**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**

Maria Amélia Doná Aguilar

Jane Harumi Yamamoto

# SISTEMAS OPERACIONAIS

ALGORITMO DE FILA DE ATENDIMENTO - RESTAURANTE

POÇOS DE CALDAS

2024

Maria Amélia Doná Aguilar

Jane Harumi Yamamoto

# SISTEMAS OPERACIONAIS

ALGORITMO DE FILA DE ATENDIMENTO - RESTAURANTE

Relatório apresentado a disciplina obrigatória de Sistemas Operacionais, apresentação de um algoritmo que simula uma fila de atendimento de um restaurante, utilizando pipes e threads.

Professor: João Carlos de Moraes Morselli Junior

POÇOS DE CALDAS

2024

**SUMÁRIO**

## Introdução

A comunicação entre processos (ICP) é um agrupamento de processos que permite a transferência de informações entre si. Para que a execução ocorra, pressupõe por parte do sistema operativo, entre outras coisas, a criação de um contexto de execução próprio, abstrai o processo dos componentes reais do sistema. Esse mecanismo permite que os processos diferentes troquem mensagem e colaborem entre si, sendo que podem ser executados no mesmo computador ou em diferentes computadores conectados em rede.

Desempenhando um papel crucial em sistemas de multitarefa o IPC permite que os aplicativos compartilhem informações, sincronizem suas operações e trabalhem em conjunto para realizar tarefas complexas. Os mecanismos podem ser classificados em “Locais”, que são comunicação entre processos no mesmo computador e “Distribuídos”, que são comunicação entre processos em computadores diferentes.

Os pipes são um mecanismo de IPC usado para transferir dados sequencialmente entre processos. Existem dois tipos principais: pipes anônimos e pipes nomeados. Pipes anônimos são usados para comunicação entre processos relacionados, como pai e filhos, sendo úteis para redirecionar entrada e saída padrão. Eles funcionam apenas no mesmo computador e requerem dois pipes para comunicação bidirecional.

Os pipes nomeados permitem comunicação entre processos não relacionados, sejam eles localmente ou em rede, suportando comunicação bidirecional em uma única conexão. Sendo o ideal para cenários cliente-servidor, onde vários clientes conectam ao servidor central. Sendo os pipes eficientes, simples de usar e oferecem suporte para transferência contínua, uteis para comunicação síncrona ou/e assíncrona.

O multithreading é uma técnica que permite a execução simultânea de várias tarefas em um aplicativo, aumentando sua capacidade de resposta e, em sistemas com múltiplos processadores ou núcleos, sua eficiência e taxa de processamento. Um processo é um programa em execução que o sistema operacional gerencia para isolar aplicativos. Já um thread é a menor unidade de execução atribuída a um processo, contendo registros de CPU e uma pilha que permitem continuar sua execução sem interrupções. Vários threads podem existir dentro de um único processo, compartilhando seu espaço de endereço virtual e podendo executar partes do mesmo código.

Threads são particularmente úteis para melhorar a responsividade do aplicativo e otimizar o uso de múltiplos núcleos. Por exemplo, operações demoradas podem ser delegadas a threads de trabalho, permitindo que o thread principal continue gerenciando a interface do usuário ou respondendo a eventos. Com uma implementação adequada, o multithreading pode tornar os aplicativos mais rápidos, responsivos e eficientes.

## Objetivo

O objetivo do código é apresentar uma simulação de um processamento de pedidos entre dois processos, um que representa o atendente e uma cozinha. O atendente recebe os pedidos do usuário e os envia para cozinha, que processa e retorna o pedido pronto. Assim o sistema demonstra o uso de pipes para comunicação entre os processos e threads para monitorar os pedidos processados de forma concorrente. Assim, sendo possível compreender como os processos podem se comunicar e como os threads podem ser usadas para gerenciar tarefas paralelas.

