

**UNIDADE CURRICULAR:** Sistemas Operativos

**CÓDIGO:** 21111

**DOCENTE:** Paulo Shirley

**A preencher pelo estudante**

**NOME:** Luís Carlos Crispim Pereira

**N.º DE ESTUDANTE:** 2300163

**CURSO:** LEI – Licenciatura em Engenharia Informática

**DATA DE ENTREGA:** 24/05/24

**TRABALHO / RESOLUÇÃO:**

**Introdução:**

Este relatório descreve a implementação do programa mtxor.c, seguindo as orientações descritas no EFolio B. O objetivo do programa é realizar operações de XOR em uma sequência de números utilizando um buffer compartilhado entre múltiplas threads (produtor e consumidores). O programa foi concebido em linguagem C e faz uso extensivo de threads e mecanismos de sincronização para coordenar a produção e o consumo de itens no buffer.

**Opções:**

Bibliotecas: As cinco bibliotecas incluídas neste programa têm funções específicas que são usadas para fornecer acesso a diversas funcionalidades necessárias para manipulação de arquivos, processos e entrada/saída de dados, que são essenciais para a implementação do programa:

1. stdio.h: Usada para entradas e saídas padrão. Ela fornece funções como printf, fprintf, scanf, fscanf, getchar, putchar, entre outras.
2. stdlib.h: Fornece funções como malloc, calloc, realloc e free para alocação e liberação de memória dinâmica. Também inclui funções como exit e abort para sair ou abortar um programa. Embora esta biblioteca não seja utilizada no programa, optei por manter pois é uma biblioteca “padrão”.
3. unistd.h: Faculta o acesso a um grande número de funções do sistema operativo. Ela inclui funções para manipulação de arquivos (open, close, read, write), manipulação de diretórios (opendir, readdir, closedir), controle de processos (fork, exec, wait, exit) e manipulação de pipelines (pipe, dup, dup2).
4. pthread.h: Inclui funções para criação e manutenção de threads (pthread\_create, pthread\_join) e mecanismos de sincronização (pthread\_mutex\_lock, pthread\_mutex\_unlock).
5. sched.h: Proporciona funções para manipulação de políticas de escalonamento, como sched\_yield.

Variáveis utilizadas:

* Globais:
  + buffer buf: Estrutura que representa o buffer compartilhado entre tarefa produtora e tarefas consumidoras.
  + pthread\_mutex\_t\_mtx\_buf: Mutex usado para sincronização e acesso do buffer.
  + int dimbuf, N, nt: Parâmetros obtidos dos argumentos da linha de comando.
* Estruturas:
  + buffer, Estrutura que representa o buffer, contendo:
    - items: Ponteiro para o array itens.
    - count: Número atual de itens no buffer.
    - dimbuf: Capacidade máxima do buffer.
    - total\_items\_produced: Total de itens produzidos.
    - total\_items\_consumed: Total de itens consumidos.
    - N: Número total de itens a serem produzidos.
  + Consumer\_args: Estrutura para passar argumentos para as threads consumidoras, contendo:
    - id: Identificador da thread.
    - block\_size: Tamanho do bloco de itens que a thread vai consumir.
    - total\_operations: Total de operações realizadas pela thread.
    - local\_xor: Valor XOR calculado pela thread.
* Macros:
  + print\_test\_ativo: Macro para habilitar ou desabilitar a impressão de mensagens de test.
* Variáveis nas funções:
  + main:
    - argc: Contador de argumentos da linha de comando.
    - Argv: Array de strings com os argumentos da linha de comando.
    - pthread\_t consumers[nt]: Array de identificadores de threads consumidoras.
    - consumer\_args args[nt]: Array de estruturas para passar argumentos para as threads consumidoras.
    - int total\_operations: Total de operações realizadas por todas as threads consumidoras.
    - int final\_xor: Valor XOR final calculado após todas as threads consumidoras terem terminado.
  + producer:
    - int items\_produced: Contador de itens produzidos pela função.
  + consumer:
    - \*consumer\_args args: Ponteiro para a estrutura consumer\_args passada como argumento.
    - int items\_to\_consume: Número de itens que a thread vai consumir em uma iteração.

Abordagem:

Optei por uma programação modular e uma abstração funcional clara para facilitar a leitura e a manutenção do código. Cada parte do programa é responsável por uma funcionalidade específica, o que ajuda a manter o código organizado e eficiente.

