****

**实验报告：关键活动**

2351041

刘浩田

完成日期:2024.10.7

# 索引

### 概述

### 项目目的

### 项目环境

### 项目背景

### 项目要求

### 项目实现

### 项目总结

# 概述

在现代项目管理中，任务调度问题是一个核心议题，它涉及到如何在有限的资源和时间约束下，有效地安排和调度项目中的各个子任务，以确保项目的顺利完成。本实验项目的核心目标是开发一个程序，用于解决任务调度问题，并特别关注于识别和计算工程项目中的“关键活动”。关键活动是指那些对项目总工期有着直接影响的任务，它们的延迟将直接导致整个项目的延期。因此，识别这些关键活动对于项目管理和资源优化配置具有重要意义。

本项目要求程序能够接收用户输入的任务交接点数量和子任务的数量，以及每个子任务的起始交接点、结束交接点和所需时间。程序需要根据这些输入信息，判断给定的任务调度方案是否可行，并计算出完成整个项目所需的最短时间。如果调度方案可行，程序还需输出所有的关键活动，包括它们的起始和结束交接点。这不仅涉及到对任务之间依赖关系的分析，还需要对任务的执行顺序和时间进行优化计算。

在项目实施过程中，程序将采用拓扑排序算法来确定任务的执行顺序，并计算每个任务的最早开始时间和最晚完成时间。通过比较这些时间，程序能够识别出关键活动，并计算出项目的最短完成时间。关键活动的识别基于每个任务的最早开始时间和最晚完成时间是否相同，如果相同，则该任务为关键活动。此外，程序还需按照特定的顺序输出关键活动，即任务开始的交接点编号小者优先，如果起点编号相同，则按照输入时任务的逆序输出。

本项目不仅要求程序能够准确地计算和输出结果，还要求程序具备良好的用户交互界面和错误处理机制。用户交互界面需要简洁明了，能够引导用户正确输入所需信息，并在输入错误时给出清晰的提示。错误处理机制则需要确保程序在面对不合理的输入或计算错误时，能够稳定运行并给出适当的反馈。

在技术实现上，本项目将采用C++语言进行编程，利用其强大的面向对象编程特性和STL（标准模板库）来构建程序。程序将包含多个类和数据结构，如Graph类用于表示任务之间的依赖关系，MyVector和MyStack类用于实现动态数组和栈，以支持任务的存储和处理。此外，程序还将自定义PathInfo结构体来存储关键活动的路径信息，以及Edges和Arcnode结构体来表示图中的边和节点。

为了确保程序的广泛适用性，我们在设计时特别考虑了跨平台兼容性，使得该程序不仅在Windows操作系统上运行流畅，同时也兼容Linux环境，满足了不同用户群体的需求。通过使用标准C++语言和遵循Linux标准的函数，我们的程序能够在两大主流操作系统上提供一致的用户体验。

综上所述，本项目旨在通过编程实践，深入理解和掌握任务调度问题的核心概念和解决方法，同时提升编程技能和算法应用能力。通过本项目的实现，我不仅能够为工程项目管理提供有力的工具支持，还能够为后续更复杂的项目管理问题提供解决思路和方法。

# 项目目的

本实验项目的核心目的在于通过编程实践，深入理解和掌握任务调度问题的核心概念和解决方法，同时提升编程技能和算法应用能力。在项目管理领域，任务调度是一个至关重要的环节，它直接关系到项目能否在预定时间内顺利完成。通过本项目，我旨在开发一个能够判定任务调度可行性、计算项目最短完成时间和识别关键活动的程序，以辅助项目管理者进行决策和资源优化配置。

首先，实验旨在加强学生对数据结构和算法的理解和应用。任务调度问题涉及到大量的数据处理，如任务的存储、查询、修改和删除等，这些操作都需要高效的数据结构来支持。通过本实验，学生可以深入理解图、栈、队列等数据结构的特点和适用场景，掌握它们在实际问题中的应用方法。特别是在处理动态数据和复杂依赖关系时，合适的数据结构显得尤为重要。

其次，实验强调了算法的重要性。一个优秀的算法能够显著提高程序的执行效率，对于任务调度系统而言，这直接关系到用户体验和项目管理的效率。例如，在实现任务信息查询功能时，选择合适的搜索算法可以加快检索速度；在进行任务排序和关键路径计算时，合理的算法可以减少计算的复杂度。通过本实验，学生可以练习设计和优化算法，提高解决实际问题的效率。

