Student: Cristiam David Martin Jackson 08.11.2018
Indeks: 302134

[SOI] - Systemy Operacyjne

Laboratory Nr. 1 - Kernel compilation and system calls

Purpose

The purpose of the laboratory is to get to know the mechanism of making calls. A new call should be added to the system; the function of this call should return the process identification (PID) of the process in the table given by an argument. The function should be implemented in the system Minix 2.0 provided.

Solution

System calls are functions performed by the kernel of the operative system, in the MINIX 2 system, the system calls are implemented inside of the modules MM (Memory Manager) or FS (File System). For the present laboratory the module MM is used and the steps followed for creating and executing the system call are as shown:

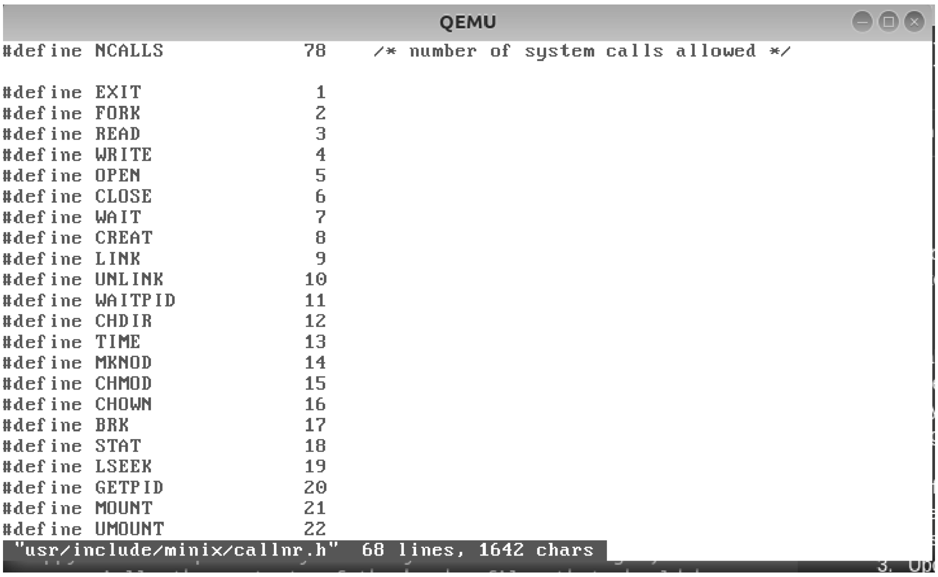
1. Define the system call **GETPROCNR (*number*)** in (/usr/include/minix/callnr.h)
2. Create the function prototype **_PROTOTYPE (int do_getprocnr, (void));** in (/usr/src/mm/proto.h)
3. Update the system call function **do_getprocnr,** in (/usr/src/**mm**/table.c)
4. Update the system call function **no_sys,** in (usr/src/**fs**/table.c)
5. Add a function for the System call **do_getprocnr()** in (/usr/src/mm/main.c)
6. Recompile and reload the Minix2 system with a new kernel
7. Create and execute the test function for the system call

Step 1.

```
vi /usr/include/minix/callnr.h
```

A table with definitions is shown, NCALLS shows the maximum number of calls allowed; if the number of call for the new function is a non existent number, the NCALLS number should be incremented, otherwise the system will show an error.
For the present case, a non used number call is used for the new function GETPROCNR.

