

**HÉRICK VITOR VIEIRA BITTENCOURT**

**ANÁLISE DE PAGE FAULTS**  
Trabalho M2

Itajaí (SC), novembro de 2024



**UNIVERSIDADE DO VALE DO ITAJAÍ**  
**CURSO DE BACHAREL EM**  
**CIÊNCIA DA COMPUTAÇÃO**

**ANÁLISE DE PAGE FAULTS**  
Trabalho M2

por

Hérick Vitor Vieira Bittencourt

Relatório da análise de page faults em aplicações  
Windows e Linux como parte do trabalho M2 da  
matéria de Sistemas Operacionais  
Orientador: Felipe Viel

Itajaí (SC), novembro de 2024

## 1 SOBRE O PROJETO

Para consolidar o aprendizado sobre escalonamento e memória, o projeto consiste na análise da geração de page faults em três programas diferentes, o primeiro sendo um alocador de memória com o tamanho de buffer predefinido durante tempo de compilação, o segundo programa analisado deve ser o sistema de comunicação entre processos e multithread desenvolvido durante o trabalho M1, por fim, deve ser feita a análise de page faults de um software multiplataforma, para avaliar a diferença dos resultados entre diferentes os sistemas operacionais Windows e Linux.

### 1.1 ANÁLISE DO PROGRAMA “MEMORY COST”

O programa C++ fornecido para a execução do projeto possui como parâmetro o valor que será usado para alocar em memória e realizar as operações, estas sendo:

- Alocação de memória

```
{
    Timer timer;
    for (int i = 0; i < iterationCount; ++i)
    {
        int* p = new int[bufSize / sizeof(int)];
        delete[] p;
    }
    printf("%.3f s to allocate %d MB %d times.\n", timer.GetElapsed(), bufSize / (1024 * 1024), iterationCount);
}

{
    Timer timer;
    double deleteTime = 0.0;
    for (int i = 0; i < iterationCount; ++i)
    {
        int* p = new int[bufSize / sizeof(int)];
        Timer deleteTimer;
        delete[] p;
        deleteTime += deleteTimer.GetElapsed();
    }
    printf("%.3f s to allocate %d MB %d times (%.3f s to delete).\n", timer.GetElapsed(), bufSize / (1024 * 1024), iterationCount, deleteTime);
}
```

Figura 1 - Código de alocação do vetor

- Escrita e leitura de valores

```

{
    // Initialize and zero the memory.
    int* p = new int[bufSize / sizeof(int)]();
    {
        // Repeatedly write to the already allocated memory.
        Timer timer;
        for (int i = 0; i < iterationCount; ++i)
        {
            memset(p, 1, bufSize);
        }
        printf("%.4f s to write %d MB %d times.\n", timer.GetElapsed(), bufSize / (1024 * 1024), iterationCount);
    }
    {
        // Repeatedly read from the already allocated memory.
        Timer timer;
        int sum = 0;
        for (int i = 0; i < iterationCount; ++i)
        {
            for (size_t index = 0; index < bufSize / sizeof(int); ++index)
            {
                sum += p[index];
            }
        }
        printf("%.4f s to read %d MB %d times, sum = %d.\n", timer.GetElapsed(), bufSize / (1024 * 1024), iterationCount, sum);
    }
    delete[] p;
}

```

Figura 2 - Código de escrita e leitura de valores do vetor

- Alocação + Escrita repetidamente

```

{
    // Repeatedly allocate, write, and free memory.
    Timer timer;
    double deleteTime = 0.0;
    for (int i = 0; i < iterationCount; ++i)
    {
        int* p = new int[bufSize / sizeof(int)];
        memset(p, 1, bufSize);
        Timer deleteTimer;
        delete[] p;
        deleteTime += deleteTimer.GetElapsed();
    }
    printf("%.4f s to allocate and write %d MB %d times (%.4f s to delete).\n", timer.GetElapsed(), bufSize / (1024 * 1024), iterationCount, deleteTime);
}

```

Figura 3 - Código de alocação e escrita

- Alocação + Leitura repetidamente

```

{
    // Repeatedly allocate, read, and free memory.
    Timer timer;
    int sum = 0;
    for (int i = 0; i < iterationCount; ++i)
    {
        int* p = new int[bufSize / sizeof(int)];
        for (size_t index = 0; index < bufSize / sizeof(int); ++index)
        {
            sum += p[index];
        }
        delete[] p;
    }
    printf("%.4f s to allocate and read %d MB %d times, sum = %d.\n", timer.GetElapsed(), bufSize / (1024 * 1024), iterationCount, sum);
}

```

Figura 4 - Código de alocação e leitura

Os resultados foram registrados e carregados no Qlik Sense para a geração dos gráficos, facilitando a interpretação das tarefas sob diferentes tamanhos de buffer.

