

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. E pode-se pausar a execução com P ou p, e retomá-la do estado em que estava com R ou r.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SQL CONSOLE COMMENTS
Contagem total de itens: 544
Contagem total de itens: 570
Contagem total de itens: 596
Contagem total de itens: 622
Contagem total de itens: 648
Contagem total de itens: 674
Contagem total de itens: 700
Contagem total de itens: 725
Contagem total de itens: 751
Contagem total de itens: 777
Contagem total de itens: 803
Contagem total de itens: 829
Contagem total de itens: 855
Contagem total de itens: 881
Contagem total de itens: 907
Contagem total de itens: 933
Contagem total de itens: 959
Contagem total de itens: 985
Contagem total de itens: 1011
Contagem total de itens: 1037
Contagem total de itens: 1063
Contagem total de itens: 1089
Contagem total de itens: 1115
Contagem total de itens: 1141
Contagem total de itens: 1167
Contagem total de itens: 1193
Contagem total de itens: 1219
Contagem total de itens: 1245
Contagem total de itens: 1271
Contagem total de itens: 1297
Contagem total de itens: 1323
Contagem total de itens: 1349
Contagem total de itens: 1375
Contagem total de itens: 1401
Contagem total de itens: 1427
Contagem total de itens: 1453
Contagem total de itens: 1479
Quantidade total de Itens: 1500, Peso total dos itens: 1618.50 Kg
Contagem total de itens: 1505
Contagem total de itens: 1531
Contagem total de itens: 1557
^CWARNING: Contagem interrompida pelo operador
o [mateusbarbosa@ArchLinux m1 1]$
```

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.

```

1 void* sensorThread(void* param) {
2     Sensor* sensor = (Sensor*)param;
3     char sender_message[100];
4
5     sprintf(sender_message, "Esteira %d: Iniciando...\n", sensor->id);
6     write(pipe_fd[1], sender_message, sizeof(sender_message));
7
8     while (1)
9     {
10
11         if (!sensor->is_running)
12             omp_set_lock(&sensor->lock);
13
14         sensor->items_count++;
15         sensor->weight += WEIGHTS[sensor->id];
16
17         #pragma omp critical
18         {
19             totalItemsCount++;
20             totalWeight += WEIGHTS[sensor->id];
21
22             // Verificar se é necessário atualizar o peso total
23             if (totalItemsCount % ITEMS_UPDATE_WEIGHT == 0) {
24                 sprintf(sender_message, "Quantidade total de Itens: %d, Peso total dos itens: %.2f Kg\n", totalItemsCount, totalWeight);
25                 write(pipe_fd[1], sender_message, sizeof(sender_message));
26             }
27
28             // Verificar se é necessário enviar a contagem para exibição
29             current_time = time(NULL);
30             if (current_time - last_display_time >= DISPLAY_SECONDS_INTERVAL) {
31                 sprintf(sender_message, "Contagem total de itens: %d\n", totalItemsCount); // Exibir a contagem total de itens
32                 write(pipe_fd[1], sender_message, sizeof(sender_message));
33
34                 last_display_time = current_time;
35             }
36         }
37
38         usleep(INTERVALS[sensor->id] * 1000000); // usleep usa microssegundos
39     }
40 }

```

Imagem 02: Código relacionado às esteiras

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

```

1 void displayProcess() {
2     char reciever_message[100];
3     while (1)
4     {
5         read(pipe_fd[0], reciever_message, sizeof(reciever_message));
6         printf("%s", reciever_message);
7     }
8 }

```

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

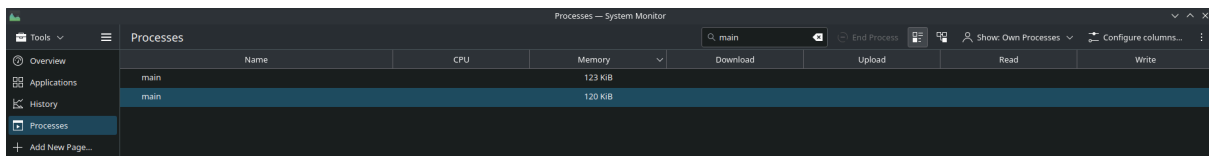
- Manipulador de sinal **keyboardInput**: Para lidar com o input do teclado, possibilitando interromper a contagem (tecla P ou p) e retomá-lá (tecla R ou r), quando necessário. Essa função bloqueia as threads (lock) da contagem e libera elas (unlock).

