Hendrick Felipe Scheifer
João Victor Briganti
Luiz Gustavo Takeda

Manipulação de *Threads*

Relatório técnico de atividade prática solicitado pelo professor Rodrigo Campiolo na disciplina de Sistemas Operacionais do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Universidade Tecnológica Federal do Paraná – UTFPR
Departa mento Acadêmico de Computação – DACOM
Bacharelado em Ciência da Computação – BCC

Campo Mourão
Outubro / 2024

Resumo

Este trabalho visa aplicar conceitos práticos sobre operações com *threads*, explorando o uso da biblioteca POSIX Threads para a criação de programas. Utilizando o hipervisor VirtualBox versão 7.1.2 para configurar uma máquina virtual com o sistema operacional GNU/Linux Debian. Os procedimentos realizados foram a listagem das *threads* em execução, verificação do número máximo de *threads* que o sistema suporta e a medição do tempo de execução de um programa que realiza cálculos de matrizes. Os resultados foram satisfatórios, alcançando os objetivos propostos e proporcionando uma melhor compreensão sobre a utilização de *threads* na programação.

Palavras-chave: VirtualBox. POSIX Threads. GNU/Linux Debian.

Sumário

1 Introdução

A princípio, os sistemas operacionais não suportavam mais de uma tarefa por processo, o que se tornou um grande problema conforme a complexidade das aplicações desenvolvidas se tornou maior. Uma mesma aplicação atualmente exerce diversas funções simultâneas, o que gerou a necessidade de mais de uma tarefa em um mesmo processo, operando sob os mesmos recursos (MAZIERO, 2019).

Para resolver o problema citado anteriormente, surgem fluxos de execuções independentes menores em um mesmo processo, denominados *threads*, que compartilham recursos entre si (MAZIERO, 2019). As *threads* são essenciais em aplicações que exigem a execução simultânea de várias funções. São muito mais leves, o que as torna mais rápidas para criar e destruir, além disso, favorecem o desempenho das aplicações ao otimizar a computação e as operações de entrada/saída, permitindo a sobreposição dessas atividades. (TANENBAUM, 2016).

2 Objetivos

Este trabalho visa explorar e detalhar a criação e manipulação de *threads* em um ambiente GNU/Linux, fornecendo explicações abrangentes sobre a biblioteca utilizada para essa manipulação.

3 Fundamentação

Threads de núcleo são utilizadas para mapear *threads* de usuário no núcleo do sistema operacional. Além de serem utilizadas por atividades internas do próprio núcleo. A maneira como essas *threads* de usuário são mapeadas para as de núcleo dependem do modelo de implementação utilizado (MAZIERO, 2019).

Os principais modelos de *threads* são os seguintes modelos: N:1, onde várias *threads* de usuários são mapeadas para uma *thread* de núcleo, reduzindo a carga de gerência, visto que o núcleo enxerga tudo como uma única *thread*. Temos também o modelo 1:1, onde cada *thread* de usuário gera uma de núcleo, tornando a distribuição do processamento entre *threads* mais justa, porém é pouco escalável devido à grande carga de *threads* de núcleo geradas em aplicações maiores. E por fim temos o modelo N:M, sendo uma abordagem híbrida entre os dois modelos citados anteriormente, onde N *thread* de usuário são mapeadas para M *threads* de núcleo (sendo M<N), esta abordagem, embora mais complexa, explora o melhor de cada um dos outros dois modelos (MAZIERO, 2019).

Antes de 1995, cada sistema operacional criava sua própria interface para a programação de *threads*. Mas no ano citado, foi estabelecido o padrão IEEE POSIX 1003.1c,

popularmente conhecido como POSIX Threads, este padrão determina uma interface para a utilização de *threads* na linguagem C e ainda é amplamente suportado e disseminado atualmente (MAZIERO, 2019).

