Implement a Kernel Call

VE482 Group 6

December 2, 2021

Section 1

How to write a new kernel call?

Introduction

- In Minix3, the servers handle system calls.
- Implement a system call consists of 2 steps
  - Implement a system-call handler
  - Implement a user library

System-call Handler

- The system-call handler should be placed in an appropriate server
- The server processes a user request by invoking the matching handler
- For example
  - A system call updates filesystem and `fproc` data structures
  - It should be placed in the `FS` server.

Section 2

Which files need to be changed?

Implement a System Call `printmessage()`

- Task
  1. Implement a system-call handler `do_printmessage()`
  2. Add a user-library to call the handler `do_printmessage()`
- Goal
  1. Call in user space with `printmessage()`
  2. Print "`I am a system call.`"

Create a System-call Handler

- The source code for all servers are located in: `/usr/src/servers`
- Each server has a separate directory
  - i.e. Filesystem is located in: `/usr/src/servers/fs`
- Each of the server source directories contain two files:
  - `table.c`: contains definition for the `call_vec` table
  - `proto.h`: contains the prototypes of all system-call handler functions

table.c

- The `call_vec` table is an array of function pointers that is indexed by the system-call number

`/usr/src/servers/fs/table.c`:

```
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 */
    ...
```

table.c

- Identify one unused entry
- Replace `no_sys` with `do_printmessage()`

```
...
    no_sys,                 /* 67 = unused */
    no_sys,                 /* 68 = unused */
    do_printmessage,        /* 69 = printmessage */
};
...
```

proto.h

- Declare the ~~prototype~~ function of our system-call handler

  /usr/src/servers/fs/proto.h

```
// _PROTOTYPE( int do_printmessage, (void) );
int do_printmessage(void);
```

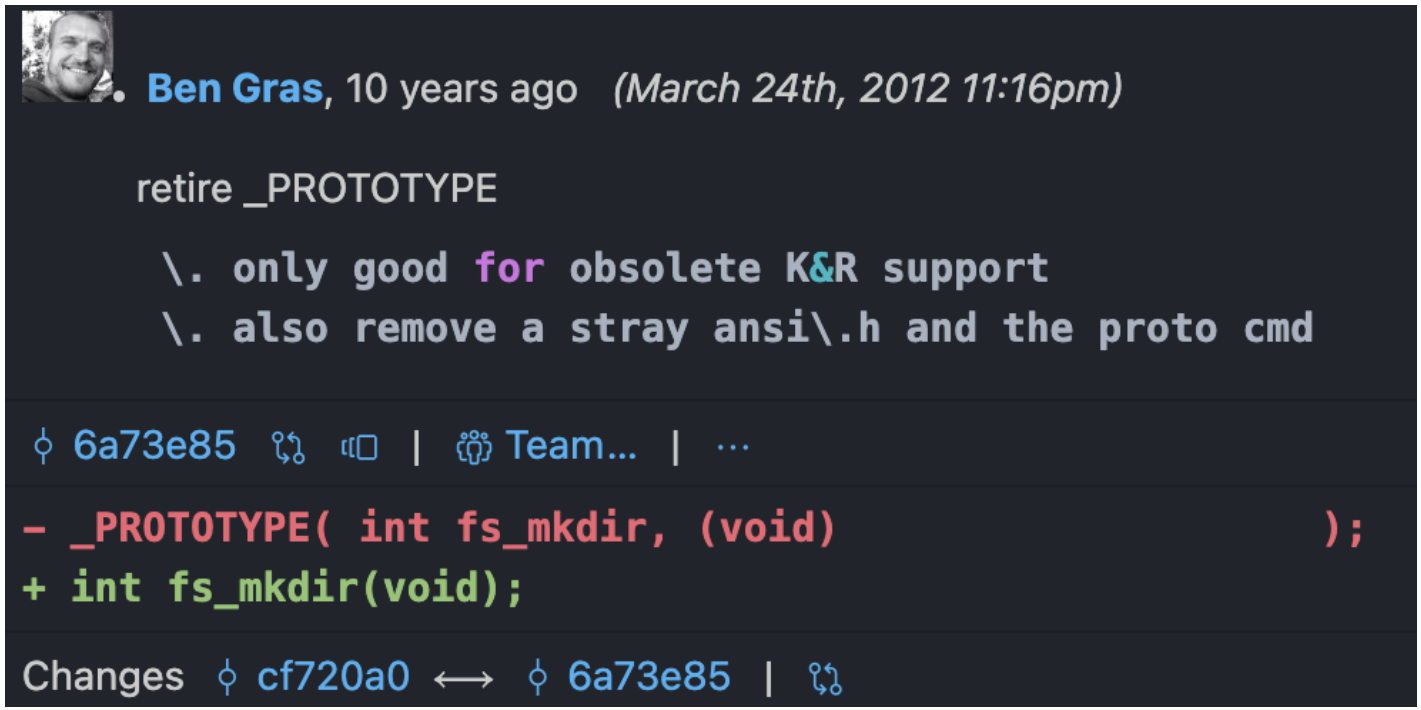

Figure 1: retire _PROTOTYPE

Implement the System-call Handler

- Add our funtion in source file

  `/usr/src/servers/fs/misc.c`

```
#include <stdio.h>
/*===========================================*
*                 do_printmessage           *
*===========================================*/
int do_printmessage()
{
    printf("\I am a system call.\n");
    return(OK);
}
```

Implement a User Library Function

- Modify file: `/usr/include/minix/callnr.h`
  - Increase the value of `NCALLS` by one
  - Add macro

    ```
    #define printmessage 69
    ```
- Modify file: `/usr/include/unistd.h`
  - Add the function declaration

    ```
    void printmessage(void);
    ```

Implement a User Library Function

- Create file: `/usr/include/printmessage.h`

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

int printmessage()
{
    message m;
    return( _syscall( FS, PRINTMESSAGE, &m ) );
}
```

Recompile the Kernel

- Recompile Mnix Kernel

```
cd /usr/src/tools/
make hdboot
sync
shutdown
```

- Reboot Minix

Using the New System Call

- Tester

```c
#include <unistd.h>
#include "printmessage.h"
int main()
{
    printmessage();
    return 0;
}
```

- Output

```
I am a system call.
```