**Avaliação M1 - IPC, Threads e Paralelismo**

**Mateus José da Silva, Mateus Barbosa, Matheus de Oliveira Rocha**

Universidade do Vale do Itajaí - UNIVALI

Escola Politécnica

Ciência da Computação

{silvamateus, mateus.barbosa, matheus.rocha}@edu.univali.br,

**1. Contexto da aplicação para compreensão da solução proposta**

A aplicação desenvolvida visa simular um sistema de contagem e exibição de itens em três esteiras em uma indústria de alimentos. O sistema é composto por dois principais componentes: o processo de contagem, responsável por simular a contagem dos itens em cada esteira e atualizar o peso total dos itens processados; e o processo de exibição, responsável por receber os dados de contagem do processo de contagem e exibir as informações no display.

O contexto do problema trata-se de uma indústria de alimentos que precisa monitorar o fluxo de produtos em suas esteiras para garantir a segurança dos alimentos. Cada esteira processa produtos de diferentes pesos, e é necessário realizar a contagem dos itens e atualizar o peso total dos itens processados a cada 1500 unidades. Além disso, é importante exibir periodicamente as informações de contagem no display para que os operadores possam acompanhar o progresso.

**2. Resultados obtidos com as simulações**

Durante a simulação da aplicação, foi possível observar que o processo de contagem funcionou conforme o esperado, com cada esteira contando os itens e atualizando o peso total de forma correta. O uso de threads garantiu que a contagem pudesse ser realizada de forma concorrente e eficiente, enquanto o uso de mutex assegurou a consistência dos dados compartilhados entre as threads.

O processo de exibição também funcionou conforme o esperado, recebendo os dados de contagem do processo de contagem através do pipe e exibindo as informações no display a cada 2 segundos. Isso permitiu que os operadores acompanhassem o progresso da contagem de forma clara e em tempo real.

E também a funcionalidade de atualizar a contagem dos pesos ao atingir um determinado ponto está ocorrendo como esperado. E caso o operador deseje parar a execução das esteiras usando CTRL+C, ela irá parar todas as threads e eliminar os processos necessários, para evitar consumo desnecessário de poder computacional.

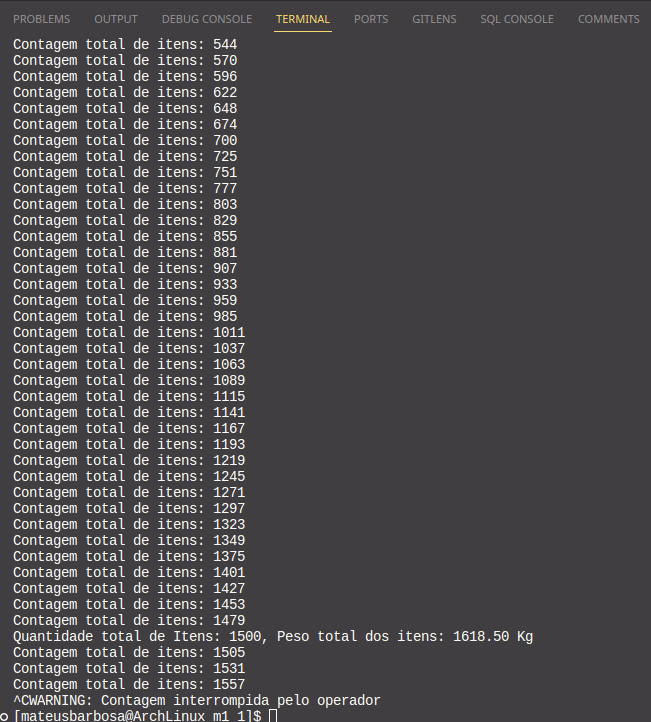


Image 01: Execução da aplicação e interrupção pelo Operador

**3. Códigos Importantes da Implementação**

Os códigos mais importantes da implementação são:

- Função **sensorThread**: Responsável por simular o funcionamento de cada esteira, contando os itens e atualizando o peso.