4 Materiais e Equipamentos

- Especificações do computador utilizado:
  - Modelo: Notebook Lenovo Thinkpad E14
  - CPU: AMD Ryzen 5-3500U
  - Memória Principal: 8 GB RAM
  - Memória Secundária: SSD 256 GB NVMe
  - Sistema Operacional: Fedora 40
- Hipervisor: VirtualBox 7.1.2
- Sistema Operacional no Hipervisor: GNU/Linux Debian 12.7
- Núcleo: Linux 6.10.11

5 Procedimentos e Resultados

Nesta seção, apresentaremos os procedimentos utilizados para a manipulação de *threads* em um ambiente GNU/Linux. Abordaremos a verificação do número de *threads* em execução, a capacidade máxima suportada e o impacto no desempenho de programas gerados por essas *threads*.

5.1 Listagem de *Threads*

No sistema GNU/Linux, é possível obter informações sobre o estado do sistema através da leitura de arquivos localizados em `/proc` (NEGUS, 2012). Na Figura 1, apresentamos a leitura do arquivo `/proc/loadavg`, que fornece dados sobre a carga média de execução do sistema, na quarta coluna, após o símbolo "/", verificamos que o sistema possui 115 *threads* em execução.

```
joao@linux:~$ cat /proc/loadavg
0.00 0.00 0.00 1/115 1254
```

Figura 1 – Saída da leitura do arquivo `/proc/loadavg`.

A listagem das *threads* pode ser realizada com a execução do seguinte comando `ps -eLo pid,lwp,nlwp,user,cmd -sort -nlwp`, cuja saída é apresentada na Figura 2.

```
joao@linux:~$ ps -eLo pid,lwp,nlwp,user,cmd --sort -nlwp
    PID     LWP NLWP USER     CMD
    570     570    2 systemd+ /lib/systemd/systemd-timesyncd
      1       1    1 root     /sbin/init
      2       2    1 root     [kthreadd]
      3       3    1 root     [pool_workqueue_release]
      4       4    1 root     [kworker/R-rcu_gp]
      5       5    1 root     [kworker/R-sync_wq]
      6       6    1 root     [kworker/R-slub_flushwq]
      7       7    1 root     [kworker/R-netns]
     11      11    1 root     [kworker/u16:0-events_unbound]
     12      12    1 root     [kworker/R-mm_percpu_wq]
     13      13    1 root     [rcu_tasks_kthread]
     14      14    1 root     [rcu_tasks_rude_kthread]
     15      15    1 root     [rcu_tasks_trace_kthread]
     16      16    1 root     [ksoftirqd/0]
     17      17    1 root     [rcu_preempt]
     18      18    1 root     [rcu_exp_par_gp_kthread_worker/0]
     19      19    1 root     [rcu_exp_gp_kthread_worker]
     20      20    1 root     [migration/0]
     21      21    1 root     [idle_inject/0]
     22      22    1 root     [cpuhp/0]
     23      23    1 root     [cpuhp/1]
     24      24    1 root     [idle_inject/1]
     25      25    1 root     [migration/1]
     26      26    1 root     [ksoftirqd/1]
     28      28    1 root     [kworker/1:0H-events_highpri]
     29      29    1 root     [cpuhp/2]
```

Figura 2 – Saída do comando `ps -eLo pid,lwp,nlwp,user,cmd -sort -nlwp`.

O comando *ps* utilizado para a visualização das *threads* apresenta as seguintes *flags* (SHOTTS, 2017):

- **e:** Seleciona todos os processos.
- **o:** Permite ao usuário definir o formato da saída. No comando, foram selecionadas as seguintes informações:
  - **pid:** ID do processo.
  - **lwp:** ID da *thread*.
  - **nlwp:** Número de *threads* associadas a este processo.
  - **user:** Nome do usuário proprietário do processo em execução.
  - **cmd:** Comando que está sendo executado.
- **sort:** Ordena a saída de forma decrescente com base no número de *threads*, conforme especificado pela *-nlwp*.

Na saída apresentada na Figura 2, o processo com o maior número de *threads* é o comando `/lib/systemd/systemd-timesyncd`, que possui um total de 2 *threads*. Os demais processos listados têm apenas 1 *thread* cada.

