

TSO - 2024.2 - JOÃO PEDRO FERREIRA DUARTE

João Pedro F. Duarte¹

¹Bacharelado em Engenharia de Computação
Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Minas Gerais – MG – Brasil

joaopedrofduarte@outlook.com

1. Implementação da Syscall no SO Minix3

Abstract. *In Operating Systems, system calls, Syscalls, are functions renewed to perform some type of specific service in favor of the processes carried out by the OS. With this same objective, the present study presents the implementation of a Syscall, which we will call **useless**, within the Minix3.2.1 Process Manager Server, for purposes study on this process.*

Resumo. *Em Sistemas Operacionais, chamadas de sistema, Syscalls, são funções implementadas para realizar algum tipo de serviço específico em favor dos processos realizados pelo SO. Com este mesmo objetivo, o presente estudo apresenta a implementação de uma Syscall, que a chamaremos de **inútil**, dentro do Servidor Gerenciador de Processo do Minix3.2.1, para fins de estudo sobre este processo.*

1.1. Introdução

O presente texto, com objetivo de instrução, serve para que qualquer programador com o intuito de estudar mais sobre os conceitos relacionados a Sistemas Operacionais, através do uso do Minix, em sua versão 3.2.1, possa realizar todos os passos necessários para implementar, desde o início, a criação de uma chamada de sistema. Em maiores detalhes, temos aqui as referências [Beletti 2015] e [Tanenbaum and Bos 2014] como fontes base para entendimento do Kernel do sistema operacional em uso, como também da documentação presente no site do Minix para desenvolvedores conforme [Tanenbaum 2024]. Link da documentação.

Sobre a ambientação necessária para o desenvolvimento deste projeto, o programador deve realizar a instalação de uma máquina virtual minix, na versão correta para este projeto e realizar todas as etapas de instalação do sistema operacional conforme documentação oficial supracitada. O link referente à plataforma onde deve ser criada a máquina virtual, como também ao local onde estará a versão correta do sistema operacional e a parte da documentação onde constam as etapas de instalação do sistema, podem ser consultados a seguir: [Virtual Box], [Minix versão 3.2.1] e [processo de instalação].

1.2. Primeiro caso

Inicialmente, para dar início ao processo de criação da chamada de sistema, tem-se como focos de alteração e estudo para fins de entendimento do processo interno do Kernel do sistema operacional, a aplicação das seguintes rotas para atualização de dados e manipulação.

1. /callnr.h: Aqui está a listagem de todas as chamadas de sistema presentes no Minix.
2. /table.c: Aqui estão os nomes de cada uma das rotinas das chamadas de sistema do Minix.
3. /proto.h: Protótipo de todas as rotinas de chamadas de sistema do sistema operacional.

4. /misc.c: Chamada de sistema de todas as Syscalls do tipo Misc.
5. /minhachamadolib.h: Este é o "include file" que realiza a chamada da chamada de sistema inútil.
6. /teste1.c: Arquivo de teste criado para realizar o teste da chamada inútil.

Para realizar a manipulação dos diretórios, de forma a acessar, listar, ver conteúdo, editar e salvar arquivos, pode-se usar comandos Linux padrão, conforme pode ser consultado em [Lista de comandos básicos pela DevMedia].

1.3. Segundo Caso

O primeiro passo da atividade consiste em acessar o arquivo 1 através do comando `vi`, em `/usr/src/include/minix/callnr.h`, assim, através do comando `vi /usr/src/include/minix/callnr.h`. Neste arquivo, deve ser inserida a seguinte linha entre as chamadas de sistema 68 e 71. `#define MYSYSCALL 69`. Após realizar a inserção do valor na linha, basta salvar o arquivo e sair do editor de texto `vi`, através do comando `wq`. O arquivo referenciado pode ser visto abaixo na figura 1.

```

#define TRUNCATE      93 /* to UFS */
#define FTRUNCATE     94 /* to UFS */
#define FCHMOD        95 /* to UFS */
#define FCHOWN        96 /* to UFS */
#define SPROF         98 /* to PM */
#define CPROF         99 /* to PM */

/* Calls provided by PM and FS that are not part of the API */
#define PM_NEWEXEC    100 /* from UFS or RS to PM: new exec */
#define SRV_FORK      101 /* to PM: special fork call for RS */
#define EXEC_RESTART  102 /* to PM: final part of exec for RS */
#define GETPROCNR     104 /* to PM */
#define ISSETUGID      106 /* to PM: ask if process is tainted */
#define GETEINFO_0     107 /* to PM: get pid/uid/gid of an endpoint */
#define SRV_KILL       111 /* to PM: special kill call for RS */

#define GCOV_FLUSH     112 /* flush gcov data from server to gcov files */
#define PM_GETSID      113 /* PM getsid() */

#define TASK_REPLY     121 /* to UFS: reply code from drivers, not
                          * really a standalone call.
                          */
#define MAPDRIVER      122 /* to UFS, map a device */

