高级任务调度器 (Advanced Task Scheduler) 项目说明文档

1. 项目概述

本项目是一个基于 C++ (Win32 API) 开发的多线程任务调度系统。它模拟了一个复杂的后台处理环境，具备任务队列管理、并发执行、异常处理、系统死锁模拟及数据可视化功能。

核心目标：

- 展示面向对象编程（多态、工厂模式、单例模式）。
- 演示多线程编程与同步机制（互斥锁、条件变量）。
- 实现原生 Windows GUI 交互与数据可视化布局。
- 模拟系统故障（死锁/冻结）与人工干预恢复流程。

2. 系统架构

系统采用分层架构设计，主要包含以下核心组件：

2.1 类结构设计

- `ITask` (接口层)：定义所有任务的基类，包含纯虚函数 `Execute()`，实现多态调用。
- 具体任务类 (实现层)：
  - `TaskBackup` (Task A): 磁盘 I/O 操作。
  - `TaskMatrix` (Task B): 密集计算与数据格式化。
  - `TaskHttp` (Task C): 网络请求模拟。
  - `TaskReminder` (Task D): 跨线程 UI 交互。
  - `TaskStats` (Task E): 统计学计算与报表生成。
  - `TaskChaos` (Task F): 异常注入与故障触发。
- `TaskScheduler` (核心控制层)：
  - 采用 单例模式 (Singleton)。
  - 维护后台工作线程（`Worker Loop`）和任务队列（`taskList`）。
  - 负责线程同步、任务调度、死锁冻结与系统恢复。
- `TaskFactory` (工厂层)：根据 UI 事件 ID 生成对应的任务实例，解耦 UI 与逻辑。
- `LogWriter` (基础设施)：提供线程安全的日志写入功能。

2.2 UI 布局架构

界面采用 三栏式布局，高度像素级对齐：

1. 左栏 (Control Panel)：包含任务触发按钮及系统工具（死锁/重置）。
2. 中栏 (Status Hub)：

- Task Queue：显示待处理任务列表。
- System Log：显示系统状态流水（支持水平滚动）。

3. 右栏 (Data Visualization Board)：专用大屏，用于展示 Task B 的矩阵和 Task E 的统计报表。

3. 功能模块详解

3.1 任务类型

ID	任务名称	功能描述	技术细节
Task A	真实备份	将 C:\Data 目录打包压缩至 D:\Backup 。	自动创建测试文件，调用 PowerShell Compress-Archive 命令。
Task B	矩阵计算	生成 200x200 矩阵并进行乘法运算。	在 Data Board 输出格式化的 10x10 预览，带循环计数 Iteration: N)。
Task C	网络请求	模拟 HTTP GET 请求。	模拟网络延迟，验证异步执行不阻塞 UI。
Task D	定时提醒	弹出系统提示框。	演示工作线程如何安全地调用主线程 UI (MessageBox)。
Task E	数据统计	生成 1000 个随机数并计算均值/方差。	输出整齐的 20列 x 50行 表格，去除冗余分割线。
Task F	系统熔断	注入致命错误 (Chaos)。	抛出 runtime_error ，触发调度器的异常捕获机制，导致系统冻结。

3.2 队列管理功能

- Revoke Selected：从等待队列中撤销选中的任务。
- Clear Queue：一键清空所有等待中的任务。
- Clear Log/Data：分别清空系统日志和数据大屏的文本内容。

3.3 故障模拟与恢复 (核心亮点)

- 冻结 (Freeze)：当 Task F 抛出异常时，调度器捕获异常并将状态标志 isFrozen 设为 true 。工作线程进入 cv.wait() 挂起状态，停止处理任何后续任务，模拟系统“死锁”或“假死”。
- 重置 (Reset)：点击 RESET 按钮调用 UnfreezeSystem() ，将 isFrozen 设为 false 并通过 cv.notify_all() 唤醒工作线程，系统恢复正常处理积压任务。