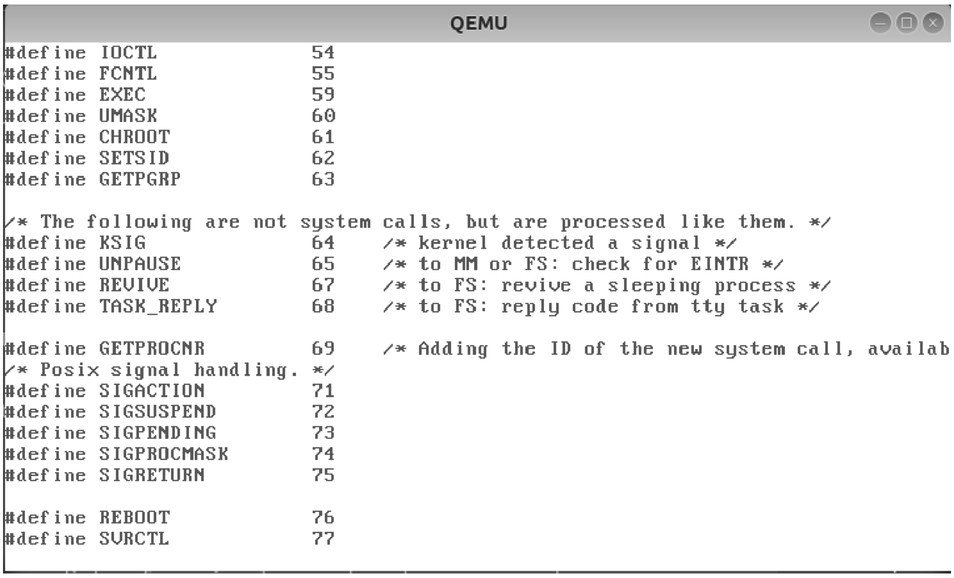
```
#define GETPROCNR          69
```



QEMU

```
#define NCALLS            78     /* number of system calls allowed */

#define EXIT               1
#define FORK               2
#define READ               3
#define WRITE              4
#define OPEN               5
#define CLOSE              6
#define WAIT               7
#define CREAT              8
#define LINK               9
#define UNLINK            10
#define WAITPID           11
#define CHDIR             12
#define TIME              13
#define MKNOD             14
#define CHMOD             15
#define CHOWN             16
#define BRK               17
#define STAT              18
#define LSEEK             19
#define GETPID            20
#define MOUNT             21
#define UMOUNT            22
 "usr/include/minix/callnr.h"  68 lines, 1642 chars
```



QEMU

```
#define IOCTL             54
#define FCNTL             55
#define EXEC              59
#define UMASK             60
#define CHROOT            61
#define SETSID            62
#define GETPGRP           63

/* The following are not system calls, but are processed like them. */
#define KSIG              64     /* kernel detected a signal */
#define UNPAUSE           65     /* to MM or FS: check for EINTR */
#define REVIVE            67     /* to FS: revive a sleeping process */
#define TASK_REPLY        68     /* to FS: reply code from tty task */

#define GETPROCNR         69     /* Adding the ID of the new system call, availab
/* Posix signal handling. */
#define SIGACTION         71
#define SIGSUSPEND        72
#define SIGPENDING        73
#define SIGPROCMASK       74
#define SIGRETURN         75

#define REBOOT            76
#define SVRCTL            77
```

Step 2.

The prototype function should be added in any position, it will establish the new function for the call system.

Vi /usr/src/mm/proto.h
_PROTOTYPE (int do_getprocnr, (void));

QEMU

```
/* forkexit.c */
_PROTOTYPE( int do_fork, (void)                                         );
_PROTOTYPE( int do_mm_exit, (void)                                      );
_PROTOTYPE( int do_waitpid, (void)                                      );
_PROTOTYPE( void mm_exit, (struct mproc *rmp, int exit_status)          );

/* getset.c */
_PROTOTYPE( int do_getset, (void)                                       );

/* main.c */
_PROTOTYPE( void main, (void)                                           );

/* Prototype function will be added in main.c */
_PROTOTYPE( int do_getprocnr, (void)

/* misc.c */
_PROTOTYPE( int do_reboot, (void)                                       );
_PROTOTYPE( int do_svrctl, (void)                                       );

#if (MACHINE == MACINTOSH)
_PROTOTYPE( phys_clicks start_click, (void)                             );
#endif
```

Step 3.

