lab4实验报告

并行化代码思路

void* calculate_life(void* arg)

该函数用于每个线程模拟生命游戏的一部分，具体步骤如下：

1、**计算邻居数**：对给定区域内的每个细胞，调用 `count_live_neighbors` 函数计算该细胞的活邻居数量。

2、更新细胞状态

如果细胞当前是活的（值为 1），并且其邻居数不等于 2 或 3，则该细胞死亡（更新为 0）。

如果细胞当前是死的（值为 0），并且其邻居数恰好为 3，则该细胞复活（更新为 1）。

3、使用互斥锁进行保护

void simulate_life_parallel(int threads, LifeBoard* state, int steps)

1、**初始化互斥锁**：在开始并行模拟前，初始化一个互斥锁 `mutex`，确保对共享资源（棋盘）的访问是线程安全的。

分配任务给线程：

- 计算每个线程需要处理的行数：`rows_per_thread = state->height / threads`。
- 对每个线程，计算其要处理的行范围（`start_row` 到 `end_row`），并创建一个 `ThreadArgs` 结构体来存储这些信息。

创建线程：

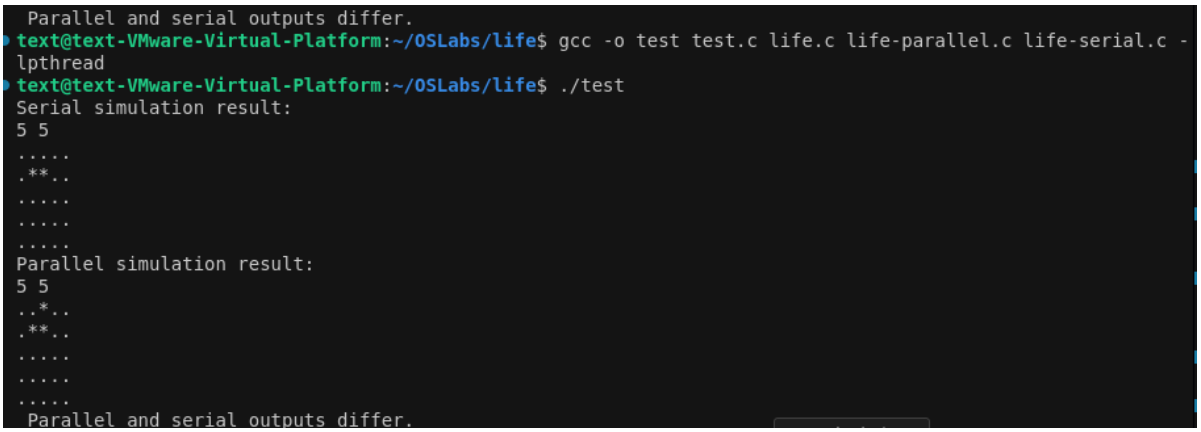
- 对每个线程调用 `pthread_create`，并将 `ThreadArgs` 作为参数传递给 `calculate_life` 函数，指示线程处理哪部分棋盘。

等待所有线程完成：

- 在每次迭代完成后，调用 `pthread_join` 等待所有线程的执行完成，确保在进行下一步之前，所有线程都完成了对棋盘的更新。

销毁互斥锁：在所有线程完成后，销毁互斥锁。

运行test.c比较串行和并行程序的结果


Parallel and serial outputs differ.
text@text-VMware-Virtual-Platform:~/OSLabs/life$ gcc -o test test.c life.c life-parallel.c life-serial.c -
lpthread
text@text-VMware-Virtual-Platform:~/OSLabs/life$ ./test
Serial simulation result:
5 5
.....
.**..
.....
.....
.....
Parallel simulation result:
5 5
..*..
.**..
.....
.....
.....
Parallel and serial outputs differ.