Chapter 5c: Mutex Example: A Non-Contiguous Critical Section

Gene Cooperman

THIS IS A WORK IN PROGRESS.

1 Non-Contiguous Critical Sections with Mutexes

1.1 The Bank Account Example

Consider a simple bank that maintains accounts. Two people – Alice and Bob – share a joint checking account. Each person is represented by a separate thread. Both threads can deposit to and withdraw from the same account concurrently.

The shared data is the account balance, which is a global variable. Since both threads can read and modify it at the same time, we need a mutex to protect it.

1.2 C Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

/* ---------------------------------------------------------
 * Shared data: the joint account balance and its mutex
 * --------------------------------------------------------- */
double balance = 1000.00;           /* global -- shared by all threads */
pthread_mutex_t account_lock = PTHREAD_MUTEX_INITIALIZER;

/* ---------------------------------------------------------
 * deposit(): add amount to the account
 * --------------------------------------------------------- */
void deposit(double amount)
{
    pthread_mutex_lock(&account_lock);       /* enter critical section */
    balance = balance + amount;
    printf("  Deposited $%.2f  -> balance = $%.2f\n", amount, balance);
    pthread_mutex_unlock(&account_lock);     /* exit critical section */
}

/* ---------------------------------------------------------
```

```c
 * withdraw(): subtract amount from the account
 * -------------------------------------------------------- */
void withdraw(double amount)
{
    pthread_mutex_lock(&account_lock);       /* enter critical section */
    if (balance >= amount) {
        balance = balance - amount;
        printf("  Withdrew  $%.2f  -> balance = $%.2f\n", amount, balance);
    } else {
        printf("  Withdrew  $%.2f  -> DENIED (balance = $%.2f)\n",
               amount, balance);
    }
    pthread_mutex_unlock(&account_lock);     /* exit critical section */
}

/* --------------------------------------------------------
 * alice_thread(): Alice's sequence of transactions
 * -------------------------------------------------------- */
void *alice_thread(void *arg)
{
    printf("Alice: depositing paycheck\n");
    deposit(1500.00);                        /* critical section #1 */

    printf("Alice: paying rent\n");
    withdraw(1200.00);                       /* critical section #2 */

    printf("Alice: depositing bonus\n");
    deposit(500.00);                         /* critical section #3 */

    return NULL;
}

/* --------------------------------------------------------
 * bob_thread(): Bob's sequence of transactions
 * -------------------------------------------------------- */
void *bob_thread(void *arg)
{
    printf("Bob: withdrawing grocery money\n");
    withdraw(200.00);                        /* critical section #1 */

    printf("Bob: depositing refund\n");
    deposit(75.00);                          /* critical section #2 */

    printf("Bob: withdrawing gas money\n");
    withdraw(50.00);                         /* critical section #3 */

    return NULL;
}

/* --------------------------------------------------------
 * main(): create threads and wait for them to finish
 * -------------------------------------------------------- */
int main(void)
{
    pthread_t tid_alice, tid_bob;

    printf("Initial balance: $%.2f\n\n", balance);

    pthread_create(&tid_alice, NULL, alice_thread, NULL);
    pthread_create(&tid_bob,   NULL, bob_thread,   NULL);

    pthread_join(tid_alice, NULL);
    pthread_join(tid_bob,   NULL);

    printf("\nFinal balance: $%.2f\n", balance);
    return 0;
}
```

Compile and run:

```
gcc -o bank bank.c -lpthread
./bank
```

1.3 What Is a Non-Contiguous Critical Section?

A **contiguous** critical section is a single, unbroken block of code between one lock and one unlock:

```
lock
  ... critical code ...
unlock
```

In the bank example, each individual call to `deposit()` or `withdraw()` contains a contiguous critical section – the code between `pthread_mutex_lock` and `pthread_mutex_unlock` within that function.

However, from the perspective of each **thread**, the critical section is **non-contiguous**. Look at Alice's thread:

```
alice_thread:
    deposit(1500.00);      <-- lock ... critical code ... unlock
    withdraw(1200.00);     <-- lock ... critical code ... unlock
    deposit(500.00);       <-- lock ... critical code ... unlock
```

Alice accesses the shared `balance` variable in three separate places, with ordinary (non-critical) code in between. The overall region of code that touches shared data is not one continuous block – it is spread across multiple lock/unlock pairs. This is a **non-contiguous critical section**.

The same is true for Bob's thread, which also accesses the shared balance in three separate, non-adjacent lock/unlock pairs.

1.4 Why Non-Contiguous Critical Sections Matter

The key insight is that we **cannot** simply lock the mutex once at the beginning of Alice's thread and unlock it at the end. If we did that:

```
alice_thread:
    lock
    deposit(1500.00);
    withdraw(1200.00);     // Bob is completely locked out
    deposit(500.00);       // for the entire duration
    unlock
```

Then Bob would be blocked for the entire duration of Alice's execution. This defeats the purpose of having multiple threads – we want Alice and Bob to be able to interleave their transactions.

Instead, we lock the mutex only for the **minimum necessary duration** – just long enough to complete one atomic operation on the shared data. Between transactions, the mutex is released, giving the other thread an opportunity to run. The result is a non-contiguous critical section: multiple short, separate critical regions within each thread, rather than one long one.

1.5 Visualizing the Interleaving

Because the critical sections are non-contiguous, Alice's and Bob's transactions can interleave freely:

```
Time
 |    Alice                          Bob
 |    ------                         ---
 |    lock
 |      deposit $1500
 |    unlock
 |                                   lock
 |                                     withdraw $200
 |                                   unlock
 |    lock
 |      withdraw $1200
 |    unlock
 |                                   lock
 |                                     deposit $75
 |                                   unlock
 |                                   lock
 |                                     withdraw $50
 |                                   unlock
 |    lock
 |      deposit $500
 |    unlock
 v
```

Each individual transaction is atomic (protected by the mutex), but the threads are free to interleave between transactions. This is the essence of a non-contiguous critical section: the shared data is accessed in multiple, separate, protected regions rather than one monolithic block.