UNIVERSIDADE DE SÃO PAULO - ICMC
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO
SSC0140 – SISTEMAS OPERACIONAIS I
Profa: Kalinka Regina Lucas Jaquie Castelo Branco
Trabalho 1 – Confecção de um chamada de sistema

Especificação do trabalho de geração da sua System Call

Objetivos

Compreender o processamento inerente a uma chamada ao sistema. Conhecer as várias fases do processamento de uma chamada ao sistema no Linux (acredito ser mais fácil que no Windows). Saber como introduzir novos serviços (chamadas ao sistema) no Linux. Compreender os mecanismos básicos internos ao Linux para gestão de processos (bloqueio/suspensão de processos que não podem continuar e reativação do mesmos quando adequado).

Introdução

Neste trabalho pretende-se que o aluno implemente um novo serviço no núcleo do Sistema de Operação Linux. Este novo serviço deverá estar acessível a qualquer aplicação através de uma chamada ao sistema. O trabalho está definido em três fases, sendo a primeira relacionada à familiarização com o núcleo (*kernel*) do Sistema Operacional Linux. Para isso irá proceder-se à introdução de uma nova e muito simples chamada ao sistema, que no caso exemplo constitui em imprimir uma string pré-definida. Nas duas fases seguintes,
irá:

1. aprender como se criam módulos para o Linux, e como estes podem ser carregados e removidos dinamicamente.;
2. implementar a sua própria chamada de sistemas.

Adicionar um novo serviço ao núcleo do Linux

Um processo, executando em modo usuário, está bastante limitado no conjunto de operações que pode realizar (por exemplo, não pode acessar diretamente o *hardware* da máquina). Assim, para realizar operações mais delicadas, o processo deverá solicitar a execução de um serviço no núcleo do SO. Este serviço será executado pelo próprio processo, mas com o processador em modo kernel (seguro), podendo assim fazer uso das instruções privilegiadas.

Para realizar uma chamada ao sistema, o processo deverá gerar uma interrupção por *software*. No caso do Linux esta interrupção tem o código 80h. Quando é gerada uma interrupção, seja ela por *software* ou por *hardware*, o processador consulta um registro interno denominado IVR (*Interrupt Vector Register*) para saber em que endereço de memória está o vetor de interrupções. Este vetor contém, em cada posição $0 <= i <= n$, um apontador para a rotina de tratamento da interrupção com o código *i*. Por exemplo, na posição 80h deste vetor está o apontador para a rotina que deverá tratar o *interrupt 80h*. Ou seja, quando de uma interrupção N, a execução é transferida para a sub-rotina indicada no vetor_de_interrupções[N]. No caso da interrupção int 80h, na versão 2.6 do kernel do Linux, essa sub-rotina chama-se *system_call* e está localizada no arquivo arch/i386/kernel/syscall_table.S (todas as referências e arquivos terão como base o diretório raiz do código fonte do kernel do Linux, que no nosso caso é /usr/src/linux-2.6). No Linux,

uma aplicação realiza uma chamada ao sistema colocando no registo EAX o código da chamada a que pretende executar e gerando um *int 80h em* seguida. Assim, esta rotina system_call() será executada sempre que um processo solicitar uma chamada ao sistema. A rotina system_call() é consideravelmente simples. Usa o registro EAX (que contém o código do serviço solicitado) para indexar uma outra tabela, a sys_call_table, também localizada no arquivo arch/i386/kernel/syscall_table.S, e saltando para o endereço por esta indicado.
Para a versão 2.6 do kernel do Linux, a tabela sys_call_table é definida do seguinte modo:

```
data
ENTRY(sys_call_table)
.long sys_restart_syscall
.long sys_exit
.long sys_fork
.long sys_read
.long sys_write
...
.long sys_tee /* 315 */
.long sys_vmsplice
.long sys_move_pages
.long sys_getcpu
.long sys_epoll_pwait
```

Procedimento
Como vai adicionar uma nova chamada de sistema ao núlceo/kernel do Linux, terá de adicionar uma nova entrada a esta tabela (no fim da mesma), apontando para a sua função que implementa o serviço. Apesar de não ser obrigatório, o nome da sua função deverá conter o prefixo sys_, (por exemplo, sys_myservice). A partir deste momento poderá implementar a sua chamada. Por convenção, todos os arquivos de cabeçalho utilizados por chamadas ao sistema independentes da arquitetura são colocados no diretório include/linux, sendo os dependentes colocados no diretório include/asm-i386 (neste trabalho não iremos lidar com chamadas dependentes da arquitetura). Os arquivos que contêm o código fonte da implementação são normalmente colocados em um diretório dedicado ao tipo de serviço que se pretende oferecer. Por exemplo, serviços de rede são colocados no diretório net e serviços relativos a sistemas de arquivos em fs. Sugerimos que inclua os seus no diretório arch/i386/kernel dedicada aos serviços do kernel para arquiteturas x86. Terá em seguida que alterar o Makefile desse diretório para incluir o seu arquivo na variável obj-y. Para ser ligada (*linked*) corretamente o protótipo da função que implemente o serviço deve ser precedida da palavra asmlinkage e o seu nome prefixado por sys_, como exemplificado abaixo:

```
asmlinkage int sys_myservice(int arg1, char* arg2) {
/* implementação do serviço */
}
```