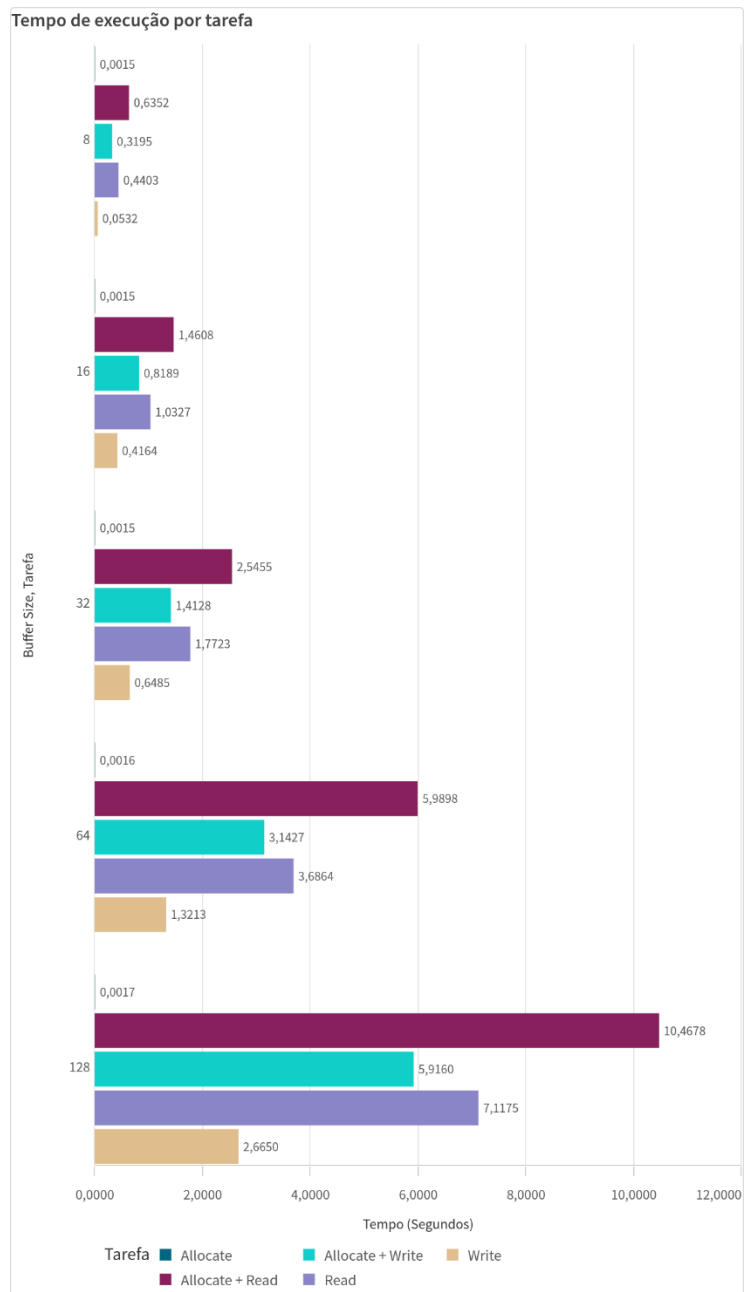


Figura 5 - Tempo de execução por tipo de tarefa executada no "Memory Cost"

Ao comparar os resultados, torna-se aparente que a tarefa de alocar memória é consistente entre todos os tamanhos de buffer fornecidos, no entanto, a leitura e escrita aumentam consideravelmente, dobrando de tempo para cada incremento no tamanho de buffer pelo próximo

valor na potência de dois, isto serve como indício de que as leituras e escrita de blocos consideráveis de dados podem afetar drasticamente o desempenho, ainda que realizados na memória principal.

