# 摘　　要

在线评测系统（Online Judge System，一般简称 OJ），是一个为程序设计竞赛爱好者和 ACM/ICPC（国际大学生程序设计竞赛）训练队伍提供训练及交流的平台，同时也可以为程序设计语言、数据结构、算法等课程提供作业练习、实现及检测的平台。目前的在线评测系统一般采用单服务器模式，在同时服务大量用户时由于系统效能不足而进入无法处理用户请求的不利状况。

本课题将在武汉大学已有的在线评测系统上进行全面重构，将原始的单服务器模式转换为由一个中心服务模块作为系统核心组件，一个底层数据库服务模块提供数据支持，多个 Web 服务模块和多个评测判题服务模块组成的服务集群，本集群中的任何一个部件均可部署在不同的服务器上，且负载高峰期可以用普通 PC 临时替代，由局域网或高速 Internet 负责连接，从而大大减轻单个服务器的压力并提升整个集群的服务性能。在提供服务集群的同时本课题还提供了评测系统的客户端软件，即在实现 B/S 架构的同时也实现了 C/S 架构，让用户在网络不稳定的情况下也能通过对网络状况需求相对较低客户端一样进行练习。

本文所讨论的是本课题的判题服务模块，即整个系统的内核模块。本文从在线评测系统的内核模块的处理流程，Linux 操作系统相关资源限制及信息获取，设计实现所需的大量繁杂背景知识介绍开始展开，详细讨论了在 Linux 中内核模块的设计与实现的过程，并就内核模块内各功能子模块的关键点进行了仔细的分析，最后对系统的安全保障提出了相对全面的需求并给出目前所使用的以 Linux 操作系统自身安全机制和 ptrace 监控为主的对应解决方案。

本课题的系统实现已经大致完成，并在实验环境中稳定运行，为本课题提供了充分的实践支持，验证了本文的理论基础和研究成果。

**关键字：** 在线评测系统；评测内核；分布式部署；系统安全

# Abstract

Online Judge System (OJ) is a training platform for programming lovers and the members of ACM/ICPC (International Collegiate Programming Contest) training team, and it can be used as a homework test platform for the courses like Programming Language, Data Structures, Algorithms or other courses which need students to write programs to solve their homework. The OJs on using generally deployed on a single server, they always breakdown when a large number request arrival on the same time.

This subject is based on the oak (the current Online Judge System of Wuhan University), converts the single-server mode to a services cluster. The services cluster includes a single Core Manage module, a single DBMS module, a number of Web Service modules and a number of Judge Kernel modules. All the modules can be distributed deployed on any server, connected by high-speed network, or use a large number of PC instead server on peak load. Without the services cluster, we afford a Client software for users who need to use our system in a horrible network environment. We implemented both B/S structure on the services cluster side and C/S structure on the Client software side.

This article is discussed in the issue of Judge Services, the Judge Kernel module in the whole system. We started on the processes of Judge Kernel, resource constraints of Linux and a number of related background knowledge which are used for develop on Linux in this paper, then discussed the implement details on the design and develop periods, and get deeply on many key points, at last we discussed the system security problem with the current solutions which use the self-safe techniques of Linux and monitor of ptrace function.

The system implemented this paper is nearly completed now, and running stably on the experimental environment for a long time, it strongly supports the theories appeared this paper and the results of our research.

**Keywords:** Online Judge System; Judge Kernel; Distributed Deploy; System Security

# 目　　录

# 第1章　　引言

## 1.1 课题背景及意义

在线评测系统（Online Judge System，一般简称 OJ），是一个为程序设计竞赛爱好者和 ACM/ICPC（国际大学生程序设计竞赛）训练队伍提供训练及交流的平台，同时也可以为程序设计语言、数据结构、算法等课程提供作业练习、实现及检测的平台。

对国内计算机专业培养的学生理论功底一般会很好，但实践能力总是略显不足，从而无法在科研学术领域及工业界有较强的竞争能力这一现象，一个合理的在线评测系统的搭建正是为了培养学生的实际动手能力和提高教学辅导的工作效率来扭转这一问题。而武汉大学目前已稳定运行两年多的既有在线评测系统很好的达到了这个目标。

但随着参加 ACM/ICPC 及有课程实践需要的同学越来越多，包括武汉大学在内的国内外众多的在线评测系统所采用的所有模块均部署在单服务器上的模式在同时服务大量用户时已暴露出大量的问题。具体表现在 Web 服务的响应速度无法跟上用户的大面积访问需求，以及判题服务的响应时间慢于用户提交新程序的时间，从而让整个在线评测系统进入无法提供服务给新登录用户，且对已登录用户的服务也会无法得到及时响应的恶性循环。近两年国内 ACM/ICPC 举行的亚洲区各赛点的网络预选赛充分暴露了这一问题。

本课题将在武汉大学已有的在线评测系统上进行全面重构，将原始的单服务器模式转换为由一个中心服务模块作为系统核心组件，一个底层数据库服务模块提供数据支持，多个 Web 服务模块和多个评测判题服务模块组成的服务集群，本集群中的任何一个部件均可部署在不同的服务器上，且负载高峰期可以用普通 PC 临时替代，由局域网或高速 Internet 负责连接，从而大大减轻单个服务器的压力并提升整个集群的服务性能。在提供服务集群的同时本课题还提供了评测系统的客户端软件，即在实现 B/S 架构的同时也实现了 C/S 架构，让用户在网络不稳定的情况下也能通过对网络状况需求相对较低客户端一样进行练习。

本文所讨论的是本课题的判题服务模块，即整个系统的内核模块。

## 1.2 课题需解决的问题

作为一个在线评测系统，核心的功能是要保证本系统能够在保证宿主机安全的前提下，对用户提交的程序进行实时编译、运行，并评判最后的结果，最后将精准获取的运行耗费时间、内存及运行结果或错误信息返回给提交程序的用户。

