

## System Call

### Projeto 1 - System Call

#### QEMU

Utilize o QEMU para dar seguimento nesta atividade.

1. Faça download do arquivo `Projeto_01`.

2. Surgirá um diretório contendo a versão 3.17.2 do kernel Linux contendo um arquivo `.config` adequado para o experimento. Além disso, iremos trabalhar com a imagem `SO_DC.img` que foi criada por Glauber de Oliveira Costa.

3. Compile o kernel com

```
$ cd linux-3.17.2
```

```
$ make -j 5 ARCH=i386
```

Para não ter problemas com a quota, configure previamente o diretório `CCACHE` utilizando o comando:

```
$ export CCACHE_DIR="/tmp/.ccache"
```

4. Teste o kernel com a imagem utilizando o QEMU:

```
$ qemu-system-i386 -hda SO_DC.img -kernel linux-3.17.2/arch/i386/boot/bzImage -append "ro root=/dev/hda"
```

5. Quando o sistema entrar poderemos fazer login com

usuário: root

senha: root

#### Primeira chamada

Antes de fazermos uma chamada de sistema mais elaborada, vamos testar uma que não recebe parâmetros e retorna uma constante. Chamaremos esta chamada de `mycall`.

1. Alterar a tabela `linux-3.17.2/arch/x86/syscalls/syscall_32.tbl`, acrescentando uma nova linha ao final do arquivo:

2.   357   i386   mycall                   sys\_mycall

3. Incluir uma declaração da função `sys_mycall` em um ponto adequado do arquivo `linux-3.17.2/include/linux/syscalls.h`

4. `asmlinkage long sys_mycall(void);`

5. Incluir o código para função no diretório `linux-3.17.2/arch/x86/kernel/`. Podemos utilizar o arquivo `mycall.c`, cujo conteúdo é:

6. `#include <linux/unistd.h>`

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

8.

9. `asmlinkage long sys_mycall(void) {`

10. `return(4096);`

11. `}`

12. Alterar o `Makefile` do diretório `linux-3.17.2/arch/x86/kernel/`, incluindo uma linha:

13. `obj-y += mycall.o`

14. No diretório `linux-3.17.2` executar

15. `$ make -j 5 ARCH=i386`

16. Escrever um programa para testar a chamada em modo usuário. Pode ser um programa bem simples como `ex-mycall.c`:

17. `/*`

18. `* Teste da nova chamada de sistema`

19. `*/`

20. `#include <stdio.h>`

21. `#include <stdlib.h>`

22. #include <unistd.h>

23.

24. int main() {

25. int r = syscall(357);

26. printf("Retorno da chamada de sistema: %d.\n", r);

27. return r;

}

28. Compilar o programa, considerando que a imagem foi feita para i386. As opções são:

- Compilar normalmente em uma máquina i386
- Utilizar o gcc cross compiler:

○ `$ gcc -m32 -static ex-mycall.c -o ex-mycall`

29. Rodar o programa no QEMU:

Devemos tornar o executável disponível no ambiente do QEMU.

- Uma opção simples é exportar o arquivo que contém o executável como um disco:

`$ qemu-system-i386 -hda SO_DC.img -kernel linux-`

`3.17.2/arch/i386/boot/bzImage -append "ro root=/dev/hda" -hdb ex-mycall`

1.

- Após logar no sistema, devemos executar:

○ `$ cat /dev/hdb > ex-mycall`

○ `$ chmod +x ex-mycall`

`$ ./ex-mycall`

## Projeto 1:

Em grupo (máx. 4 ), implemente uma chamada de sistema que receba ao menos um parâmetro e que trabalhe com pelo menos um elemento do sistema operacional. A ideia é que você adicione uma pequena funcionalidade ao kernel. Assim, como realizado no teste acima, `ex-mycall`.

## Entrega

Cada grupo deverá fazer um relatório contendo a descrição da chamada de sistema implementada e o seguinte conteúdo disponível:

- Arquivo README contendo a descrição da chamada e como os outros arquivos devem ser instalados/testados.
- Arquivo com código da chamada de sistema.
- Arquivo teste.

**Data de entrega:** 12/09

### **Exemplos:**

**Obs. Caso seja entregue projetos iguais as notas serão zeradas.**

1. mycall - verifica o estado de um processo.
2. superuser - atribui privilégios de root ao processo que invoca esta chamada.
3. mycall - retorna informações sobre um processo (tipo de thread, nome e quantidade de páginas em uso).
4. forkn - executa n forks de um processo.
5. kern\_buf - armazena e recupera dados de um buffer no espaço de kernel.
6. mymemset - chamada de sistema que preenche vetores.
7. mkuniquedir - cria um diretório com o número do processo como sufixo.
9. open\_n - abre uma lista de arquivos passada como parâmetro.
10. timed\_stop - envia um sinal SIGSTOP para um processo após um tempo passado como parâmetro.
11. mycall - envia um SIGTERM para um processo e, após um intervalo de tempo, um SIGKILL.
12. getccount e getcpid - retornam um contador para um número de filhos e um vetor com os PIDs dos processos filhos.
13. multikill - envia sinal para n processos.
14. forkLimit - limita o número de forks que o sistema pode executar.
15. imortal - registra processos que não morrerão ao receber SIGKILL.
16. mycall - atribui prioridade máxima ao processo passado como parâmetro.
17. mycall - altera o diretório de trabalho, procurando primeira ocorrência do nome passado como parâmetro no caminho atual.

### **Dicas**

Veja outras dicas para implementação de system calls.

### **Referências**

Gracia, I, C. (2017) Implementação de Chmdadas de Sistemas. MC504 - Unicamp