System Calls

Em um sistema operacional seguro e coerente, os processos não tem acesso direto ao hardware, não importa se o processo pertence ao usuário comum ou ao administrador do sistema. Só quem pode ter acesso ao hardware é o sistema operacional (SO), mais precisamento o Kernel do SO. Então se um processo precisa acessar algum dispositivo de hardware ele deve solicitar ao SO. Essa solicitação ao sistema operacional é chamado de System Call, ou traduzindo para o português, chamada ao sistema. A System Call, também chamada de syscall, surge por conta da preocupação em protegem o hardware e o próprio Kernel das ações do usuário, que pode vir a danificar um dispositivo acidentalmente ou não (software mal intencionados, também conhecidos como vírus).

As instruções que têm o poder de comprometer o sistema são conhecidas como instruções privilegiadas, enquanto as instruções não-privilegiadas são as que não oferecem perigo ao sistema. Para que uma aplicação possa executar uma instrução privilegiada, o processador implementa o mecanismo de modos de acesso. Existem basicamente dois modos de acesso implementados pelo processador: modo usuário e modo kernel . Quando o processador trabalha no modo usuário, uma aplicação só pode executar instruções não-privilegiadas, tendo acesso a um número reduzido de instruções, enquanto no modo kernel a aplicação pode ter acesso ao conjunto total de instruções do processador.

POSIX

Um modelo para syscall é o POSIX “Portable Operating System Interface [for Unix]”, especificado pela IEEE. Ele é a API de acesso ao recursos do sistema. Nesse padrão é descrito uma série de funções que o Kernel deve ter para ser considerado valido. Se todas essas funções não estiverem implementas, o Kernel não poder ser chamado de Kernel padrão POSIX.

WIN32

Padrão utilizando pela Microsoft. É a API de acesso as system calls dos sistemas operacionais da empresa. Ele se utiliza de DLLs para implementação de acesso aos dispositivos. O que torna mais fácil a substituição de uma dll por outra com um desempenho melhor para determinado tipo de aplicativo.

Incluindo uma syscall no kernel

Nesse tutorial de inclusão de system call no Kernel estou utilizando o Kernel 2.6.32-21, na distribuição Ubuntu 10.04. Nesse sistema o Kernel está localizado no caminho /usr/src/linux-source-2.6.32-21/, e a partir desse ponto do tutorial todas a referencia a caminhos e arquivos será partindo desse caminho, sendo escrito o caminho da seguinte forma: ./umaPasta/umArquivo.arq Todos os comando estão sendo executados em modo superusuário.

Para encontrar esse caminho, primeiro tem de baixar o fonte do kernel, que no Ubuntu pode ser feito com um apt-get.

```
apt-get install linux-source-2.6.32
```

Esse comando irá baixar o fonte compactado, então temos de descompactá-lo com

```
cd /usr/src
tar xjf linux-source-2.6.32.tar.bz2
```

Iremos criar uma pasta mySyscall onde iremos colocar todos os arquivos necessários para a inclusão de uma system call personalizada. Dentro dessa pasta criaremos o arquivo mySyscall.c, que contem o fonte de nossa system call, e o Makefile, o qual irá fazer a compilação da nossa system call.

```
mkdir ./mySyscall
touch ./mySyscall/mySyscall.c
touch ./mySyscall/Makefile
```

O arquivo `mySyscall.c` irá conter o seguinte codigo:

```
#include <linux/linkage.h>
#include <linux/kernel.h>

static int numVezes = 0;

asmlinkage void sys_mySyscall(){
        printk("\nA system call mySyscall foi chamanda %d
```

```
vezes\n", ++numVezes);
}
```

e o `Makefile` conterá isso:

```
obj-y := mySyscall.o
```

Vamos agora modificar os arquivos necessários para que o Kernel encontre nossa system call. Vamos começar pelo arquivo ./arch/x86/kernel/syscall_table_32.S, adicionar no final desse arquivo a seguinte linha

```
.void sys_mySyscall
```

O próximo arquivo a ser alterado é o ./arch/x86/include/asm/unistd 32.h , nesse será adicionado a linha:

```
#define __NR_mySyscall numSyscall
```

onde numSyscall deve ser substituído pelo número da última system call somado de 1. Essa linha deve ser inserida após a ultima declaração de system call. Por exemplo, se a ultima system call declarada for a system call ultimaDeclaracao (que provavelmente não está no fim do arquivo) teriamos o seguinte:

```
#define __NR_ultimaDeclaracao 336
#define __NR_mySyscall 337
```

Ainda no arquivo unistd.h, após adicionar a nossa system call, tem-se que aumentar o número de system calls declaradas, nesse exemplo passaria para 338, alterando a linha

```
#define RN_syscalls 338
```

Agora vamos modificar o ./arch/x86/include/asm/unistd 32.h , deve ser inserir a seguinte linha logo após o último `asmlinkage`:

```
asmlinkage void sys_mySyscall();
```

Bom, agora precisamos alterar o ./Makefile, nesse tem que procurar a linha semelhante a essa:

```
core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/
```

e adicionar `mySyscall/` no final da linha.

Vamos compilar tudo agora, mas antes vamos copiar a configuração já existente do sistema

```
cp /boot/config-2.6.24-19-generic ./.config
```

```
make menuconfig
```

na caixa que irá aparecer selecione a opção “Load an Alternate Configuration File”, ao entrar nessa opção provavelmente irá estár preenchido com '.config' que é o arquivo de configuração que copiamos anteriormente, caso não esteja, mude para '.config', então é só ir em EXIT e salvar. O próximo comando é

```
make-kpkg clean
```

e finalmente

```
make-kpkg --initrd --append-to-version=-custom kernel_image
kernel_headers
```

agora é só instalar os .deb resultantes

```
dpkg -i *.deb
```