为了实现能监控用户程序所耗费的时间、内存，可以通过读取用户程序的进程在系统中的相关记录来获取，或者直接运行一个守护进程来统计该用户程序的运行时间，并通过不断查看用户程序的内存使用情况来获取用户程序的内存使用。考虑到对用户程序的干扰程度，通过比较系统中能看到记录了和程序所耗费的时间和内存有关的所有数据来源，最后采用的是通过实时读取 Linux 系统中/proc/伪文件目录下相关信息和在关键系统调用时检测来实现时间和内存的获取。

对用户程序的结果判断及错误信息处理相对容易，通过文件对比、程序智能对比、判断系统信号量和错误输出重定向等方式即可获取用户程序的结果和错误信息。

对内核最重要的是要保证判题服务模块的宿主机不被用户程序的异常所破坏或者被恶意破坏者的恶意代码所攻击，只有保证了宿主机的安全才可以保证能准确编译运行用户程序

并获取正确结果并提供安全稳定的持续服务。对这一问题的结局方案，最直观也是最安全的即是将用户程序严格限制在某一运行空间中，包括对内存的限制、对文件系统访问及修改权限的限制以及对相关系统库函数调用的限制。为实现这一目标，一般的限制可直接通过系统默认支持的对指定进程的资源的限制函数来进行，而对系统默认支持外相对高端的如对系统相关文件系统的访问及修改权限或底层系统调用的限制，本课题考虑对 Linux 系统进行内核编程来实现，在比较了使用需求和实现难易度后，最后选择了使用 ptrace 来进行监控。

## 1.3 课题取得的成果

本课题所实现的可分布式部署在线评测系统目前已在实验室内网的临时服务集群上稳定运行超过两个星期，为整个武汉大学 ACM/ICPC 集训队提供了练习、比赛的平台。在结合其他高校的经验和教训基础上，对在线评测系统提出了更符合队伍训练及教学练习发展方向和更好提升用户体验的需求，并对核心功能做了深入的探讨并实现了需求中提出的所有核心功能及提升用户体验的绝大部分功能。

## 1.4 本文的组织

因为在线评测系统本身是一个比较复杂的系统，加上本系统要求可分布式部署，虽然本文只讨论内核的设计及实现，单本文所覆盖的内容和知识体系还是不可避免的会比较复杂。为简化阅读，现将整篇论文各章节的结构介绍如下：

第一章为引言，本章介绍论文选题的课题背景、课题意义、课题所需要解决的问题、课题实际上取得的成果及本论文的结构安排。

第二章为在线评测系统内核设计背景知识，本章从评测系统对用户程序的编译、运行及最后判断的标准，Linux 及其内核编程及本模块实现的灵魂部分 ptrace 等方面介绍了本系统内核设计实现所需的相关背景知识。

第三章为在线评测系统内核的设计与实现，本章是论文的重点部分，在从全局角度介绍了本系统内核的大体规划和功能划分后，详细介绍了各功能模块的设计实现过程。

第四章是在线评测系统内核的安全机制，本章讨论了本系统内核在编译运行用户程序时该考虑的安全机制问题，并说明介绍当前本系统所采用的实现机制。

第五章是总结和展望，本章对整篇论文及系统进行总结，阐述课题的研究成果和意义，并就其他需要解决的问题及优化发展方向给出建议。

# 第2章　在线评测系统内核设计背景知识

由于在线评测系统已经是一个成熟的概念，且目前国内外已经有大量的在线评测系统实现，故本章首先介绍在线评测系统的相关背景知识，然后介绍本在线评测系统开发和运行的宿主机操作系统 Linux 及在 Linux 上进行开发所需相关背景知识。

## 2.1 在线评测系统简介

在线评测系统（Online Judge System，一般简称 OJ），是一个为程序设计竞赛爱好者和 ACM/ICPC（国际大学生程序设计竞赛）训练队伍提供训练及交流的平台，同时也可以为程序设计语言、数据结构、算法等课程提供作业练习、实现及检测的平台。

使用在线评测系统的用户首先阅读其欲解决的问题描述、问题对应的输入输出格式及数据描述，自行在任意场合用在线评测系统支持的程序设计语言来尝试解决此问题。用户将自己确定无误的源代码提交给在线评测系统，获取评测结果，如果结果正确则可以继续尝试解决更多的问题，或在得到相关错误信息后修改自己的源代码以继续尝试解决当前问题。

### 2.1.1 在线评测系统对用户程序的处理流程

由于本文只讨论在线评测系统的内核模块，所以在此处的处理流程只涉及用户程序在内核模块中所要经历的处理流程。

用户程序在评测内核中所要经历的处理过程为编译、运行和对比三大过程。首先由内核模块通过调用相应的编译器将用户程序编译成可执行文件（解释型的程序设计语言免去这一步，如 Java），然后调用程序所对应的标准输入做输入，在对应的资源限制下运行用户程序可执行文件，获得用户程序输出，最后对比用户程序输出和对应的标准输出得到结果。在这一系列过程中，只要期间发生任何异常，评测系统内核模块均会终止后续操作并返回相应错误结果，包括在编译阶段的编译错误，运行阶段的运行错误或者超过对应资源限制。如果在期间没有异常产生，则按照 2.1.2 节中的判断机制返回用户程序的最终评测结果。

在后续章节的图 3-1 在线评测系统的内核模块工作流程图给出了一个更直观的处理流程描述。

### 2.1.2 在线评测系统对用户程序输出的判断标准

一般情况下，在线评测系统通过对比用户程序输出和对应的标准输出两个文本文件的相似度来判断用户程序是否给出了正确的输出，即得到正确结果。

为简化处理流程及增大对用户的挑战性，在评测系统中将相似度只分成三个层次，即完全一致，格式错误和不一致。完全一致要求两个文本文件内所有字符均一致。格式错误是指如果两个文件不完全一致，但是在去除两个文件的所有分隔符（如空格，制表符及换行符等）后所有字符均一致。如果两个文本文件，不完全一致且不是格式错误，均判为不一致。

