PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS

FACULDADE DE ENGENHARIA DE COMPUTAÇÃO - ESCOLA POLITÉCNICA

Amanda Soares da Silveira – 21018595

Beatriz Cupa Newman – 22002150

Eduardo de Faria Rios Perucello – 22009978

Júlia Machado Duran – 22009210

Luana Bresciani Baptista – 22006563

Luigi Bertoli Menezes – 22000113

Luiz Henrique Souza Custódio da Silveira – 21019531

IMPLEMENTAÇÃO DE NÚCLEO DE SISTEMA OPERACIONAL

CAMPINAS

2025

1. Introdução

Este projeto consiste no desenvolvimento de dois módulos de kernel para o sistema operacional Linux: [1] [2]

- **`kfetch_module`**: Fornece informações do sistema através de um dispositivo virtual `/dev/kfetch`. [1]
- **`risk_module`**: Avalia o risco de processos específicos com base no tempo de uso da CPU e chamadas de sistema (`syscall`), acessível via `/proc/process_risk/risk_score`. [2] [4]

Dessa forma, o objetivo do projeto é desenvolver módulos de kernel e interações com subsistemas do Linux, para melhor entendimento do sistema operacional.

2. Racional de design

a. Escolha dos módulos

- **`kfetch_module`**: Proporciona uma interface simples para obtenção de informações do sistema, útil para monitoramento e diagnóstico. Os componentes são:

    - **`Kernel:`** versão da release do kernel do Linux
    - **`NOME-DO-HOST`**: nome decidido pelo kernel do Sistema Operacional
    - **`CPU:`** `distribuidora | versão | modelo | frequência de clock` do processador do computador
    - **`CPUs:`** número de núcleos (ou threads) disponíveis para uso
    - **`Mem:`**
        - `FreeRam: (número de páginas livres * tamanho da página)/1024² MB`
        - `TotalRam: (número de páginas totais* tamanho da página)/1024² MB`
    - **`Uptime:`** duração em minutos do sistema rodando
    - **`Proc:`** número de processos rodando no computador (há uma diferença mínima entre os processos contados no momento exato em que o módulo é printado e quando o comando `ps -e --no-headers | wc -l` é executado depois. Além dos processos do

comando, processos podem ter morrido ou nascido. O sistema está sempre dinâmico e essas diferenças são normais.)

- **`risk_module`**: Permite a avaliação de risco de processos, facilitando a identificação de processos com alto consumo de CPU.
    - **Gerenciamento de Arquivos e Descritores**
        - `files_struct`: Representa os arquivos abertos por um processo.
        - `file`: Estrutura de arquivo aberta.
        - `fdtable` (via `files_fdtable()`): Tabela de descritores de arquivos.
        - `S_ISSOCK`: Macro para identificar se um arquivo é um socket.
    - **Informações de Processo**
        - `task_struct`: Estrutura central com todas as informações do processo.
            - `utime`, `stime`: Tempos de CPU em modo usuário e kernel.
            - `nvcsw`, `nivcsw`: Trocas de contexto voluntárias/involuntárias.
            - `cred->uid.val`: Verifica privilégios de root (UID 0).
            - `start_time`: Tempo de criação do processo.
            - `mm->exe_file`: Arquivo executável do processo.
            - `files`: Descritores de arquivos do processo.
            - `ioac.read_bytes`, `ioac.write_bytes`: Operações de I/O (se compilado com `CONFIG_TASK_IO_ACCOUNTING`).
    - **Rede**
        - `S_ISSOCK`: Para detectar sockets em descritores de arquivos.
    - **Acesso ao Sistema de Arquivos /proc**
        - `/proc`: Interface de pseudo-sistema de arquivos.
        - `proc_mkdir`, `proc_create`, `remove_proc_entry`: Criação e remoção de entradas em `/proc`.
        - `proc_ops`: Define operações de leitura e escrita no arquivo do módulo.

b. Arquitetura

Ambos os módulos interagem diretamente com o kernel do Linux, utilizando interfaces como `/dev` e `/proc` para comunicação com o espaço de usuário.

- **Estrutura do Diretório:**
  - `kfetch_module.c`: Código fonte do módulo `kfetch_module`.
  - `risk_module.c`: Código fonte do módulo `risk_module`.
  - `Makefile`: Script para compilação dos módulos.

3. Implementação técnica

a. Escolha dos módulos

- **Sistema Operacional**: Linux *(6.8.0-55 generic)*
- **Ferramentas Utilizadas**: GCC, Make, `insmod`, `rmmod`
  - **`kfetch_module:`**
    - **`<linux/module.h>`**: Biblioteca principal para a criação de módulos no kernel, fornece funções como `module_init`, `module_exit`, e macros como `MODULE_LICENSE` e `MODULE_DESCRIPTION`.
    - **`<linux/fs.h>`**: Fornece funções relacionadas a arquivos e sistemas de arquivos, essencial para trabalhar com dispositivos de caractere e suas operações (`open`, `read`, `write`, `release`).
    - **`<linux/cdev.h>`**: Utilizada para lidar com dispositivos de caractere dinâmicos (cdev), como a inicialização de `cdev` e seu registro.
    - **`<linux/uaccess.h>`**: Fornece funções para acessar dados entre o espaço de usuário e o kernel, como `copy_to_user` e `copy_from_user`.
    - **`<linux/device.h>`**: Usada para criar e destruir dispositivos e classes de dispositivos no espaço de usuário (com funções como `class_create`, `device_create`).
    - **`<linux/utsname.h>`**: Contém informações sobre o sistema, como nome do host e versão do kernel, acessados via a função `utsname()`.
    - **`<linux/sched.h>`**: Fornece acesso às estruturas de agendamento de processos, como `task_struct` e funções como `for_each_process` para iterar sobre os processos.