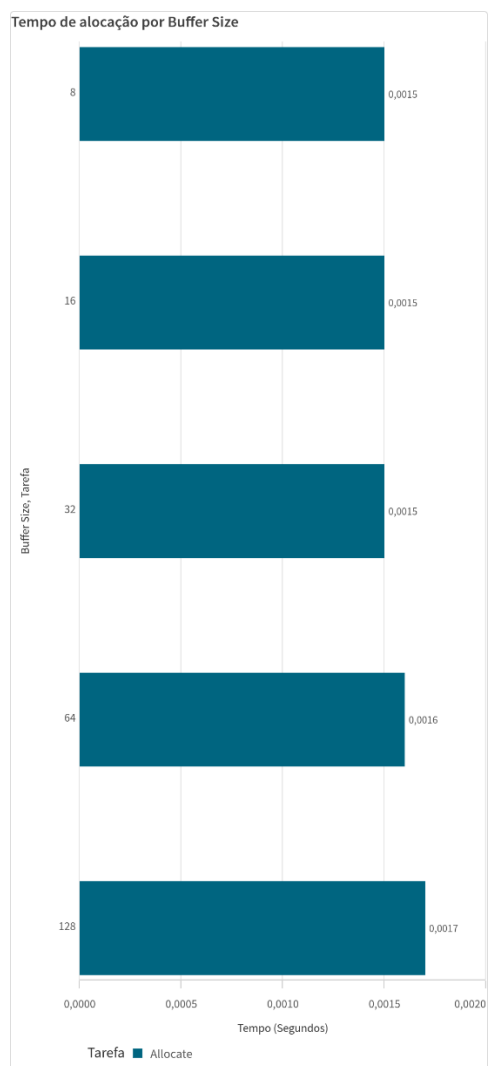


Figura 6 - Tempo de alocação por tamanho do buffer

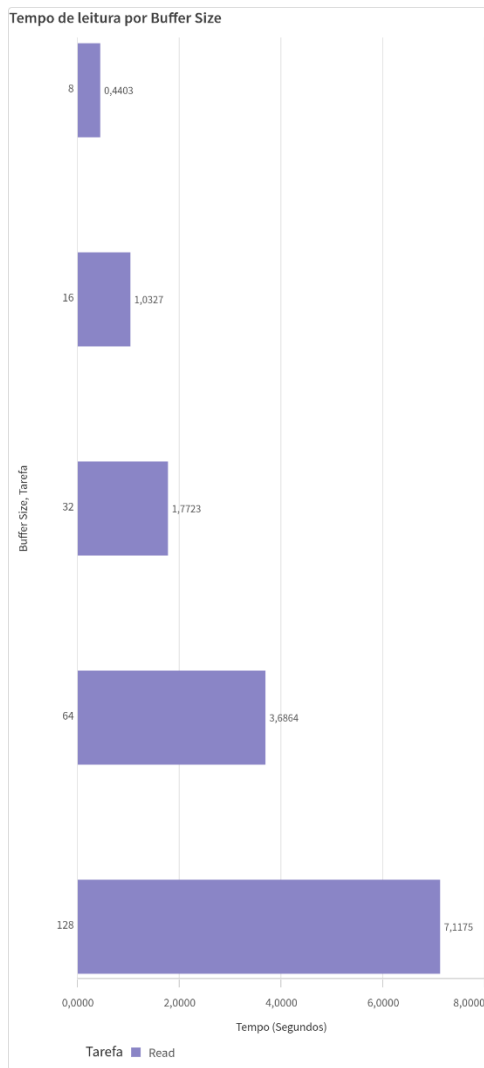


Figura 7 - Tempo de leitura por tamanho do buffer

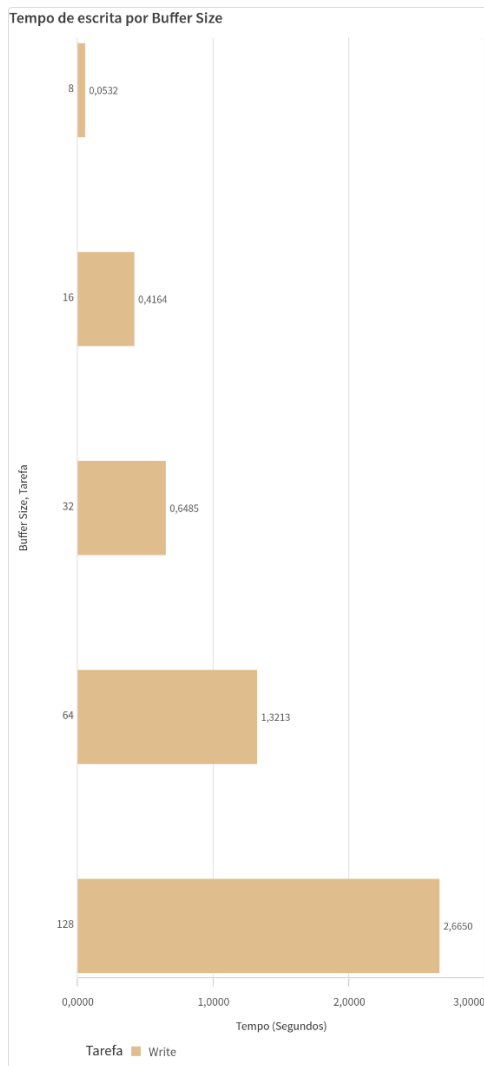


Figura 8 - Tempo de escrita por tamanho do buffer

Ao analisar o Memory Cost no Process Explorer, é observado que o programa inicia com poucas page faults, porém rapidamente aumenta ao chegar na seção de leitura, visto que necessita acessar endereços não presentes na memória principal, porém, como o programa não possui vazamentos de memória, a quantidade de page faults tornou-se previsível entre múltiplas execuções e o crescimento esperado é que o valor dobre a cada incremento do tamanho de buffer.



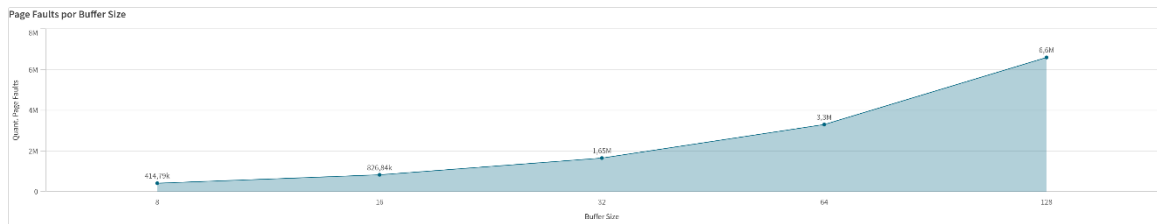


Figura 9 - Page faults por tamanho de buffer

Uma observação que ocorreu após alguns ajustes nas bibliotecas do código para torna-lo compatível com Linux é que ao analisar as page faults no Linux, ocorre um salto considerável em page faults a partir do momento em que o buffer size é maior que 16MB.

```
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ # Buffer size = 8
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ ps -p 148982 -o minflt,majflt,cmd
MINFL MAJFL CMD
660 0 ./MemoryCost
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ # Buffer size = 16
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ ps -p 150031 -o minflt,majflt,cmd
MINFL MAJFL CMD
666 0 ./MemoryCost
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ # Buffer Size 32
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ ps -p 151190 -o minflt,majflt,cmd
MINFL MAJFL CMD
106473 0 ./MemoryCost
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ # Buffer Size 64
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ ps -p 151944 -o minflt,majflt,cmd
MINFL MAJFL CMD
109691 0 ./MemoryCost