在评测内核中，只有完全一致的比较结果被认为是正确的结果。

### 2.1.3 在线评测系统返回结果状态

用户提交自己的程序给在线评测系统，在系统的评测过程中和评测结束后，可以得到自己所提交程序的结果，所有结果、结果简写及导致该结果可能的原因如下表 2-1 所示：

表 2-1 在线评测系统返回结果列表

```
ACCEPTED                // 1. AC, 结果正确
WRONG_ANSWER            // 2. WA, 结果错误
PRESENTATION_ERROR      // 3. PE, 结果格式错误
COMPILE_ERROR           // 4. CE, 编译错误, 并有附带错误信息
TIME_LIMIT_EXCEEDED     // 5. TLE, 运行时超过时间限制
MEMORY_LIMIT_EXCEEDED   // 6. MLE, 运行时超过内存限制
OUTPUT_LIMIT_EXCEEDED   // 7. OLE, 输出文件大小超限制
RUNTIME_ERROR_SIGSEGV   // 8. RE, 运行时错误, 堆栈溢出或非法文件访问
RUNTIME_ERROR_SIGFPE    // 9. RE, 运行时错误, 除零
RUNTIME_ERROR_SIGBUS    // 10. RE, 运行时错误, 系统总线异常
RUNTIME_ERROR_SIGABRT   // 11. RE, 运行时错误, 程序被异常终止
RUNTIME_ERROR_JAVA      // 12. RE, Java 程序运行时错误, 并有附带错误信息
RESTRICTED_FUNCTION     // 13. RF, 被禁止函数, 如非法系统调用等
SYSTEM_ERROR            // 14. SE, 系统未知错误
PENDING                 // 15. 该提交请求目前在判题队列中
COMPILING               // 16. 该提交请求目前正在编译用户源代码
RUNNING                 // 17. 该提交请求目前正在运行用户程序
JUDGING                 // 18. 该提交请求目前正被对比结果
```

其中前 14 个均为终止结果状态，15-18 为中间结果状态，即中间结果会随着在线评测系统对用户程序的处理流程进行而改变，最终结果会是终止结果状态中的一种。各项被简写为 RE 的结果状态在某些在线评测系统将不进行区分，统一作为 RUNTIME_ERROR 返回给用户。

只有 Accepted 一种终止结果状态表示用户的程序完全正确，其他的终止状态都是由于用户程序有各种问题而无法得到正确的结果。

### 2.1.4 在线评测系统的安全问题

作为一个评测系统，在执行用户程序的时候，要考虑用户程序刻意或者无意中对系统造成的安全隐患，包括打开和访问的文件、运行时间、使用内存、输出大小、工作文件夹、运行的用户和用户组、进程数等问题。因为其中的任何一个非法操作均有可能破坏在线评测系统的内核或其他模块，甚至破坏在线评测系统的宿主操作系统，从而带来无法估量的损失。

一般的解决方案是想要尽可能的构造一个让用户程序运行的沙盒(SandBox)，让用户程序在这个可控的沙盒限制区域内运行而不超出这一区域的时间空间限制，且最好在这一过程中能对用户程序进行监控，防止不可知的破坏产生。

## 2.2 Linux 及相关背景知识

Linux，是一种基于 Unix 操作系统而来的完全免费和开源的计算机操作系统。在经过十多年的发展后，现在 Linux 内核支持从个人电脑到大型主机甚至包括嵌入式系统在内的各种硬件设备，和其他的商用 Unix 系统以及微软 Windows 相比，作为自由软件的 Linux 具有低成本、安全性高、更加可信赖的优势。

### 2.2.1 在 Linux 上进行程序编译

Linux 中默认的编译器为 GCC，是致力于开发自由并且完整的类 Unix 操作系统，包括软件开发工具和各种应用程序的 GNU 计划的一部分。最初 GCC 只是表示 GNU C 编译器（GNU C Compiler），而发展到现在，GCC 的含义扩展到了 GNU 编译器集合(GNU Compiler Collection)，其中包括目前广泛使用的 gcc(C 编译器)、g++(C++编译器)、gpc(Pascal 编译器)和 gcj(Java 编译器) 等。

在使用 GCC 编译程序时，编译可以被细分为四个阶段：

**1. 预编译 preprocess**

源代码中的预编译指示以“#”为前缀。可以通过在 gcc 后加上-E 选项来调用预编译器。预编译的过程通过完成三个主要任务给了代码很大的灵活性。

(1) 将“include”的文件拷贝到编译的源文件中

(2) 用实际值替代“define”的文本

(3) 在调用宏的地方进行宏替换

**2. 编译 compile**

作为一个中间步骤，gcc 把源代码翻译成汇编语言。这是一项极其复杂的工作，包括词法分析、语法分析、中间代码产生、编译优化和汇编代码生成。人们有时会把这一步误解为整个过程，但是，实际上还有许多工作要 gcc 去做。

**3. 汇编 assemble**

GCC 通过 as 把汇编语言代码转换为目标代码。事实上，目标代码并不能在 CPU 上运行。编译器选项-c 把.c 文件转换为以..o 为扩展名的目标文件。

**4. 连接 link**

连接器 1d 使用命令，接受前面由 as 创建的目标文件并把它转换为可执行文件。

上述过程特指 C 语言的编译器，事实上对 C++和 Pascal，整个过程大同小异。在编译的过程中，通过对编译参数的选择，GCC 还能支持诸如编译器优化级别、标准兼容、错误信息级别控制等功能。

