Recompilação do Núcleo como Atividade Prática no gerenciamento de "Chamadas de Sistemas" no Ensino de Sistemas Operacionais

Ian Andrade Moreira[1], **Swami de Paula Lima**[1], **Hamilton José Brumatto**[1]

[1]Departamento de Ciências Exatas e Tecnológicas
Universidade Estadual de Santa Cruz (UESC)

ianmoreira80@gmail.com, {sbtux,brumatto}@yahoo.com.br

Abstract. *This article proposes a practical experience in the teaching of Operating Systems, specifically in the demonstration of the theme: "System Calls", whose goal is to present to the student the experience of executing a code that performs a system call, as well as the code that responds to the system call, evidencing the change of control flow between the user level and the core level. The article describes the practical experience of simple achievement that results in great benefits for learning the theme.*

Resumo. *Este artigo propõe uma experiência prática no ensino de Sistemas Operacionais, em específico na demonstração do tema: "Chamadas de Sistemas", cujo objetivo é apresentar ao aluno a vivência da execução de um código que realiza uma chamada de sistema, bem como do código que responde à chamada do sistema, evidenciando a troca do fluxo de controle entre o nível de usuário e o nível de núcleo. O artigo descreve a experiência prática de simples realização que resulta em grandes benefícios para o aprendizado do tema.*

1. Introdução

O estudo de Sistema Operacional geralmente apresenta ao aluno as diversas funcionalidades do mesmo que é o gerenciamento de: chamadas de sistema, processos, memória, recursos, arquivos e entrada e saída. O laboratório prático desta matéria geralmente envolve dois tipos de recursos: sistemas reais (comerciais ou de ensino) ou simuladores. Cada qual com suas vantagens e desvantagens. O uso de um ou de outro pode depender da funcionalidade que se queira trabalhar.

O uso de simuladores oferecem a vantagem de ilustrar, de forma animada, as interações entre processos seja na fila de escalonamento, seja no acesso à memória ou aos recursos de entrada e saída. A desvantagem é que apesar de animado, a versão entregue aos alunos é puramente teórica, o mesmo resultado os alunos conseguem construindo um mapa passo a passo.

O uso de sistemas reais permite que o aluno consiga manipular diretamente o resultado do hardware, analisando: pilhas de memória, natureza de arquivos, portas de entrada/saída. Porém, se o aluno quiser entender de forma mais aprofundada o funcionamento é necessário uma capacidade grande de depuração do código para análise das funcionalidades. Mesmo para sistemas acadêmicos, como o MINIX ou TROPIX, a implementação do sistema como atividade prática requer um domínio grande do hardware e construção de algoritmos para um resultado básico.

Um aspecto básico do gerenciamento do Sistema Operacional: “Gerenciamento de Chamadas de Sistemas” possui conceitos abstratos que são de difícil aprendizado, como: “nível do usuário” e “nível do núcleo”. Uma prática que requer a recompilação do núcleo incluindo uma nova funcionalidade na chamada de sistema é um exemplo que leva o aluno a programar em ambos os níveis vivenciando todos os aspectos envolvidos com este gerenciamento.

Com base nesta proposta, este artigo apresenta uma implementação prática da experiência de recompilar o núcleo com a inclusão de uma nova chamada de sistema. Primeiramente são apresentadas algumas iniciativas realizadas no ensino prático da disciplina como também os conceitos envolvidos na proposta desta metodologia. Em seguida o desenvolvimento prático da experiência e por final considerações envolvendo o resultado desta aplicação no aprendizado.

2. Atividades Práticas

O desenvolvimento do aprendizado de Sistemas Operacionais baseado apenas em aulas teóricas traz mais dificuldades na assimilação dos processos quando comparado com o uso de atividades práticas na ilustração e demonstração dos processos envolvidos.

Estamos interessados em atividades práticas para o “Gerenciamento de Chamadas de Sistemas”. Uma aplicação prática que ilustra o gerenciamento de chamadas de sistema requer a diferenciação do código no nível do usuário e no nível do núcleo e como a chamada evolui entre um nível e outro.

O uso de simuladores como SOSim[Machado and Maia 2004] ou Nachos[Christopher et al. 1993] é prejudicado, pois este tipo de recurso não oferece um modelo prático para apresentação das chamadas de sistema.

