Trabalho 1

Sistemas Operacionais

15/12/2021

Módulos de kernel Linux

Neste projeto, você trabalhará com módulos de kernel do sistema operacional Linux. Todas as instruções desse trabalho foram testadas usando o sistema Ubuntu 20.04.3 (kernel 5.11). Se você estiver usando uma distribuição Linux diferente, você precisará adaptar as instruções, ou instalar uma distribuição compatível.

Como discutido em nossas aulas, código do kernel é executado em modo privilegiado, isto é, com permissão total. Isso permite que qualquer instrução seja executada, qualquer operação de I/O seja iniciada e qualquer área da memória possa ser acessada diretamente. Isso significa que qualquer erro no código pode ser desastroso e travar o sistema. Portanto, é mais seguro realizar este trabalho usando uma máquina virtual (como com o virtualbox), pois neste caso, qualquer falha, na pior das hipóteses, exigirá apenas a reinicialização do sistema.

1. Parte 1: Compilando módulos do kernel

Primeiramente, verifique se os pacotes make, binutils e gcc estão instalados e atualizados. Se eles não estiverem instalados, eles podem ser instalados com os seguintes comandos:

sudo apt update

sudo apt install make binutils gcc

A primeira parte deste trabalho consiste em seguir uma série de etapas para criar e carregar um módulo no kernel do Linux.

Para listar todos os módulos do kernel que estão carregados no momento digite o comando lsmod no terminal. Este comando irá listar os módulos do kernel carregados com três colunas de informações: nome, tamanho e quem está usando o módulo.

O programa listado na Figura 1, denominado simple.c, ilustra um módulo de kernel muito básico que imprime algumas mensagens quando ele é carregado, isto é, passa a fazer parte do kernel, e descarregado, isto é, deixa de fazer parte do kernel.

A função int simple\_init() é o ponto de entrada do módulo, ou seja, a função que é disparada automaticamente quando o módulo é carregado no kernel. Similarmente, a função void simple\_exit() é o ponto de saída do módulo, ou seja, a função que é chamada quando o módulo é removido do kernel.

A função usada como ponto de entrada do módulo deve retornar um valor inteiro, com 0 representando sucesso e qualquer outro valor representando fracasso. A função usada como ponto de saída do módulo deve retornar void. Nem o ponto de entrada nem o ponto de saída do módulo recebem quaisquer parâmetros. As duas macros a seguir são usadas para registrar com o kernel, respectivamente, os pontos de entrada e saída do módulo:

Module\_init(simple\_init);

Module\_exit(simple\_exit);