再次，实验着重于提升学生的面向对象编程能力。面向对象编程是一种重要的编程范式，它通过将数据和操作封装成对象，提高了代码的可重用性和可维护性。在本实验中，学生需要构建多个类来表示不同的概念，如任务、交接点等，并设计类的属性和方法来实现系统的功能。这要求学生不仅要掌握面向对象编程的语法，还要学会如何运用面向对象的思想进行系统设计。

此外，实验还旨在培养学生的软件工程意识。在实际的软件开发过程中，除了编写代码，还需要进行需求分析、系统设计、测试和维护等工作。通过本实验，学生可以体验软件开发的全过程，了解各个阶段的任务和要求。特别是在系统设计阶段，学生需要考虑系统的结构、模块划分、接口设计等问题，这对于提高学生的系统分析和设计能力具有重要意义。

最后，实验还关注于提升学生的用户体验意识。一个好的软件不仅要功能强大，还要易于使用。在本实验中，学生需要设计用户友好的交互界面，确保用户能够方便地进行操作。这要求学生不仅要关注程序的内部逻辑，还要考虑用户的需求和习惯，培养从用户的角度思考问题的能力。

综上所述，本实验目的在于通过实际操作，使学生在多方面得到锻炼和提升，为将来的学习和工作打下坚实的基础。通过本实验，学生不仅能够提高自己的编程能力和系统设计能力，还能够培养良好的软件开发习惯和用户意识，为成为一名优秀的软件工程师奠定基础。通过本项目的实施，学生将能够将理论知识与实践相结合，深入理解任务调度问题，并掌握解决此类问题的关键技术和方法。这不仅能够提升学生的技术能力，还能够增强他们解决复杂问题的能力，为未来的职业生涯做好准备。

# 项目环境

## 系统：Windows11

## 编译器：VisualStudio2022x86

## 语言：C++

# 项目背景

在项目管理领域，任务调度问题是一个核心议题，它涉及到如何在有限的资源和时间约束下，有效地安排和调度项目中的各个子任务，以确保项目的顺利完成。随着信息技术的不断发展，项目管理越来越依赖于电子化、自动化的工具来提高效率和准确性。在这些子任务中，有些任务即使推迟几天完成，也不会影响全局的工期；但是有些任务必须准时完成，否则整个项目的工期就要因此而延误，这些任务被称为“关键活动”。

本项目正是在这样的需求驱动下进行的设计和实现。项目旨在为项目管理提供一个电子化的解决方案，以应对日益增长的项目调度需求，并解决传统调度方式中存在的问题。通过计算机程序自动化处理任务信息，不仅大大提高了数据处理的速度和准确性，还减少了人力资源的消耗，并提高了用户（项目管理者及团队成员）的体验。

在这样的背景下，开发一个功能完善、操作简便、安全可靠的项目调度系统变得尤为重要。本项目通过编程实现一个能够判定任务调度可行性、计算项目最短完成时间和识别关键活动的程序，旨在为项目管理者提供一个强有力的工具，以优化资源配置和提高项目执行效率。

