Universidade do Vale do Itajaí
Escola Politécnica
Curso de Ciência da Computação – Campus Itajaí
Sistemas Operacionais
Processos, Threads, Concorrência e Paralelismo

Relatório da Avaliação 01

Acadêmicos: Matheus Baron Lauritzen e Gustavo Baron Lauritzen

16/04/2024

Introdução

Este trabalho apresenta o desenvolvimento e a avaliação de um sistema de monitoramento e controle de esteiras transportadoras para a indústria alimentícia. O projeto foi desenvolvido em duas plataformas: Windows e Linux, utilizando duas formas de comunicação interprocessos (IPC): pipe nomeado e multithreading. O objetivo é comparar o desempenho das diferentes plataformas e métodos de IPC em termos de tempo de execução.

Contexto

Uma Indústria de Alimentos de Santa Catarina chamada FoodSec S.A. possui a tarefa de escanear alimentos por meio de câmeras e verificar se os mesmos estão corretos. Os alimentos podem passar por uma das duas esteiras disponíveis. As duas esteiras são controladas por um por um único computador centralizado. Esse computador recebe dados de um sensor em cada uma das esteiras que captura a contagem de itens que são identificados como seguros. A contagem é exibida em um display perto das esteiras (todos os displays são controlados pela mesma função, é apenas uma replicação).

Descrição da Implementação

Implementação com Multithread:

Bibliotecas:

- stdio.h: Fornece funções de entrada e saída padrão, como printf.
- stdlib.h: Oferece funções utilitárias gerais, como pthread_cancel.
- time.h: Define funções para trabalhar com hora e data, usada pelo clock_gettime.
- pthread.h: Declara funções para programação multithread com POSIX Threads.
- unistd.h: Oferece funções relacionadas ao sistema operacional, como usleep.

Funções:

- esteira1: Simula uma esteira que adiciona 1 item de peso 5 a cada 2 segundos.
    - Utiliza pthread_mutex_lock para obter acesso exclusivo às variáveis compartilhadas.
    - Incrementa cont_itens e soma 5 à cont_peso.
    - Desbloqueia o mutex com pthread_mutex_unlock.
    - Utiliza usleep para esperar 2 segundos (2.000.000 microsegundos).
- esteira2: Simula outra esteira que adiciona 1 item de peso 2 a cada 1 segundo.
    - Similar a esteira1, mas incrementa cont_peso com 2.
    - Utiliza usleep para esperar 1 segundo (1.000.000 microsegundos).
- display: Atualiza a tela a cada 2 segundos mostrando a quantidade total de itens e o peso total.

- Limpa a tela com system("clear").
- Bloqueia o mutex para garantir acesso seguro às variáveis compartilhadas.
- Exibe os valores de cont_itens e cont_peso.
- Verifica se a quantidade de itens processados aumentou em pelo menos 500 desde a última atualização (atualizacao).
  - Se sim, atualiza o peso total (aux) somando o peso dos novos itens.
  - Incrementa atualizacao para acompanhar o progresso.
- Desbloqueia o mutex.
- Utiliza usleep para esperar 2 segundos.

Função Principal (main):

- Declara variáveis para armazenar o tempo de início e término da execução.
- Cria três threads: thread_esteira1, thread_esteira2 e thread_display.
- Inicia a medição de tempo com clock_gettime.
- Aguarda o usuário digitar 'q' ou 'Q' para sair.
- Cancela as threads criadas com pthread_cancel.
- Espera todas as threads terminarem com pthread_join.
- Finaliza a medição de tempo e calcula o tempo total de execução.
- Exibe o tempo total de execução e finaliza o programa.

Implementação com Pipe nomeado:

Foram criados 3 arquivos: Display, Esteira1 e Esteira2.