Availability of the number 69 for the system call is confirmed, since its default function was no_sys.

vi /usr/src/mm/table.c
do_getprocnr, /* 69 = update table*/

QEMU

```
        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 */

        do_ksig,        /* 64 = KSIG: signals originating in the kernel */
        no_sys,         /* 65 = UNPAUSE */
        no_sys,         /* 66 = unused  */
        no_sys,         /* 67 = REVIVE  */
        no_sys,         /* 68 = TASK_REPLY  */
        do_getprocnr,   /* 69 = Update table with address of the function call
        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  */
};
/* This should not fail with "array size is negative": */
```

Step 4.

vi /usr/src/fs/table.c

```
QEMU
        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 = SIGACTION */
        no_sys,         /* 72 = SIGSUSPEND */
        no_sys,         /* 73 = SIGPENDING */
        no_sys,         /* 74 = SIGPROCMASK */
        no_sys,         /* 75 = SIGRETURN */
        no_sys,         /* 76 = REBOOT */
        do_svrctl,      /* 77 = SVRCTL */
};
/* This should not fail with "array size is negative": */
```

The setted number is shown as unused, since FS is no the module chosen for the system call, the table is not modified.

Step 5.

The function is implemented in the main program of the memory manager (MM).

vi /usr/src/mm/main.c

Int do_getprocnr(void){
Int n = mproc[mm_in.m1_i1].mp_pid;
printf("\nPosition: %d\tPID> %d",mm_in.m1_i1,n);
return n;
}

```
QEMU
/*=========================================================================*
 *                      FUNCTION do_getprocnr()
 *=========================================================================*/
 int do_getprocnr(void){

int n=mproc[mm_in.m1_i1].mp_pid;
 printf("\nPosition: %d\tPID: %d",mm_in.m1_i1,n);


 printf("\nmproc[%d].mp_pid:%d ",mm_in.m1_i1,n);


/*
 mm_call = mm_in.m_type;*/      /* system call number */
/* printf("\nmm_call:%d ",mm_call);
 printf("\nNR_PROCS: %d",NR_PROCS);
 */

/*
 printf("\nENOENT:%d ",ENOENT);
  return ENOENT;
  */

  return n;
```

For implementing this functions, the inspection of the headers **mproc.h** (/usr/src/mm/mproc.h) and **param.h** (/usr/src/mm/param.h) were required. All headers were already included in the main.c function.

mproc.h: Contains a structure which can be accessed for retrieving the process id, the useful fragment is shown:

```
EXTERN struct mproc {
  struct mem_map mp_seg[NR_SEGS];/* points to text, data, stack */
  char mp_exitstatus;           /* storage for status when process exits */
  char mp_sigstatus;            /* storage for signal # for killed procs */
  pid_t mp_pid;                 /* process id */
  pid_t mp_procgrp;             /* pid of process group (used for signals) */
  pid_t mp_wpid;                /* pid this process is waiting for */
  int mp_parent;                /* index of parent process */
```

The argument for choosing the process number is given by mm_in.m1_i1, which can be checked by accessing the table of definitions param.h, it can be either referred as **pid** as well as **mm_in.m1_i1**

```
QEMU
/* The following names are synonyms for the variables in the input message. */
#define addr            mm_in.m1_p1
#define exec_name       mm_in.m1_p1
#define exec_len        mm_in.m1_i1
#define func            mm_in.m6_f1
#define grpid           (gid_t) mm_in.m1_i1
#define namelen         mm_in.m1_i1
#define pid             mm_in.m1_i1
#define seconds         mm_in.m1_i1
#define sig             mm_in.m6_i1
#define stack_bytes     mm_in.m1_i2
#define stack_ptr       mm_in.m1_p2
#define status          mm_in.m1_i1
#define usr_id          (uid_t) mm_in.m1_i1
#define request         mm_in.m2_i2
#define taddr           mm_in.m2_l1
#define data            mm_in.m2_l2
#define sig_nr          mm_in.m1_i2
#define sig_nsa         mm_in.m1_p1
#define sig_osa         mm_in.m1_p2
#define sig_ret         mm_in.m1_p3
#define sig_set         mm_in.m2_l1
#define sig_how         mm_in.m2_i1
#define sig_flags       mm_in.m2_i2
 "/usr/src/mm/param.h"  40 lines, 1206 chars
```

Recompile

cd /usr/src/tools
make clean
make hdboot
reboot

Make clean: Clean all the object files
Make hdboot: This installs the kernel in the disk

