplă care să afișeze ceva pe ecran și demonstrați că merge apelând-o dintr-un program. Exemplu apel `syscall(id-functie, "world")`. **Atenție**, trebuie să recompilați kernelul și să reporniți sistemul de operare cu kernelul nou!
3. Modificați funcția de sistem de mai devreme să copieze un număr dat de bytes dintr-un buffer sursă într-altul destinație. La ieșire, funcția va scrie numărul de bytes copiat efectiv. Verificați intrările primite și semnalăți eventualele erori. Exemplu apel: `sz = kcp(src,dst,10)`.