A solução prática seria o uso de um sistema real, seja comercial ou acadêmico. Uma experiência que aborda todos os aspectos do gerenciamento de uma chamada de sistema seria implementar uma nova chamada de sistema no sistema operacional. Apenas criar programas que usam de chamadas de sistema não dá dimensão ao aluno da visão prática que ocorre no nível do núcleo durante a execução da chamada.

Apesar de sistemas acadêmicos como MINIX[Tanenbaum 1987] e outros serem uma versão simplificada de um sistema operacional comercial, usá-los neste caso em particular, não traz simplificação na experiência prática. Tanto no sistema acadêmico, como no sistema comercial, é necessário recompilar o núcleo para inserir uma nova chamada de sistema. Portanto optamos por recompilar o núcleo do sistema linux em uma distribuição.

Há muitos trabalhos sobre a recompilação do núcleo, e muitos trabalhos que envolvem atividades práticas em aulas de sistemas operacionais, mas não há trabalhos anteriores que consideram a recompilação do núcleo como atividade prática. Não que isto seja um feito novo, mas os tutores que já optaram em utilizar esta prática ainda não apresentaram resultados sobre isto.

3. A Prática

3.1. Descrição

O gerenciamento dos recursos da máquina requer programas eficientes e com privilégio de acesso, a CPU tem de estar no modo privilegiado para realizar algumas das ações. por

segurança, a grande maioria dos programas rodam no nível de usuário, sem tal privilégio, uma chamada de sistema (System Call) é um recurso muito usado para que se transfira para o núcleo (Kernel) a realização de alguma tarefa que deve ser feita pelo sistema operacional, dentro de seu programa com tal privilégio. Assim, quando um processo de usuário deseja utilizar uma chamada de sistema, o Mecanismo de chamada de sistema verifica se o processo tem a permissão para realizar aquela tarefa. Caso negativo o Sistema Operacional barra a realização da tarefa, e informa o processo que a requisição não foi atendida.

Na parte técnica, ocorre temporariamente uma troca do controle de fluxo. Os dados da aplicação são salvos, e é alternado para o modo de núcleo. A rotina do Sistema Operacional é executada no modo de núcleo, caso haja privilégios. Após o término, retorna para o modo usuário e restaura os dados salvos.

Dentre os serviços que podemos realizar com as chamadas de sistema estão:

- Criação de processos e seu gerenciamento;
- Gerenciamento da memória principal;
- Acesso a arquivos, diretórios e gerenciamento do sistema de arquivos;
- Manuseamento do E/S;
- Proteção;
- Redes, etc.

As chamadas de sistema podem ter nomes diferentes dependendo do sistema operacional que se esteja usando:

- Unix: System Call;
- Windows: Application Program Service (API);
- OpenVMS: System Services.

3.2. Materiais e Métodos

O mais importante na criação de uma chamada de sistema, é possuir o código fonte de um "kernel linux" que possa receber nossa função ou programa, que será chamado. Este código foi baixado diretamente do repositório disponibilizado pela Linux Foundation através do site Kernel.org[Linux Kernel Org Inc.]. Para efetivamente fazer uso do kernel, foi utilizada a distribuição Ubuntu 16.04 LTS, com o Kernel versão 4.16.17. O código foi todo escrito utilizando ferramentas já disponíveis na distribuição, como o editor de texto Gedit. Além disso, as seguintes ferramentas e bibliotecas que foram utilizadas:

- gcc
- make

Além disso, foram utilizadas as seguintes bibliotecas:

- ncurses-dev
- fakeroot
- build-essential
- xz-utils
- libssl-dev
- bc

As configurações do computador usado são irrelevantes para o sucesso da tarefa, e serão ignorados. A tarefa foi realizada, também, em outros computadores, com as mesmas configurações, com sucesso igualmente observado.

3.3. A experiência

O programa que foi decidido ser escrito faz uso apenas de bibliotecas contidas no próprio kernel Linux, sem uso de nenhum programa ou biblioteca externa. Espera-se que seja uma experiência de simples realização, para servir como atividade prática, cujo principal objetivo é demonstrar o fluxo de controle através dos modos usuário e núcleo. Desta forma optou-se por um programa simples que utiliza a biblioteca `printk` (descrita abaixo), logo, utiliza-se a linguagem C e uma receita passo a passo que o aluno deve reproduzir. Neste código busca-se realizar uma chamada que, ao ser invocada, captura e guarda em um arquivo de log, informações referentes a todos os processos que estão em execução na máquina no momento. Os resultados são salvos utilizando o `DMESG`, e o log normalmente está no caminho `/var/log`.