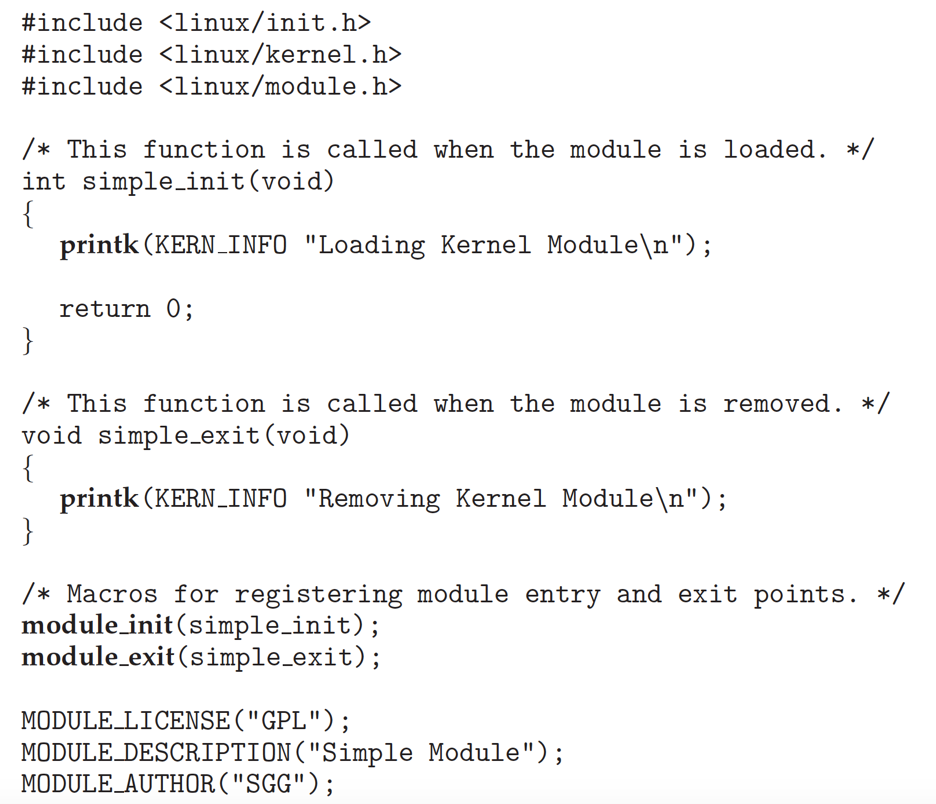


Figura 1: modulo de kernel simple.c

Módulos de kernel são carregados dinamicamente no sistema operacional. Eles não são executados na linha de comando como programas do usuário. Consequentemente, eles não têm I/O padão com teclado (stdin) e monitor (stdout).

Observe as funções printk() no código do módulo apresentado na figura 1. Esta função é no kernel o equivalente a um printf() em um programa de usuário, mas a sua saída é enviada para o buffer de log do kernel, cujo conteúdo pode ser lido usando o comando dmesg. Uma diferença entre printf() e printk() é que printk() nos permite especificar um flag de prioridade, cujos valores são fornecidos no arquivo de include <linux/printk.h>. Neste caso, a prioridade é KERN\_INFO, que é definida como uma mensagem informativa.

As linhas finais, onde encontramos MODULE\_LICENSE(), MODULE\_DESCRIPTION() e MODULE\_AUTOR(), representam os detalhes relativos à licença do software, descrição do módulo e autor. Para nossos propósitos, não exigimos essas informações, mas nós as incluímos porque é uma prática padrão no desenvolvimento de módulos de kernel.

Para compilar o módulo de kernel simple.c use o Makefile disponibilizado junto com a descrição do trabalho no sistema AVA. Um Makefile é apenas um arquivo de configuração que instrui o que o utilitário make deve fazer. O make é usado para automatizar o processo de construção de aplicações chamando o compilador, linkeditor, executando testes, e até mesmo fazendo o deploy, entre outras operações possíveis. Para compilar o módulo, digite no terminal o comando make na pasta onde estiver os arquivos simple.c e Makefile. A compilação produz vários arquivos. O arquivo simple.ko contém o módulo de kernel compilado. A etapa a seguir descreve como o módulo compilado no kernel do Linux.

1. Parte 2: Carregando e removendo módulos do kernel

Módulos do kernel são carregados usando o comando insmod. Assim, para carregar o módulo compilado na parte 1, excute o seguinte comando no terminal:

sudo insmod simple.ko

Para verificar se o módulo foi carregado, digite o comando lsmod e procure pelo módulo simple. Lembre-se de que o ponto de entrada do módulo é invocado quando o módulo é inserido no kernel. Para verificar o conteúdo da mensagem no buffer de registro do kernel, digite o comando dmesg no terminal. Você deverá ver a mensagem "Loading Kernel Module". Para remover o módulo do kernel use o comando rmmod (observe que o sufixo .ko é desnecessário):

sudo rmmod simple

Certifique-se de com o comando dmesg que o módulo foi removido. Como o buffer de log do kernel pode encher rapidamente, faz sentido limpa-lo periodicamente. Para isso digite no terminal o seguinte comando.

