1 Modifiche al kernel di MINIX 3.1.2a

1.1 Aggiunta di una chiamata di sistema

L'obiettivo è aggiungere una chiamata di sistema `foo` gestita dal server di MINIX, cioè il processo identificato dalla macro PM_PROC_NR. In realtà in `<lib.h>` viene definita la macro MM per comodità.

1a ⟨*Macro MM* 1a⟩≡

```
/* In /usr/src/include/lib.h */

#define MMPM_PROC_NR
```

Definisce:
MM, usato nella porzione 1c.
Usa PM_PROC_NR 1b.

1b ⟨*Macro PM_PROC_NR* 1b⟩≡

```
/* In /usr/src/include/minix/com.h */
/* User-space processes, that is, device drivers, servers, and INIT. */
#define PM_PROC_NR        0      /* process manager */
```

Definisce:
PM_PROC_NR, usato nella porzione 1a.

Una volta aggiunta la *syscall* identificata dal numero definito dalla macro FOO, un programma utente potrà chiamarla nel modo seguente. Si noti che il messaggio m non trasporta alcuna informazione.

1c ⟨*testfoo.c* 1c⟩≡

```
#include <lib.h>
void main(){
  message m;
  return(_syscall(MM, FOO, &m));
}
```

Usa FOO 2a e MM 1a.

1.1.1 L'elenco delle chiamate di sistema

L'elenco delle chiamate di sistema è contenuto in `<minix/callnr.h>`. Bisogna aggiornare il numero delle chiamate: da 95 diventano 96.

1d ⟨*Macro NCALLS* 1d⟩≡

```
/* In /usr/src/include/minix/callnr.h */
#define NCALLS        96     /* number of system calls allowed */
```

Usiamo poi il primo numero libero (se sono 96, si va da 0 a 95) per la nostra nuova chiama FOO

2a ⟨*Macro FOO* 2a⟩≡

```
/* In /usr/src/include/minix/callnr.h */
#define FOO          95     /* to PM */
```

Definisce:
FOO, usato nella porzione 1c.

A questo punto, avendo modificato uno *header* di sistema, è opportuno operare un `make headers`, in modo che la modifica venga propagata ai file che vengono caricati dal compilatore C.

1.1.2 La gestione della chiamata

Poiché le chiamate di sistema necessitano spesso del contributo congiunto di `PM` e `FS`, l'elenco è unico in `/usr/include/minix/callnr.h` e condiviso fra i due server. Quindi bisognerà prevedere che una richiesta FOO possa arrivare anche a `FS`. Entrambi i server predefiscono uno *handler* no_sys che serve nei casi in cui il server non sia interessato dalla chiamata.

2b ⟨*Gestore no_sys* 2b⟩≡

```
/* In /usr/src/servers/fs/utility.c */
PUBLIC int no_sys()
{
/* Somebody has used an illegal system call number */
  return(EINVAL);
}
```

Definisce:
no_sys, usato nella porzioni 5, 7, e 8.

La gestione delle chiamate avviene nel main del server. Per esempio in FS

3 ⟨*Main di FS* 3⟩≡

```
/* In /usr/src/servers/fs/main.c */
  /* This is the main loop that gets work, processes it, and sends replies. */
  while (TRUE) {
        get_work();              /* sets who and call_nr */
        fp = &fproc[who_p];      /* pointer to proc table struct */
        super_user = (fp->fp_effuid == SU_UID ? TRUE : FALSE); /*su? */

        /* Check for special control messages first. */
        if (call_nr == PROC_EVENT) {
                /* Assume FS got signal. Synchronize, but don't exit. */
                do_sync();
        } else if (call_nr == SYN_ALARM) {
                /* Alarm timer expired. Used only for select(). Check it. */
                fs_expire_timers(m_in.NOTIFY_TIMESTAMP);
        } else if ((call_nr & NOTIFY_MESSAGE)) {
                /* Device notifies us of an event. */
                dev_status(&m_in);
        } else {
                /* Call the internal function that does the work. */
                if (call_nr < 0 || call_nr >= NCALLS) {
                        error = ENOSYS;
                /* Not supposed to happen. */
                        printf("FS, warning illegal %d system call by %d\n", call_nr, who_e);
                } else if (fp->fp_pid == PID_FREE) {
                        error = ENOSYS;
                        printf("FS, bad process, who = %d, call_nr = %d, endpt1 = %d\n",
                                who_e, call_nr, m_in.endpt1);
                } else {
⟨Gestione delle chiamate 4⟩
                }

                /* Copy the results back to the user and send reply. */
                if (error != SUSPEND) { reply(who_e, error); }
                if (rdahed_inode != NIL_INODE) {
                        read_ahead(); /*do block read ahead */
                }
        }
  }
  return(OK);                          /* shouldn't come here */
}
```

