Working with Threads in C on Linux

Objective: Understand and implement multithreading using the POSIX threads (pthreads) library. Explore synchronization primitives like mutexes and condition variables.

Prerequisites:

- Basic C programming

- GCC compiler

- Linux OS with pthread library installed (-lpthread)

- Editor of choice (e.g., vim, VSCode, gedit)

Exercise 1: Hello from Threads

Goal: Create multiple threads that print messages independently.

Instructions:

1. Write a C program that creates 5 threads.

2. Each thread should print: 'Hello from thread X', where X is the thread number.

3. Wait for all threads to finish before exiting the program.

Hints:

- Use pthread_create and pthread_join.

- Pass the thread number as an argument to the thread function.

Exercise 2: Summing Array Segments with Threads

Goal: Use threads to compute the sum of an array in parallel.

Instructions:

1. Initialize an array of 100 integers with values from 1 to 100.

2. Create 4 threads. Each thread sums 25 elements.

3. Use a global total_sum variable.

4. Use a mutex to protect access to total_sum.

Hints:

- Use pthread_mutex_t to avoid race conditions.

- Initialize the mutex with pthread_mutex_init.

Exercise 3: Producer-Consumer with Condition Variables

Goal: Implement a bounded-buffer (circular queue) using threads, mutexes, and condition variables.

Instructions:

1. Create a buffer with a fixed size (e.g., 5 items).

2. Implement a producer thread that adds numbers to the buffer.

3. Implement a consumer thread that removes and prints numbers.

4. Use pthread_mutex_t and pthread_cond_t to synchronize access.

Hints:

- Use pthread_cond_wait and pthread_cond_signal.

- Ensure the producer waits if the buffer is full, and the consumer waits if it's empty.

Function	**Purpose**
pthread_cond_t	Defines a condition variable for signaling between threads.
pthread_cond_init	Initializes a condition variable before use.
pthread_cond_wait(&cond, &mutex)	Waits for a signal on cond. Atomically unlocks mutex and suspends the thread. Re-locks mutex when signaled and thread resumes.
pthread_cond_signal(&cond)	Wakes one waiting thread (if any) on the condition variable.
pthread_mutex_unlock(&mutex)	Unlocks a previously locked mutex. Always used after modifying shared state.

Exercise 4: Simulate a Bank Account with Multiple Clients

Goal: Simulate multiple clients withdrawing and depositing money using proper synchronization.

Instructions:

1. Create a shared bank account with a starting balance of 1000.

2. Launch 3 deposit threads and 3 withdrawal threads.

3. Each deposit thread deposits 100 ten times.

4. Each withdrawal thread withdraws 50 twenty times.

5. Protect the shared balance with a mutex.

Optional Challenge:

- Ensure withdrawals only happen if the balance is sufficient using a condition variable.

Hints:

- Carefully lock/unlock the mutex around read/write to the balance.

- Use pthread_cond_wait to pause withdrawals when funds are insufficient.

Final Notes

Clean-up & Best Practices:

- Always pthread_join all threads.

- Destroy mutexes and condition variables using pthread_mutex_destroy and pthread_cond_destroy.

- Test for race conditions by running with valgrind or under high load.