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 memoria, 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. 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

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Figura 1 - Código de alocação do vetor

* Escrita e leitura de valores

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

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

* Alocação + Escrita repetidamente

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Figura 3 - Código de alocação e escrita

* Alocação + Leitura repetidamenteTexto

  Descrição gerada automaticamente

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.

Gráfico, Gráfico de barras

Descrição gerada automaticamente

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 memoria 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, 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.

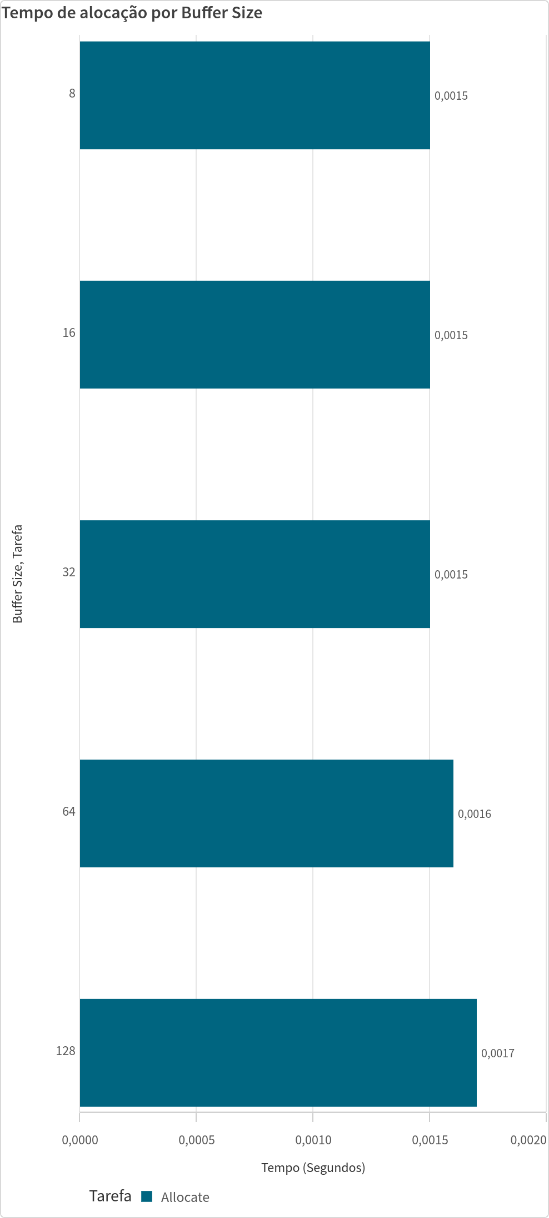


Figura 6 - Tempo de alocação por tamanho do buffer

Gráfico, Gráfico de caixa estreita

Descrição gerada automaticamente

Figura 7 - Tempo de leitura por tamanho do buffer

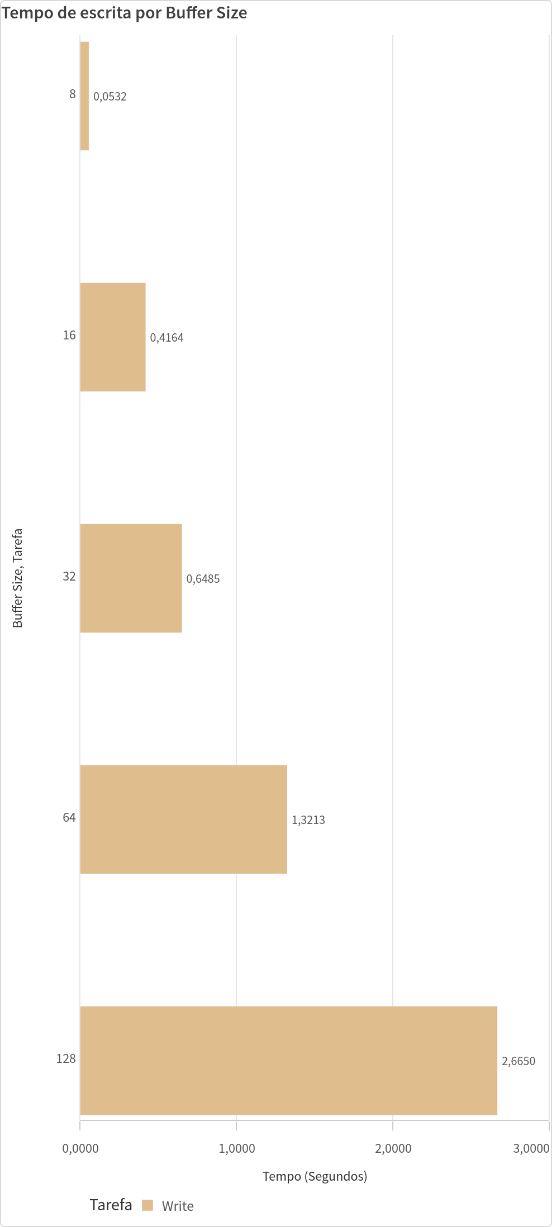


Figura 8 - Tempo de escrita por tamanho do buffer

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.

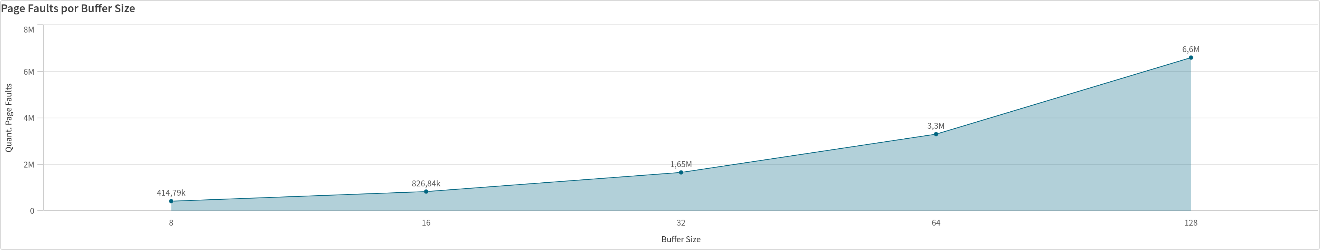


Figura 9 - Page faults por tamanho de buffer

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.

Texto

Descrição gerada automaticamente

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

* 1. 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)

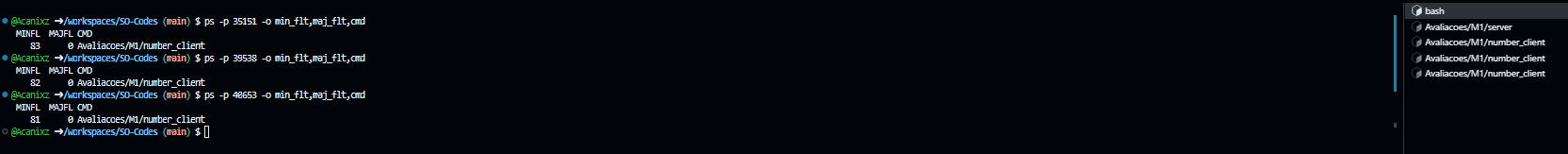


Figura 11 - Analise do processo "Number\_client"

* 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)

Forma

Descrição gerada automaticamente com confiança baixa

Figura 12 - Analise 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

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

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)

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Figura 14 - Page faults do servidor no Windows

* + Number Client conectado



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



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.

Tela de computador

Descrição gerada automaticamente

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

* 1. 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. **Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente**

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

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

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

* + - 1. **Tempo de clock fornecido**

**Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente**

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

**Texto

Descrição gerada automaticamente**

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

* + - 1. **Arquivo fornecido**

**Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente**

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

**Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente**

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

* + - 1. **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 analise) e aumentando as page faults.

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

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

**Texto

Descrição gerada automaticamente**

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

1. Conclusão

A analise 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 memoria 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.