本系统在处理用户提交的程序时，采用的即是 GCC 来对用户提交的各种源代码（包括 C、C++、Pascal，在未来还将提供对 Java 的支持）进行编译，最终运行可执行文件来判断用户程序的正确与否。另外，本系统的内核模块及中心服务器模块的所有 C/C++源代码也是由 GCC 来完成编译连接成可执行文件的。

### 2.2.2 在 Linux 中创建新进程并执行相应程序

Linux 作为一个稳定的操作系统，拥有一系列创建、执行、终止和等待终止等完善的进程控制原语。在 Linux 中，通过 fork 创建新进程、exec 执行新程序、exit 和 wait 处理终止和等待终止，其中的每个操作都是基于一个或多个底层系统调用的多个函数，且有相关的宏定义完善，它们共同构成了 Linux 完善的进程控制原语。

Linux 中有两个创建进程的系统调用：fork()和 clone()。由于 clone()系统调用是用户创建新线程而使用，所以我们采用 fork()，这是一个在类 Unix 操作系统中都会提供的一个底层系统调用，该函数所处的头文件和函数原型为：

```
#include <unistd.h>
pid_t fork(void);
```

一个现有进程可以调用 fork 函数创建一个新进程，这个新进程称之为调用进程的子进程。fork 函数被调用一次，但是返回两次。两次返回的唯一区别是子进程的返回值是 0，而

父进程的返回值则是新的子进程的进程 ID。子进程和父进程继续执行 fork 调用后的指令。子进程是父进程的副本。例如，子进程获得父进程数据空间、堆和栈的副本，且这只是子进程所拥有的副本，父进程和子进程并不共享这些存储部分，父进程和子进程共享正文段。

fork 完成后父进程和子进程将都执行自己的后续语句，子进程往往会调用 exec 系列函数来执行另一个程序。当进程调用一种 exec 函数时，该进程执行的程序完全替换为新程序，而新程序将从其 main 函数开始执行。因为调用 exec 并不创建新进程，所以前后的进程 ID 并不改变。exec 只是用一个全新的程序替换了当前进程的正文、数据、堆和栈段。

Linux 一共提供了 6 种不同的 exec 函数来让我们使用，这些函数所处的头文件和函数原型分别为：

```
#include <unistd.h>
int execl(const char *pathname, const char *arg0, … /* (char *)0 */ );
int execv(const char *pathname, char *const argv[]);
int execle(const char *pathname, const char *arg0, …
           /* (char *)0, char *const envp[] */);
int execve(const char *pathname, char *const argv[], char *const envp[]);
int execlp(const char *filename, const char *arg0, … /* (char *)0 */);
int execvp(const char *filename, char *const argv[]);
```

这六个函数都具有相同的功能，均为调用第一个参数给出的新程序文件，并按照后续参数给出的参数表来执行新程序文件。这些函数的区别只是对参数的选择上，限于篇幅问题，本文在这里不详细介绍，读者可以在相关书籍资料中找到详细的参数说明。如果函数调用成功，将没有任何返回，因为它们调用的程序不再运行。如果函数调用失败，则返回-1，同时将错误代码保持在 errno 中，与其他系统调用一样。

在进程创建后，还需要用 exit 来终止进程，一般这个操作将由进程创建后转换的用户程序来终止，或者发生异常而被系统终止。

进程终止后，需要父进程通过 wait 来等待终止，获取相关的退出信息，调用系统定义好的宏来查看退出信息的具体意思。

### 2.2.3 在 Linux 中设定进程资源限制

在 Linux 操作系统中，可以通过 setrlimit 函数对每个进程都有的一组资源限制进行更改，该函数所处头文件和函数原型为：

```
#include <sys/resource.h>
int setrlimit(int resource, const struct rlimit *rlptr);
```

对该函数的每次调用都会指定一个资源和一个指向下列结构的指针：

```
struct rlimit {
  rlim_t rlim_cur;  // 软限制，当前的限制
  rlim_t rlim_max;  // 硬限制，软限制所能达到的最大值
}
```

setrlimit 函数中的 resource 参数部分取值参考下表，由于本在线评测系统并没有用到所有的取值，故此表只列出当前系统用到了的。

表 2-2 setlimit 函数资源限制部分选项

```
RLIMIT_CPU      // CPU 时间的最大量值(秒)，超过时向该进程发送 SIGXCPU 信号
RLIMIT_DATA     // 数据段的最大字节长度
RLIMIT_FSIZE    // 可创建的文件的最大字节长度，超过时向该进程发送 SIGXFSZ 信号
RLIMIT_NOFILE   // 可打开的最大文件数
RLIMIT_NOPROC   // 可拥有的最大子进程数
```

除 setrlimit 函数外，限制程序运行时间的还有 alarm 函数。两者区别在于 setlimit 设置

的是 CPU 时间，而 alarm 函数设置的是限制用户程序能运行的最大墙上时钟时间，当超过所对应的时间后，两个函数都会向该进程发送 SIGXCPU 信号。

文件系统的权限限制可以通过 Linux 系统本身的安全机制来进行，即让用户程序以较低权限用户和用户组运行，通过 setuid 和 setgid 函数可以改变程序运行使用 uid 和 gid。对工作目录的更改可以通过 chdir 函数来进行，也可以在执行 setuid 和 setgid 之前，用超级用户运行父进程，通过执行 chroot 来实现。函数原型和具体实现此处略，读者可以查看相应书籍资料来获取。

## 2.3 ptrace 的介绍

ptrace 是一个可以使父进程在用户层拦截和修改系统调用的函数，可以监控和控制其他进程，该函数还能够改变子进程中的寄存器和内核映像，因而可以实现断点调试和系统调用的跟踪。