Código do Display:

- **Bibliotecas:**
  - stdio.h: funções básicas de input e output.
  - stdlib.h: funções gerais utilitárias.
  - sys/types.h, sys/socket.h, sys/un.h: funções de rede para sockets Unix domain.
  - unistd.h: funções do sistema operacional POSIX.
  - fcntl.h: funções de controle de descritores de arquivo.
  - time.h: funções de data e hora.
- **Função atualiza_display:**
  - Limpa a tela e posiciona o cursor no topo.
  - Exibe o título "MONITORAMENTO DAS ESTEIRAS".
  - Mostra a quantidade total e o peso total de itens.
- **Processo principal:**
  - Cria sockets local para comunicação com as esteiras.
  - Faz o bind dos sockets locais com os caminhos especificados.
  - Escuta conexões nas portas dos sockets locais.
  - Aguarda a conexão das esteiras 1 e 2.
  - Configura os sockets para leitura não bloqueante.

- Entra em um loop infinito para receber e processar dados:
  - Lê dados dos sockets das esteiras (quantidade e peso) até que não haja mais dados para serem lidos.
  - Calcula a quantidade e peso total.
  - Atualiza a quantidade no display a cada 2 segundos e o peso a cada 500 unidades da variável quantidade.
  - Verifica se o usuário pressionou 'q' para sair.
- Encerra os sockets e exibe o tempo total de execução.

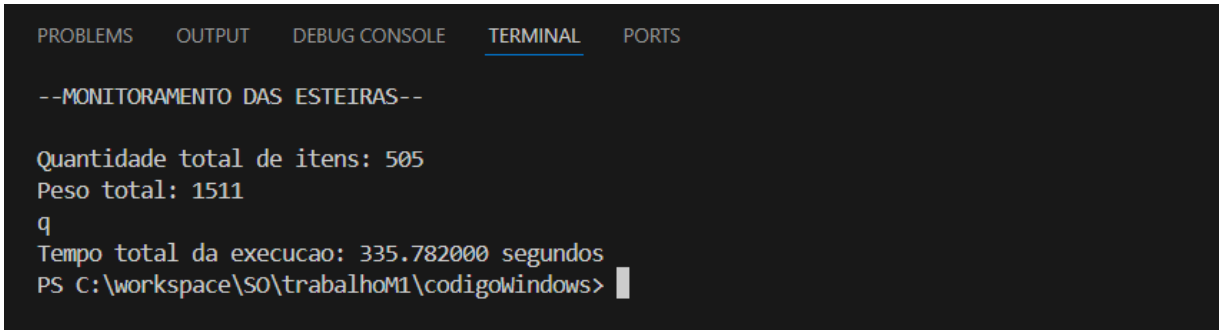
Código Esteira1 e Esteira2:

- **Bibliotecas:** mesmas bibliotecas utilizadas no código do display.
- **Processo principal:**
  1. Cria um socket para comunicação com o display.
  2. Conecta o socket ao caminho do socket do display.
  3. Configura o socket para leitura não bloqueante.
  4. Entra em um loop infinito para gerar e enviar dados:
     - Incrementa as variáveis de quantidade e peso.
     - Exibe a quantidade e peso gerados.
     - Envia a quantidade e peso para o display através do socket.
     - Verifica se o usuário pressionou 'q' para sair.
  5. Encerra o socket e exibe o tempo total de execução.

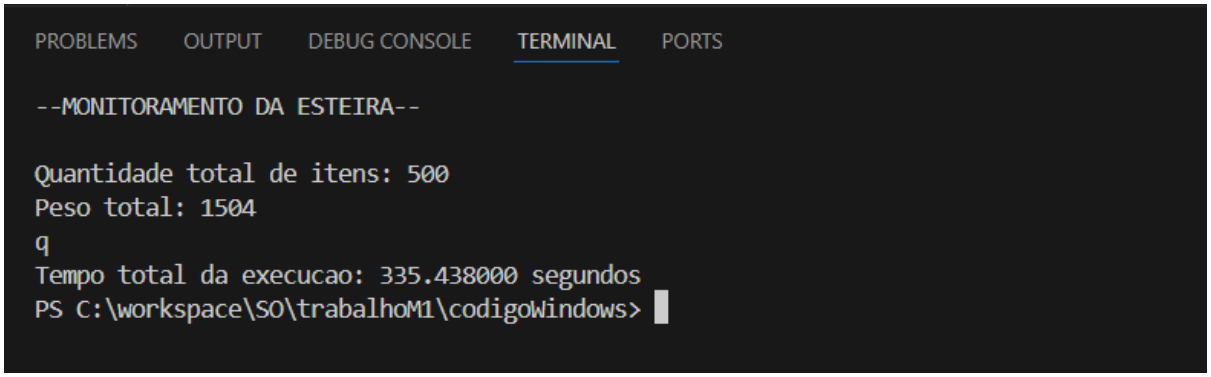
Resultados Obtidos

Sistema Operacional	Método IPC	Tempo (segundos)
Windows	Pipe nomeado	335
Windows	Multithreading	335
Linux	Pipe nomeado	342
Linux	Multithreading	333

