Criando uma System Call no Minix

Tarcísio E. M. Crocomo

Universidade Federal de Santa Catarina

25 de Maio de 2011

Sumário

- Introdução ao sistema de mensagens.
- Criando nossa syscall
- Criando nossa função wrapper
- Testando nossa syscall

A função _syscall

No Minix, as chamadas de sistema, quando não estão sendo acessadas por funções "wrapper" de bibliotecas, são feitas por mensagens.

Essas mensagens são passadas pela função

/usr/include/lib.h

```
_syscall(int who, int syscallnr, message *messageptr)
```

Dissecando a função _syscall

`int who`: O destinatário da mensagem. Mais à frente veremos quais podem ser.

`int syscallnr`: O código da chamada desejada. A chamada que implementaremos terá o seu código único.

`message *messageptr`: Ponteiro para uma struct que será utilizada para passar informações para dentro da syscall, como se fossem parâmetros, já que não estamos usando uma função específica.

Quem é who?

Os valores para `who` estão definidos em `lib.h`, e são os seguintes:

/usr/include/lib.h

```
#define MM PM_PROC_NR //servidor de processos
#define FS FS_PROC_NR //servidor de filesystem
```

Definindo nosso syscallnr

Os números das chamadas estão definidos em `minix/callnr.h`.

Para definirmos o nosso, adicionamos uma linha em `/usr/src/include/minix/callnr.h`. No Minix 3.1.4 o número 64 está livre.

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

```
#define MYCALL 64
```

Definindo o servidor

Agora vamos ao nosso servidor. Utilizaremos o FS para atender nossa syscall.

No arquivo `/usr/src/servers/vfs/table.c` definiremos a função interna (handler) que será executada quando nossa chamada for passada ao FS.

Definindo o servidor

O arquivo contém um grande array de ponteiros para funções. Vamos colocar a nossa na posição correta. Guie-se pelos comentários.

/usr/src/servers/vfs/table.c

```
no_sys, /* 64 = unused */
```

virará

/usr/src/servers/vfs/table.c

```
do_mycall, /* 64 = minha syscall */
```

Observação: Documente bem o que está fazendo, nos padrões dos comentários que já estão no arquivo, para ter uma boa forma de saber o que já fez.

Definindo o servidor

No arquivo `/usr/src/servers/vfs/proto.h` iremos colocar o protótipo de nossa função handler.

/usr/src/servers/vfs/proto.h

```
/* mycall.c */
_PROTOTYPE( int do_mycall, (void) );
```

Implementando o handler

Para implementar nosso handler, temos duas opções:

- Utilizar um arquivo preexistente
- Criar nosso próprio arquivo

Criaremos um novo arquivo. O que isso muda é que teremos depois que adicioná-lo no Makefile, mas ajudará na organização.

Implementando o handler

Criaremos então o arquivo `mycall.c`, com o seguinte conteúdo

/usr/src/servers/vfs/mycall.c

```
#include "fs.h"
#include <stdio.h>

PUBLIC int do_mycall()
{
  int nro = m_in.m1_i1; /* m_in é a mensagem
                           de entrada para toda
                           syscall */
  printf("O numero e %d\n", nro);

  return(OK);
}
```

Modificando o Makefile

Por fim, modificaremos o Makefile adicionando `mycall.o` na variável `OBJ`:

/usr/src/servers/vfs/Makefile

```
OBJ = (...)  mycall.o
```

Compilando o sistema

Podemos então recompilar o sistema.

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

E temos nossa syscall pronta, para usá-la basta reiniciar o sistema!

Fim!

Ou será que não?

O que falta?

Syscalls não são utilizadas por aplicações externas diretamente por chamadas à `_syscall()`.

Utiliza-se o que se chama de "wrapper", ou seja, uma função de biblioteca que "envolve" a chamada à `_syscall()`.

Criando nosso wrapper

Criar o wrapper é bem simples. Basicamente, iremos até `/usr/src/lib/posix` e criaremos o arquivo `_mycall.c` com a nossa função:

`/usr/src/lib/posix/_mycall.c`

```
#include <lib.h>

PUBLIC int mycall(int nro)
{
  message m;
  m.m1_i1 = nro;

  return(_syscall(FS, MYCALL, &m));
}
```

Alterando o Makefile

Agora abrimos o arquivo Makefile.in e adicionamos nosso arquivo a ele:

/usr/src/lib/posix/Makefile.in

```
libc_FILES = " \
             (...) \
             _mycall.c \
             (...) "
```

E na mesma pasta, execute o comando `make Makefile`.

Recompilando as bibliotecas

Por fim, vá até `/usr/src/tools` e execute `make libraries`.

Teste final

Testar nossa syscall é simples. Façamos um arquivo teste.c:

/root/teste.c

```
int main (void)
{
  mycall(15);

  return 0;
}
```

E por fim, `cc -o teste teste.c` dentro da pasta `/root`. Agora basta rodar (`./teste`) e se o resultado for

```
O numero e 15
```

nossa syscall foi implementada com sucesso!

E-mail: tarcisio.crocomo@inf.ufsc.br