Tarefa 7: Algoritmo de Peterson

17 de junho de 2025

1 Por que usar enter_region e leave_region para estabelecer uma região crítica não funciona com compiladores modernos?

Os compiladores modernos realizam diversas otimizações de código que comprometem o funcionamento do algoritmo de Peterson:

- **Reordenamento de código:** Compiladores podem alterar a ordem das instruções. No algoritmo de Peterson, se `turn = process` for executado antes de `interested[process] = TRUE`, o algoritmo quebra.
- **Cache em registradores:** Variáveis compartilhadas podem ser armazenadas em registradores. Um processo pode ler valores desatualizados em vez dos atuais na memória.
- **Otimização de loops:** O loop de espera (`while (turn == process && interested[other] == TRUE)`) pode ser otimizado porque aparentemente não faz nada.
- **Eliminação de armazenamentos:** O compilador pode eliminar operações de memória consideradas desnecessárias.

2 Como os problemas descritos na pergunta anterior podem ser resolvidos?

As questões de otimização do compilador podem ser resolvidas através de:

- **Palavra-chave volatile:** Informa ao compilador que variáveis podem mudar inesperadamente:

```
volatile int turn;
volatile int interested[N];
```

- **Barreiras de memória:** Evitam reordenamento de instruções:

```
interested[process] = TRUE;
__sync_synchronize(); // Barreira de compilador
turn = process;
```

- **Diretivas específicas:** Para impedir otimizações:

```
#pragma optimize("", off)
// Código do algoritmo de Peterson
#pragma optimize("", on)
```

- **Primitivas de sincronização:** Usar mutexes do `<threads.h>` do C11 ou threads POSIX.

3 Com os problemas dos compiladores modernos resolvidos, qual problema ainda impede que o algoritmo de Peterson funcione em processadores modernos?

Mesmo resolvendo os problemas do compilador, as arquiteturas de processadores modernos introduzem desafios:

- **Ordenação de memória relaxada:** Processadores modernos não garantem consistência sequencial - as operações podem ocorrer em ordem diferente da especificada mesmo sem reordenamento do compilador.
- **Buffers e caches:** Uma CPU pode escrever em uma variável, mas outra CPU pode não ver a alteração imediatamente devido a:

  - Escrita intermediária em buffer
  - Diferentes versões em cache da memória
- **Execução fora de ordem:** Processadores executam instruções fora da sequência para otimizar o desempenho.
- **Operações não-atômicas:** Atribuições de variáveis podem não ser atômicas, especialmente para variáveis maiores.

4 Como os problemas descritos na pergunta anterior podem ser resolvidos?

Para resolver problemas em nível de processador:

- **Barreiras de memória de hardware:** Forçam ordenação correta:

```
interested[process] = TRUE;
__atomic_thread_fence(__ATOMIC_SEQ_CST);
turn = process;
```

- **Operações atômicas:** Garantem ordenação no compilador e hardware:

```
__atomic_store_n(&interested[process], TRUE, __ATOMIC_SEQ_CST);
__atomic_store_n(&turn, process, __ATOMIC_SEQ_CST);
```

- **Atomicidade da linguagem C11:**

```
#include <stdatomic.h>
atomic_int turn;
atomic_int interested[N];
```

- **Mecanismos de sincronização:** Mutexes (`pthread_mutex_t`), semáforos (`sem_t`) ou locks de leitura-escrita (`pthread_rwlock_t`) são projetados para lidar com desafios de concorrência em sistemas modernos.

5 Demonstrações Práticas e Resultados

Implementamos cinco versões do algoritmo para demonstrar os problemas e soluções:

- **Original:** Comportamento não-determinístico, falhas ou spinlock infinito.

```
interested[process] = TRUE; turn = process;
while (turn == process && interested[other] == TRUE);
```

- **Sem Otimizações (`-O0`):** Nossos testes demonstraram que mesmo com otimizações completamente desativadas, o algoritmo ainda falha gravemente (contador: 19.976.974 de 20.000.000). Este resultado é crucial porque prova conclusivamente que os problemas do algoritmo de Peterson não são causados apenas por otimizações do compilador, mas fundamentalmente pelo reordenamento de memória a nível do processador.
- **Diretivas de Pragma:** Usando `#pragma optimize(, off/on)` para desativar otimizações apenas nas funções críticas. Esta abordagem também falhou, reforçando que mesmo sem otimizações do compilador, o reordenamento a nível do processador ainda quebra o algoritmo.
- **Operações Atômicas GCC:** Usando `__atomic_store_n` com `__ATOMIC_SEQ_CST`.
- **Biblioteca C11:** Usando `atomic_int` e `memory_order_seq_cst`.
- **Mutex POSIX:** Substituição por `pthread_mutex_lock/unlock`.

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void enter_region(int process){
    pthread_mutex_lock(&mutex);
}

void leave_region(int process){
    pthread_mutex_unlock(&mutex);
}
```

Nossos testes com 10 milhões de incrementos confirmaram: apenas as implementações com operações atômicas ou mutex funcionaram corretamente, enquanto as versões original, sem otimizações e com pragmas falharam consistentemente. Estes resultados são fundamentais porque demonstram inequivocamente que o algoritmo de Peterson, como apresentado no livro de Tanenbaum, é inadequado para sistemas modernos devido ao reordenamento de memória a nível do processador - um problema que não pode ser resolvido simplesmente desativando otimizações do compilador.