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ # Buffer Size 128
@Acanixz →.../SO-Codes/Avaliacoes/M1/windows $ ps -p 153293 -o minflt,majflt,cmd
MINFL MAJFL CMD
116123 0 ./MemoryCost
```

Figura 10 - Anomalia de Page Fault em buffers maiores do que 16MB devido a limite da memória cache

## 1.2 ANÁLISE DO TRABALHO M1

Ao executar o código em Linux através do Codespaces, foi inicialmente realizado uma checagem por page faults na execução dos clients numéricos e de string separadamente.

- Number client (Apresenta aproximadamente 82 minor page faults)

```
@Acanixz →VerRespostas/SO-Codes (main) $ ps -p 35533 -o minflt,majflt,cmd
MINFL MAJFL CMD
83 0 Avaliacoes/M1/number_client
@Acanixz →VerRespostas/SO-Codes (main) $ ps -p 35533 -o minflt,majflt,cmd
MINFL MAJFL CMD
82 0 Avaliacoes/M1/number_client
@Acanixz →VerRespostas/SO-Codes (main) $ ps -p 48603 -o minflt,majflt,cmd
MINFL MAJFL CMD
83 0 Avaliacoes/M1/number_client
@Acanixz →VerRespostas/SO-Codes (main) $
```

Figura 11 - Analise do processo "Number\_client"

- String client (Devido a maior complexidade, no primeiro processo, o primeiro salto para o loop requer uma major page fault, mas ainda similar ao number client, apresenta aproximadamente 80 page faults)



Figura 12 - Análise do processo "String\_client"

Os códigos apresentaram um valor consistente de minor page faults (valor do endereço não está no cache ou memória compartilhada) uma vez que entram no loop para enviar dados repetidamente, isso mostra que o escopo do código em loop está alocado a caber em apenas uma página, sem a necessidade de realizar a troca de páginas adicionais para a execução do código, com a exceção sendo o String Client, cujo requer um salto mais distante e consequentemente precisou ser alocado em outra página que não se encontra na memória física durante a execução do primeiro processo.

