# Laboratórios de Sistemas Operativos
# Tutorial #5: Sincronização de secções críticas

*Os tutoriais práticos de SO consistem num conjunto de exercícios práticos que permitem aos alunos familiarizarem-se com um determinado tema que será necessário para resolver os projetos da disciplina. Os tutoriais podem ser resolvidos individualmente ou em grupo. A sua resolução é recomendada mas não obrigatória. Não são avaliados.*

*Cada tutorial pressupõe que os exercícios são realizados numa interface de linha de comandos (shell) de um sistema Unix/Linux ou equivalente. Assume também que os alunos já resolveram os tutoriais anteriores.*

Este guião pretende ajudar os alunos a resolverem (simples) problemas de sincronização usando as primitivas de *mutex* e *read-write lock*.

1. Descarregue o programa *contapartilhada.c* e comece por estudar o seu conteúdo.
   Neste programa, o casal Alice e Bob partilham uma conta bancária. Alice tem um emprego e, sempre que recebe um pagamento, deposita-o na conta do casal. Bob, pelo contrário, vive à custa dos rendimentos de Alice. Gosta muito de passear e ver montras e, quando não resiste, tenta levantar dinheiro da conta do casal para fazer alguma compra – embora nem sempre consiga!
2. Compile este programa. Execute-o passando diferentes valores como argumento. Experimente com 100, 1000, 10000 e valores superiores.
   Para cada valor, experimente repetir a execução algumas vezes e observe se o resultado impresso é o mesmo.
   a. Tente encontrar um exemplo em que, correndo duas vezes passando argumento idêntico, o Bob acaba por levantar diferentes montantes. Como explica este fenómeno?
   b. Tente agora encontrar um exemplo em que o saldo final da conta não reflete o total depositado pela Alice subtraído pelo total gasto pelo Bob. Como explica este caso mais grave?
3. Identifique as secções críticas neste programa.
4. Resolva o problema de sincronização existente utilizando um trinco lógico (mutex).
   a. Pode declará-lo e inicializá-lo da seguinte forma:

      ```
      pthread_mutex_t trinco;
      pthread_mutex_init(&trinco, NULL);
      ```

      Ou simplesmente:

      ```
      pthread_mutex_t trinco = PTHREAD_MUTEX_INITIALIZER;
      ```

      Lembre-se que também existe uma função para destruir o *mutex* (qual é?)
   b. De seguida, use as funções *pthread_mutex_lock* e *pthread_mutex_unlock* para sincronizar as secções críticas que identificou.
   c. Experimente e confirme que o erro grave que detetou na 2.b já não se verifica.

5. Acrescente agora 4 tarefas (*threads*) que, no seu ciclo, se limitam a chamar a função *consultar_conta*. No ciclo da Alice e Bob, acrescente também uma chamada à mesma função no final de cada iteração.
    a. Caso não tenha antes sincronizado a secção crítica em *consultar_conta* (pois essa função antes não era invocada concorrentemente), lembre-se que agora tem de o fazer!
    b. Com este novo programa, a função *consultar_conta* passa a ser aquela que é mais frequentemente executada no programa. Nota também que é uma função que apenas lê sobre dados partilhados (ou seja, nunca modifica dados partilhados). Assim sendo, o programa é um bom candidato a beneficiar do uso de um trinco de leitura-escrita (*rwlock*), em vez de um *mutex*.

        Desenvolva um esquema de sincronização baseado em read-write locks que permita que o máximo número de tarefas possa executar em paralelo.
        i. Para tal, passe a declarar um trinco deste novo tipo:

            ```
            pthread_rwlock_t rwl;

            pthread_rwlock_init(&rwl, NULL);
            ```

        ii. E passe a usar as funções *pthread_rwlock_rdlock* e *pthread_rwlock_wrlock* sempre que se iniciar uma secção crítica de leitura-apenas ou uma secção crítica em que haja pelo menos uma escrita a dados partilhados (respetivamente).

            Em ambos os casos, liberte as secções críticas com *pthread_rwlock_unlock*.
    c. Observe como muda o tempo de execução do programa ao usar *mutexes* e *rwlocks*.
        i. Experimente (i) simular atrasos crescentes no acesso em leitura/escrita à conta chamando a função *sleep* dentro das secções críticas e (ii) alterar o número de tarefas que executam consultar_conta().
        ii. Em que situações consegue observar ganhos de desempenho para a solução baseada em *rwlocks*?
6. Agora aplique estes conhecimentos no seu projeto!