5.2 Número Máximo de *Threads*

Informações sobre o número máximo de *threads* que podem ser criadas no sistema também podem ser obtidas através da leitura de arquivos em `/proc` (NEGUS, 2012). Na

Figura 3, apresentamos a leitura do arquivo `/proc/sys/kernel/max_threads`, que exibe o número máximo permitido de *threads*.

```
joao@linux:~$ cat /proc/sys/kernel/threads-max
31174
```

Figura 3 – Saída da leitura do arquivo `/proc/sys/kernel/max_threads`.

5.3 Tempo de Execução

Utilizando a biblioteca *pthread*, foi desenvolvido um programa para calcular, de maneira paralela, a média aritmética das linhas e a média geométrica das colunas de uma matriz $M \times N$. Os resultados são salvos em um arquivo de texto especificado pelo usuário.

Esta implementação adota técnicas de paralelização de funções e dados, distribuindo as tarefas de cálculo entre várias threads para maximizar a eficiência do processamento.

As especificações do sistema que executou o programa, estão detalhadas na Seção 4, e a matriz de entrada possui o tamanho 100×200. A Tabela 1 mostra o tempo de execução do programa com 1, 2, 4, 8 e 16 *threads*, destacando o impacto da paralelização.

Tabela 1 – Tempo de execução para diferentes quantidades de *threads*

Número de *threads*	**Tempo Real (ms)**	**Tempo de Sistema (ms)**	**Tempo de Usuário (ms)**
1	24	8	7
2	24	11	4
4	20	15	15
8	20	13	20
16	21	13	15

Para cada execução, foram medidos três tipos de tempo:

- **Tempo Real:** Representa o tempo total decorrido durante a execução.
- **Tempo de Sistema:** Indica o tempo gasto de execução operações do núcleo.
- **Tempo de Usuário:** Indica o tempo dedicado à execução do código de usuário do programa.

Os resultados mostram que o tempo real permaneceu estável, em torno de 20 ms, independentemente do número de *threads*. O tempo de sistema variou, com a execução em uma única *thread* apresentando um tempo menor em relação às demais, enquanto nas execuções com múltiplas *threads*, o tempo aumentou ligeiramente, atingindo valores entre 11 e 15 ms. Por outro lado, o tempo de usuário foi menor para a execução com 2 *threads*, atingindo um tempo médio de 4 ms, em comparação com as execuções com 4 *threads* ou mais, nas quais variou entre 13 e 20 ms.

6 Conclusões

Este trabalho explorou o uso da biblioteca de *threads* em um ambiente GNU/Linux. Um dos principais benefícios dessa abordagem é a sua simplicidade em comparação com o uso de processos, especialmente pela facilidade de compartilhamento do mesmo espaço de endereçamento, facilitando a divisão de carga de trabalho entre as *threads*. No entanto, vale ressaltar que está simplicidade traz que um erro em uma *thread* pode afetar todo o processo.

A decisão de empregar exclusivamente *threads* deve ser fundamentada na análise das necessidades específicas da aplicação, visando otimizar o desempenho e gerenciar os riscos associados. Como discutido na Subseção 5.3, a criação de *threads* para tarefas simples pode não resultar em ganhos significativos de desempenho.

Este trabalho destaca a importância das *threads* para o desenvolvimento de sistemas que demandam agilidade e eficiência, enfatizando a necessidade de uma análise cuidadosa na implementação.

7 Referências

MAZIERO, C. A. *Sistemas Operacionais: Conceitos e Mecanismos*. [S.l.]: UFPR, 2019. Citado 2 vezes nas páginas 4 e 5.

NEGUS, C. *Linux bible*. [S.l.]: John Wiley & Sons, 2012. v. 772. Citado 2 vezes nas páginas 5 e 6.

SHOTTS, W. *The Linux command line*. [S.l.]: LinuxCommand. org, 2017. Citado na página 6.

TANENBAUM, H. B. A. S. *Sistemas Operacionais Modernos*. [S.l.]: Pearson, 2016. Citado na página 4.