sudo dmesg -c

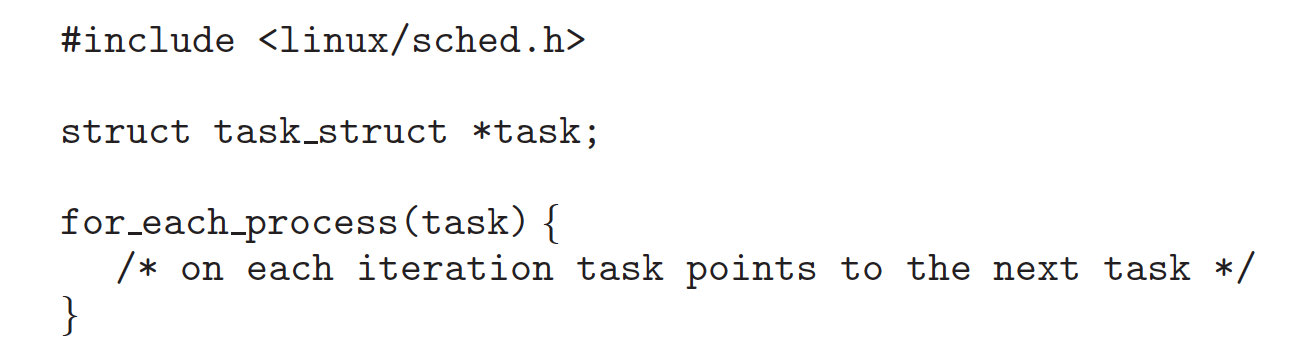
Após ter se assegurado de que o módulo foi carregado e descarregado corretamente, vamos fazer algumas modificações simples. Em nossas aulas, discutimos a função do temporizador (timer), bem como do manipulador de interrupções do temporizador. No Linux, a taxa na qual o temporizador dispara (a taxa de tique) é o valor de HZ definido no arquivo de cabeçalho <asm/param.h>. Este valor determina a frequência das interrupções do temporizador e seu valor varia de acordo com a máquina e arquitetura. Por exemplo, se o valor de HZ é 100, ocorre 100 interrupções do temporizador a cada segundo. Além disso, o kernel mantém a variável global jiffies, que mantém o número de interrupções do temporizador que ocorreram desde que o sistema foi inicializado. A variável jiffies é declarada no arquivo de cabeçalho <linux /jiffies.h>. Considerando essas informações, faça o seguinte:

1. Imprima usando printk() os valores de jiffies e HZ na função simple\_init().
2. Imprima usando printk() o valor de jiffies na função simple\_exit().
3. Usando os diferentes valores de jiffies obtidos em simple\_init() e simple\_exit(), determine o número de segundos decorridos desde o momento em que o módulo foi carregado até o momento em que foi removido.
4. Módulo de kernel Linux para listagem de Tarefas

Nesta parte do trabalho, você codificará um módulo de kernel para listar todas as tarefas em execução em um Sistema Linux. Seu módulo deve iterar pelas tarefas linearmente e fazendo busca em profundidade. No linguajar de Linux, tarefa é um termo genérico para se referir tanto a um processo quanto a uma thread.

* 1. Iterando linearmente

No kernel do Linux, a macro for\_each\_process() facilita percorrer as tarefas atualmente em execução no sistema. Como discutido em nossas aulas, no Linux a PCB é chamada de task\_struct e é definida no arquivo de cabeçalho <linux/sched.h>. Para iterar sobre as tarefas e acessar os vários campos da estrutura, basta passar um ponteiro para a macro for\_each\_process(), a cada iteração o ponteiro apontará para a próxima task\_struct. No Ubuntu a macro for\_each\_process() é definida no arquivo <Linux/sched/signal.h>.