Verificação de Argumentos, o programa começa verificando o número de argumentos fornecidos (argc). Se não forem exatamente três argumentos, o programa imprime uma mensagem de qual o uso correto e encerra.

Inicialização do Buffer, o buffer compartilhado é inicializado com os parâmetros fornecidos (dimbuf, N) e preparado para armazenar os itens produzidos.

Criação de Threads, são criadas múltiplas threads consumidoras (nt threads) para processar os itens produzidos. Cada thread consome um bloco de itens do buffer, realizando operações de XOR e contabilizando o número de operações realizadas.

Função Produtora, a função produtora (producer) gera números aleatórios e os coloca no buffer até que o número total de itens (N) seja alcançado. Utiliza mutex para sincronizar o acesso ao buffer, está é uma região critica do programa pois pode levar a uma “corrida” pelas thread ao buffer partilhado na tentativa de adicionar itens em simultâneo, isto é evitado pela utilização do mutex.

Função Consumidora, cada thread consumidora executa a função consumer, que consome itens do buffer, realiza operações de XOR nos itens consumidos e atualiza o contador de operações. As threads continuam até que todos os itens tenham sido produzidos e consumidos, está também é uma região critica do programa pois pode levar a uma “corrida” pelas thread ao consumo simultâneo no buffer partilhado, isto é evitado pela utilização do mutex.

Finalização, após todas as threads consumidoras terminarem, os resultados são combinados para calcular o valor final de XOR e o número total de operações realizadas. O programa imprime os resultados e libera os recursos alocados.

Pseudo-código acesso de uma tarefa consumidora ao buffer:

Função consumer (id, block\_size):

Inicializar local\_xor como 0

Inicializar total\_operations como 0

Enquanto Verdadeiro:

Aguardar bloqueio do mutex mtx\_buf

Se todos os itens foram produzidos e consumidos:

Libertar bloqueio do mutex mtx\_buf

Sair do loop

Se o buffer estiver vazio:

Libertar bloqueio do mutex mtx\_buf

Ceder o processador para outras tarefas

Continuar para a próxima iteração

Calcular items\_to\_consume como o mínimo entre o tamanho atual do buffer e block\_size

Para cada item em items\_to\_consume:

Decrementar contador do buffer

Calcular XOR do item com local\_xor

Incrementar total\_operations

Incrementar total\_items\_consumed no buffer

Libertar bloqueio do mutex mtx\_buf

Imprimir id e total\_operations realizadas

Finalizar tarefa com local\_xor como valor de retorno

**Conclusão e Resultados:**

Durante a execução do programa, foram realizados vários testes para garantir que a funcionalidade de produção e consumo de itens no buffer fosse realizada corretamente. As threads consumidoras processaram os itens conforme esperado, e os resultados de XOR foram consistentes com os valores esperados. O programa mtxor.c foi implementado com sucesso, atendendo aos requisitos especificados e funcionando corretamente com múltiplas threads para realizar operações de XOR em uma sequência de números. A modularidade do código e o uso de mecanismos de sincronização garantiram a correta coordenação entre o produtor e os consumidores, resultando em uma execução eficiente e confiável.

**Testes:**

Uma imagem com captura de ecrã, software, Software de multimédia, texto

Descrição gerada automaticamente

Figura 1 – Diretoria Raiz (Pasta SO - Efolio B no Desktop onde testei o programa)

Erros possíveis de acontecer que consegui replicar:

Uma imagem com captura de ecrã, texto, software, Software de multimédia

Descrição gerada automaticamente

Figura 2 – Utilizador não colocou o número de argumentos correto.

Uma imagem com captura de ecrã, texto, software, Software de multimédia

Descrição gerada automaticamente

Figura 3 – Valores inválidos – Utilizador não colocou valores válidos..

Resultados esperados com sucesso:

Uma imagem com captura de ecrã, texto, software, Software de multimédia

Descrição gerada automaticamente

Figura 4 - Programa compila e não produz avisos (warnnings) com gcc -Wall

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 5 – Programa compila, e funciona demonstra em consola a mensagem necessárias.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 6 – Programa compila, e funciona demonstra em consola a mensagem necessárias.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Figura 7 – Programa compila, e funciona demonstra em consola as mensagem necessárias. Com prints de test ativos.

**Nota final:** O programa ‘mtxor.c’, foi concebido em linguagem C, num determinado diretório (SO – Efolio B) num sistema Unix (neste caso Ubuntu em VM num Mac). Optei por criar a macro print\_test\_ativo (que vai em inativo) conforme aconselhado pelo prof. Shirley no fórum da UC.