Usando pipes para permitir a comunicação entre processos de forma simples e eficaz, os dois pipes (ou FIFOS) são criados, assim um envia pedidos do atendente para cozinha e o outro envia os pedidos prontos de volta para o atendente. Sendo essenciais para permitir troca de informações entre processos independentes. Enquanto a thread é usada para criar um monitor separado que acompanha os pedidos processados pela cozinha. Permitindo que a monitoração dos pedidos aconteça de forma assíncrona em relação ao processo do atendente. Melhorando a eficácia e permitindo a execução simultânea de tarefas.

## Descrição do Algoritmo

1. **Configuração de Sinais e FIFOs**:

```c
// Função que lida com sinais de interrupção (ex.: Ctrl+C)
void signal_handler(int signum) {
    printf("\nRecebido sinal de interrupção. Limpando recursos...\n");
    limpar_fifos(); // Garante que os recursos serão limpos antes de sair
    exit(1);
}
```

- O programa define tratadores de sinal (signal_handler) para garantir que os recursos sejam liberados caso o programa seja interrompido (por exemplo, com Ctrl+C).
- Os FIFOs são criados com mkfifo() para possibilitar a comunicação entre processos.

2. **Criação do Processo Filho (Cozinha)**:

```
// Criar processo filho (Cozinha)
pid_t pid = fork();
if (pid < 0) {
    perror("Erro ao criar processo filho");
    limpar_fifos();
    return EXIT_FAILURE;
}

if (pid == 0) {
    // Executa o processo filho (Cozinha)
    cozinha();
    return 0;
}
```

   - O processo pai cria um processo filho usando fork(). O processo filho executa a função cozinha(), que é responsável por ler pedidos do FIFO FIFO_ATENDENTE, processá-los (simulando o tempo de preparo) e enviar a resposta para o FIFO FIFO_COZINHA.

3. **Função cozinha()**:
   - O processo filho abre os FIFOs e aguarda pedidos do atendente. Quando um pedido é recebido, ele é processado (simula o preparo com sleep()) e, em seguida, enviado de volta ao atendente. Se receber um pedido especial com a descrição "sair", a função encerra sua execução.

4. **Função de Monitoramento com Thread**:
   - Uma thread separada é criada para monitorar a chegada de pedidos prontos da cozinha. A função monitorar_pedidos() lê do FIFO FIFO_COZINHA e exibe o tempo de preparo de cada pedido.

5. **Função Principal (main())**:

- O processo pai configura e abre os FIFOs.
- Uma thread é criada para monitorar os pedidos prontos.
- O usuário pode digitar pedidos no terminal. Quando um pedido é inserido, ele é enviado para o FIFO FIFO_ATENDENTE e o processo pai atualiza as estatísticas.
- Quando o usuário digita "sair", um pedido especial é enviado para a cozinha, sinalizando o encerramento do programa.

6. **Limpeza e Estatísticas**:
   - A função limpar_fifos() é chamada para fechar e remover os FIFOs, garantindo que os recursos sejam liberados corretamente.
   - O programa exibe estatísticas finais (total de pedidos recebidos e pedidos processados).

