
**UNIVERSIDADE FEDERAL DO RIO DE JANEIRO**

CENTRO DE TECNOLOGIA

ESCOLA POLITÉCNICA


# Uso de locks e variáveis de condição

Sistemas Operacionais

2024.2


Alunos:
Felipe Vasconcellos (119154566),
Gabriel Machado(122143833) e
Karen Pacheco (123476904)


Rio de Janeiro

# Introdução

Este trabalho tem como objetivo principal implementar uma solução para o problema "Unisex Bathroom Problem", adaptado a partir do Little Book of Semaphores, utilizando locks e variáveis de condição em C, documentando o uso do ChatGPT no processo.

Além de implementar a lógica básica do problema, decidimos acrescentar à solução questões de starvation e fairness para garantir que homens e mulheres tenham acesso justo ao banheiro.

# Especificações e Funcionalidades

1. Controle de acesso ao banheiro:
    - Exclusão mútua entre gêneros.
    - Limite de 3 pessoas no banheiro simultaneamente.
1. Entrada interativa para o número de threads de homens e mulheres.
2. Tempos aleatórios entre ações, simulados com rand() e sleep().
3. Critério de parada onde o programa encerra após todas as threads finalizarem.
4. Prevenção de starvation:
    - Política de fairness com limite de 6 acessos consecutivos por gênero.
    - Alternância entre gêneros ao atingir o limite, com mensagens indicativas.

# Ferramentas utilizadas

- **Versão do ChatGPT**: 4o (plus)
- **Sistema Operacional**: Linux (Ubuntu 20.04).
- **Compilador**: GCC 9.4.0.

# Desenvolvimento

Processo de raciocínio e o registro de prompts durante a resolução do problema foi o seguinte:

**PROMPT 1:**

*“A partir do livro carregado explique como funciona o problema 'Unisex Bathroom' apresentado neste livro. Além disso, qual é o objetivo principal do problema 'Unisex Bathroom' em termos de sincronização?”*

Com este prompt o chatGPT nos ajudou a entender melhor o problema e as regras de sincronização: exclusão mútua, capacidade limitada e prevenção de deadlocks.

**PROMPT 2:**

*“Com isso, gere o código em C baseado na solução do autor proposta neste livro que te enviei para o problema “The unisex bathroom problem”.”*

A resposta do chatGPT forneceu um código inicial em C utilizando semáforos conforme a implementação abordada no livro. Embora o uso de semáforos não fosse o objetivo do trabalho, esse código foi adaptado posteriormente para usar locks e variáveis de condição, servindo como base para a lógica de controle de threads.

**PROMPT 3:**

*“Adapte esse código para utilizar locks e variáveis de condição. Não use semáforos. Utilize a API pthreads para threads, locks e variáveis de condição. Não use outra API de threads.”*

Neste prompt o chat gerou uma versão adaptada do código usando locks e variáveis de condição da API pthreads, sem utilizar semáforos. Essa versão passou a ser o ponto de partida para o desenvolvimento do nosso código final.

**PROMPT 4:**

*“Corrija este código de tal forma que deixe explícito quando uma pessoa sai do banheiro para que a gente possa verificar o funcionamento do algoritmo e se está de acordo com a lógica proposta”*

Foi adicionado ao código uma saída explícita para mostrar quando uma pessoa entra e sai do banheiro. Isso ajuda a verificar se o algoritmo está funcionando corretamente e facilita a nossa análise e validação do comportamento do programa durante os testes. Além disso, permite identificar se os estados das variáveis globais (men_in_bathroom, women_in_bathroom e total_in_bathroom) estavam sendo atualizados corretamente e se os sinais (pthread_cond_broadcast) estavam sendo enviados para liberar as threads apropriadas.

**PROMPT 5:**

*“Implemente tempo aleatório entre ações das threads, usando funções como rand() e sleep()”*

O código anterior não implementava tempos aleatórios entre as ações das threads. Ele usava um tempo fixo de 1 segundo (`sleep(1)`) para simular o tempo que uma pessoa passa no banheiro, então enviamos um prompt para que ele utilizasse funções srand para gerar tempo aleatório. Com isso, foi possível testar o algoritmo em cenários mais próximos do mundo real, analisando como as threads se comportam com tempos de execução variáveis.