Já o servidor apresenta aproximadamente 110 page faults, com pouco crescimento de page faults, considerando que os threads foram inicializados e estão executando o mesmo trecho sem a necessidade de fazer saltos para códigos em páginas alternativas

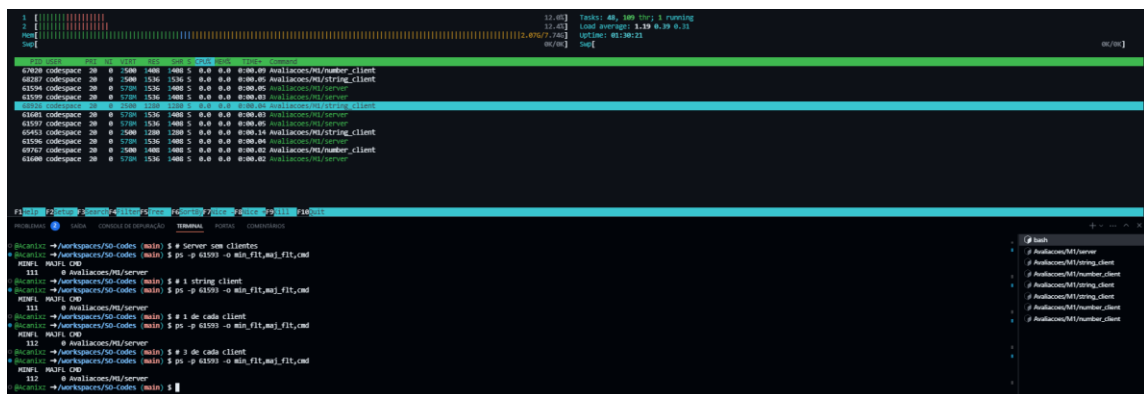


Figura 13 - Análise do servidor multithread

Para realizar a análise do projeto do trabalho M1 no Windows, foi feito a conversão do código para utilizar a biblioteca Windows.h, permitindo que o código seja executado com uma logica similar ao original, abaixo estão os resultados obtidos através do Process Explorer.

- Server
  - Aguardando conexão (1147 page faults)

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
M1_server.exe		1.104 K	4.252 K	3292			1.147

```

C:\Users\Acanixz\Desktop\Projects\SO-Codes\Avaliacoes\M2\M1_Windows\M1_server.exe
Server listening on 127.0.0.1:8080...
Server listening on 127.0.0.2:8081...
127.0.0.1 [THREAD 0]: Waiting for connection
127.0.0.2 [THREAD 2]: Waiting for connection
127.0.0.1 [THREAD 1]: Waiting for connection
127.0.0.1 [THREAD 3]: Waiting for connection
127.0.0.2 [THREAD 1]: Waiting for connection
127.0.0.2 [THREAD 0]: Waiting for connection
127.0.0.1 [THREAD 2]: Waiting for connection
127.0.0.2 [THREAD 3]: Waiting for connection
  
```

Figura 14 - Page faults do servidor no Windows

- Number Client conectado

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
M1_number_client.exe	< 0.01	664 K	3.868 K	2984			1.048
M1_server.exe	< 0.01	1.060 K	4.252 K	3292			1.153

Figura 15 - Page faults do number\_client no Windows

- 1 client de cada tipo
  - Quando há uma nova conexão, cada processo ganha algumas page faults até se estabilizar

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
M1_number_client.exe	< 0.01	676 K	3.880 K	2984			1.051
M1_server.exe	< 0.01	1.100 K	4.284 K	3292			1.162
M1_string_client.exe	< 0.01	716 K	3.884 K	16356			1.046

Figura 16 - Page fault de cada programa do trabalho M1 no Windows

- 3 clients de cada tipo
  - Novas instâncias dos processos geram menos page faults do que as primeiras instâncias, possivelmente devido a páginas compartilhadas estarem influenciando no carregamento das instruções.