Por organização, é criado o diretório `info` no fonte do kernel, onde será armazenado todos os arquivos que fazem parte dessa chamada de sistema

O objetivo da chamada do sistema é realizar, para cada processo em execução, uma busca de informações sobre o processo e registrar estas informações em um arquivo de log. Os parâmetros registrados são obtidos a partir da estrutura dinâmica `task_struct`:

- Nome do Processo
- Número de ID do Processo
- Estado do Processo
- Prioridade
- Prioridade em Tempo Real
- Prioridade Estática
- Prioridade Dinâmica
- Política do Processo
- Número de CPUs permitidas
- CPU em uso
- Flags
- Trace
- Se for um processo filho, também: Nome do Processo Pai e ID do Processo Pai

Listagem 1. Biblioteca de Chamada de Sistema criada

```
  asmlinkage long sys_listProcessInfo(void) {
    struct task_struct *proces;

    for_each_process(proces) {

    printk("Process:~%s\nPID_Number: %ld\nProcess State: %ld\nPriority: %ld\n \
       RT_Priority: %ld\nStatic Priority: %ld\nNormal Priority: %ld\n \
       Policy: %u\nCpus Allowed: %d\nCPU: %u \nFlags: %u \nTraces: %u \n",
       proces->comm, (long)task_pid_nr(proces), (long)proces->state,
       (long)proces->prio, (long)proces->rt_priority, (long)proces->static_prio,
       (long)proces->normal_prio, (unsigned int)proces->policy,
       (int)proces->nr_cpus_allowed, \
       (unsigned)proces->cpu, (unsigned)proces->flags, (unsigned)proces->ptrace
    );

   if(proces->parent)
      printk("Parent process: %s, PID_Number: %ld",proces->parent->comm,
      (long)task_pid_nr(proces->parent));

   printk("\n\n");
  }
  return 0;
}
```

Uma biblioteca extra foi criada, apenas para ter o escopo da função utilizada. O código de nossa chamada de sistema é mostrado na Listagem 1.

Algo que deve ser observado é que foi utilizada a função de sistema `printk`. Quando estamos em modo de núcleo, a função `printf` (utilizada normalmente pelo biblioteca stdio.h) não pode ser utilizada, ela não é implementada no modo de núcleo. Assim, a função `printk` tem o mesmo escopo e realiza igualmente as mesmas funções, com a diferença que funciona apenas em modo de núcleo e serve para registrar informações normalmente usadas para depuração das ações no modo de núcleo.

Neste ponto, os alunos tomam ciência de que o código que estão preparando é um código que será rodado em um outro nível no sistema operacional, em um nível com mais privilégio.

Outro detalhe importante é o uso do modificador `asmlinkage`. Normalmente em uma chamada de função, na execução do código, os primeiros parâmetros da função estão gardados no registradores e os demais na pilha. A chamada de sistema funciona de forma diferente, todos os parâmetros estão na pilha. O que esse modificador faz é simplesmente informar para o compilador que ele deve procurar os parâmetros utilizados na chamada de sistema diretamente na pilha da CPU, ao invés de olhar os registradores.

3.4. Fazendo rodar

Para que uma função seja considerada uma chamada de sistema, ela deve ser inserida diretamente nas funções de Kernel (núcleo). Para que isso aconteça, o kernel deve ser recompilado contento a função que escrevemos. Para tanto é garantir que nossa chamada está presente na tabela contendo a lista com as chamadas de funções disponíveis para aquele sistema.

Cada chamada de sistema possui para si um número inteiro associado a aquela chamada. Normalmente, cada chamada deve possuir um *wrapper* no nível do usuário que faça o tratamento e faça a chamada. Para este artigo, entretanto, faremos nossa chamada usando diretamente o número de chamada de função presente na tabela de chamadas, isto evita a criação de uma biblioteca no nível do usuário que seria a responsável por realizar a chamada de sistema. A tabela `syscall_64.tbl` está presente no código fonte do SO que será recompilado. Nossa chamada é inserida no final das chamadas, utilizando um número que esteja disponível, como podemos ver na figura 1

E por fim, como última configuração antes de ter o núcleo recompilado, é necessário que seja incluído o protótipo da chamada de sistema no arquivo de cabeçalhos `syscalls.h` para que a chamada seja reconhecida, como podemos ver na figura 2

Podemos agora recompilar o kernell, atendendo aos passos para incluir o módulo criado na imagem do SO gerada. A imagem é instalada no diretório de carregamento e a opção de escolha criada no menu de carregamento do sistema.

