PONTIFÍCIA UNIVERSIDADE CATÓICA DE GOIÁS – PUC GO
CIÊNCIAS DA COMPUTAÇÃO

JOÃO VITOR PEIXOTO MARTINS
GUILHERMY FERREIRA DA SILVA
AUGUSTO DE PAULA E CAMPOS

SISTEMAS OPERACIONAIS ||

GOIÂNIA, GOIÁS
2025

Introdução e Cenário Escolhido

Este trabalho tem como objetivo desenvolver um sistema multiprocessado utilizando memória compartilhada como meio de comunicação entre processos distintos, aplicando mecanismos de sincronização como semáforos para garantir o acesso concorrente seguro aos dados.

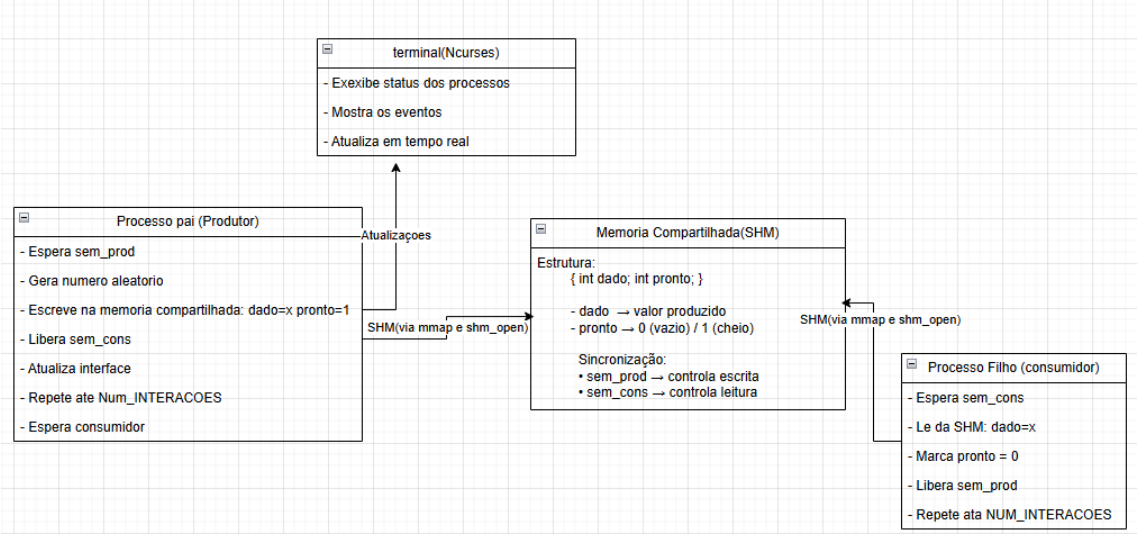
O sistema foi implementado em linguagem C, utilizando chamadas de sistema do POSIX para criação e mapeamento de memória compartilhada (shm_open, mmap) e controle de semáforos (sem_open, sem_wait, sem_post). Além disso, foi integrada uma interface textual interativa desenvolvida com a biblioteca NCurses, permitindo acompanhar em tempo real o funcionamento do sistema.

O cenário escolhido para o projeto foi o modelo Produtor-Consumidor. O Produtor é responsável por gerar valores inteiros aleatórios e armazená-los em uma memória compartilhada. O Consumidor, executando em um processo separado, lê esses valores e os “consome”, liberando o espaço para que o produtor possa gerar novos dados. A comunicação entre os dois processos é controlada por semáforos nomeados, que garantem que apenas um processo acesse o recurso compartilhado de cada vez, evitando condições de corrida, inconsistências e deadlocks.

A interface desenvolvida com NCurses permite acompanhar o comportamento do sistema em tempo real, mostrando o estado da memória (vazia ou cheia), os valores produzidos e consumidos, e os logs de eventos — como bloqueios, liberações e finalização do processo.

Arquitetura da Solução

A aplicação foi desenvolvida seguindo o modelo cliente-servidor, no qual dois processos distintos (Produtor e Consumidor) interagem por meio de uma memória compartilhada, utilizando semáforos nomeados para garantir a sincronização.



Justificativa das Decisões Tomadas

O sistema utiliza memória compartilhada (SHM) por permitir comunicação direta e rápida entre processos, sem precisar de arquivos intermediários. A sincronização foi feita com semáforos POSIX, garantindo que produtor e consumidor acessem a memória de forma alternada e segura.

A estrutura { int dado; int pronto; } foi escolhida por sua simplicidade:

• dado armazena o valor produzido;

• pronto indica se há dado disponível (0 = vazio, 1 = cheio).

A interface NCurses foi usada para exibir o funcionamento do sistema em tempo real, tornando a execução mais visual e didática. Além disso, o código foi estruturado de forma modular, facilitando manutenção e portabilidade.