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
M1_server.exe	< 0.01	1,064 K	4,268 K	3292			1,163
M1_number_client.exe	< 0.01	712 K	3,908 K	2984			1,059
M1_string_client.exe	< 0.01	668 K	3,868 K	16356			1,049
M1_string_client.exe	< 0.01	584 K	3,372 K	17860			894
M1_number_client.exe	< 0.01	588 K	3,376 K	16104			894
M1_string_client.exe	< 0.01	588 K	3,372 K	6148			893
M1_number_client.exe	< 0.01	580 K	3,376 K	12748			893

Figura 17 - Page faults do trabalho M1 em stress test no Windows

### 1.3 ANÁLISE DE SOFTWARE

O software escolhido para o teste livre foi um leitor de instruções do RISC-V, além de analisar as instruções, ele também faz o calculo de desempenho do através da quantidade de ciclos por instrução e fornece possíveis soluções para conflitos de hazards presentes no código assembly, que impediriam o código de fazer execução paralela, as opções incluem inserção de NO-Operators para atrasar a execução do código, reordenamento de instruções sem afetar o resultado do programa e simulação da técnica de forwarding, reduzindo a quantidade de NOPs necessários para que a correção aconteça.

Os resultados do programa foram obtidos usando os parâmetros abaixo:

- Tempo de clock: 1
- Arquivo: código\_dump
- Tecnica: 5 – Todas as opções

#### 1. Análise por passos:

##### (1) Inicializando

(main) O programa começa alocando as variáveis para guardar as informações do clock, arquivo escolhido e quais operações realizar. Esta fase gera aproximadamente 144 page faults. Os próximos parâmetros fornecidos não causam page faults, pois os

valores estão na memória cache e não requerem buscas adicionais.

The screenshot shows a Linux terminal window with two panes. The left pane displays system statistics: tasks (28, 84 thr; 1 running), load average (0.45 0.27 0.22), uptime (07:47:03), and a table of processes. The right pane shows a bash terminal with commands: `ps -p 286599 -o min_fit,maj_fit,cmd` and `ps -p 286599 -o min_fit,maj_fit,cmd`. The output of the first command is: `min_fit maj_fit cmd` and `144 0 ./main`. The output of the second command is: `min_fit maj_fit cmd` and `144 0 ./main`. Below the panes, the program output is visible: `sh: 1: cls: not found`, `Criando organizacoes:`, `ORGANIZACAO A`, and `Forneca o tempo de clock da organizacao A: []`.

Figura 18 - Page faults do analisador RISC-V (Fase 1 - Linux)

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
main.exe		1.040 K	4.700 K	18048			1.229

C:\Users\Acanixz\Desktop\Projects\SO-Codes\Avaliacaoes\M2\3-Desempenho\_Instrucoes\main.exe

Criando organizacoes:  
-----  
ORGANIZACAO A  
Forneca o tempo de clock da organizacao A:

Figura 19 - Page faults do analisador RISC-V (Fase 1 - Windows)

## (2) Tempo de clock fornecido

The screenshot shows a Linux terminal window with two panes. The left pane displays system statistics: tasks (31, 84 thr; 1 running), load average (0.43 0.27 0.22), uptime (07:47:21), and a table of processes. The right pane shows a bash terminal with commands: `ps -p 286599 -o min_fit,maj_fit,cmd` and `ps -p 286599 -o min_fit,maj_fit,cmd`. The output of the first command is: `min_fit maj_fit cmd` and `144 0 ./main`. The output of the second command is: `min_fit maj_fit cmd` and `144 0 ./main`. Below the panes, the program output is visible: `sh: 1: cls: not found`, `Criando organizacoes:`, `ORGANIZACAO A`, `Forneca o tempo de clock da organizacao A: 1`, and `Forneca o nome/caminho do arquivo contendo as instrucoes:`.

Figura 20 - Page faults do analisador RISC-V (Fase 2 - Linux)

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
main.exe		944 K	4.732 K	18048			1,249

```

C:\Users\Acanixz\Desktop\Projects\SO-Codes\Avaliacoes\M2\3-Desempenho_Instrucoes\main.exe

Criando organizacoes:
-----
ORGANIZACAO A
Forneca o tempo de clock da organizacao A: 1
-----

Forneca o nome/caminho do arquivo contendo as instrucoes:

```

Figura 21 - Page faults do analisador RISC-V (Fase 2 - Windows)

### (3) Arquivo fornecido

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
main.exe		944 K	4.732 K	18048			1,249

```

C:\Users\Acanixz\Desktop\Projects\SO-Codes\Avaliacoes\M2\3-Desempenho_Instrucoes\main.exe

Criando organizacoes:
-----
ORGANIZACAO A
Forneca o tempo de clock da organizacao A: 1
-----

Forneca o nome/caminho do arquivo contendo as instrucoes:
codigo_dump

Escolha a tecnica:
1 - Sem solucao em hardware, insercao de NOPs
2 - Forwarding, insercao de NOPs
3 - Sem solucao em hardware, reordenamento, insercao de NOPs
4 - Forwarding, reordenamento, insercao de NOPs
5 - Todas as opcoes acima