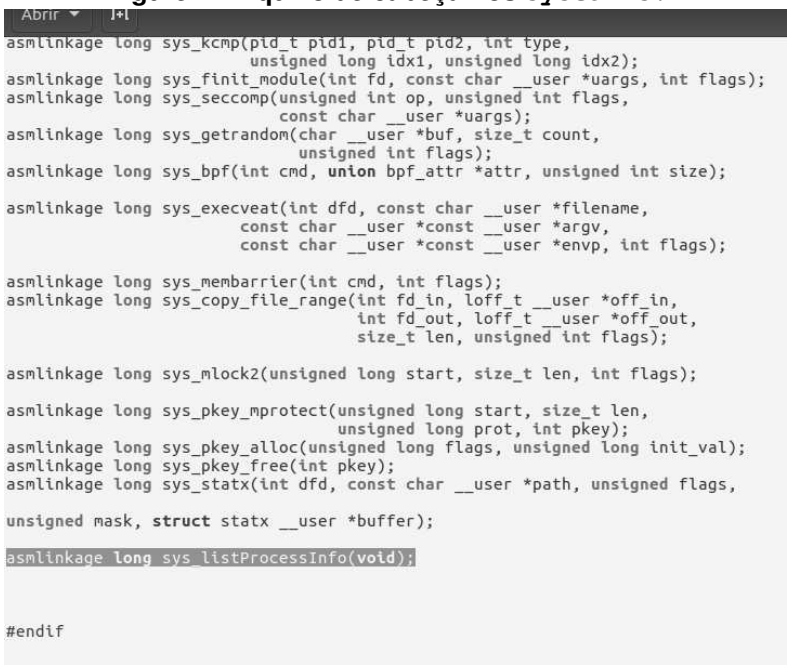
3.5. Vendo o resultado

Para testar a chamada de sistema, é criado um programa simples que apenas faz a chamada de sistema utilizando o número que foi registrado em `syscall_64.tbl`. Como é uma chamada de núcleo, o programa pode estar em qualquer lugar do seu sistema, que irá funcionar normalmente, o código pode ser visto na Listagem 2

Figure 1. Tabela das chamadas de sistemas do SO

```
Abrir
319     common  memfd_create            sys_memfd_create
320     common  kexec_file_load         sys_kexec_file_load
321     common  bpf                     sys_bpf
322     64      execveat                sys_execveat/ptregs
323     common  userfaultfd             sys_userfaultfd
324     common  membarrier              sys_membarrier
325     common  mlock2                  sys_mlock2
326     common  copy_file_range         sys_copy_file_range
327     64      preadv2                 sys_preadv2
328     64      pwritev2                sys_pwritev2
329     common  pkey_mprotect           sys_pkey_mprotect
330     common  pkey_alloc              sys_pkey_alloc
331     common  pkey_free               sys_pkey_free
332     common  statx                   sys_statx
333     common  listProcessInfo         sys_listProcessInfo
#
# x32-specific system call numbers start at 512 to avoid cache impact
# for native 64-bit operation.
#
512     x32     rt_sigaction            compat_sys_rt_sigaction
```

Figure 2. Arquivo de cabeçalhos `syscalls.h`



```
Abrir
asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_getrandom(char __user *buf, size_t count,
                              unsigned int flags);
asmlinkage long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);

asmlinkage long sys_execveat(int dfd, const char __user *filename,
                        const char __user *const __user *argv,
                        const char __user *const __user *envp, int flags);

asmlinkage long sys_membarrier(int cmd, int flags);
asmlinkage long sys_copy_file_range(int fd_in, loff_t __user *off_in,
                                    int fd_out, loff_t __user *off_out,
                                    size_t len, unsigned int flags);

asmlinkage long sys_mlock2(unsigned long start, size_t len, int flags);

asmlinkage long sys_pkey_mprotect(unsigned long start, size_t len,
                                  unsigned long prot, int pkey);
asmlinkage long sys_pkey_alloc(unsigned long flags, unsigned long init_val);
asmlinkage long sys_pkey_free(int pkey);
asmlinkage long sys_statx(int dfd, const char __user *path, unsigned flags,

unsigned mask, struct statx __user *buffer);

asmlinkage long sys_listProcessInfo(void);


#endif
```