### 2.3.1 Linux 内核编程及对系统调用的监控

内核，是一个操作系统的核心。它负责管理系统的进程、内存、设备驱动程序、文件和网络系统，决定着系统的性能和稳定性。Linux 的一个重要的特点就是其源代码的公开性，所有的内核源程序都可以在相关目录下找到，大部分应用软件也都是遵循 GPL 而设计的，都可以获取相应的源程序代码。全世界任何一个软件工程师都可以将自己认为优秀的代码加入到其中，由此引发的一个明显的好处就是 Linux 修补漏洞的快速以及对最新软件技术的利用。而 Linux 的内核则是这些特点的最直接的代表。

拥有了内核的源程序可以让我们了解系统是如何工作的，通过通读源代码，我们就可以了解系统的工作原理。其次，我们可以针对自己的情况，量体裁衣，定制适合自己的系统。再次，如果需要增加对某部分功能的支持，除了把相应部分编译到内核中(build-in)外，也可以将该部分来编译成模块(module)，在需要使用的时候动态调用，这样可以很方便快捷的发布和使用。由于 Linux 的源程序是完全公开的，任何人只要遵循 GPL，就可以对内核加以修改并发布给他人使用。

所有的操作系统都提供多种服务的入口点，由此程序向内核请求服务。各种版本的 Linux 都提供经过良好定义的有限数目的入口点，经过这些入口点进入内核，这些入口点被称为系统调用(system call)。系统调用是不能更改的一种 Linux 特征。Linux 所使用的技术是为每个系统调用在标准 C 库中设置一个具有同样名字的函数。用户进程用标准 C 调用序列来调用这些函数，然后，函数又用系统所要求的技术调用相应的内核服务。例如函数可将一个或多个 C 参数送入通用寄存器，然后执行某个产生软中断进入内核的机器指令。从应用角度考虑，可将系统调用视作为 C 函数，从而实现了 Unix 操作系统的内核 C 编程。

在 Linux 2.4 及更早的内核版本中，系统均只支持 INT 0x80 来完成系统调用，所以监控模块只需要检测到这一系统调用就可以完成对程序进行监控，来达到保护系统安全的功能。但是在 Linux 2.6 版本内核开始，内核新加入了对 sysenter/sysexit(Intel P6+架构)和 vsyscall(AMD 架构) 来完成系统调用的支持，从而使得要想通过拦截系统调用来达到安全监测的目的会更复杂。

### 2.3.2 ptrace 简介

在 2.3.1 节中我们介绍了 Linux 内核编程，通过改变系统调用的参数，我们可以拦截系统调用，从而对用户的程序进行全面监控，并对操作系统自身进行保护。事实上 Linux 中已经提供了一个可以在用户层拦截和修改系统调用的函数：ptrace。ptrace 提供了一种使父进程得以监控和控制其他进程的方式，该函数还能够改变子进程中的寄存器和内核映像，因而可以实现断点调式和系统调用的跟踪。

操作系统提供了一种标准的服务来让程序员实现对底层硬件和服务的控制(比如文件系统)，叫做系统调用（system calls)。当一个程序需要做系统调用的时候，该程序将相关参数放进系统调用的相关寄存器，然后调用软中断 0x80（在 Windows XP 及 Linux 2.6 版本内核开始加入了对 sysenter/sysexit 和 vsyscall 的支持)，这个中断就像一个让程序得以接触到内核模式的窗口，程序将参数和系统调用号交给内核，内核来完成系统调用的执行。

在执行系统调用前，内核会先检查当前进程是否处于被“跟踪”（traced）状态，如果是被跟踪的，内核会暂停当前进程并将控制权交给跟踪进程，使跟踪进程得以查看或者修改被跟踪进程的寄存器。这就是 ptrace 进行监控的时候，我们可以在这时候通过 ptrace，选择合理的调用参数来获取当前寄存器的内容，从而判断用户程序想进行什么样的系统调用，继而决定是否给用户程序以进行次系统调用的权限。

ptrace 函数的定义是：

```
long ptrace(enum __ptrace_request request,
            pid_t pid,
            void *addr,
            void *data);
```

第一个参数即 ptrace 的行为选择与决定其他参数用法的枚举量，常用的取值及其参数的中文大意如下表 2-3 所示：

表 2-3 ptrace 行为取值表

```
PTRACE_TRACEME      // 通知调用跟踪
PTRACE_PEEKTEXT     // 获取用户程序正文段内容
PTRACE_PEEKDATA     // 获取用户程序数据段内容
PTRACE_PEEKUSER     // 获取用户程序用户空间数据
PTRACE_POKETEXT     // 设置用户程序正文段内容
PTRACE_POKEDATA     // 设置用户程序数据段内容
PTRACE_POKEUSER     // 设置用户程序用户空间内容
PTRACE_GETREGS      // 获取用户程序所有通用寄存器内容
PTRACE_GETFPREGS    // 获取用户程序所有浮点运算寄存器内容
PTRACE_SETREGS      // 设置用户程序所有通用寄存器内容
PTRACE_SETFPREGS    // 设置用户程序所有浮点运算寄存器内容
PTRACE_CONT         // 继续被监控的用户程序
PTRACE_KILL         // 杀死被监控的用户程序
PTRACE_SYSCALL      // 继续被监控的用户程序，并在下次系统调用时暂停
```

从上述行为取值可以看出，ptrace 已经能够获取并设置非常多的系统内存或寄存器内容，从而达到监控程序运行并在发生意外情况时及时终止的功能。

# 第3章　在线评测系统内核的设计和实现

在线评测系统是一个复杂、综合的系统，本系统内核模块的核心功能就是在线编译并运行用户提交的程序，并且评测其正确性，最后能迅速准确的得出用户程序的运行结果。而且在此过程中，要考虑到用户程序运行时可能出现的各种异常情况，或者是用户提交的恶意代码对评测系统或者评测系统所在宿主机的破坏。所以，本在线评测系统的内核模块设计要求对 Linux 系统内核编程和操作系统相关知识有相当好的综合运用能力。