# 项目要求

本项目旨在开发一个能够处理任务调度问题并识别关键活动的程序，该程序需要满足以下详细要求：

## 6.1 功能要求

系统需要实现以下核心功能：

输入任务信息：程序能够接收用户输入的任务交接点数量和子任务的数量，以及每个子任务的起始交接点、结束交接点和所需时间。

调度可行性判断：程序需要判断给定的任务调度方案是否可行，并在不可行的情况下输出相应的提示。

计算项目最短完成时间：如果任务调度方案可行，程序应计算并输出完成整个项目所需要的最短时间。

识别关键活动：程序应识别并输出所有关键活动，包括它们的起始和结束交接点。

排序输出关键活动：输出的关键活动应按照特定的顺序，即任务开始的交接点编号小者优先，起点编号相同时，与输入时任务的顺序相反。

## 6.2 数据结构要求

数据结构：建议使用图结构来存储任务和交接点信息，以支持高效的数据插入和删除操作。

数据完整性：确保所有任务信息的完整性和准确性，如检查必填项、格式等。

## 6.3 用户界面要求

操作简便：用户界面应直观易用，操作流程简单明了。

输入验证：系统应提供输入验证，确保用户输入的数据符合要求。

错误处理：对于用户的错误操作，系统应提供清晰的错误提示。

## 6.4 可复用性和可维护性

模块化设计：系统应采用模块化设计，便于后期维护和升级。

代码可读性：代码应具有良好的可读性，注释清晰，便于他人理解和修改。

## 6.5 测试要求

测试用例：为每个功能编写详细的测试用例，确保功能的正确性和稳定性。

边界测试：进行边界条件测试，确保系统在极端情况下也能正常工作。

通过满足以上要求，项目将能够为用户提供一个功能全面、操作简便、安全可靠的任务调度管理工具，有效提升任务调度工作的效率和质量。程序将采用C++语言进行编程，利用其强大的面向对象编程特性来构建程序。程序将包含多个类和数据结构，如Graph类用于表示任务之间的依赖关系，MyVector和MyStack类用于实现动态数组和栈，以支持任务的存储和处理。此外，程序还将自定义PathInfo结构体来存储关键活动的路径信息，以及Edges和Arcnode结构体来表示图中的边和节点。通过这些技术实现，程序将能够有效地处理任务调度问题，并为用户提供准确的项目调度分析。

# 项目实现

本实验项目的核心在于实现一个程序，该程序能够判定给定的工程项目任务调度的可行性，并在可行的情况下计算完成整个项目所需的最短时间，同时识别并输出所有的关键活动。

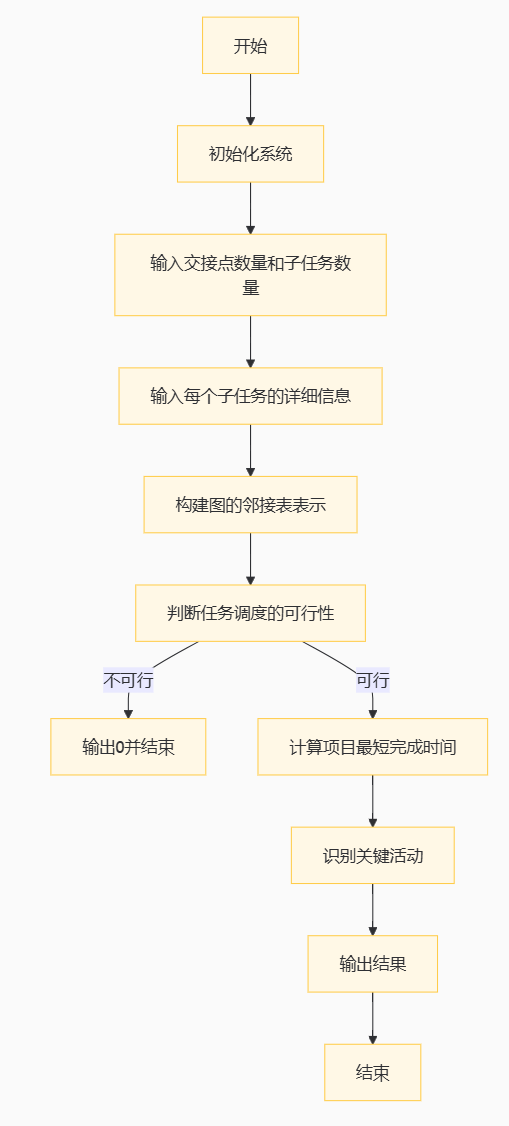


图 项目流程图

## 7.1 类设计

**MyString 类**

MyString 类用于安全地处理字符串操作，避免传统 C 风格字符串带来的潜在问题。它提供了构造函数、析构函数、以及一些基本的字符串操作功能。MyString 类的设计允许程序更加安全和方便地处理字符串数据，特别是在需要存储和操作任务名称或其他字符串数据时。

|  |
| --- |
| 附件、MyString 类 |
| class MyString {  private:  char\* data;  int length;  public:  MyString()  : data(nullptr)  , length(0)  {  }  MyString(const char\* str)  {  length = strlen(str);  data = new char[length + 1];  strcpy(data, str);  }  ~MyString()  {  delete[] data;  }  const char\* c\_str() const  {  return data;  }  int size() const  {  return length;  }  }; |