Listagem 2. Rotina que realiza a chamada de sistema

```c
#include <stdio.h>
#include <linux/kernel.h>
#include <sys/syscall.h>
#include <unistd.h>
int main()
{
   printf("Invoking 'listProcessInfo' system call");
   long int ret_status = syscall(333); // is the syscall number

   if(ret_status == 0)
        printf("System call 'listProcessInfo' executed correctly. \
        Use dmesg to check processInfo\n");
   else
        printf("System call 'listProcessInfo' did not execute as expected\n");
   return 0;
}
```

O resultado da execução deste código será um registro no log de depuração do núcleo `dmesg`, a figura 3 mostra uma pequena parte desta listagem, lá fica evidente que foi de fato realizada a impressão que esperávamos do núcleo do sistema operacional.

Figure 3. Parte da listagem obtida no `dmesg`

```
[ 8570.583477] Parent process: kthreadd,          PID_Number: 2
[ 8570.583479]

[ 8570.583487] Process: gnome-terminal-
                        PID_Number: 22943
                        Process State: 1
                        Priority: 120
                        RT_Priority: 0
                        Static Priority: 120
                        Normal Priority: 120
                        Policy: 0
                        Cpus Allowed: 4
                        CPU: 3
                        Flags: 4194304
                        Traces: 0
[ 8570.583490] Parent process: upstart,          PID_Number: 1130
[ 8570.583491]

[ 8570.583500] Process: bash
                        PID_Number: 22950
                        Process State: 1
                        Priority: 120
                        RT_Priority: 0
                        Static Priority: 120
                        Normal Priority: 120
                        Policy: 0
                        Cpus Allowed: 4
                        CPU: 1
                        Flags: 4194304
                        Traces: 0
[ 8570.583503] Parent process: gnome-terminal-,          PID_Number: 22943
[ 8570.583504]
```

4. Resultados

Está prática foi introduzida inicialmente na disciplina de Sistemas Operacionais para uma dupla de alunos voluntários por sugestão do professor da disciplinas. O resultado obtido foi a gratificação da dupla em conseguir realizar o feito, bem como, ao fazer um relatório da prática, identificar o funcionamento de uma chamada de sistema de forma completa, o funcionamento de um código no nível do usuário e o momento em que a chamada é executada no nível de núcleo.

A experiência permite uma fácil aplicação em sala, ou até mesmo como uma tarefa fora da sala, pois uma vez criada a imagem do núcleo e usada na experiência prática, ela pode ser descartada retornando o sistema à situação original.

Um outro resultado importante com esta prática é a familiarização do manuseio do sistema operacional linux pelos alunos. Este sistema é importante no meio da ciência da computação e muitas vezes negligenciado pelos alunos que preferem partir para uma solução mais popular.

5. Conclusões e Trabalhos Futuros

Com o objetivo de uma demostração prática da funcionalidade do sistema opercional de "gerenciamento das chamadas de sistemas" a proposta deste artigo revela um resultado satisfatório. O aprendizado do aluno no assunto é facilitado a partir do momento que na prática ele consegue vivenciar os espaços de execução de um código envolvendo o sistema operacional e um pouco da estrutura interna do mesmo.

A facilidade de execução desta experiência abre um leque de possibilidades de novas funcionalidades que, a título de experiência, o aluno pode inserir no sistema operacional.

A próxima etapa será incluir esta prática como atividade para toda a turma, com isto poderemos confirmar se os ganhos neste caso podem ser reproduzidos como benefício para o aprendizado da disciplina.dois do

References

Christopher, W. A., Procter, S. J., and Anderson, T. E. (1993). The nachos instructional operating system. Technical Report UCB//CSD-93-739, University of California, Berkeley.

Linux Kernel Org Inc. kernel.org. https://www.kernel.org/ acessado em Julho/2018.

Machado, F. B. and Maia, L. P. (2004). Um framework construtivista no aprendizado de sistemas operacionais - uma proposta pedagógica com o uso do simulador sosim. XII Workshop de Educação em Computação.

Tanenbaum, A. S. (1987). *Operating Systems: Design and Implementation*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.