在本课题所设计的可分布式部署的在线评测系统中，内核模块是作为一个相对独立的模块存在，无论系统最后会应用于什么领域，由本系统的中心服务模块发送过来的评测请求均为经过了严格的数据封装后的标准评测请求，这样降低了评测内核模块的设计复杂度，但同时也对内核模块的效率和严谨性提出了更高的要求。因此，在对本系统内核模块进行开发的过程中，需要精心基于在线评测系统的目标、内容、规模、现有的配置等具体情况，对内核模块工作流程做出相应的整体规划和功能划分，以方便后续的具体设计和实现。

本章第一节将进行内核模块的总体规划和功能划分，后几节将按照此规划和功能划分依次给出每个具体功能子模块的设计和实现思路。由于本系统代码量非常大，本文讨论期间将尽可能用流程图、伪代码或逻辑描述来说明是各个模块是如何设计实现的，有涉及具体实现细节的请参见附录或本系统源代码中的相关文件。

## 3.1 在线评测系统内核模块的总体规划及功能划分

目前国内外已经存在非常多的在线评测系统，比如几乎是目前所有在线评测系统鼻祖、汇集全球程序设计竞赛爱好者于一地的 UVA Online Judge，或者是国内甚至是亚洲参与选手最多的北京大学在线评测系统，抑或其他任何一个大学都可能存在的在线评测系统，它们的功能和设计模式都是非常相似的，那就是收集题目，然后上传到该在线评测系统，由用户针对该系统中特定的题目做出合适的解答，最后将程序代码提交至在线评测系统获取改解答代码的运行结果。但是这些在线评测系统大多都有一个致命的缺陷，就是整个系统的所有模块都高度耦合，当最终部署于单服务器上的设计在面对突发性大规模的用户时，会导致整个系统瘫痪，并且非常不利于系统投入使用后的内核维护，因为每一次修正都可能是对整个系统的全面改动。

鉴于上述情况，为了避免重复发明轮子的无创新无谓劳动，本课题提出的要求是能让整个在线评测系统的每个模块都尽可能在一个很容易扩展的自定义协议通信的前提下保持单个模块的独立性。本课题中所提到的可分布式部署的精髓也正是各模块之间高度的独立，因为一旦能把每个模块都独立出来，则每个模块都可以部署到不同的服务器上。在服务压力非常重的时候，甚至是可以动用实验室为数众多的普通 PC 来扮演评测内核模块的宿主机的角色，因为最终用户也是用的 PC 机在进行练习和比赛，对所有用户来说反而是更公平的一种安排。一个独立的内核模块，是非常容易进行维护的，因为只要保证和中心服务模块的协议不改变，内部的监控方法或其他的规则可以随时修改，且工作量非常少。所有与用户交互的功能，均由中心服务模块完成，包括一般设计模式下的用户提交，将由系统的 Web 服务模块或者用户客户端传输到中心服务模块后，由中心服务模块封装成统一格式后传送给内核模块，而内核模块所有的返回，均是传输到中心服务模块，最后由中心服务模块分发给相应的模块展示给用户。所以，内核模块可以不必理会诸多的错误可能，只需要专注于内核的独立

功能即可。

具体来看，本评测系统的内核模块内部应该也是由多个相对独立的功能子模块组成，再由一个统一的总控子模块来进行统一调度和对外通信构成，这样也可以保证每个独立的子模块的功能或者逻辑改动不会影响其他的模块，符合本课题提出的易维护和易部署的要求。

在线评测系统内核模块所需要完成的主要功能是从中心服务模块获取用户提交的源代码，与源代码对应的系统标准输入输出，以及对应题目的资源限制，在编译用户源代码后，用系统标准输入作为输入来运行编译后的可执行文件，最后用系统标准输出来对比用户的输出，检测用户程序是否正确，并将结果返回给中心服务模块。当编译或者运行阶段出现了任何异常，都需要立即终止当前进行的活动并取消后续活动，且需要将异常错误信息反馈给中心服务模块。如果是需要用程序来判断用户的输出是否符合要求，还需要从中心服务模块中获取验证程序，并将其编译成可执行文件后去使用用户的输出来做验证程序的输入并获取比较结果并将此结果返回给中心服务模块。

在线评测系统的返回结果列表已经在节中进行了简要介绍，且该定义以头文件形式定义于整个系统的头文件目录内，保证了更新和修改时对整个系统的影响会保持一致且不影响逻辑功能。