**MyStack 类**

MyStack 类是一个模板类，用于实现栈这一数据结构。它提供了 push、pop、top 和 empty 方法，用于元素的入栈、出栈、查看栈顶元素和检查栈是否为空。MyStack 类的设计使得程序能够使用栈来管理数据，特别是在需要后进先出的数据管理场景中，如在拓扑排序中管理顶点。

|  |
| --- |
| 附件、MyStack 类 |
| template <typename T>  class MyStack {  private:  MyVector<T> data;  public:  void push(const T& value)  {  data.push\_back(value);  }  T top()  {  if (data.getSize() == 0) {  throw std::out\_of\_range("Stack is empty");  }  return data[data.getSize() - 1];  }  void pop()  {  if (data.getSize() == 0) {  throw std::out\_of\_range("Stack is empty");  }  data.resize(data.getSize() - 1);  }  bool empty() const  {  return data.getSize() == 0;  }  }; |

**MyVector 类**

MyVector 类是一个模板类，用于模拟标准模板库中的 vector 容器。它提供了动态数组的功能，包括构造函数、析构函数、push\_back 方法用于添加元素，以及 operator[] 用于访问元素。此外，MyVector 还提供了 getSize 方法来获取当前容器中元素的数量，以及 resize 方法用于调整容器的大小。

|  |
| --- |
| 附件、MyVector类部分程序 |
| template <typename T>  class MyVector {  private:  T\* data;  int size;  int capacity;  public:  MyVector()  : data(nullptr)  , size(0)  , capacity(0)  {  }  MyVector(int n)    {    }  ~MyVector()  {    }  void push\_back(const T& value)  {    }  T& operator[](int index)  {    }  const T& operator[](int index) const  {    }  int getSize() const  {    }  void resize(int newSize)  {    }  T\* begin() { return data; }  T\* end() { return data + size; }  const T\* begin() const { return data; }  const T\* end() const { return data + size; }  }; |

**Graph 类**

Graph 类用于表示工程项目的任务调度图。它包含了顶点数量 vexnum、边数量 arcnum 以及一个 MyVector 类型的 vertices 容器，用于存储图中的顶点信息。Graph 类提供了 CreateGraph 方法，根据用户输入构建图的邻接表表示，并计算每个顶点的入度。此外，CriticalPath 方法用于计算项目的关键路径，即完成整个项目所需的最短时间，并识别所有关键活动。

|  |
| --- |
| 附件、Graph类部分程序 |
| struct Graph {  int vexnum;  int arcnum;  MyVector<Vnode> vertices;  Graph(int v, int a)  : vexnum(v)  , arcnum(a)  {  }  // 创建一个有向图，并根据用户输入的信息构建边  void CreateGraph(const int vexnum, const int arcnum, const int task\_num)  {    }  // 计算项目的关键路径，并返回关键路径的起始和终止节点  int CriticalPath(Graph& graph, MyVector<PathInfo>& result)  {  return max\_ve;  } |

**PathInfo 结构体**

PathInfo 结构体用于存储关键活动的路径信息，包含起始交接点 start 和结束交接点 end。这个结构体的设计使得程序能够清晰地记录和输出关键活动的路径信息，为项目管理者提供了直观的关键活动视图。

|  |
| --- |
| 附件、PathInfo 结构体 |
| struct PathInfo {  int start;  int end;  PathInfo()  : start(0)  , end(0)  {  } // 添加默认构造函数  PathInfo(int s, int e)  : start(s)  , end(e) {};  }; |

## 7.2 主函数设计

主函数 main 是程序的入口点，负责处理用户输入和显示输出结果。程序首先提示用户输入交接点数量 N 和子任务数量 M，然后读取每个子任务的起始交接点、结束交接点和所需时间。接着，程序创建 Graph 对象，并调用 CreateGraph 方法构建图。之后，程序调用 CriticalPath 方法计算项目的关键路径，并输出完成项目所需的最短时间以及所有关键活动。