```

Figure 1. Caminho de referência de `callnr.h`.

1.4. Terceiro Caso

Agora é necessário navegar até 2 através do comando `vi`, que permita a abertura do editor do arquivo. Procure neste arquivo pela seguinte linha: `no_sys, /*69 = unused */`, e nesta mesma linha, faça a substituição do conteúdo por `do_mysyscall, /*69 = my sys call */`. O caminho completo para este acesso é `vi /usr/src/servers/pm/table.c`

```

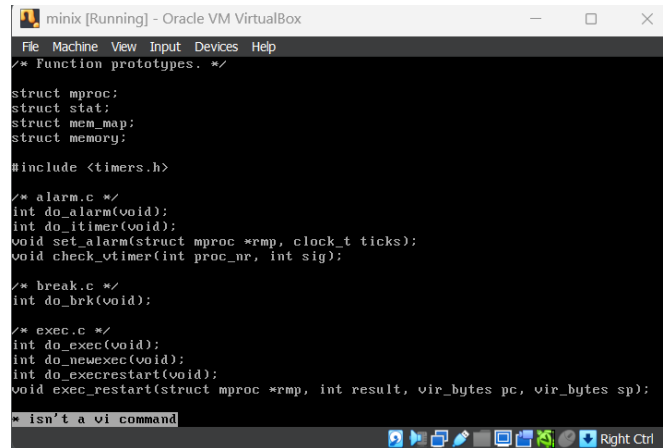
no_sys, /* 93 = (truncate) */
no_sys, /* 94 = (ftruncate) */
no_sys, /* 95 = (fchmod) */
no_sys, /* 96 = (fchown) */
no_sys, /* 97 = unused */
do_sprofile, /* 98 = sprofile */
do_cprofile, /* 99 = cprofile */
do_newexec, /* 100 = newexec */
do_srv_fork, /* 101 = srv_fork */
do_execrestart, /* 102 = exec_restart */
no_sys, /* 103 = unused */
do_getprocnr, /* 104 = getprocnr */
no_sys, /* 105 = unused */
do_get, /* 106 = issetugid */
do_geteinfo_0, /* 107 = geteinfo XXX: old implementation */
no_sys, /* 108 = unused */
no_sys, /* 109 = unused */
no_sys, /* 110 = unused */
do_srv_kill, /* 111 = srv_kill */
no_sys, /* 112 = gcov_flush */
do_get, /* 113 = getsid */

```

Figure 2. Caminho de referência de `table.c`.

1.5. Quarto Caso

Como terceira etapa do processo de criação da chamada de sistema, agora é necessário navegar com o comando `vi` e abrir o arquivo 3, através da instrução `vi /usr/src/servers/pm/proto.h`. Aqui, devemos procurar pela linha `int do_getsetpriority(void);` (inside `/*misc.c */`) e realizar a adição, uma linha acima da já referida linha, a seguinte função, `int do_mysyscall(void);`. Dada finalização da edição, basta agora, assim como anteriormente, salvar e sair do presente arquivo por `wq`.



```
minix [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
/* Function prototypes. */

struct mproc;
struct stat;
struct mem_map;
struct memory;

#include <timers.h>

/* alarm.c */
int do_alarm(void);
int do_itimer(void);
void set_alarm(struct mproc *rmp, clock_t ticks);
void check_vtimer(int proc_nr, int sig);

/* break.c */
int do_brk(void);

/* exec.c */
int do_exec(void);
int do_newexec(void);
int do_execrestart(void);
void exec_restart(struct mproc *rmp, int result, vir_bytes pc, vir_bytes sp);

/* isn't a vi command
```

Figure 3. Conteúdo interno de `proto.h`.