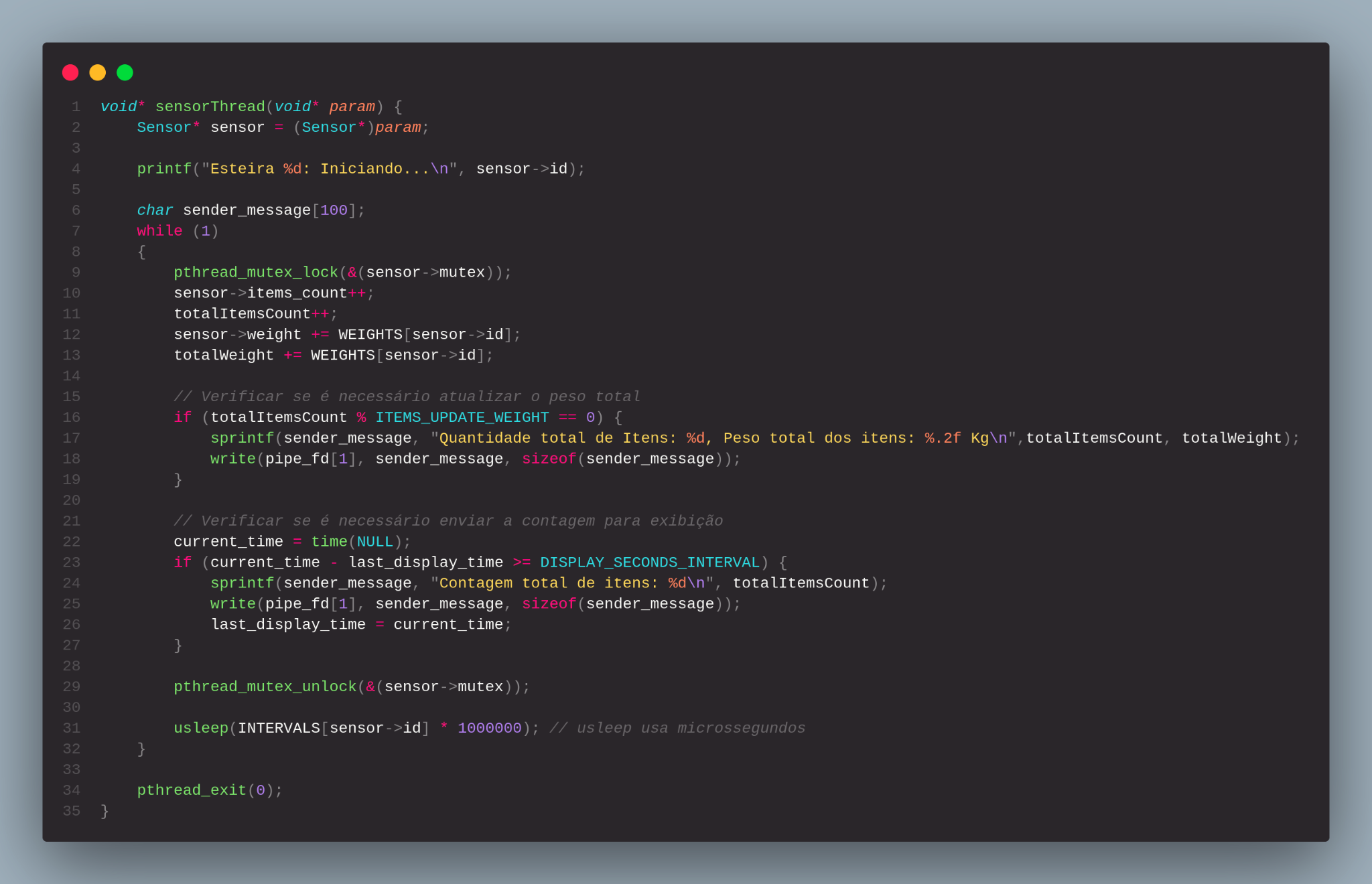


Imagem 02: Código relacionado às esteiras

- Função **displayProcess**: Processo filho responsável por ler os dados do pipe e exibir as informações no display.



Imagem 03: Código referente ao processo de Display das mensagem ao terminal

- Manipulador de sinal **signalHandler**: Para lidar com o sinal SIGINT e interromper a contagem quando necessário e “matar” as threads e processos.



