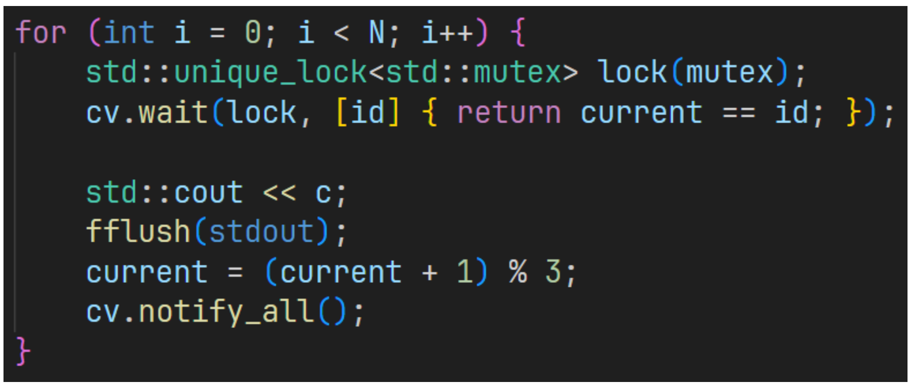
作业 7 多线程同步

1 设计思路

为每个线程分配一个 id, 用全局变量 `current` 标识现在需要输出的线程。当线程 id 与 `current` 相等时输出。每个线程输出结束时，改变 `current` 的值，通过环境变量唤醒所有线程，通知所有线程检查现在 `current` 与自身 id 的关系，不符合条件的线程继续休眠，符合条件的线程输出。

2 关键代码

如图 1，每个线程需要循环 N 次。线程通过环境变量判断是否符合要求，符合时，环境变量将锁释放，进行后续输出。输出后马上刷新缓冲区，确保结果正确输出到屏幕。最后，改变 `current` 值，唤醒所有线程进行检查。

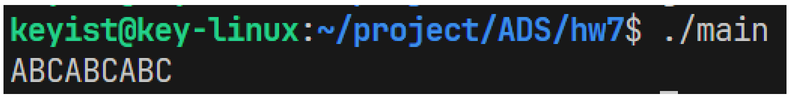


```
for (int i = 0; i < N; i++) {
    std::unique_lock<std::mutex> lock(mutex);
    cv.wait(lock, [id] { return current == id; });

    std::cout << c;
    fflush(stdout);
    current = (current + 1) % 3;
    cv.notify_all();
}
```

图 1: 关键代码

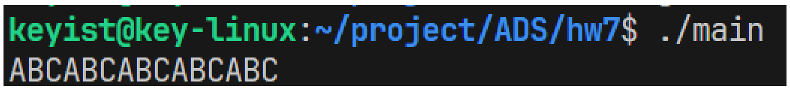
3 输出结果

N 取 3、5、7 时的结果如下图。



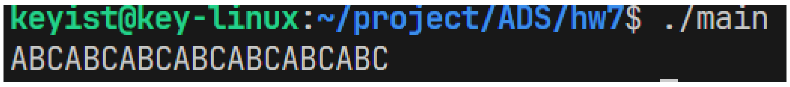
```
keyist@key-linux:~/project/ADS/hw7$ ./main
ABCABCABC
```