1.6. Quinto Caso

Agora é necessário através do comando de edição de arquivos `vi`, devemos entrar dentro do arquivo `misc.c` pelo caminho `vi /usr/src/servers/pm/misc.c`. Ao final do arquivo, deve-se inserir a função 1, conforme pode ser visto abaixo.

```
1      int do_minhachamada(void)
2      {
3          int input_number = m_in.m1_i1;
4          printf("ola, sou a syscall inutil que imprime o numero % d na
              tela \n ", input_number);
5          return 0;
6      }
```

Listing 1. Programa em C que mostra o que faz a syscall

1.7. Sexto Caso

Dentro do arquivo `/usr/src/servers/pm`, com o comando `cd`, tente compilar o arquivo criado antes, digite `make` e veja se o arquivo antes criado está compilando de forma adequada. Lembrando que o sistema operacional Minix usa `cc` como compilador C pré-instalado.

1.8. Sétimo Caso

De forma semelhante ao já realizado, com o comando `vi`, no caminho `/usr/include/mysyscalllib.h`, e adicione o código em 2. Após a finalização, basta salvar o arquivo e passar para o próximo passo.

```
1      #include <lib.h>
2      #include <unistd.h>
3      int minhachamada(int n)
4      {
5          message m;
6          m.m1_i1 = n;
7          return ( _syscall(PM_PROC_NR, MINHACHAMADA, &m) );
8      }
```

Listing 2. Programa em C para somar dois números

Agora através do comando `cd`, vá para o seguinte caminho `/usr/src/releasetools`, e digite `make install`. Após o comando anterior compilar digite `sync` e por fim reinicialize o sistema operacional com o comando `reboot`. Caso ocorram erros ao final, por exemplo se ao reinicializar aparecer algum erro, basta desligar e ligar novamente a máquina virtual.

1.9. Oitavo caso

Como último passo de alterações, vá para o diretório `root` e lá crie o arquivo C de testes com o conteúdo mostrado no código 3. Após finalizar a edição, assim como anteriormente, basta agora salvar e fechar o arquivo.

```
1  #include <mycalllib.h>
2  int main(void)
3  {
4      int n = 10;
5      minhachamada(n);
6      return 0;
7  }
```

Listing 3. Programa em C para somar dois números

Com o fim das alterações, agora é necessário compilar o arquivo de testes, então, através do comando de terminal que utiliza o compilador C, `cc`, digite o seguinte dentro do diretório onde o arquivo C criado está localizado. `cc test1.c -o test1.out` e depois compile o arquivo com o comando `./test1.out`. Estando tudo feito de forma adequada, ao final dessas alterações, a chamada de sistema deve funcionar de maneira adequada.

1.10. Conclusão

A implementação de uma syscall no sistema operacional Minix 3.2.1 proporcionou uma compreensão aprofundada sobre o funcionamento interno do kernel e o processo de integração de novas funcionalidades no mesmo. Durante o desenvolvimento, explorei os principais arquivos e diretórios responsáveis por gerenciar as chamadas de sistema, bem como as interações entre o kernel e o servidor de gerenciamento de processos.

A criação da syscall "inútil" demonstrou de forma prática como configurar, prototipar e testar uma funcionalidade personalizada. Além disso, o trabalho reforçou a importância de ferramentas como o editor `vi`, comandos básicos de terminal e o compilador `cc`, que são indispensáveis para manipular sistemas baseados em Unix.

Este estudo serve como referência prática para desenvolvedores e estudantes interessados em sistemas operacionais, oferecendo um caminho claro para explorar os conceitos fundamentais de syscalls e suas aplicações. Assim, contribui para o aprofundamento teórico e técnico no campo da Engenharia de Computação.

2. References

- [Beletti 2015] Beletti (2015). Creating a system call - english. https://github.com/rhiguita/lab-minix/blob/master/eng_us/Seen2-Creating_a_System_Call-English.txt. Guide available on GitHub for learning purposes related to MINIX system calls.
- [Tanenbaum 2024] Tanenbaum, A. S. (2024). Minix 3 documentation. <https://www.minix3.org/doc/>. Official documentation for MINIX 3, a microkernel-based operating system.
- [Tanenbaum and Bos 2014] Tanenbaum, A. S. and Bos, H. (2014). *Modern Operating Systems*. Pearson Education, Upper Saddle River, NJ, 4th edition. Available in multiple editions and widely used in academia.