Experimento #4

Sistemas Operacionais A

**Introdução:**

Este experimento tem com a intenção apresentar e testar os conhecimentos sobre o uso de threads na linguagem C.

A primeira parte do experimento consiste no problema entre produtores e consumidores que irão disputar por um buffer de números, os produtores tentam inserir N vezes o número 10, enquanto os consumidores retiram os valores já inseridos. Para tal o código deve ser corrigido e compilado sem erros de sintaxe e/ou lógica, com atenção também às verificações sobre o estado do buffer (se este encontra cheio, vazio, ou se os ponteiros de produtor e consumidor chegaram ao seu fim). Após compilar sem erros, deve-se executar o programa e verificar quantas vezes os produtores e consumidores conseguiram produzir ou consumir os elementos do buffer.

A segunda parte é a implementação da Solução de Jantar das Filósofas apresentada em aula, com a utilização de threads e semáforos para a garantia da exclusão mútua entre 5 threads que concorrem pela “comida”.

**Apresentação Dos Erros de Sintaxe e/ou Lógica do programa Exemplo:**

* **Erros de Sintaxe:**
* Linha 84: operação de incremento inválida



* Linha 111: operação de incremento inválida



* **Erros de Lógica:**
  + Biblioteca stdlib.h faltante (uso da função de exit())
  + Linha 53: cont\_p não inicializado.

****

* + Linhas 63 a 73: remoção do If que verifica incorretamente a posição dos ponteiros em myadd e como consequência a também remoção do else

****

* + Linha 83: alteração da condição do If de myremove(), que verifica se o buffer está vazio, substituído por if(cont\_p != 0)

****

* + Linhas 100 e 113: sum não inicializado e sendo decrementado ao invés de incrementado.

****

* + Linha 132: sinal de comparação entre cont\_c e NO\_OF\_ITERATIONS no while() é "<" ao invés de "<".

****

* + Linhas 158 e 159: para evitar warnings após a compilação do programa, foi realizado cast para intptr\_t do valor que representa os parâmetros da função que será gerada como thread. Além disso, para que as exibições do número do thread produtor ou consumidor sejam realizadas corretamente (Linhas 103, 116, 130 e 139) o valor de ID passado por parâmetro é recebido também por cast com intptr nas respectivas funções.

** **

**Respostas às perguntas:**

* **Contidas no texto do Experimento:**
* **Explique por que a vantagem do uso de *threads* é condicional**

A vantagem do uso de threads depende da forma o código foi estruturado, se ele pode ser executado de forma concorrente entre as rotinas o uso de threads é válido por apresentar maior velocidade na troca de contexto entre threads de um mesmo processo, mas caso seja um programa com execução sequencial não há nexo na utilização de threads em sua execução.

* **Apresente um quadro comparativo com, pelo menos, três aspectos para processos e *threads*.**

|  |  |  |
| --- | --- | --- |
| Aspectos | Processos | Threads |
|
| Compartilham Uma mesma memória | Não | Sim |
|
|
| Tempo para Troca de Contexto | Leva Mais Tempo | Leva Menos Tempo |
|
|
| Contexto | Todo Processo | Parte do Processo |
|

* **O que é a área de *heap*?**

A área de heap é uma parte da memória reservada para alocação dinâmica das variáveis a serem criadas durante a execução de um processo.

* **Quais são as funções do *dispatcher*?**

Este se encarrega da realização das trocas de contexto dos processos executados na CPU.

* **O que vem a ser a memória *cache*?**

A memória cache faz o intermédio entre a RAM e a CPU, por ser muito lenta em comparação a Central de processamento e RAM gera um desperdício na hora de enviar as informações para a CPU e para evitar esse gargalo foi criada uma memória mais rápida, a cache, que se comunica diretamente com o processador enviando o deve ser realizado por ele.

* **Contidas no código fonte:**
* **Por que ret não está sendo comparado a algum valor?**

Pois o valor de retorno (ret) pode ser 0, ou 1, caso 0, é considerado falso pelo if(), caso 1 considerado verdadeiro.

* **Por que não há necessidade de um cast?**

Pois apesar de ser um retorno na base decimal a função if() interpreta 0 ou 1 como resultado booleano (falso ou verdadeiro).

* **Para que serve cada um dos argumentos usados com pthread\_create?**

O primeiro argumento é uma variável do tipo p\_thread\_t criada para receber o id do thread criado.

O segundo é usado para adicionar ou não atributos ao thread criado, no caso do experimento não se deseja nenhum atributo então este campo receberá NULL.

O terceiro argumento se refere à qual rotina do programa será executada como thread.

O quarto e último argumento são passados os parâmetros que serão utilizados pela rotina do thread a ser criado.

* **O que ocorre com os threads criados, se ainda estiverem sendo executadas e a thread que as criou termina através de um pthread\_exit()?**

Não ocorre nada com os threads criados, pois a chamada apenas finaliza a thread que a chamou e não o processo.

* **Idem à questão anterior, se o termino se dá através de um exit()?**

Diferente da chamada pthread\_exit(), a chamada exit finaliza o processo, e com isso também as threads criadas nele.

**Resultados da Execução do Programa Exemplo:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Produtor | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
| Produção Obtida | 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Produção Esperada | 1000 | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Consumidor | #1 | #2 | #3 | #4 | #5 | #6 | #7 | #8 | #9 | #10 |
| Consumo Obtido | 1000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Consumo Esperado | 1000 | | | | | | | | | |

Como apresentado nas tabelas, por não haver métodos que garantem a exclusão mútua, houve produção/consumo monopolizado dentre os threads de mesma função, ou seja, um produtor produziu tudo e um consumidor consumiu tudo que foi produzido.

**Conclusão:**

No experimento 1 foi possível perceber qual o comportamento dos threads concorrentes sem uma implementação da exclusão mútua, causando monopolização da produção e do consumo em um único thread de sua respectiva função, justamente o comportamento esperado ao deixar um recurso sendo concorrido sem métodos que garantem exclusividade de seu uso.