Ricevuto e analizzato il messaggio, il cuore della gestione è la seguente istruzione. call_nr contiene il numero della chiamata (p.es. FOO): si tratta quindi di eseguire il corrispondente *puntatore a funzione* contenuto nell'*array* call_vec.

4 ⟨*Gestione delle chiamate* 4⟩≡ (3)

```
error = (*call_vec[call_nr])();
```

Usa call_vec 5 8.

Bisogna quindi aggiornare il call_vec di FS in modo che all'indice FOO contenga un puntatore a no_sys.

5 ⟨*call_ vec di FS* 5⟩≡

```
PUBLIC _PROTOTYPE (int (*call_vec []), (void) ) = {
no_sys,         /*  0 = unused */
        do_exit,        /*  1 = exit    */
        do_fork,        /*  2 = fork    */
        do_read,        /*  3 = read    */
        do_write,       /*  4 = write   */
        do_open,        /*  5 = open    */
        do_close,       /*  6 = close   */
no_sys,         /*  7 = wait    */
        do_creat,       /*  8 = creat   */
        do_link,        /*  9 = link    */
        do_unlink,      /* 10 = unlink  */
no_sys,         /* 11 = waitpid */
        do_chdir,       /* 12 = chdir   */
no_sys,         /* 13 = time    */
        do_mknod,       /* 14 = mknod */
        do_chmod,       /* 15 = chmod */
        do_chown,       /* 16 = chown  */
no_sys,         /* 17 = break   */
        do_stat,        /* 18 = stat    */
        do_lseek,       /* 19 = lseek   */
no_sys,         /* 20 = getpid  */
        do_mount,       /* 21 = mount */
        do_umount,      /* 22 = umount */
        do_set,         /* 23 = setuid  */
no_sys,         /* 24 = getuid  */
        do_stime,       /* 25 = stime   */
no_sys,         /* 26 = ptrace  */
no_sys,         /* 27 = alarm   */
        do_fstat,       /* 28 = fstat   */
no_sys,         /* 29 = pause   */
        do_utime,       /* 30 = utime   */
no_sys,         /* 31 = (stty)  */
no_sys,         /* 32 = (gtty)  */
        do_access,      /* 33 = access  */
no_sys,         /* 34 = (nice)  */
no_sys,         /* 35 = (ftime) */
        do_sync,        /* 36 = sync    */
no_sys,         /* 37 = kill    */
        do_rename,      /* 38 = rename */
```

```
        do_mkdir,       /* 39 = mkdir  */
        do_unlink,      /* 40 = rmdir  */
        do_dup,         /* 41 = dup    */
        do_pipe,        /* 42 = pipe   */
no_sys,         /* 43 = times  */
no_sys,         /* 44 = (prof) */
        do_slink,       /* 45 = symlink */
        do_set,         /* 46 = setgid */
no_sys,         /* 47 = getgid */
no_sys,         /* 48 = (signal)*/
        do_rdlink,      /* 49 = readlink*/
        do_lstat,       /* 50 = lstat  */
no_sys,         /* 51 = (acct) */
no_sys,         /* 52 = (phys) */
no_sys,         /* 53 = (lock) */
        do_ioctl,       /* 54 = ioctl  */
        do_fcntl,       /* 55 = fcntl  */
no_sys,         /* 56 = (mpx) */
no_sys,         /* 57 = unused */
no_sys,         /* 58 = unused */
        do_exec,        /* 59 = execve */
        do_umask,       /* 60 = umask */
        do_chroot,      /* 61 = chroot */
        do_setsid,      /* 62 = setsid */
no_sys,         /* 63 = getpgrp */

no_sys,         /* 64 = KSIG: signals originating in the kernel */
        do_unpause,     /* 65 = UNPAUSE */
no_sys,         /* 66 = unused */
        do_revive,      /* 67 = REVIVE */
no_sys,         /* 68 = TASK_REPLY */
no_sys,         /* 69 = unused */
no_sys,         /* 70 = unused */
no_sys,         /* 71 = si */
no_sys,         /* 72 = sigsuspend */
no_sys,         /* 73 = sigpending */
no_sys,         /* 74 = sigprocmask */
no_sys,         /* 75 = sigreturn */
        do_reboot,      /* 76 = reboot */
        do_svrctl,      /* 77 = svrctl */

no_sys,         /* 78 = unused */
        do_getsysinfo,  /* 79 = getsysinfo */
no_sys,         /* 80 = unused */
```

```
        do_devctl,      /* 81 = devctl */
        do_fstatfs,     /* 82 = fstatfs */
  no_sys,         /* 83 = memalloc */
  no_sys,         /* 84 = memfree */
        do_select,      /* 85 = select */
        do_fchdir,      /* 86 = fchdir */
        do_fsync,       /* 87 = fsync */
  no_sys,         /* 88 = getpriority */
  no_sys,         /* 89 = setpriority */
  no_sys,         /* 90 = gettimeofday */
  no_sys,         /* 91 = seteuid */
  no_sys,         /* 92 = setegid */
        do_truncate,    /* 93 = truncate */
        do_ftruncate,   /* 94 = truncate */
  ⟨Gestione FOO di FS 7⟩
  };
  /* This should not fail with "array size is negative": */
  extern int dummy[sizeof(call_vec) == NCALLS * sizeof(call_vec[0]) ? 1 : -1];
```

