|  |  |  |
| --- | --- | --- |
| http://www.uvanet.br/common/images/brasao_oficial.jpg | **Universidade Estadual Vale do Acaraú – UVA**  Ciências da Computação Sistemas Operacionais Prof. Walisson Pereira |  |

**Aluno: Luiz Roberto Rodrigues Nobre**

**Relatório do Experimento com o Uso de Threads:**

**Questionamentos:**

1. **O que o programa faz;**

**Codigo 1:**

O programa cria 100 threads (filhos), que simulam processos executando tarefas simples em paralelo. Cada thread imprime uma mensagem de "alô" e "tchau", enquanto o processo principal (pai) espera todos os filhos terminarem antes de encerrar com sua mensagem final.

**Codigo 2:**

Simula um produtor e um consumidor que compartilham um vetor de tamanho 3. O produtor preenche com números aleatórios e o consumidor zera. Isso se repete 5 vezes.

**Codigo 3:**

Cria 100 produtores e 100 consumidores que acessam um buffer de 2 posições. Os produtores colocam valores no buffer e os consumidores retiram, tudo controlado com semáforos e mutex para evitar problemas.

1. **Quais as estratégias usadas;**

**Codigo 1:**

Uso da biblioteca pthread para criação de threads,utilização de pthread\_join para garantir que o pai espere todos os filhos finalizarem e a função sleep(1) foi usada para simular que cada thread está realizando uma tarefa.

**Codigo 2:**

Uso de pthread para criar produtor e consumidor,uso de mutex para proteger o vetor,uso de uma variável turn para alternar a vez de cada thread.

**Codigo 3:**

Uso de semáforos (sem\_t) para controlar posições livres e ocupadas no buffer,uso de mutex para proteger o acesso simultâneo ao vetor,uso de malloc para garantir que cada thread tenha seu próprio ID exclusivo.

1. **Quais as dificuldades encontradas;**

**Codigo 1:**

Entender como passar os IDs corretamente para as threads sem que eles fiquem repetidos ou bagunçados.

**Codigo 2:**

A parte da sincronização usando a variável turn manualmente pode ser confusa e arriscada, e também o uso imediato de pthread\_join fez as execuções acontecerem em sequência, e não em paralelo.

**Codigo 3:**

Gerenciar 200 threads ao mesmo tempo e garantir que não haja conflitos de acesso, principalmente com IDs repetidos.

1. **Como você contornou esta dificuldade;**

**Codigo 1:**

Usei um vetor de inteiros (ids[i]) fora do loop, garantindo que cada thread pegasse seu próprio número fixo.

**Codigo 2:**

Manteve-se o comportamento sequencial como estava proposto. Controlei bem o uso do turn e do mutex pra evitar que as threads acessem o vetor ao mesmo tempo.

**Codigo 3:**

Usei malloc para alocar dinamicamente cada ID, garantindo que cada thread tivesse sua própria cópia. E também usei mutex e semáforos de forma correta para controlar o acesso ao buffer

1. **Quais fontes de pesquisa você consultou?**

**Em todos o códigos usei como fonte:**

Documentação oficial da biblioteca POSIX Threads (pthread)

Livros e materiais de estudo de Sistemas Operacionais:

- "Operating System Concepts" de Silberschatz, Galvin, e Gagne.

Tutoriais e exemplos de código no Stack Overflow para soluções específicas do meu problema.

**4:**

**Saída do código ao executá-lo:**

pré execução

meio da execução

produtor 0: produziu 7

consumidor 1: consumiu 7

produtor 0: produziu 89

consumidor 1: consumiu 89

consumidor 3: consumiu 89

consumidor 1: consumiu 89

produtor 2: produziu 89

produtor 2: produziu 62

consumidor 1: consumiu 62

produtor 2: produziu 39

consumidor 1: consumiu 39

consumidor 3: consumiu 39

consumidor 1: consumiu 73

produtor 2: produziu 73

produtor 2: produziu 72

consumidor 3: consumiu 72

consumidor 1: consumiu 72

consumidor 3: consumiu 29

produtor 2: produziu 29

produtor 2: produziu 56

consumidor 1: consumiu 56

pós-execução

Program ended with exit code: 0

**Você consegue responder:**

1. **Por que o consumidor 1 consumiu duas vezes o número 89?**

Porque o código não usa nenhum tipo de trava (mutex ou semáforo) para controlar o acesso à variável valor, que é compartilhada. Assim, dois consumidores podem acabar lendo o mesmo valor antes de um deles conseguir zerar. Isso fez com que o consumidor 1 conseguisse consumir o mesmo número duas vezes.

1. **Por que tanto o consumidor 1 e 3 consumiram o número 39? (caso similar com o número 72)**

Pelo mesmo motivo: como não tem controle na hora de acessar a variável, dois consumidores podem “ver” o mesmo valor ao mesmo tempo e os dois conseguem consumir. Eles basicamente acessam valor tão rápido e tão juntos que os dois passam pela verificação (if valor != 0) antes que um deles consiga zerar.

1. **Por que o consumidor 3 consumiu o número 29, mas o produtor 2 só o produziu depois?**

Na verdade, o produtor produziu antes sim. O que aconteceu foi que o printf que mostra “produtor produziu 29” apareceu **depois** na tela. Isso acontece porque as threads estão todas rodando ao mesmo tempo e o sistema não garante a ordem exata das mensagens. Então, as mensagens podem sair fora de ordem no terminal, mesmo que as ações tenham ocorrido em outra ordem real.

1. **Por que a execução sempre ocorre em ordem diferente, mesmo com o mesmo código?**

Porque o programa usa threads, que são executadas de forma concorrente. O sistema operacional é quem decide a ordem de execução das threads, e isso pode mudar toda vez que o programa roda. Pode depender de vários fatores, como o uso do processador no momento, ou até o tempo que uma thread leva pra ser escalada. Por isso, cada vez que o programa roda, a ordem pode sair diferente.