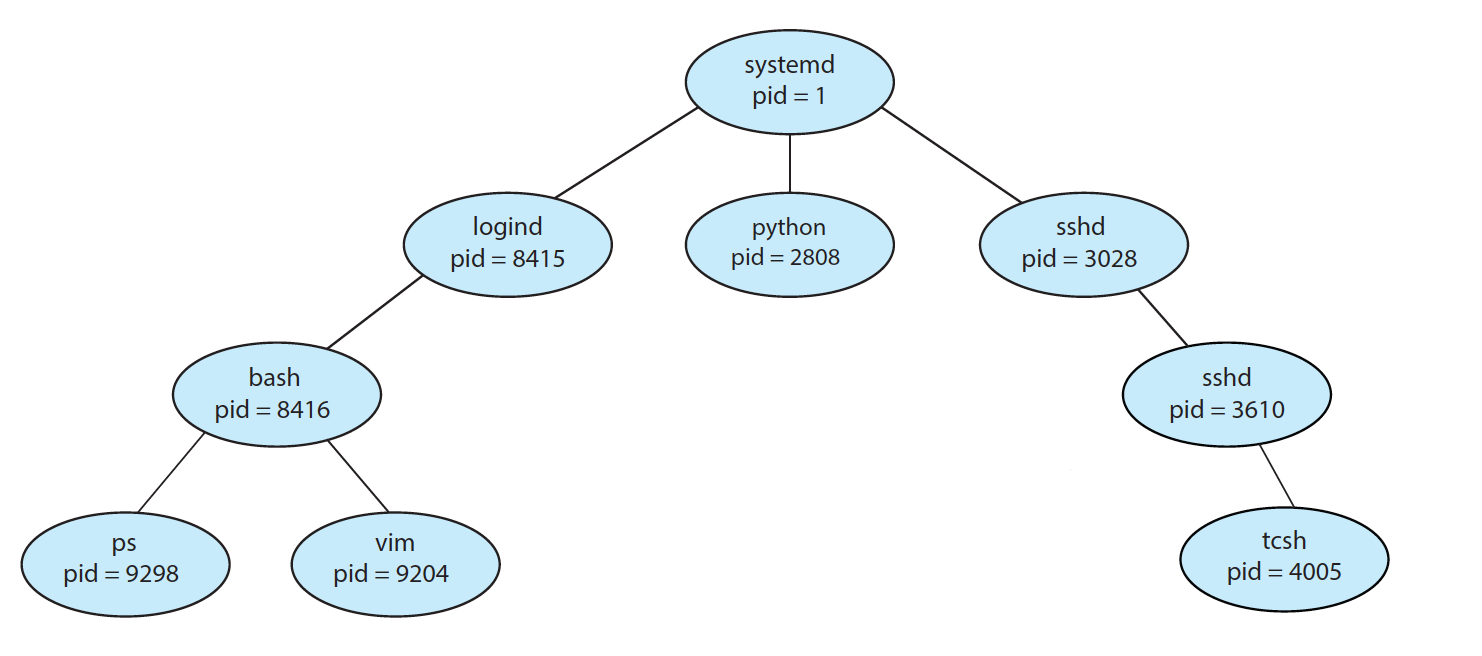
Programe um módulo de kernel que itera por todas as tarefas do sistema usando a macro for\_each\_process(). Em particular, imprima o comando da tarefa, o estado, e a id de cada do processo de cada tarefa (Você provavelmente terá que ler o código da estrutura task\_struct em <linux/sched.h> para obter os nomes destes campos). Escreva o código no ponto de entrada do módulo, e faça com que o que foi solicitado apareça no buffer de log do kernel, para que possamos visualizar usando o comando dmesg. Para verificar se o seu código está funcionando corretamente, compare o conteúdo do buffer de log do kernel com a saída do comando que lista todas as tarefas do sistema:

ps -el

Os dois valores devem ser muito semelhantes. Como as tarefas são dinâmicas, no entanto, é possível que algumas tarefas apareçam em uma lista, mas não na outra.

* 1. Iterando nas tarefas com busca em profundidade

O objetivo agora será iterar por todas as tarefas do sistema usando busca em profundidade. Como um exemplo, a busca em profundidade na árvore abaixo resultaria na seguinte ordem de visita: 1, 8415, 8416, 9298, 9204, 2808, 3028, 3610, 4005.



O Linux armazena sua árvore de processos como uma série de listas. Examinando a estrutura task\_struct em <linux/sched.h>, vemos dois atributos struct list\_head: children e sibling. Cada um desses atributos são ponteiros para uma lista, a primeira contendo os filhos da tarefa e a segunda contendo seus irmãos. O Linux também mantém uma referência à tarefa inicial no sistema(init\_task) que é do tipo task\_struct. Usando essas informações, bem como operações macro em listas, podemos iterar sobre os filhos de init\_task da seguinte maneira:

struct task\_struct \*task;

struct list\_head \*list;

list\_for\_each(list, &init\_task->children) {

task = list\_entry(list, struct task\_struct, sibling);

/\* task aponta para o próximo filho na lista \*/

}

A macro list\_for\_each() recebe dois parâmetros, ambos do tipo list\_head:

* O segundo parâmetro é um ponteiro para a cabeça da lista a ser percorrida.
* O primeiro parâmetro é um ponteiro para o nó cabeça da lista a ser percorrida.

A cada iteração de list\_for\_each(), o primeiro parâmetro aponta para estrutura de lista do próximo filho. Em seguida, usamos esse valor para obter cada estrutura na lista usando a macro list\_entry(). Primeiramente, seu trabalho é pesquisar sobre essas macros e entender como usá-las corretamente. Em seguida, começando pela tarefa init\_task, implemente um módulo de kernel que itera sobre todas as tarefas do sistema usando busca em profundidade (dica: use uma função recursiva). Assim como no exercício da subseção 3.1, imprima o nome, estado e pid de cada tarefa no buffer de log do kernel. Para verificar se seu resultado está correto use o comando

ps -eLf

Este comando lista todas as tarefas - incluindo threads - no sistema. Para verificar se você realmente realizou uma iteração apropriada, você terá que examinar as relações entre as várias tarefas geradas pelo comando ps.