4. 程序执行流程

4.1 正常任务流程

1. 用户操作：点击 "Task A: Backup" 按钮。
2. 消息处理： WndProc 接收 WM_COMMAND ，调用 TaskFactory::CreateTask(ID_BTN_A) 。
3. 任务创建：工厂返回 TaskBackup 的智能指针。
4. 加入队列： TaskScheduler::AddTask 加锁并将任务推入 taskList ，同时通知 (notify_one) 后台线程。
5. 后台调度：
   - Worker Loop 被唤醒。
   - 检查时间戳（支持延迟/周期任务）。
   - 取出任务，执行 task->Execute() 。
6. 执行与反馈：
   - Task A 调用 PowerShell 进行磁盘操作。
   - 调用 Log() 发送 WM_UPDATE_LOG 消息给主窗口。
   - 主窗口更新 System Log 界面。

4.2 死锁与恢复流程

1. 触发故障：用戶点击 "FREEZE" (Task F)。
2. 异常抛出： TaskChaos::Execute 抛出 std::runtime_error 。
3. 异常捕获： Worker Loop 中的 try-catch 块捕获异常。
4. 进入冻结：
   - 输出 "SYSTEM FROZEN" 日志。
   - 设置 isFrozen = true 。
   - 线程进入 cv.wait(lock) ，永久挂起，不再从队列取任务。
5. 用户排队：此时用戶点击Task A, B 等，任务被加入队列，显示在Task Queue 中，但不会被执行。
6. 系统恢复：用戶点击 "RESET"。
7. 唤醒线程： UnfreezeSystem 修改 isFrozen = false 并通知后台线程。
8. 恢复运行： Worker Loop 继续循环，开始处理刚才积压的任务。

5. 关键代码逻辑说明

5.1 像素级 UI对齐算法

为了解决界面拥挤和对齐问题，采用了 底边锚定法 (Bottom Anchor Strategy)：

```
// 设定统一的底边 Y 坐标
int Y_BOTTOM = 640;

// 右侧高度倒推
int rightEditH = Y_BOTTOM - rightEditTop;
```

```
// 左侧高度分配
int availableH = Y_BOTTOM - startY - midFixedOverhead;
int hQueue = availableH / 2;
int hLog = availableH - hQueue; // 自动补齐剩余像素
```

这确保了无论窗口如何调整，System Log 和 Data Visualization 的底部实线永远在同一水平线上。

5.2 格式化输出 (Task B & E)

使用 `stringstream` 和 `iomanip` 库实现类似 Excel 的表格输出：

- `setw(N)`：设置固定宽度。
- `fixed + setprecision(1)`：控制小数位数。
- 使用 ASCII 字符 (`+`, `-`, `|`) 构建边框。

5.3 线程同步

```
// 生产者 (UI线程)
{ lock_guard<mutex> lock(listMutex); taskList.push_back(st); }
cv.notify_one();

// 消费者 (后台线程)
unique_lock<mutex> lock(listMutex);
cv.wait(lock, [this] { return (!taskList.empty() && !isFrozen) || !running; });
```

通过条件变量 `cv` 实现了高效的等待/唤醒机制，避免了轮询（Busy Waiting）带来的 CPU 浪费。

6. 环境要求与编译

- 操作系统：Windows 10 / 11 (依赖 Win32 API 和 PowerShell)。
- 编译器：MSVC (Visual Studio 2019/2022)。
- 链接库：
  - `user32.lib`
  - `gdi32.lib`
  - `comctl32.lib`
- 运行权限：由于 Task A涉及 `C:\` 和 `D:\` 根目录写入，建议以管理员身份运行，或确保对应路径有读写权限。

7. 总结

该程序不仅仅是一个任务调度演示，更是一个完整的 Windows 桌面应用程序框架。它展示了如何处理前后台交互、如何优雅地处理系统级错误（软死锁），以及如何在纯Win32API 下构建美观、对齐的复杂用户界面。