- **<linux/mm.h>**: Usada para obter informações sobre memória, como funções de gerenciamento de memória e estrutura `sysinfo`.
- **<linux/timekeeping.h>**: Fornece funções para acessar o tempo do sistema, como `ktime_get_boottime_seconds()`.
- **<linux/mutex.h>**: Usada para criar e gerenciar mutexes, garantindo exclusão mútua entre operações de leitura e escrita.
- **<linux/sched/signal.h>**: Contém funções e estruturas relacionadas ao controle de sinais e manipulação de processos no kernel.
- **<asm/processor.h>**: Fornece informações relacionadas à arquitetura do processador, como `boot_cpu_data`, usado para obter informações sobre o modelo da CPU.
- **<linux/string.h>**: Utilizada para manipulação de strings no kernel, com funções como `snprintf` e `strstr`.
- **<linux/sysinfo.h>**: Fornece funções para acessar informações sobre o sistema, como a função `si_meminfo()` para obter dados sobre memória.

- **risk_module:**
  - **<linux/module.h>**: Macro `module_init`, `module_exit`, e suporte a módulos.
  - **<linux/kernel.h>**: Funções básicas do kernel.
  - **<linux/init.h>**: Inicialização e finalização de módulos.
  - **<linux/fs.h>**: Estruturas e funções para arquivos.
  - **<linux/file.h>** e **<linux/fdtable.h>**: Tabela de arquivos e descritores.
  - **<linux/proc_fs.h>**: Manipulação do sistema `/proc`.
  - **<linux/uaccess.h>**: Comunicação entre espaço de usuário e kernel.
  - **<linux/sched.h>**: Acesso à `task_struct` e agendador.
  - **<linux/seq_file.h>**: Suporte a arquivos de leitura sequencial.
  - **<linux/pid.h>**: Manipulação de IDs de processo.

b. Fluxo de Execução

1. Compilação dos módulos com o comando `make`.
2. Carregamento dos módulos no kernel utilizando `sudo insmod [kfetch_module.ko || risk_module.ko]`.

3. Verificação da criação dos dispositivos `ls /dev/kfetch` e `ls /proc/process_risk/risk_score`.
4. Interação com os módulos através de `sudo cat /dev/kfetch || cat /proc/process_risk/risk_score`

4. Resultado dos testes e validação

a. Testes Realizados

- **`kfetch_module`**:
  - Verificação da criação do dispositivo `/dev/kfetch`.
  - Leitura das informações fornecidas pelo módulo.
- **`risk_module`**:
  - Carregamento do módulo e verificação da criação de `/proc/process_risk/risk_score`.
  - Escrita e leitura de valores em `/proc/process_risk/risk_score` para diferentes PIDs

b. Resultados Obtidos

Ambos os módulos funcionaram conforme o esperado, fornecendo as informações e funcionalidades propostas. No **`risk_module,`** dependendo dos diferentes resultados da análise de risco, a saída é em cores diferentes:

- Risco alto: vermelho;
- Risco médio: amarelo e
- Risco baixo: verde.

5. Desafios e Soluções

a. Desafios Encontrados

- **Gerenciamento de Recursos**: Garantir que os módulos não causassem vazamentos de memória ou conflitos com outros módulos.
- **Interação com o Kernel**: Compreender as interfaces do kernel para criar dispositivos e manipular informações de processos.[4]

b. Soluções Implementadas

- **Gerenciamento de Recursos**: Utilização adequada das funções `kmalloc` e `kfree` para alocação e liberação de memória.
- **Interação com o Kernel**: Estudo e aplicação das APIs do kernel para criação de dispositivos e manipulação de informações de processos.

6. Conclusão

O desenvolvimento dos módulos kfetch_module e risk_module aprofundou o conhecimento em programação de módulos de kernel no Linux, incluindo manipulação de estruturas internas e criação de interfaces em /proc. Ambos os módulos funcionaram conforme esperado, extraindo informações detalhadas de processos e avaliando riscos com base em critérios normalizados e ponderados.

O projeto destacou a importância de um design cuidadoso para garantir segurança, estabilidade e eficiência no kernel. O risk_module demonstrou como combinar métricas quantitativas e qualitativas para gerar uma avaliação de risco eficaz, útil na detecção de comportamentos anômalos.

Por fim, o trabalho reforça o valor da programação de módulos para monitoramento avançado e segurança do Linux, contribuindo para a formação técnica em sistemas operacionais e segurança.

7. Referências Bibliográficas

[1] THE LINUX KERNEL ARCHIVE. *Linux Kernel Documentation*. Disponível em: https://www.kernel.org/doc/html/latest/. Acesso em: 29 maio 2025.

[2] LOVE, Robert. *Linux Kernel Programming*. 3. ed. Nova York: McGraw-Hill, 2010.

[3] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO/IEC 12207: Software Life Cycle Processes*. Genebra: ISO, 2017.

[4] DING, Mingyu. *Linux Kernel Module Programming Guide*. Disponível em: https://sysprog21.github.io/lkmpg/#hello-world. Acesso em: 29 maio 2025.

[5] SILVEIRA, Amanda Soares da; NEWMAN, Beatriz Cupa; PERUCELLO, Eduardo de Faria Rios; DURAN, Júlia Machado; BAPTISTA, Luana Bresciani; MENEZES, Luigi Bertoli; SILVEIRA, Luiz Henrique Souza Custódio da. *projeto_SO* [repositório GitHub]. Disponível em: https://github.com/JuliaDuran15/projeto_SO. Acesso em: 29 mai. 2025.

x