RESULTS

The result of the call system was obtained by implementing a .c function in the desired location, whose function is to call the new system call GETPROCNR, input parameters are given so that the expected results can be accomplished.

vi testfunc.c

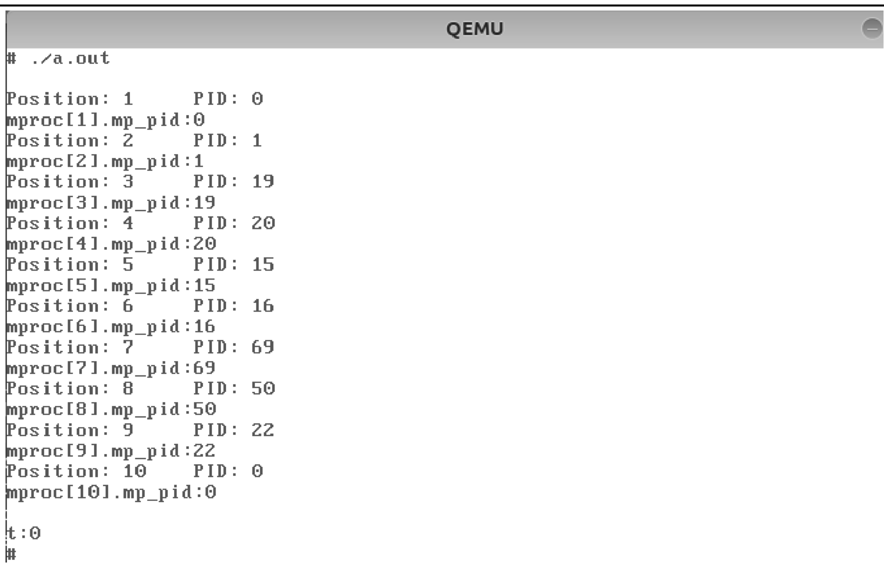
```
#include <lib.h>
#include <stdio.h>

Int (void){
Message m;
Int t;
Int i;
For (i=1;i<11;i++){
   m.m1_i1=i;/*Sending the argument for the mproc table*/
   t=_syscall(MM_PROC_NR, GETPROCNR, &m);
}
Return 0;
}
```

The _syscall function requires three parameters for calling the implemented function:

1. Who sends the signal (MM_PROC_NR), managed by the memory manager (MM)
2. The system call number (GETPROCNR), defined in the callnr.h table
3. A pointer to a message, which can be used for sending the input arguments to the system call GETPROCNR, through the structure **message.**

cc testfunc.c
./a.out


QEMU

```
# ./a.out

Position: 1     PID: 0
mproc[1].mp_pid:0
Position: 2     PID: 1
mproc[2].mp_pid:1
Position: 3     PID: 19
mproc[3].mp_pid:19
Position: 4     PID: 20
mproc[4].mp_pid:20
Position: 5     PID: 15
mproc[5].mp_pid:15
Position: 6     PID: 16
mproc[6].mp_pid:16
Position: 7     PID: 69
mproc[7].mp_pid:69
Position: 8     PID: 50
mproc[8].mp_pid:50
Position: 9     PID: 22
mproc[9].mp_pid:22
Position: 10    PID: 0
mproc[10].mp_pid:0

t:0
#
```

The acceded table of operations have a limit of 32 positions, given by the NR_PROC definition, the argument shall not exceed this point, the first 10 positions of the mproc table are shown in the previous image, as well as their respective process identification (PID), this output is written inside of the system call **do_getprocnr(void)**, the PID is returned to the testing function **testfunc.c.**

The implementation of system calls is a task which requires documentation about the functions and processes involved in the operation of Minix since it is booted. Manipulating

the kernel allows the operator, to achieve specific tasks as well as implement specialized functions.
Different types of values can be used as input as well as output, everyone of the libraries used in this report contain information about the definitions and content of every structure and function used for executing tasks while running the operating system.

REFERENCES

- **Modern operating systems 3ed - A. Tanenbaum**
- **https://wiki.minix3.org/**