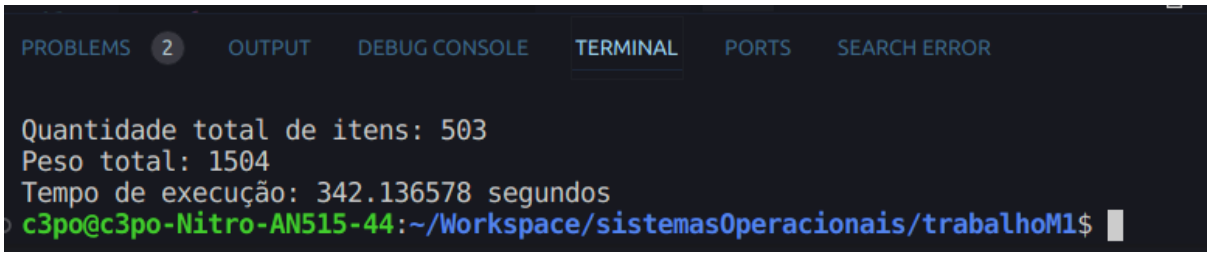
Pipe Nomeado (Windows):



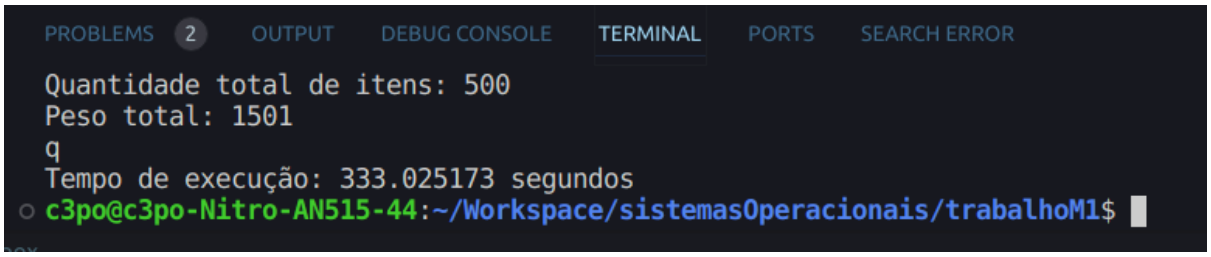
Multithreading (Windows):



Pipe Nomeado (Linux):



Multithreading (Linux):



Código fonte:

Segue o link do Github para o repositório com o código fonte:

- https://github.com/matheus-lauri/trabalhoM1SO;

Conclusão

Ao analisar os resultados das implementações de monitoramento e controle de esteiras transportadoras para a indústria alimentícia, fica evidente que a escolha do método de comunicação e plataforma de desenvolvimento desempenha um papel crucial no desempenho do sistema.

Os dados mostram que, as implementações realizadas no ambiente Windows, observou-se que tanto o método de Multithreading quanto o uso de Pipe nomeado apresentaram desempenho praticamente idêntico. Essa similaridade nos tempos de execução sugere que, as duas abordagens foram igualmente eficazes na gestão das tarefas de monitoramento e controle das esteiras transportadoras.

Já no ambiente do Linux, ficou evidente que o Multithreading apresentou um desempenho ligeiramente melhor em relação ao uso de Pipe nomeado. Esta tendência sugere que, sob as condições específicas de teste e ambiente de execução no Linux, a execução paralela de threads proporcionou uma vantagem na eficiência do sistema em comparação com a comunicação via Pipe nomeado.

Portanto, a variação nos tempos de execução entre as plataformas Windows e Linux ressalta a importância de adaptar as soluções de software às características específicas do ambiente de execução.