Definisce:
call_vec, usato nella porzione 4.
Usa no_sys 2b.

Semmai `FS` dovesse ricevere una chiamata FOO, deve ignorarla.

7 ⟨*Gestione FOO di FS* 7⟩≡ (5)

```
  no_sys,         /* 95 = foo unused */
```

Usa no_sys 2b.

La gestione in PM

Anche qui si tratta di aggiornare call_vec.

8 ⟨*call_vec di PM* 8⟩≡

```
/* In /usr/src/servers/pm/table.c */
_PROTOTYPE (int (*call_vec[NCALLS]), (void) ) = {
no_sys,         /*  0 = unused */
        do_pm_exit,   /*  1 = exit    */
        do_fork,      /*  2 = fork    */
no_sys,         /*  3 = read    */
no_sys,         /*  4 = write   */
no_sys,         /*  5 = open    */
no_sys,         /*  6 = close   */
        do_waitpid,   /*  7 = wait    */
no_sys,         /*  8 = creat   */
no_sys,         /*  9 = link    */
no_sys,         /* 10 = unlink  */
        do_waitpid,   /* 11 = waitpid */
no_sys,         /* 12 = chdir   */
        do_time,      /* 13 = time    */
no_sys,         /* 14 = mknod   */
no_sys,         /* 15 = chmod   */
no_sys,         /* 16 = chown   */
        do_brk,       /* 17 = break   */
no_sys,         /* 18 = stat    */
no_sys,         /* 19 = lseek   */
        do_getset,    /* 20 = getpid  */
no_sys,         /* 21 = mount   */
no_sys,         /* 22 = umount  */
        do_getset,    /* 23 = setuid  */
        do_getset,    /* 24 = getuid  */
        do_stime,     /* 25 = stime   */
        do_trace,     /* 26 = ptrace  */
        do_alarm,     /* 27 = alarm   */
no_sys,         /* 28 = fstat   */
        do_pause,     /* 29 = pause   */
no_sys,         /* 30 = utime   */
no_sys,         /* 31 = (stty)  */
no_sys,         /* 32 = (gtty)  */
no_sys,         /* 33 = access  */
no_sys,         /* 34 = (nice)  */
no_sys,         /* 35 = (ftime) */
no_sys,         /* 36 = sync    */
        do_kill,      /* 37 = kill    */
```

```
no_sys,         /* 38 = rename */
no_sys,         /* 39 = mkdir  */
no_sys,         /* 40 = rmdir  */
no_sys,         /* 41 = dup    */
no_sys,         /* 42 = pipe   */
        do_times,       /* 43 = times  */
no_sys,         /* 44 = (prof) */
no_sys,         /* 45 = unused */
        do_getset,      /* 46 = setgid */
        do_getset,      /* 47 = getgid */
no_sys,         /* 48 = (signal)*/
no_sys,         /* 49 = unused */
no_sys,         /* 50 = unused */
no_sys,         /* 51 = (acct) */
no_sys,         /* 52 = (phys) */
no_sys,         /* 53 = (lock) */
