Atividade Modulo 8 – Threads

3 - A) O que é o problema do produtor consumidor e como podemos resolver este problema com threads?

R: Este problema também é conhecido como o problema do buffer limitado, e consiste em coordenar o acesso de tarefas (processos ou threads) a um buffer compartilhado com capacidade de armazenamento limitada a N itens(que podem ser inteiros, registros, mensagens, etc).Esse tipo de problema pode-se resolver o problema dos produtores/consumidores de forma eficiente usando um mutex e dois semáforos, um para cada aspecto de coordenação envolvido.

B) Indique mais um problema classico da computação e como podemos resolve-los com threads explicando de forma detalhada?

R:Leitores/escritores - Outra situação que ocorre com frequência em sistemas concorrentes é o problema dos leitores/escritores. Neste problema, um conjunto de tarefas acessam de forma concorrente uma área de memória compartilhada, na qual podem fazer leituras ou escritas de valores. De acordo com as condições de Bernstein (Seção 10.1.3), as leituras podem ser feitas em paralelo, pois não interferem umas com as outras, mas as escritas têm de ser feitas com acesso exclusivo à área compartilhada, para evitar condições de disputa. A Figura 12.2 mostra leitores e escritores acessando de forma concorrente uma matriz de números inteiros M. O estilo de sincronização leitores/escritores é encontrado com muita frequência em aplicações com múltiplas threads. O padrão POSIX define mecanismos para a criação e uso de travas com essa funcionalidade, acessíveis através de chamadas como pthread\_rwlock\_init(), entre outras.

Solução simplista - Uma solução simplista para esse problema consistiria em proteger o acesso à área compartilhada com um mutex ou semáforo inicializado em 1; assim, somente um Essa solução deixa muito a desejar em termos de desempenho, porque restringe desnecessariamente o acesso dos leitores à área compartilhada: como a operação de leitura não altera os valores armazenados, não haveria problema em permitir o acesso paralelo de vários leitores à área compartilhada, desde que as escritas continuem sendo feitas de forma exclusiva

|  |  |  |
| --- | --- | --- |
| **OBJETOS** | **DESCRIÇÃO** | **COMANDO DE INICIALIZAÇÃO** |
| THREAD | Thread, em inglês, significa "fio" ou "linha", e no contexto dos microprocessadores representa uma ordem de execução, com instruções encadeadas que são desempenhadas uma por vez. | Thread.start() |
| LOCK | Um objeto Lock pode ser liberado por qualquer thread. | Threading.lock() |
| RLOCK | Um objeto RLock só pode ser liberado pelo thread que o adquiriu. | Threading.Rlock() |
| CONDITION | Uma variável de condição está sempre associada a algum tipo de bloqueio; isso pode ser passado ou um será criado por padrão. Passar um é útil quando várias variáveis ​​de condição devem compartilhar o mesmo bloqueio. O bloqueio faz parte do objeto de condição: você não precisa rastreá-lo separadamente | Threading.condition() |
| EVENT | Este é um dos mecanismos mais simples de comunicação entre threads: uma thread sinaliza um evento e outras threads esperam por ele. | Threading.event() |
| SEMAPHORE | Os semáforos são frequentemente usados ​​para proteger recursos com capacidade limitada, por exemplo, um servidor de banco de dados. Em qualquer situação em que o tamanho do recurso seja fixo, você deve usar um semáforo limitado. | Threading.semaphore() |
| BOUNDEDSEMAPHORE | Classe que implementa objetos de semáforo limitados. Um semáforo limitado verifica se seu valor atual não excede seu valor inicial. Se isso acontecer, ValueErroré gerado. Na maioria das situações, os semáforos são usados ​​para proteger recursos com capacidade limitada. Se o semáforo for lançado muitas vezes é sinal de um bug. Se não for fornecido, o valor padrão é 1 | Threading.boundedsemaphore() |
| TIMER | Essa classe representa uma ação que deve ser executada somente após um certo período de tempo — um cronômetro. [Timer](https://docs.python.org/pt-br/3/library/threading.html" \l "threading.Timer" \o "threading.Timer)é uma subclasse de [Thread](https://docs.python.org/pt-br/3/library/threading.html#threading.Thread) e, como tal, também funciona como um exemplo de criação de threads personalizados. | Threading.timer() |
| BARRIER | Essa classe fornece uma primitiva de sincronização simples para uso por um número fixo de threads que precisam esperar uns pelos outros. Cada uma das threads tenta passar a barreira chamando o [wait()](https://docs.python.org/pt-br/3/library/threading.html" \l "threading.Barrier.wait" \o "enfiar.Barreira.esperar)método e bloqueará até que todas as threads tenham feito suas [wait()](https://docs.python.org/pt-br/3/library/threading.html" \l "threading.Barrier.wait" \o "enfiar.Barreira.esperar)chamadas. Neste ponto, os threads são liberados simultaneamente. | Threading.barrier() |