本在线评测系统内核模块的总工作流程图如下图 3-1 所示：

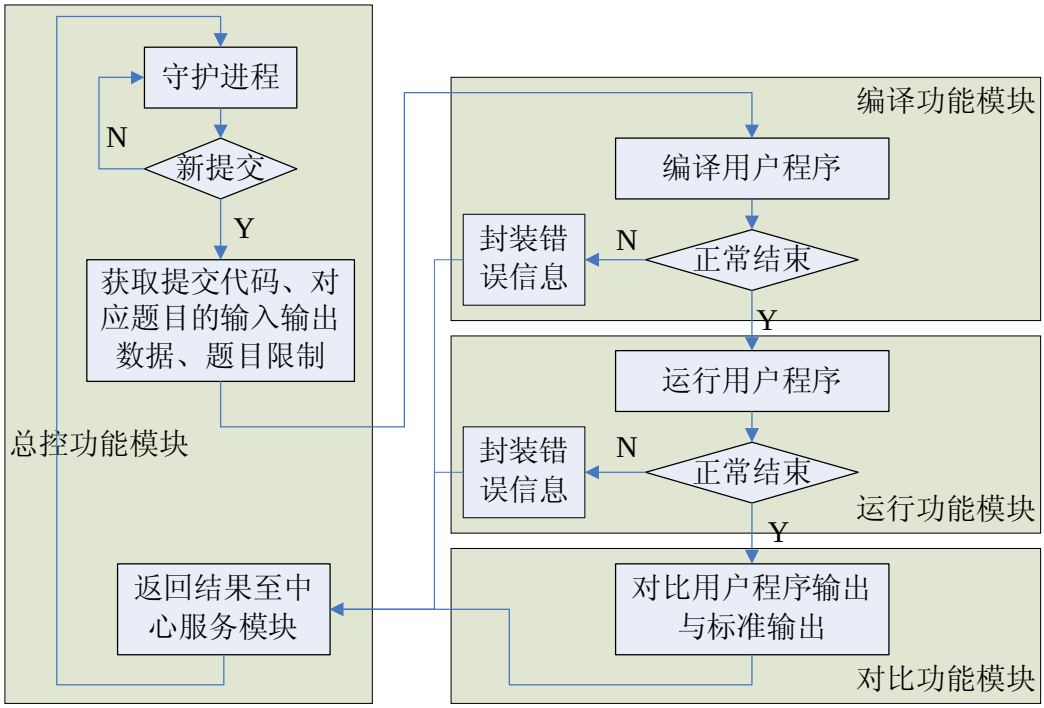


图 3-1 在线评测系统内核模块工作流程图

本课题所设计的在线评测系统内核模块应该由总控子模块，编译功能子模块、运行功能子模块和对比功能子模块这三个大的逻辑处理模块，以及监控功能模块和辅助功能模块两个相对独立且多处调用的子模块构成。

其中总控模块负责进行对整个评测过程的控制和与本在线评测系统的中心服务模块通信。详细来说包括建立本内核模块运行时所需要的工作目录并将当前模块的工作目录转移到此目录，绑定相关系统信号量 SIGTERM 和 SIGPIPE 以响应操作系统发出的终止信号或者在通信管道异常时也可以按正常途径来终止内核模块的运行，打开通信端口并连接到在线评测系统的中心服务模块接收中心服务模块发过来的判题请求和后续的用户程序源代码、对应

题目的输入输出数据、题目的资源限制信息，运行守护进程以能够在接收中心服务模块发过来的判题请求后能顺序调用编译、运行和对比三个功能子模块来进行判题过程，同时如果中间有出现异常则将对应子模块封装好的错误信息返回给在线评测系统的中心服务模块，或者是在整个判题流程正常终止后将判题结果返回给中心服务模块。

编译、运行和对比三个功能子模块正是对应了内核对用户程序所进行处理的三个阶段。编译子模块负责将获取的用户源代码调用对应程序设计语言的编译器，在保障系统安全的前提下，按照规定的优化级别和编译参数将其编译成可执行文件，如果在编译过程中有错误产生，则封装编译器返回的错误信息并传回在线评测系统的中心服务模块，否则返回正常信息给总控子模块。运行子模块负责在保障系统安全的前提下，将编译子模块编译后的用户程序在规定的题目资源限制下运行，并将对应题目的输入作为标准输入重定向给用户程序，将用户程序的输出重定向到本系统指定的文件，且在运行过程中监控所有系统调用，防止用户程序进行违规操作或有破坏系统的行为，如果在运行过程中有错误产生，封装错误信息并传回在线评测系统的中心服务模块，否则返回程序运行的时间内存等相关信息。对比子模块将比较用户程序的输出文件和对应题目的标准输出文件，给出对比结果返回，如果题目需要使用特殊判断程序来运行，则将对应题目的标准输入文件、标准输出文件及用户程序输出文件作为文件输入流按约定规则输入至特殊判断程序获取对比结果，返回给总控子模块。

监控功能模块独立出来是因为在编译、运行和对比三个阶段都需要进行监控，防止用户程序出现异常或者对系统的攻击。本功能子模块需要详细的示例来说明功能及工作原理，且涉及到系统安全，故详细的设计和实现机制将在 3.7 节和第 4 章中讲解。

辅助功能模块完成了部分网络通信接口的封装，以及一些如 Linux 信号量处理函数的绑定、创建新进程并运行某程序等经常用到的小函数。本功能子模块功能较多且繁琐，基本涵盖了本在线评测系统内核模块其他功能子模块外的所有功能。

## 3.2 辅助功能模块

按照 3.1 节中进行的功能划分，辅助功能模块需要完成其他模块未实现的功能，为其他模块提供服务。由于辅助功能模块的函数或者定义的结构体都需要在其他功能模块设计实现讨论中多次出现，故将辅助功能模块的设计与实现放在最开始讨论。

辅助功能模块需要实现的功能包括设置系统限制、读取指定进程所耗费的时间、读取指定进程所耗费的内存、创建新进程并按照指定参数运行指定可执行文件、创建新进程并运行指定的 Shell 脚本、修改系统信号量相关联的处理动作、通过指定地址及端口连接服务器创建通信文件描述符和锁定指定文件，同时在辅助功能模块定义了进程创建信息这一结构体。

修改系统信号量相关联的处理函数、对服务器的连接及锁定指定文件功能属于对一系列底层操作的二次封装而将其作为一个函数出现方便其他功能子模块调用，对这些功能的设计实现可以在大量书籍资料或网络资源上找到，且其设计实现并非本课题关注的重点，故对这些功能的设计实现在此提及后并不讨论其详细设计实现。

进程创建信息结构体和创建新进程等相关操作均涉及 Linux 的进程创建函数和对相关的系统资源进行限制，故后续详细讨论中将进程创建信息结构体、设置系统限制函数、创建新进程并按照指定参数运行指定可执行文件和创建新进程并运行指定的 Shell 脚本放在一起讨论。

