| OBJETO | DESCRIÇÃO | COMANDO DE INICIALIZAÇÃO |
|---|---|---|
| Thread | Fluxos de programas que executam em paralelo dentro de uma aplicação, ou seja, uma ramificação de uma parte da aplicação que é executada de forma independente e escalonada independentemente do fluxo inicial da aplicação. | t1 = Thread(target=proc1).start() |
| Lock | Ferramenta de sincronização que possui dois estados: bloqueado e desbloqueado. Quando uma thread está bloqueada, as tentativas de acessá-la são impedidas até ela ser desbloqueada. O lock só pode ser adquirido uma vez, sendo necessário desbloqueá-lo para ser adquirido novamente (pode ser readquirido por qualquer thread). Pode ser liberado por qualquer thread. | lock = threading.Lock() |
| RLock | Ferramenta de sincronização que possui dois estados: bloqueado e desbloqueado. RLock pode ser adquirido múltiplas vezes pela mesma thread e deve ser liberado o mesmo número de vezes que foi adquirido para ser desbloqueado. Só pode ser liberado pela thread que o adquiriu. | lock = threading.RLock() |
| Condition | Uma variável de condição permite que uma ou mais threads aguardem até serem notificadas por uma outra thread. | condicao = threading.Condition() |
| Event | Um mecanismo simples de comunicação entre threads. Uma das threads manda o sinal de um evento e outras threads aguardam por ele. | evento = threading.Event() |
| Semaphore | Um primitivo de sincronização que gerencia um contador | threading.Semaphore() |

| | | |
|---|---|---|
| | interno que decrementa por cara chamada “acquire()” e incrementa a cada chamada “release()”. O contador nunca pode ser menor do que zero. quando uma chamada “acquire()” se depara com o contador zero, é bloqueado e aguarda até outra thread realizar uma chamada “release()”. | - caso argumento não seja passado, valor = 1 |
| BoundedSemaphore | Mecanismo semelhante ao Semaphore, porém o valor do contador nunca excede o valor inicial. | threading.BoundedSemaphore()<br><br>- caso argumento não seja passado, valor = 1 |
| Timer | Mecanismo que possibilita que um determinado período de tempo se passe antes de executar algo. | Timer(intervaloDeTempo, funcao, *args=None*, *kwargs=None*).start()<br><br>- caso nenhum argumento ou argumento keyword seja passado serão utilizados respectivamente: lista vazia e dicionário vazio |
| Barrier | Mecanismo para sincronização em que é necessário ter um número específico de threads para que os processos sejam inicializados. Os processos são inicializados simultaneamente. | threading.Barrier(*5, action=None, timeout=None*) |