以下是主函数的详细步骤：

**初始化系统**：程序启动后，首先提示用户输入交接点数量和子任务数量。

**输入任务信息**：程序读取用户输入的每个子任务的详细信息，包括起始交接点、结束交接点和所需时间，并构建图的邻接表表示。

**调度可行性判断**：程序通过计算每个顶点的入度来判断任务调度的可行性。如果存在顶点的入度在所有前驱任务完成后仍然不为零，则任务调度不可行，程序输出 0。

**计算项目最短时间**：如果任务调度可行，程序通过拓扑排序算法计算每个顶点的最早开始时间 ve 和最晚开始时间 vl。项目的最短完成时间是所有顶点的最晚开始时间中的最大值。

**识别关键活动**：程序通过比较每个边的最早开始时间和最晚开始时间来识别关键活动。如果两者相等，则该边对应的活动是关键活动。程序将这些关键活动按照规定的顺序输出。

**输出结果**：程序输出完成项目所需的最短时间以及所有关键活动的详细信息。

|  |
| --- |
| 附件、main函数 |
| int main()  {  int point\_node;  cout << "请输入任务交接点数量\n";  CheckCin(point\_node, 1, 100, 0);  int task\_num;  cout << "请输入任务数量\n";  CheckCin(task\_num, point\_node, 100);  Graph task\_graph { point\_node, task\_num };  task\_graph.CreateGraph(point\_node, task\_num, task\_num);  MyVector<PathInfo> result;  Graph result\_graph = task\_graph;  int project\_duration = result\_graph.CriticalPath(result\_graph, result);  cout << "任务完成时间为：\n";  cout << project\_duration << endl;  cout << "对应路径为：\n";  for (const auto& path : result) {  cout << path.start << " -> " << path.end << endl;  }  #ifdef \_WIN32  system("pause");  #elif defined(\_\_linux\_\_) || defined(\_\_APPLE\_\_)  cout << "按回车键继续..." << endl;  cin.get();  #endif  return 0;  } |

## 7.3 功能实现

**输入任务信息**

程序首先读取交接点数量 N 和子任务数量 M。然后，对于每个子任务，程序读取其起始交接点、结束交接点和所需时间，并构建图的邻接表表示。这一步骤是任务调度问题的核心，因为它涉及到对项目中各个任务及其依赖关系的理解和表示。

**调度可行性判断**

程序通过构建图并计算每个顶点的入度来检查任务调度的可行性。如果存在顶点的入度在所有前驱任务完成后仍然不为零，则任务调度不可行，程序输出 0。这一步骤确保了程序能够识别并提示用户关于任务调度方案的可行性问题。

**计算项目最短时间**

如果任务调度可行，程序通过拓扑排序算法计算每个顶点的最早开始时间 ve 和最晚开始时间 vl。项目的最短完成时间是所有顶点的最晚开始时间中的最大值。这一步骤是关键路径分析的核心，因为它直接关系到项目能否在预定时间内完成。

**识别关键活动**

程序通过比较每个边的最早开始时间和最晚开始时间来识别关键活动。如果两者相等，则该边对应的活动是关键活动。程序将这些关键活动按照规定的顺序输出。这一步骤为项目管理者提供了关键活动的具体信息，帮助他们优化资源配置和调整项目计划。

通过上述实现，程序能够有效地处理任务调度问题，并为用户提供关键路径分析的结果，从而辅助项目管理者进行决策和资源优化配置。

|  |
| --- |
| 附件、CriticalPath函数 |
| int CriticalPath(Graph& graph, MyVector<PathInfo>& result)  {  int n = graph.vexnum;  MyVector<int> ve(n);  MyVector<int> vl(n);  for (int i = 0; i < n; i++) {  vl[i] = numeric\_limits<int>::max();  }  MyStack<int> S;  // 拓扑排序，计算ve  for (int i = 0; i < n; i++) {  if (graph.vertices[i].indegree == 0) {  S.push(i);  }  }  while (!S.empty()) {  int v = S.top();  S.pop();  for (Arcnode\* p = graph.vertices[v].firstarc; p; p = p->nextarc) {  int k = p->adjvex - 1;  if (--graph.vertices[k].indegree == 0) {  S.push(k);  }  if (ve[v] + p->Weight.weight > ve[k]) {  ve[k] = ve[v] + p->Weight.weight;  }  }  }  // 计算vl  int max\_ve = ve[0];  for (int i = 1; i < n; i++) {  if (ve[i] > max\_ve) {  max\_ve = ve[i];  }  }  for (int i = 0; i < n; i++) {  vl[i] = max\_ve;  }  for (int i = n - 1; i >= 0; i--) {  for (Arcnode\* p = graph.vertices[i].firstarc; p; p = p->nextarc) {  int k = p->adjvex - 1;  if (vl[k] - p->Weight.weight < vl[i]) {  vl[i] = vl[k] - p->Weight.weight;  }  }  }  // 计算关键活动  for (int i = 0; i < n; i++) {  for (Arcnode\* p = graph.vertices[i].firstarc; p; p = p->nextarc) {  int k = p->adjvex - 1;  int ee = ve[i];  int el = vl[k] - p->Weight.weight;  if (ee == el) {  result.push\_back(PathInfo(i + 1, k + 1));  }  }  }  return max\_ve;  } |