图 2: N = 3



```
keyist@key-linux:~/project/ADS/hw7$ ./main
ABCABCABCABCABC
```

图 3: N = 5

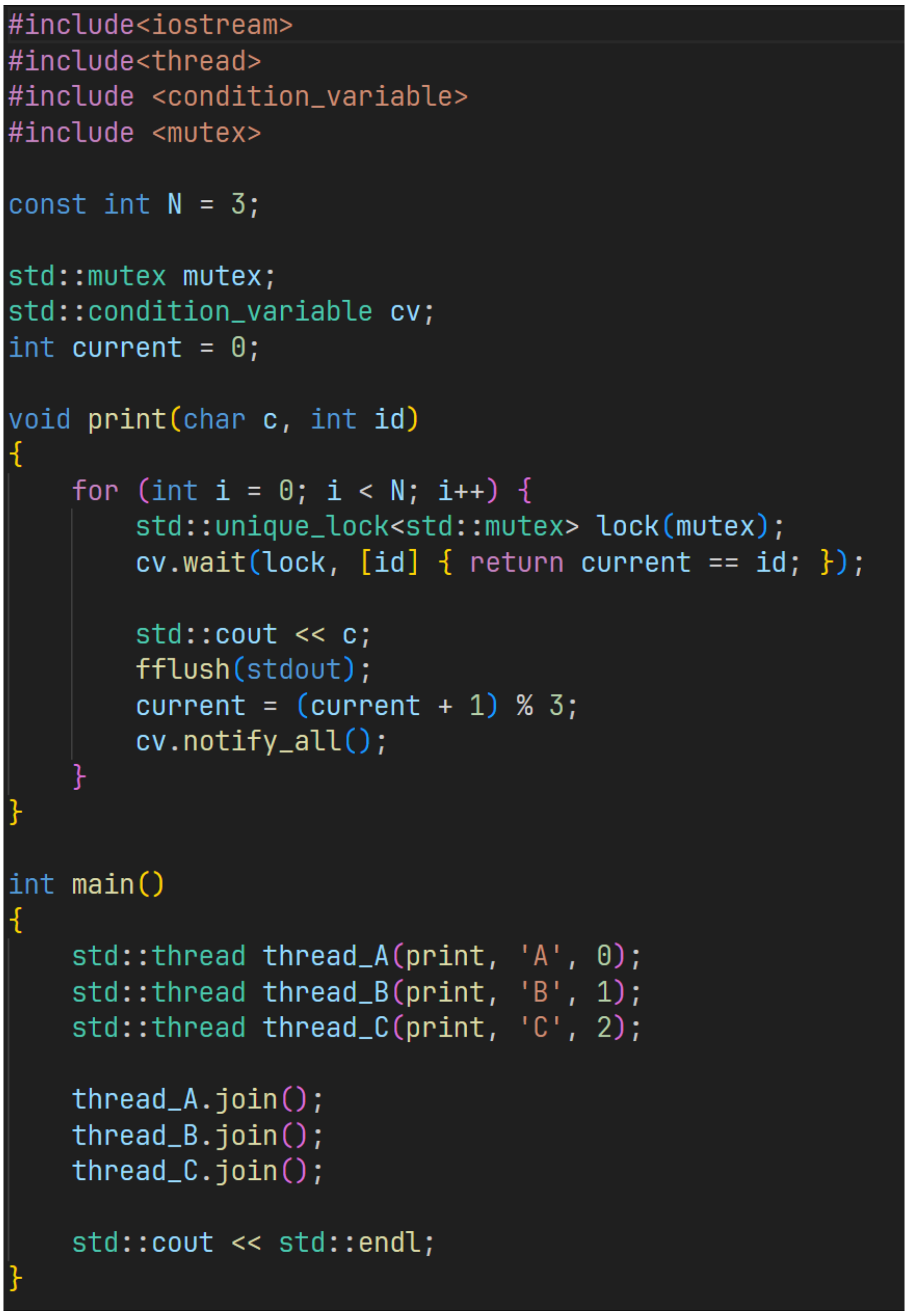


```
keyist@key-linux:~/project/ADS/hw7$ ./main
ABCABCABCABCABCABCABC
```

图 4: N = 7

4 完整代码

```
#include<iostream>
#include<thread>
#include <condition_variable>
#include <mutex>

const int N = 3;

std::mutex mutex;
std::condition_variable cv;
int current = 0;

void print(char c, int id)
{
    for (int i = 0; i < N; i++) {
        std::unique_lock<std::mutex> lock(mutex);
        cv.wait(lock, [id] { return current == id; });

        std::cout << c;
        fflush(stdout);
        current = (current + 1) % 3;
        cv.notify_all();
    }
}

int main()
{
    std::thread thread_A(print, 'A', 0);
    std::thread thread_B(print, 'B', 1);
    std::thread thread_C(print, 'C', 2);

    thread_A.join();
    thread_B.join();
    thread_C.join();

    std::cout << std::endl;
}
```



```
#include<iostream>
#include<thread>
#include <condition_variable>
#include <mutex>

const int N = 3;

std::mutex mutex;
std::condition_variable cv;
int current = 0;

void print(char c, int id)
{
    for (int i = 0; i < N; i++) {
        std::unique_lock<std::mutex> lock(mutex);
        cv.wait(lock, [id] { return current == id; });

        std::cout << c;
        fflush(stdout);
        current = (current + 1) % 3;
        cv.notify_all();
    }
}

int main()
{
    std::thread thread_A(print, 'A', 0);
    std::thread thread_B(print, 'B', 1);
    std::thread thread_C(print, 'C', 2);

    thread_A.join();
    thread_B.join();
    thread_C.join();

    std::cout << std::endl;
}
```

图 5: 完整代码