Imagem 04: Código referente à interrupção e eliminação de processos e threads

**4. Resultados obtidos com a implementação**

O uso do pipe para comunicação entre processos permite que o processo de contagem envie os dados de contagem para o processo de exibição de forma assíncrona. Isso significa que o processo de contagem não precisa esperar pela confirmação ou resposta do processo de exibição para continuar sua execução. Isso reduz o tempo de espera e aumenta a eficiência da aplicação como um todo.

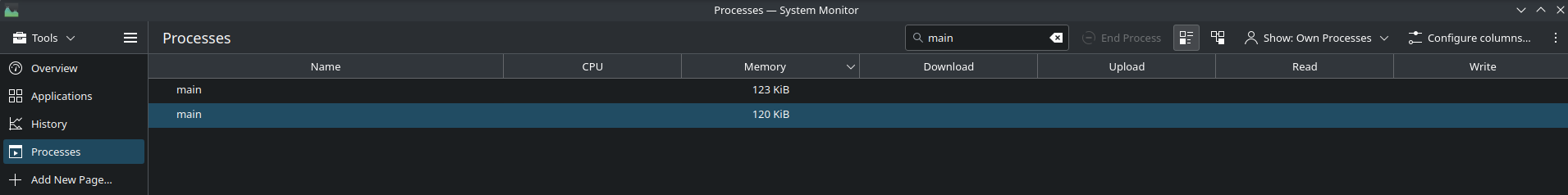


Imagem 05: Visualização de processos criados a partir do Pipe

A utilização de threads permite que a contagem dos itens em cada esteira seja realizada de forma concorrente e eficiente. Cada esteira é representada por uma thread separada, o que permite que a contagem seja realizada de forma independente, sem a necessidade de esperar pelo término de uma esteira para começar a contagem na próxima. Isso otimiza o uso dos recursos do sistema multicore e acelera o processo de contagem como um todo.

A frequência com que o display é atualizado está diretamente relacionada ao intervalo de tempo entre cada iteração do processo de exibição. Neste caso, a esteira 2 tem o menor intervalo de tempo entre cada item processado (0.1 segundos), o que significa que ela está processando itens a uma taxa mais rápida em comparação com as outras esteiras.

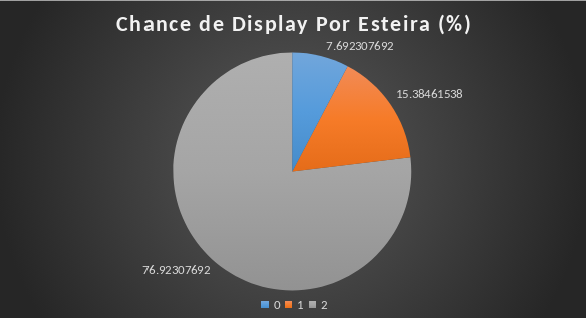


Imagem 06: Gráfico mostrando a porcentagem de display de cada esteira em 100 segundos

Como o display é atualizado a cada 2 segundos, a esteira que processa os itens mais rapidamente terá mais oportunidades de atualizar o display durante esse intervalo de tempo. A esteira 2, com um intervalo de 0.1 segundos entre cada item, terá 20 vezes mais oportunidades de atualizar o display em comparação com a esteira 1 (intervalo de 1 segundo) e 200 vezes mais oportunidades em comparação com a esteira 3 (intervalo de 0.5 segundos).

Assim, a esteira 2 é a que tem o display mais frequente porque processa itens a uma taxa mais rápida, proporcionando mais atualizações de contagem durante o intervalo de tempo determinado para a exibição. Isso garante que os operadores possam acompanhar o progresso da esteira 2 com mais detalhes em comparação com as outras esteiras.

Verificando a quantidade de pesos que cada esteira passa, pode-se perceber que a esteira 0 (1 segundo) e a esteira 2 (0.1 segundos) possuem a mesma quantidade de pesos. Mas em contrapartida, a esteira 2 precisa de 1000 itens, enquanto a esteira 0 precisa apenas de 100 itens, mostrando que a eficiência das 2 será a mesma.