A palavra *asmlinkage* requer a inclusão do arquivo linux/include/linux.linkage.h: #include <linux/linkage.h>, que sugerimos que seja incluído num arquivo myservice.h. Portanto os arquivos necessários à implementação de uma chamada ao sistema *myservice* que escreva uma string serão:

```
myservice.h
#ifndef __LINUX_MYSERVICE_H
#define __LINUX_MYSERVICE_H
#include <linux/kernel.h> // para o printk
#include <linux/linkage.h> // para o asmlinkage
#endif

myservice.c
asmlinkage int sys_myservice(char* arg1) {
printk(%s, arg1);
return 1;
}
```

Para que a chamada seja inserida no sistema terá de compilar o *kernel* e definir a imagem resultante (vmlinuz) como a que deve ser carregado na inicialização do sistema. Poderá fazer automaticamente com o comando: make bzlilo.

Disponibilização do serviço aos utilizadores/programadores

Neste momento os programas dos utilizadores podem usar a nova chamada ao sistema. No entanto para os programas em C poderem usar uma interface de forma conveniente, sem necessidade de recorrer à programação em *assembly*, é necessário fornecer uma interface através dos arquivos include do sistema. Deverá verificar qual a posição na tabela onde inseriu o apontador para a sua função e, em seguida, atribuir um identificador à sua chamada, editando o arquivo /usr/include/asm/unistd.h (atenção ao path, que é absoluto e não relativo à raiz dos sources do kernel) adicionando uma entrada que permita fazer a correspondência entre o identificador/nome do serviço que vai implementar e o seu código (índice na tabela syscall_table). O identificador do serviço deve conter o prefixo __NR_ e o valor deve corresponder à posição da chamada na tabela de chamadas ao sistema (syscall_table).

```
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
...
#define __NR_tee 315
#define __NR_vmsplice 316
#define __NR_move_pages 317
#define __NR_getcpu 318
#define __NR_epoll_pwait 319
```

Neste caso a definição associaria __NR_myservice a 320. Atualize em seguida a variável que define o número de chamadas ao sistema. Agora você poderá executar o seu serviço da seguinte de forma:

```
#include <linux/unistd.h>
syscall(__NR_myservice, "hello SO")
```

Para fornecer uma sintaxe mais simpática e transparente deverá encapsular a chamada, por exemplo num arquivo de cabeçalho myservice.h:

```
#include <linux/unistd.h>
#define myservice(X) syscall(__NR_myservice, X);
```

Este arquivo teria de ser colocado em um diretório de onde o utilizador pudesse importar, normalmente em
`/usr/include/sys.`

Teste do sistema
Agora que a nova chamada ao sistema está disponível para os programas C dos utilizadores, você pode fazer um programa de teste como o seguinte:

```
#include <sys/myservice.h>
main() {
myservice("hello SO");
}
```

Como deve ter notado, no arquivo myservice.c foi utilizada a função printk() para fazer a escrita da string, em vez de printf(). Isto se deve ao fato de a função printf() não existir ao nível do núcleo do S.O. (apenas existe na biblioteca libc.a/libc.so). Ambas as funções têm a mesma sintaxe. No entanto, a função printk() escreve as mensagens para o console do sistema, que poderá não estar visível. Se for esse o caso, poderá executar às últimas mensagens enviadas para a consola do sistema através do comando dmesg | tail.
Segue mais algumas funções disponíveis ao nível do kernel.

printk()
NAME
Printk – print messages to console log
SYNOPSIS

```
#include <linux/kernel.h>
int printk(const char*fmt, ...)
```

copy_from_user(), copy_to_user()
NAME
get_user, put_user, copy_from_user, copy_to_user copy data between kernel space and user space

SYNOPSIS

```
#include <asm/uaccess.h>
err = get_user ( x, addr );
err = put_user ( x, addr );
bytes_left = copy_from_user(void*to, const void *from,
unsigned long n );
bytes_left = copy_to_user(void*to, const void *from, unsigned
long n );
```

O trabalho deverá ser entregue via Moodle até o dia 31/10/2014.