**PROMPT 6:**

*“Altere o código de tal forma que o usuário possa escolher um número de threads por meio de uma entrada interativa”*

No código anterior o número de threads estava fixo em 10 threads e isso atrapalha na avaliação de casos e testes de casos diferentes. Com o novo prompt, o código agora permite que o número de threads seja definido interativamente pelo usuário (`num_threads`), e o programa encerra sua execução ao cumprir um critério de parada, que ocorre quando todas as threads finalizam suas tarefas.

**PROMPT 7:**

*“Introduza uma medida para evitar starvation em que o limite de acessos consecutivos de um determinado gênero é 6. O sistema deve exibir uma mensagem no terminal sempre que essa medida de alternância for ativada, indicando que a política de fairness foi aplicada para evitar starvation.”*

O comportamento “starvation” foi identificado em alguns cenários não esperados, como o caso de um número muito grande de threads de homens em comparação com os threads de mulheres. Para resolver esse problema, fizemos uma abordagem onde o contador rastreia o número de acessos consecutivos de um determinado gênero. Quando o contador atinge um limite predefinido, o sistema alterna o gênero mesmo se existir threads do gênero atual na fila garantindo a não ocorrência de starvation para ambos os gêneros.

Mas essa solução não funcionou conforme o esperado. Durante a execução, o código não ativava a política de fairness, nem exibia a mensagem que deveria ser mostrada nesse momento. Além disso, em alguns casos, o programa entrava em um loop infinito.

**PROMPT 8:**

*“Faça mudanças no código para resolver os problemas de alternância e fairness.*
*Adicione duas variáveis de controle: `fairness_policy_active`, que diz qual gênero tem prioridade (1 para homens e 2 para mulheres), e `fairness_enforced`, que controla se a alternância está ativa, bloqueando o outro gênero enquanto a política de fairness estiver em uso.*

*Coloque também um limite de acessos consecutivos de um mesmo gênero, que será 6 (`MAX_CONSECUTIVE`). Quando esse limite for atingido, crie uma função chamada `apply_fairness_policy`. Essa função ativa a política de fairness, garantindo que o outro gênero tenha prioridade até usar o banheiro pelo menos uma vez.*

*Depois que o gênero oposto usar o banheiro, desative automaticamente a alternância (`fairness_enforced = 0`), voltando ao funcionamento normal, onde homens e mulheres podem usar o banheiro sem prioridade, desde que as condições de uso sejam respeitadas.*

*Atualize também as condições de espera das threads. E ai agora, além de verificar se tem espaço no banheiro e se não há pessoas do outro gênero, veja também se a política de fairness permite o acesso ao gênero atual.*

*E ai, altere os sinais de condição `pthread_cond_signal` para liberar as threads bloqueadas. Quando uma thread sair do banheiro, envie sinais só para o gênero certo. Durante a alternância, libere apenas as threads do gênero com prioridade. Após o reset, libere threads de ambos os gêneros.”*

A resposta ajudou na correção dos problemas de alternância e fairness, utilizando as variáveis de controle fairness_policy_active e fairness_enforced para gerenciar a alternância entre os gêneros no uso do banheiro. A política foi configurada para ativar automaticamente após 6 acessos consecutivos de um gênero e desativar quando o outro gênero tivesse acesso. Isso eliminou situações de starvation e garantiu um uso equilibrado do banheiro, mesmo em cenários extremos de desigualdade no número de threads.

**PROMPT 9:**

*“Gere o arquivo Makefile para executar esse código”*

Essa resposta final incluiu a criação de um arquivo Makefile, que era um requisito do trabalho. Ele automatizou os processos de compilação, execução e limpeza do código unisex_bathroom.c, garantindo que o código fosse compilado corretamente. Com isso, foi

possível executar o programa utilizando comandos simples como make, make run e make clean. Essa automação ajudou a evitar erros manuais e tornou o fluxo de trabalho mais prático.

# Conclusão

O Chat GPT ajuda a acelerar o desenvolvimento de código e a trazer ideias novas, mas não deve ser considerado como solução única e nem como substituição do conhecimento prévio do programador. É importante o programador saber avaliar as respostas e testar se as funcionalidades estão corretas. No fim, ele é mais uma ferramenta para complementar o trabalho, não pra fazer tudo sozinho.