UNIVERSIDADE FEDERAL DE SÃO CARLOS
Bacharelado em Ciência da Computação
Laboratório de Sistemas Operacionais
Professor Gustavo Maciel Dias Vieira
***Campus* Sorocaba**

Projeto 3
Módulos e Estruturas Internas do Núcleo

Daniel Ramos Miola 438340
Giulianno Raphael Sbrugnera 408093

Sorocaba
2013

1. Introdução

O Projeto 3 tem por objetivo a criação de módulos personalizados e a consequente expansão do núcleo com a adição desses módulos, além de explorar e alterar algumas informações internas dos processos, presentes nas estruturas internas.

2. Discussão e Resultados

Ao longo do projeto, houve um total de três tarefas a serem realizadas, as quais seus resultados são discutidos abaixo.

2.1. Compilar módulo fornecido como exemplo, carregar e verificar funcionamento.

A partir do módulo fornecido, este foi compilado, criando um arquivo objeto *hello.ko*. Utilizando o comando *insmod*, inserimos o módulo na memória para uso e a efeito de teste. Ao invocar *cat /proc/hello*, obtivemos a mensagem "*Hello, World!*", garantindo que o módulo foi carregado e executado com sucesso. Em seguida, removemos o módulo através do comando *rmmod*.

2.2. Modificar módulo para exibir PID de processo que lê arquivo e de seu processo pai.

O núcleo do sistema operacional Linux guarda os dados de um processo em uma estrutura própria chamada *task_struct*, a qual podemos acessar através da macro *current* que se comporta como um ponteiro para o registro do processo atual. Assim foi possível acessar o PID do processo que executa o módulo e do processo que é seu pai, no caso o interpretador de comandos, o qual é referenciado na estrutura do processo com um ponteiro do tipo *task_struct* denominado *parent,* possibilitando a impressão dos PID's.

```
//imprime PID do processo atraves da macro current
seq_printf(m, "current PID:%d\n",current->pid);

//imprime PID do processo pai
seq_printf(m, "parent  PID:%d\n",current->parent->pid);
```

Imagem 1. Trecho do código responsável por imprimir o PID do processo que lê o arquivo e de seu processo pai

O código acima é responsável por imprimir o PID do processo que executa o módulo e de seu processo pai. É chamada a função *seq_printf*, onde seus argumentos são um ponteiro do tipo *seq_file* e um ponteiro para uma string. Ou seja, é na função *seq_printf* que os PID's são impressos. O PID, por sua vez, é acessado pela macro *current*, onde há um campo que o armazena.

2.3. Modificar módulo para elevar permissão do interpretador de comandos para root.

Para elevar as permissões do interpretador de comandos executando o módulo é necessário alterar a estrutura de credenciais do processo a qual guarda em especial os id's de usuário e grupo o qual executa o mesmo. A estrutura *cred* responsável por tais credenciais também é referenciada por um ponteiro *cred* na estrutura dos dados do processo. O id igual a 0 é o *root*, o qual pode fazer tudo no sistema. Logo, apenas alterando as credenciais na estrutura *cred* para 0 fazem com que o interpretador de comandos receba permissões de *root*.

```
/*
gera ponteiro mutavel para alteração, altera variaveis da
estrutura inserindo ID 0(root) e libera o ponteiro no fim
da edição.
*/
credential = (struct cred *) get_cred(current->parent->cred);

//altera as variaveis relativas ao id de usuario na estrutura cred
credential->uid = 0;    /* real UID of the task */
credential->gid = 0;    /* real GID of the task */
credential->suid = 0;   /* saved UID of the task */
credential->sgid = 0;   /* saved GID of the task */
credential->euid = 0;   /* effective UID of the task */
credential->egid = 0;   /* effective GID of the task */

put_cred(credential);
```

Imagem 2. Trecho responsável por dar ao interpretador de comandos executando o processo de leitura permissões de *root*

Por *cred* ser uma estrutura protegida foi necessário primeiro obter uma referência não constante, alterar a estrutura e depois liberá-la com os métodos próprios. No caso o interpretador de comandos que executa a chamada *cat* para o módulo é o processo pai, sendo acessado por *current->parent->cred.* Foram alteradas então para 0 as variáveis que guardam os id's de usuário e grupo do processo, definindo-o assim com privilégios de super usuário.

3. Dificuldades encontradas na realização

As principais dificuldades foram encontrar as estruturas de dados utilizadas pelo núcleo para armazenar os dados dos processos, bem como definir quais variáveis alterar para elevar a permissão de um processo em execução.
As variáveis da estrutura *cred* responsáveis por armazenar os id's de usuário e grupo são divididas em reais, salvas, efetivas e mais um tipo não identificado. Ficou claro que ao se alterar a variável do tipo efetiva, o privilégio é concedido, porém fica a dúvida de quais variáveis

realmente devem ser alteradas para definir uma transição de privilégio limpa e se isso não acarretará em algum problema posterior.

4. Conclusão

Concluímos então que com o conhecimento das estruturas internas ao núcleo é possível construir um módulo que aumente a flexibilidade de um núcleo monolítico, e de como é mantida a organização de tais módulos ao serem carregados e descarregados dinamicamente para suprir as necessidades do sistema operacional. Bem como são estruturados os dados dos processos e a forma como é possivel percorrer por eles através de um módulo carregado que tem acesso total ao núcleo por ser monolítico.

5. Bibliografia

[1] Slides do curso Laboratório de Sistemas Operacionais – UFSCar Sorocaba

[2] Slides do curso Sistemas Operacionais – UFSCar Sorocaba