```
1 void* keyboardInput(void* param) {
2     int input_char = 0;
3
4     while(1){
5         input_char = getchar();
6         if (input_char == 80 || input_char == 112)
7         {
8             char sender_message[100];
9             sprintf(sender_message, "WARNING: Contagem interrompida pelo operador, Contagem total de itens: %d\n", totalItemsCount);
10            write(pipe_fd[1], sender_message, sizeof(sender_message));
11            for (int i = 0; i < NUM_THREADS; i++) {
12                sensors[i].is_running = 0;
13            }
14        }
15        if (input_char == 82 || input_char == 114)
16        {
17            char sender_message[100] = "WARNING: Contagem retomada pelo operador\n";
18            write(pipe_fd[1], sender_message, sizeof(sender_message));
19            for (int i = 0; i < NUM_THREADS; i++) {
20                sensors[i].is_running = 1;
21                omp_unset_lock(&sensors[i].lock);
22            }
23        }
24    }
25 }
```

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.



Name	CPU	Memory	Download	Upload	Read	Write
main		123 KIB				
main		120 KIB				

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.

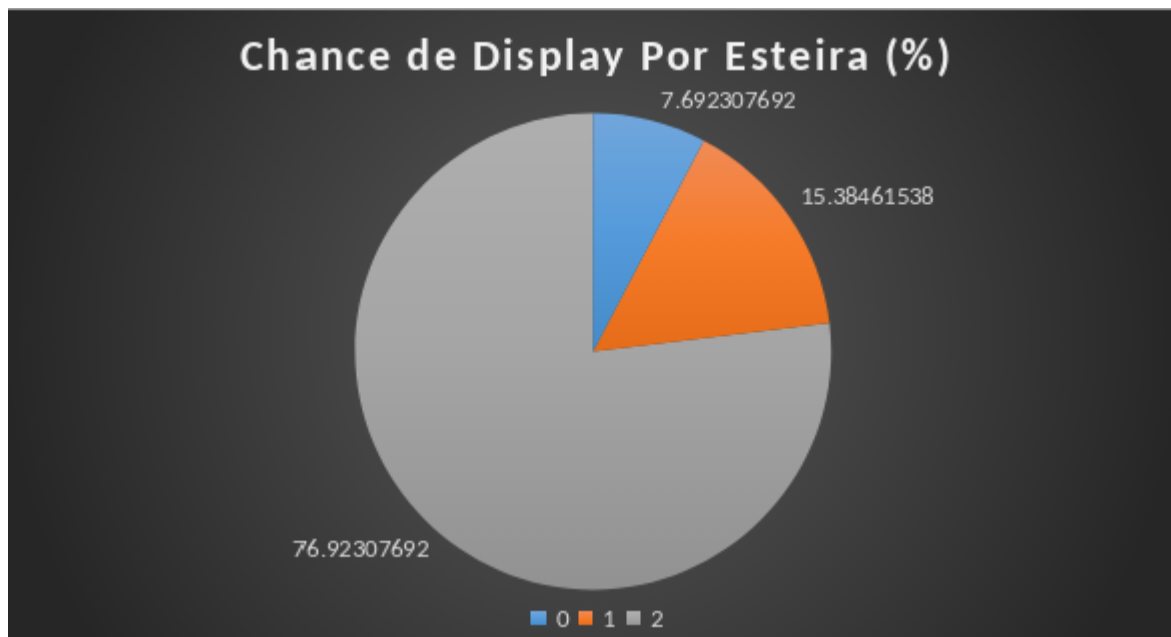


Imagem 06: Gráfico mostrando a porcentagem de display de cada esteira em 100 segundos

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.

Resultados			
Esteira Id	Qtd. de Atualizações	Chance de Display	Peso Adicionado (Kg)
0	100	7.692307692	500
1	200	15.38461538	400
2	1000	76.92307692	500

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 input do teclado, nos permitiu implementar uma funcionalidade de interrupção controlada do programa, proporcionando uma chance de parar as threads selecionadas e continuá-las, sem afetar o resto do programa (processo de display e input do teclado).

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.

TUTORIALSPPOINT. **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.