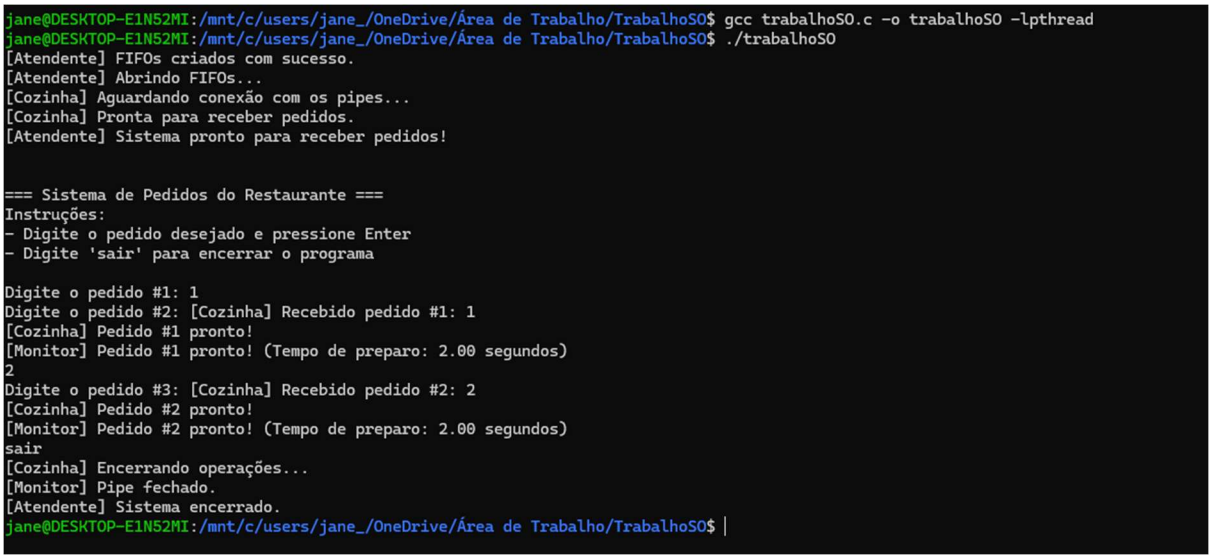
**Resumo do Fluxo de Execução**

1. O processo principal (main()) cria os FIFOs e o processo filho (cozinha()).
2. A thread de monitoramento é iniciada para observar pedidos prontos.
3. O usuário interage com o atendente, inserindo pedidos que são enviados para a cozinha.
4. A cozinha processa os pedidos, envia-os de volta e a thread de monitoramento exibe os tempos de preparo.
5. O processo pai exibe estatísticas finais após a saída do loop.

**Pontos Importantes**

- **Concorrência**: O uso de mutexes garante que os acessos aos FIFOs e às variáveis estatísticas sejam realizados de maneira segura, evitando condições de corrida.
- **Comunicação Interprocessos (IPC)**: Os pipes permitem a comunicação de maneira eficiente entre processos separados.
- **Uso de Threads**: A thread é usada para que a monitoração dos pedidos seja feita de forma independente e em paralelo ao processo de inserção de pedidos.

## Fluxo de execução




O código é compilado para ser executado no Linux, garantindo a utilização da biblioteca pthread seja usada para gerenciar threads. Então o código é executado.

O sistema inicializa, havendo as mensagens afirmando a inicialização correta do programa, além do aviso para digitar “sair” para encerrar o programa. O usuário pode então solicitar o pedido, que é recebido pela cozinha via FIFO_ATENDENTE. Enquanto a cozinha processa o pedido simula o preparo com sleep (2). Assim a thread que monitora o FIFO_COZINHA e exibe o tempo de preparo.

Enquanto o pedido dois ocorre em paralelo ao “preparo” da cozinha, e o processo ocorre idêntico ao primeiro preparo. Para encerrar o programa é dito para o programa sair e o programa finaliza e limpa os FIFOS.

## Conclusão

O programa é uma representação do uso de comunicação entre processo e multithreading para demonstrar sincronização e troca de informações em um sistema multitarefa. Utilizando pipes nomeados e os processos atendente e cozinha para demonstrar a troca de informações mesmo em processos distintos.

Com os FIFOS permitindo a comunicação entre o atendente e a cozinha, sendo possível visualizar os pipes, que permitem comunicação entre processos. A thread de monitoramento permite a execução de tarefas simultâneas, operando de forma independente, assim beneficiando do paralelismo.

Em resumo, o programa simula a execução concorrente de um cenário de processamento de pedidos. Mostrando os conceitos de IPC e multithreading. Essa abordagem mostra como o uso combinado dessas técnicas pode criar sistemas eficientes, organizados e escaláveis, reforçando a importância desses conceitos no desenvolvimento de software para multitarefa e sistemas distribuídos.

## Referências


COMUNICAÇÕES entre processos. [*S. l.*], 15 fev. 2024. Disponível em: https://learn.microsoft.com/pt-br/windows/win32/ipc/interprocess-communications. Acesso em: 6 dez. 2024.

THREADS e threading. [*S. l.*], 10 maio 2023. Disponível em: https://learn.microsoft.com/pt-br/dotnet/standard/threading/threads-and-threading. Acesso em: 6 dez. 2024.