no_sys,         /* 54 = ioctl  */
no_sys,         /* 55 = fcntl  */
no_sys,         /* 56 = (mpx)  */
no_sys,         /* 57 = unused */
no_sys,         /* 58 = unused */
        do_exec,        /* 59 = execve */
no_sys,         /* 60 = umask */
no_sys,         /* 61 = chroot */
        do_getset,      /* 62 = setsid */
        do_getset,      /* 63 = getpgrp */

no_sys,         /* 64 = unused */
no_sys,         /* 65 = UNPAUSE */
no_sys,         /* 66 = unused */
no_sys,         /* 67 = REVIVE */
no_sys,         /* 68 = TASK_REPLY */
no_sys,         /* 69 = unused */
no_sys,         /* 70 = unused */
        do_sigaction,   /* 71 = sigaction   */
        do_sigsuspend,  /* 72 = sigsuspend  */
        do_sigpending,  /* 73 = sigpending  */
        do_sigprocmask, /*74 = sigprocmask */
        do_sigreturn,   /* 75 = sigreturn   */
        do_reboot,      /* 76 = reboot  */
        do_svrctl,      /* 77 = svrctl  */
        do_procstat,    /* 78 = procstat */
        do_getsysinfo,  /* 79 = getsysinfo */
        do_getprocnr,   /* 80 = getprocnr */
```

```
no_sys,          /* 81 = unused */
no_sys,          /* 82 = fstatfs */
        do_allocmem,  /* 83 = memalloc */
        do_freemem,   /* 84 = memfree */
no_sys,          /* 85 = select */
no_sys,          /* 86 = fchdir */
no_sys,          /* 87 = fsync */
        do_getsetpriority,      /* 88 = getpriority */
        do_getsetpriority,      /* 89 = setpriority */
        do_time,        /* 90 = gettimeofday */
        do_getset,      /* 91 = seteuid */
        do_getset,      /* 92 = setegid */
no_sys,          /* 93 = truncate */
no_sys,          /* 94 = ftruncate */
⟨Gestione FOO di PM 10a⟩
};
```

Definisce:
call_vec, usato nella porzione 4.
Usa no_sys 2b.

Questa volta però affidiamo la gestione a una funzione do_foo, che poi definiremo nel paragrafo 1.1.3.

10a ⟨*Gestione FOO di PM* 10a⟩≡ (8)

```
do_foo,          /* 95 = foo */
```

1.1.3 Il servizio FOO

Innanzitutto aggiungiamo il prototipo secondo le convenzioni di MINIX.

10b ⟨*Prototipo di do_foo* 10b⟩≡

```
/* In /usr/src/servers/pm/proto.h */
_PROTOTYPE( int do_foo, (void) );
```

E poi la funzione vera e propria

10c ⟨*Definizione di do_foo* 10c⟩≡

```
/* va bene ovunque in PM: per esempio in /usr/src/servers/pm/misc.c */
PUBLIC int do_foo()
{
        printf("Foo syscall called!\n");
        return OK;
}
```

1.2 Indici

1.2.1 Porzioni di codice

1.2.2 Identificatori