```

Figura 22 - Page faults do analisador RISC-V (Fase 3 - Linux)

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
main.exe		944 K	4.732 K	18048			1,249

```

C:\Users\Acanixz\Desktop\Projects\SO-Codes\Avaliacoes\M2\3-Desempenho_Instrucoes\main.exe

Criando organizacoes:
-----
ORGANIZACAO A
Forneca o tempo de clock da organizacao A: 1
-----

Forneca o nome/caminho do arquivo contendo as instrucoes:
codigo_dump

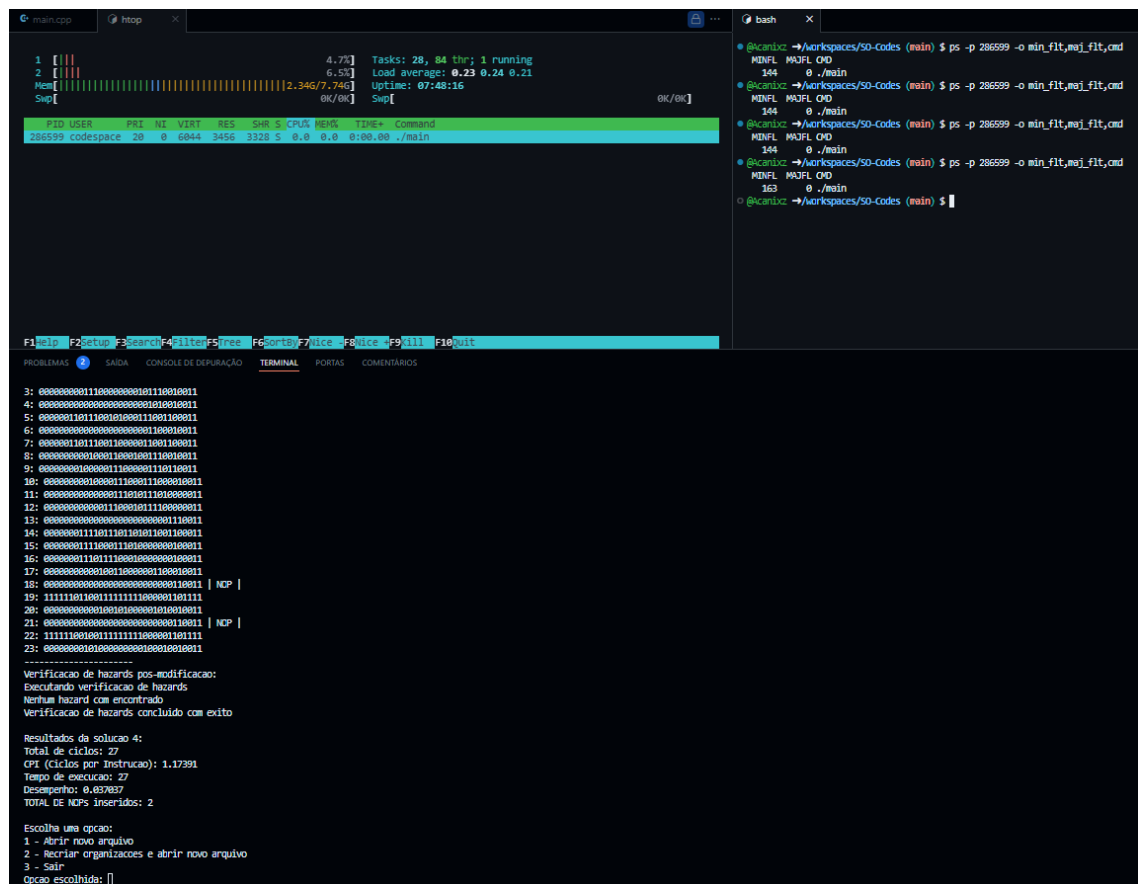
Escolha a tecnica:
1 - Sem solucao em hardware, insercao de NOPs
2 - Forwarding, insercao de NOPs
3 - Sem solucao em hardware, reordenamento, insercao de NOPs
4 - Forwarding, reordenamento, insercao de NOPs
5 - Todas as opcoes acima

```

Figura 23 - Page faults do analisador RISC-V (Fase 3 - Windows)

