Lista 01 e 02 - Criação de uma syscall

Laís Saloum Deghaide, nUSP: 11369767

Objetivos:

- Compreender o processamento inerente a uma chamada ao sistema;
- Conhecer as várias fases do processamento de uma chamada ao sistema no Linux;
- Saber como introduzir novos serviços no Linux;
- Compreender os mecanismos básicos internos ao Linux para gestão de processos.

Desenvolvimento da chamada de sistema:

Etapa 1: Escolhi realizar a lista por meio do WSL2 e compilar o kernel através de uma máquina virtual. Para tal, precisei instalar o staging, a máquina virtual e suas dependências, executando os seguintes códigos:

```
git clone

git://git.kernel.org/pub/scm/linux/kernel/git/gregkh/staging.git -b
staging-testing

cd staging

pip3 install --user git+https://github.com/ezequielgarcia/virtme.git

sudo apt-get -y -q install           \
  bc                              \
  flex                            \
  bison                           \
  build-essential                 \
  git                             \
  libncurses-dev                  \
  libssl-dev                      \
  libelf-dev                      \
  u-boot-tools                    \
  wget                            \
  xz-utils                        \
  qemu-kvm                        \
  iproute2                        \
  python3                         \
  python3-pip
```

Depois compilei meu kernel através da virtme para checar se estava tudo correto:

```
virtme-configkernel --defconfig
make -j$(nproc)
```

Etapa 2: criação da syscall em si e adição dos caminhos dela nos arquivos necessários.

1. **Adicionando a minha chamada de sistema na tabela de chamadas de sistema:**

   Abrindo a tabela de syscalls no meu editor:

```
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging$ cd arch/x86/entry/syscalls
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/arch/x86/entry/syscalls$ code syscall_64.tbl
```

   Inserindo na última linha da tabela a nova chama de sistema que criarei (sys_print):

syscall_64.tbl M

home > laisdeghaide > lista1 > staging > arch > x86 > entry > syscalls > syscall_64.tbl

```
368   444 common  landlock_create_ruleset sys_landlock_create_ruleset
369   445 common  landlock_add_rule   sys_landlock_add_rule
370   446 common  landlock_restrict_self  sys_landlock_restrict_self
371   447 common  memfd_secret        sys_memfd_secret
372   448 common  print           sys_print
```

2. **Adicionando a nova chamada do sistema ao arquivo de cabeçalho da chamada do sistema:**

   Abrindo o cabeçalho:

```
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging$ cd include/linux
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/include/linux$ code syscalls.h
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/include/linux$
```

   Adicionando cabeçalho da nova chamada:

C syscalls.h M

home > laisdeghaide > lista1 > staging > include > linux > C syscalls.h

```
1265                unsigned long fd, unsigned long pgoff);
1266  asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
1267
1268  asmlinkage long sys_print(char* str, int len);
1269
1270
```

3. **Agora que o kernel tem o número e o protótipo da minha chama de sistema:**

   Registro minha chamada de sistema no `include/uapi/asm-generic/unistd.h`

```
laisdeghaide@DESKTOP-AUE2BQQ:/home$ cd laisdeghaide/lista1/staging/include/uapi/asm-generic
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/include/uapi/asm-generic$ code unistd.h
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/include/uapi/asm-generic$
```

unistd.h M

home > laisdeghaide > lista1 > staging > include > uapi > asm-generic > C unistd.h

```
876  #ifdef __ARCH_WANT_MEMFD_SECRET
877  #define __NR_memfd_secret 447
878  __SYSCALL(__NR_memfd_secret, sys_memfd_secret)
879  #endif
880
881  #define __NR_print 448
882  __SYSCALL(__NR_print, sys_print)
883
884  #undef __NR_syscalls
885  #define __NR_syscalls 449
886
```

E também no **`kernel/sys_ni.c`**

```
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging$ cd kernel
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/kernel$ code sys_ni.c
```

sys_ni.c M

home > laisdeghaide > lista1 > staging > kernel > C sys_ni.c

```
479  COND_SYSCALL(setreuid16);
480  COND_SYSCALL(setuid16);
481  COND_SYSCALL(print);
482
483  /* restartable sequence */
484  COND_SYSCALL(rseq);
485
```

Isso é necessário para fornecer implementação de stub substituto de nosso syscall, que retorna -ENOSYS.

4. **Criando minha chamada de sistema:**

```
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/kernel$ code print.c
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1/staging/kernel$ code Makefile
```

print.c U

home > laisdeghaide > lista1 > staging > kernel > C print.c

```c
#include <linux/syscalls.h>
#include <linux/kernel.h>

SYSCALL_DEFINE2(print, char __user *, str, int, len)
{
    char *buf;

    buf = kmalloc(len+1, GFP_KERNEL);
    if (!buf)
        return -ENOMEM;

    if (copy_from_user(buf, str, len+1))
        return -EFAULT;

    printk("%s", buf);

    kfree(buf);
    return 0;
}
```

Minha chamada recebe 2 parâmetros, a minha string (str) que vou printar e o tamanho dela (len)

E no Makefile do kernel, insiro o .o da minha chamada de sistema:

Makefile M

home > laisdeghaide > lista1 > staging > kernel > Makefile

```
obj-y += sched/
obj-y += locking/
obj-y += power/
obj-y += printk/
obj-y += irq/
obj-y += rcu/
obj-y += livepatch/
obj-y += dma/
obj-y += entry/
obj-y += print.o
```

Etapa 3: Testando a chamada de sistema na parte do usuário

5. **Criando um .h para testar minha chamada de sistema:**

```
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1$ code print.h
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1$
```

print.h

home > laisdeghaide > lista1 > C print.h

```c
#ifndef LINUX_PRINT_H
#define LINUX_PRINT_H

#include <unistd.h>
#define print(str, len) syscall(448, str, len)

#endif
```

Obs: Devido a limitações da virtualização utilizada, o header print.h foi mantido na mesma pasta do arquivo de teste e não pode utilizar a macro __NR_print definida anteriormente, ambos problemas que deveriam ser corrigidos em uma implementação final.

6. **Criando um .c para testar/chamar minha syscall:**

teste_syscall.c

home > laisdeghaide > lista1 > C teste_syscall.c

```c
#include "print.h"
#include <string.h>


int main()
{
    const char* string = "Ola mundo! Aqui eh a Lais. Prazer\n";
    print(string, strlen(string));
    return 0;
}
```

Aqui passo minha macro print que foi definida no .h

7. **Compilando o programa teste_syscall.c e colocando o resultado no executável teste**

```
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1$ gcc teste_syscall.c -o teste
laisdeghaide@DESKTOP-AUE2BQQ:~/lista1$
```

8. **Etapa de compilação do kernel:**
   Executo os mesmos comandos que testei na etapa 1:

```
cd staging
virtme-configkernel --defconfig
make -j$(nproc)
```

   E já dentro da máquina virtual, chamo a minha chamada de sistema: