Q-program implementing the Peterson's solution for *mutual exclusion*:

```
#include <iostream>

#include <pthread.h>


using namespace std;


#define N 2 // Number of threads


bool flag[N];

int turn;


void* threadFunction(void* arg) {

   int id = *((int*)arg);

   int other = 1 - id;


   flag[id] = true;

   turn = other;


   // Entry section

   while (flag[other] && turn == other) {}


   // Critical section

   cout << "Thread " << id << " is in critical section" << endl;


   // Exit section

   flag[id] = false;
```

```
    return NULL;
}

int main() {
    pthread_t threads[N];
    int ids[N] = {0, 1};

    for (int i = 0; i < N; ++i) {
        pthread_create(&threads[i], NULL, threadFunction, &ids[i]);
    }

    for (int i = 0; i < N; ++i) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```

Q-program using `fork()` to create a child process:

```
#include <iostream>
#include <unistd.h> // For fork()

using namespace std;

int main() {
    pid_t pid = fork(); // Create a child process

    if (pid == -1) {
        // Fork failed
        cerr << "Fork failed" << endl;
        return 1;
    } else if (pid == 0) {
        // Child process
        cout << "Child process: My PID is " << getpid() << endl;
        cout << "Child process: My parent's PID is " << getppid() << endl;
    } else {
        // Parent process
        cout << "Parent process: My PID is " << getpid() << endl;
        cout << "Parent process: My child's PID is " << pid << endl;
    }

    return 0;
}
```

Q-program using threading with the `<thread>` library:

```
#include <iostream>
#include <thread>

using namespace std;

// Function to be executed by each thread
void threadFunction(int threadID) {
    cout << "Thread " << threadID << ": Hello, World!" << endl;
}
```

```
int main() {
    const int numThreads = 5;


    // Create an array of threads
    thread threads[numThreads];


    // Launch each thread
    for (int i = 0; i < numThreads; ++i) {
        threads[i] = thread(threadFunction, i + 1);
    }


    // Join each thread to the main thread
    for (int i = 0; i < numThreads; ++i) {
        threads[i].join();
    }


    return 0;
}
```