Imagem 07: Gráfico mostrando a quantidade de pesos (Kg) que cada esteira passa em 100 segundos

Em suma, a implementação da solução utilizando pipe e threads demonstrou ser eficaz para lidar com a contagem e exibição de itens em três esteiras em uma indústria de alimentos. A utilização de threads permitiu uma contagem concorrente e eficiente, enquanto o uso do pipe facilitou a comunicação assíncrona entre os processos de contagem e exibição. Essa abordagem oferece uma base sólida para futuras iterações e melhorias na aplicação.

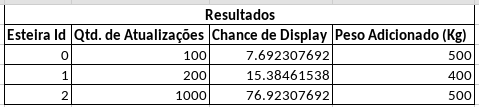


Imagem 08: Resumo dos resultados obtidos

**4. Análise e discussão sobre os resultados finais**

Ao longo deste trabalho, exploramos a implementação de um sistema de simulação de esteiras utilizando a linguagem de programação C. Durante esse processo, aprendemos diversos conceitos e técnicas essenciais para o desenvolvimento de sistemas concorrentes e comunicação entre processos.

Primeiramente, compreendemos a importância das threads na execução de tarefas paralelas e simultâneas, permitindo que múltiplas esteiras fossem simuladas de forma independente. O uso de threads nos proporcionou uma visão prática sobre como lidar com concorrência e compartilhamento de recursos em um ambiente de programação.

Além disso, exploramos técnicas para garantir a integridade dos dados compartilhados entre as threads, como a utilização de variáveis globais e a implementação de mecanismos de exclusão mútua para evitar condições de corrida.

O tratamento de sinais, particularmente o SIGINT, nos permitiu implementar uma funcionalidade de interrupção controlada do programa, proporcionando uma saída limpa e segura em caso de solicitação do usuário.

A comunicação entre processos através de pipes nos mostrou como é possível estabelecer uma interação eficiente entre diferentes partes de um programa, permitindo que informações fossem transmitidas de forma confiável entre o processo pai e o processo filho responsável pela exibição dos resultados.

Por fim, a utilização da biblioteca time.h nos proporcionou meios para lidar com operações relacionadas ao tempo, como a exibição de informações em intervalos regulares, agregando uma dimensão adicional de controle e precisão ao sistema.

Em resumo, este trabalho nos proporcionou uma valiosa experiência prática no desenvolvimento de sistemas concorrentes e comunicação entre processos, reforçando nosso entendimento sobre conceitos fundamentais de programação e nos preparando para enfrentar desafios mais complexos no futuro.

**Referências**

CPPREFERENCE. **C Programming Language**. Disponível em: https://devdocs.io/c/. Acesso em: 31 mar. 2024.

TUTORIALSPOINT. **C Library -**. Disponível em: https://www.tutorialspoint.com/c\_standard\_library/stdio\_h.htm. Acesso em: 31 mar. 2024.

CPPREFERENCE. **C Stdlib**. Disponível em: https://devdocs.io/c/program. Acesso em: 31 mar. 2024.

OPENGROUP. **C Unistd**. Disponível em: https://pubs.opengroup.org/onlinepubs/7908799/xsh/unistd.h.html. Acesso em: 31 mar. 2024.

OPENGROUP. **C Pthreads**. Disponível em: https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html. Acesso em: 31 mar. 2024.

DOCUMENTATION.HELP. **C Signal**. Disponível em: https://documentation.help/C-Cpp-Reference/signal.html. Acesso em: 31 mar. 2024.

OPENGROUP. **C Sys/Types**. Disponível em: https://pubs.opengroup.org/onlinepubs/009604499/basedefs/sys/types.h.html. Acesso em: 31 mar. 2024.

IBM. **C Sys/Wait**. Disponível em: https://www.ibm.com/docs/en/zos/3.1.0?topic=files-syswaith-hold-processes. Acesso em: 31 mar. 2024.

IBM. **C Time**. Disponível em: https://www.ibm.com/docs/en/zos/3.1.0?topic=files-timeh-time-date. Acesso em: 31 mar. 2024.

PANDEY, Durgesh. **Inter Process Communication (IPC)**. Disponível em: https://www.geeksforgeeks.org/inter-process-communication-ipc/. Acesso em: 31 mar. 2024.

TLDP. **Creating Pipes in C**. Disponível em: https://tldp.org/LDP/lpg/node11.html. Acesso em: 31 mar. 2024.