#### (4) Análise executada

O processo de análise do código e aplicação de todas as correções causa um aumento em page faults, considerando que o programa chama cada uma das funções para cada alternativa utilizada, independente se os hazards já foram verificados no mesmo arquivo anteriormente por exemplo, esta redundância faz com que diferentes trechos do código acabem saindo/voltando no escopo com mais frequência, consequentemente diminuindo o desempenho (não discernível sem análise) e aumentando as page faults.



```
1  [|||||] 4.7% Tasks: 28, 84 thr; 1 running
2  [|||||] 6.5% Load average: 0.23 0.24 0.21
Mem[|||||] 2.34G/7.74G Uptime: 07:48:16
Swp[|||||] 0K/0K Swp[|||||] 0K/0K

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
286599 codespace 20 0 6044 3456 3328 S 0.0 0.0 0:00.00 ./main

F1 Help F2 Setup F3 Search F4 Filter F5 Rec F6 Sortby F7 Nice F8 Force F9 Kill F10 Quit

PROBLEMAS SAÍDA CONSOLE DE DEBURAÇÃO TERMINAL PORTAS COMENTÁRIOS

3: 00000000111000000010110010011
4: 00000000000000000000001010010011
5: 0000011011100101000111001100011
6: 00000000000000000000001100010011
7: 0000011011100110000011001100011
8: 000000000001100110011110110011
9: 0000000100000111000001110110011
10: 000000001000011100011100010011
11: 00000000000011101011101000011
12: 00000000000110001011110000011
13: 000000000000000000000001110011
14: 0000001111011011011011001100011
15: 000000011100011101000000100011
16: 00000011101110001000000100011
17: 00000000010011000001100010011
18: 00000000000000000000000110011
19: 1111110110011111111100000110111 | NCP |
20: 00000000000100100000010010011
21: 00000000000000000000000110011 | NCP |
22: 1111110010011111111100000110111
23: 000000010100000001100010010011

-----
Verificacao de hazards pos-modificacao:
Executando verificacao de hazards
Nenhum hazard com encontrado
Verificacao de hazards concluido com exito

Resultados da solucao 4:
Total de ciclos: 27
CPI (ciclos por Instrucao): 1.17391
Tempo de execucao: 27
Desempenho: 0.837637
TOTAL DE NOPS inseridos: 2

Escolha uma opcao:
1 - Abrir novo arquivo
2 - Recriar organizacoes e abrir novo arquivo
3 - Sair
Opcao escolhida: []
```

Figura 24 - Page faults do analisador RISC-V (Fase 4 - Linux)

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Page Faults
main.exe		956 K	4.880 K	18048			1.325

```

C:\Users\Acanixz\Desktop\Projects\SO-Codes\Avaliacoes\M2\3-Desempenho_Instrucoes\main.exe
12: 00000000000011100010111100000011
13: 00000000000000000000000000001110011
14: 0000000011110111101101011001100011
15: 000000001111000111010000000100011
16: 000000001110111100010000000100011
17: 000000000001001100000001100010011
18: 0000000000000000000000000000110011 | NOP |
19: 111111011001111111111000001101111
20: 00000000000100101000001010010011
21: 0000000000000000000000000000110011 | NOP |
22: 111111001001111111111000001101111
23: 0000000010100000000100010010011
-----
Verificacao de hazards pos-modificacao:
Executando verificacao de hazards
Nenhum hazard com encontrado
Verificacao de hazards concluido com exito

Resultados da solucao 4:
Total de ciclos: 27
CPI (Ciclos por Instrucao): 1.17391
Tempo de execucao: 27
Desempenho: 0.037037
TOTAL DE NOPs inseridos: 2

Escolha uma opcao:
1 - Abrir novo arquivo
2 - Recriar organizacoes e abrir novo arquivo
3 - Sair
Opcao escolhida:

```

Figura 25 - Page faults do analisador RISC-V (Fase 4 - Windows)



## 2 CONCLUSÃO

A análise de diferentes processos em diferentes sistemas operacionais demonstram que é possível avaliar o desempenho de um programa através de sua necessidade por acesso a endereços distantes com frequência, além disso, foi observado que o uso de multithreading não afetou drasticamente o aumento de page faults na condição em que o código executado pelas threads não realiza acessos de variáveis globais e é relativamente simples para o espaço de memória caber em uma página (Figura 13), no entanto, observa-se que o custo em page faults aumenta drasticamente ao armazenar informações em buffers maiores que 16MB (Figura 10), indicando que a leitura e escrita continua não estava sendo capaz de segurar na cache todos os valores necessários para evitar page faults, necessitando da troca constante.