# 项目总结

在本次实验项目中，我成功地设计并实现了一个功能完备的任务调度系统，旨在解决工程项目中的关键路径问题。通过这个项目，我不仅巩固了对数据结构和算法的理解，还提升了面向对象编程和系统设计的能力。以下是对整个项目的总结：

## 8.1 学习体会

理论与实践相结合：通过将课堂上学到的数据结构和算法知识应用到实际项目中，我深刻体会到了理论知识在解决实际问题中的重要性。在项目实施过程中，我发现理论知识为我提供了解决问题的基本框架和方法，而实际操作则让我更加深入地理解了这些理论的实际应用。

面向对象编程：项目让我更加熟悉了面向对象的设计原则，如封装、继承和多态。通过定义清晰的类和对象，我提高了代码的可读性和可维护性。在项目中，我使用了多个类来表示不同的概念，如任务、交接点等，并设计类的属性和方法来实现系统的功能。这要求我不仅要掌握面向对象编程的语法，还要学会如何运用面向对象的思想进行系统设计。

系统设计经验：在项目开发过程中，我学习了如何规划和设计一个完整的系统，包括需求分析、系统架构设计、模块划分等。特别是在系统设计阶段，我需要考虑系统的结构、模块划分、接口设计等问题，这对于提高我的系统分析和设计能力具有重要意义。

## 8.2 技术收获

C++编程技能：通过本项目，我提高了C++编程技能，特别是在STL的使用、异常处理和输入输出操作等方面。我学会了如何利用C++的强大功能来构建复杂的系统，并解决了实际开发中遇到的各种问题。

数据结构应用：我深入理解了链表、栈等数据结构的特点和适用场景，掌握了它们在实际问题中的应用方法。特别是在处理动态数据和复杂依赖关系时，合适的数据结构显得尤为重要。

算法实现：项目中实现了多种算法，如拓扑排序、关键路径计算等，提高了我的算法设计和优化能力。我学会了如何选择合适的算法来解决特定的问题，并不断优化算法以提高程序的执行效率。

## 8.3 问题与挑战

性能优化：在处理大量数据时，如何优化算法和数据结构以提高系统性能是一个挑战。我通过优化链表操作和算法逻辑来解决这一问题，例如，通过减少不必要的数据复制和优化循环结构来提高效率。

错误处理：在用户输入错误或系统异常情况下，如何保证系统的健壮性是一个难题。我通过增加输入验证和异常捕获机制来提高系统的稳定性，确保程序在面对不合理的输入或计算错误时，能够稳定运行并给出适当的反馈。

用户界面设计：设计一个用户友好的界面也是项目中的一个挑战。我通过不断迭代和改进用户界面，使其更加直观和易用。我学会了如何从用户的角度思考问题，并设计出简洁明了的用户交互界面。

## 8.4 改进方向

功能扩展：未来可以增加更多功能，如数据导出、数据导入、成绩查询等，使其更符合实际应用需求。这些功能的添加将使系统更加完善，更好地服务于用户。

性能提升：可以探索更高效的数据结构和算法，进一步提高系统的处理速度和响应能力。例如，使用更高效的图遍历算法或优化内存管理策略。

用户体验优化：可以进一步优化用户界面和交互设计，提供更加友好和直观的用户体验。例如，通过改进布局和增加交互提示来提高用户的满意度。

安全性增强：考虑到系统处理的是个人信息，未来可以加入更多的安全特性，如数据加密、访问控制等，以保护用户数据的安全。

## 8.5 结论

这次课程设计是一次宝贵的学习经历。它不仅让我将所学知识应用于实践，还让我学会了如何分析和解决实际问题。我克服了各种挑战，最终完成了一个功能完善、性能良好的任务调度系统。我相信，这次经历将对我未来的学习和工作产生积极的影响。通过本项目的实施，我不仅提高了自己的编程能力和系统设计能力，还培养了良好的软件开发习惯和用户意识，为成为一名优秀的软件工程师奠定了基础。