Mecanismos de Sincronização e Cuidados Adotados

Para evitar erros de concorrência, o produtor sempre espera sem_prod e libera sem_cons, enquanto o consumidor faz o inverso.

Os semáforos foram inicializados com:

• sem_prod = 1 (produtor pode começar),

• sem_cons = 0 (consumidor espera).

Essa ordem evita condições de corrida e deadlocks. No final, todos os recursos (semáforos e memória) são liberados corretamente, garantindo o encerramento limpo do sistema.

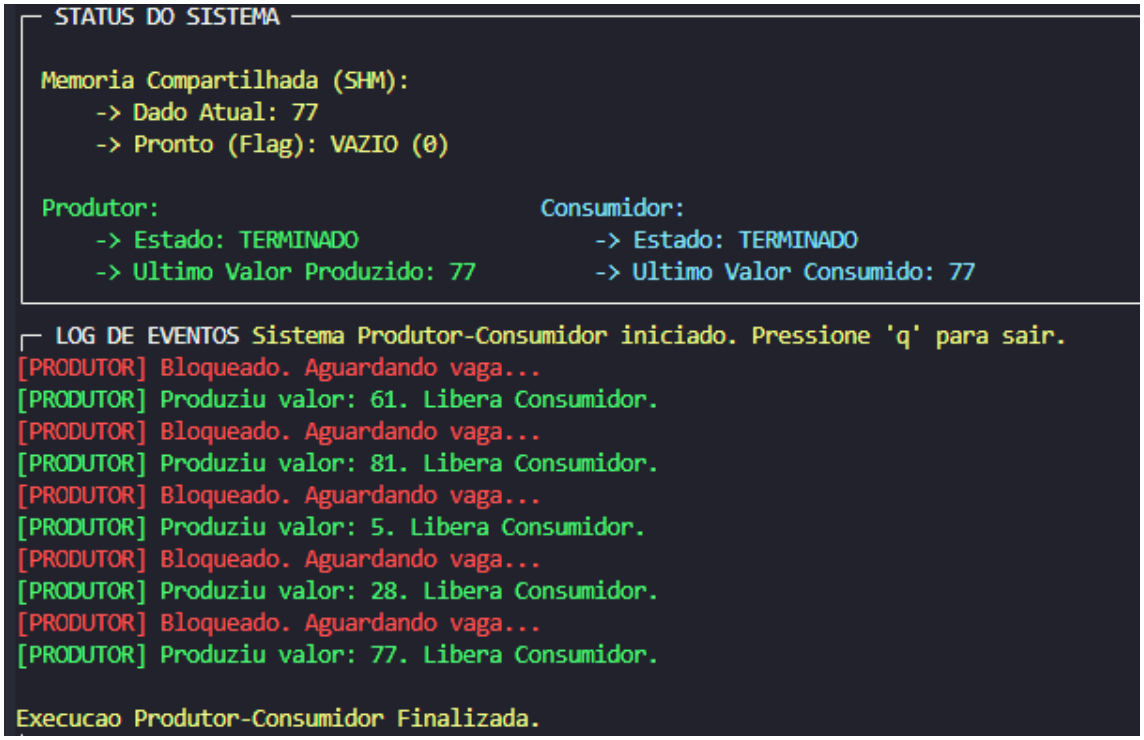
Execução do Sistema e Resultados Obtidos

Durante a execução, o programa inicia o processo pai (Produtor) e cria o processo filho (Consumidor).Ambos passam a se comunicar através da memória compartilhada (SHM), sincronizados pelos semáforos

Na interface NCurses, são exibidas três áreas principais:

•Status da memória → mostra o valor atual (dado) e o estado (pronto = 0 ou 1);

•Atividade dos processos → indica quando o produtor ou consumidor estão bloqueados, produzindo ou consumindo;



O sistema segue o ciclo:

1.O produtor espera o semáforo sem_prod, gera um número aleatório e grava na memória (dado = x, pronto = 1).

2.O consumidor, ao ser liberado por sem_cons, lê o valor, zera o campo pronto e sinaliza o produtor para continuar.

3.O processo se repete até atingir o número de interações definidas.

Esse comportamento demonstra, na prática, o funcionamento correto da sincronização entre processos usando memória compartilhada e semáforos, sem condições de corrida ou travamentos.

Conclusão

O projeto desenvolvido demonstrou de forma prática o funcionamento dos mecanismos de comunicação e sincronização entre processos em sistemas operacionais.

Através do uso de memória compartilhada e semáforos POSIX, foi possível garantir o acesso seguro aos dados, evitando condições de corrida e garantindo a alternância correta entre produtor e consumidor.

A integração com a interface NCurses proporcionou uma visualização em tempo real do comportamento do sistema, tornando a execução mais didática e compreensível.

O trabalho permitiu consolidar conceitos essenciais como processos concorrentes, regiões críticas, sincronização e gerência de recursos, demonstrando a importância desses mecanismos no desenvolvimento de sistemas confiáveis e eficiente.