读取指定进程所耗费的时间及内存均使用类似的方法实现，即读取系统相关记录文件，后续详细讨论中将放在一起。

### 3.2.1 创建新进程并运行相应程序

本功能函数按照 2.2.2 节中的背景知识描述进行。其中 exec 选用的是 execv 函数，其参数中第一个为新程序的路径，后一个参数为新程序执行的参数表。在调用 fork 创建新进程和调用 execv 来转换程序空间之间，对子进程需要设定相关的系统设置，来保证创建的新进程能在本内核模块规定的系统资源限制内运行，并由本内核模块指定能读取的文件和通知监控模块是否需要监控此子进程。此部分功能将在 3.2.2 节中进行详细讨论，此处只说明还需要进行相应操作。

这样，就可以给出创建新进程执行 Shell 脚本和用户程序的函数实现伪代码：

```
// 创建新进程执行 Shell 脚本, command 为要执行的脚本名, run_info 为资源限制信息
// 函数返回值为子进程 pid
int createShellProcess(const char* command, const RunInfo& run_info) {
  const char* commands[] = {"/bin/bash", "bash", "-c", command, NULL};
  return createProcess(commands, run_info);
}

// 创建新进程执行用户程序, commands 为执行程序名及参数, run_info 为资源限制信息
// 函数返回值为子进程 pid
int createProcess(const char* commands[], const RunInfo& run_info) {
  // 重定向标准输入/标准输出/标准错误输出文件流
  int pid = fork();
  if (pid < 0) {
    /* fork 出错, 错误处理语句 */
  } else if (pid > 0) {
    // 在父进程中, 返回子进程 ID
    return pid;
  } else if (pid == 0) {
    // 复制标准输入/标准输出/标准错误文件标识符并关闭其他文件
    // 设置系统资源限制并通知监控模块监控本进程, 此处详见 3.2.2 节
    if (execv(commands[0], (char**)(commands + 1)) == -1 {
      // execv 出错, 打印错误信息
      raise(SIGKILL);
    }
  }
}
```

在本内核模块中，由于各模块对于进程终止后所需要的信息及处理机制不尽相同，所以等待进程终止的 wait 操作均在调用创建进程的模块中处理。

### 3.2.2 设定进程资源限制

本在线评测系统中，创建每个进程，都需要设置其资源限制信息和其他运行信息，使用的信息描述结构体如下所示：

```
struct RunInfo {
  // 标准输入/标准输出/标准错误文件描述符
  int file_stdin;
  int file_stdout;
  int file_stderr;
  // 标准输入/标准输出/标准错误的重定向文件名
  const char* stdin_filename;
  const char* stdout_filename;
  const char* stderr_filename;
  // 运行时使用的用户 uid 和用户组 gid
  int uid;
  int gid;
  // 进程运行的时间(秒), 内存(KB), 输出文件大小(KB)限制
```

```
  int time_limit;
  int memory_limit;
  int output_limit;
  // 进程所能控制的子进程个数和能打开的文件个数限制
  int proc_limit;
  int file_limit;
  // 是否监控标记
  bool trace;
  // 工作目录
  const char* working_dir;
};
```

按 2.2.3 节中的背景知识介绍，表 2-2 中与此处 RunInfo 结构体的对应关系是 RLIMIT_CPU 对应 time_limit, RLIMIE_DATA 对应 memory_limit, RLIMIT_FSIZE 对应 output_limit, RLIMIE_NOFILE 对应 file_limie, RLIMIE_NOPROC 对应 proc_limit。并且对 time_limit 这一限制还使用了 alarm 函数，之所以进行此操作是因为在用户程序中，可能进入没有任何操作的死循环，这样对操作系统而言，该进程的 CPU 时间没有进行任何改变，会一直运行下去。对 memory_limit 这一限制，在后续的监控模块中还另外使用了其他相对复杂的方式进行。

对 uid 和 gid 的更改可以通过 setuid 和 setgid 函数来进行，对工作目录的更改可以通过 chdir 函数来进行。不使用 chroot 来实现是考虑到不是所有场合都能使用超级用户来运行本内核模块

通知监控模块监控此进程用 ptrace 里的下列调用实现（具体机制将在 3.7 节中讨论）。

```
ptrace(PTRACE_TRACEME, 0, NULL, NULL);
```

### 3.2.3 读取指定进程所耗费时间及内存

在 Linux 操作系统中，对每个正在运行的进程，都会在/proc 进程信息伪文件系统下有一个以该进程 ID 命名的子目录，其中的部分伪文件描述了当前进程使用时间和内存的情况。本系统中选用的是/proc/<pid>/stat 来获取时间信息，/proc/<pid>/status 来获取内存信息。

/proc/<pid>/stat 文件包含的内容太多，本文不详细解释每个参数的意思，读者可在 Linux 终端下通过 man proc 指令或查阅相关书籍资料来获取详细信息。本系统中只用到了里面第十四个参数 utime 和第十五个 stime，utime 是该进程在用户态运行的时间，stime 是该进程在核心态运行的时间，单位都是时钟嘀嗒。所以我们只要按格式读取到这两个参数，然后转化为毫秒返回即可。返回值单位采用毫秒是为了展示给用户其程序运行的更精确时间。具体实现请参见附录或本系统源代码相应文件。

/proc/<pid>/status 内也包含了相当多的内容，我们关注的只是其中包含的内存使用信息，在该文件中以 Vm 开头的几行描述了该进程的内存占用信息，单位均为 KB，对我们有用的有 VmSize（任务虚拟地址空间的大小）、VmExe（程序所拥有的可执行虚拟内存的大小、代码段，不包括任务使用的库）、VmLib（被映像到任务的虚拟内存空间的库的大小）。从 2.6 版本内核开始，此文件中加入了 VmPeak 一行，内容同 VmSize，但是在动态更新上会更精确，所以如果有 VmPeak 则优先考虑此信息。我们需要读取 VmSize（如果有 VmPeak 则用 VmPeak 替代之），再减去 VmExe 和 VmLib，就可以得到该程序实际运行所耗费的内存。

在用户程序运行的过程中不断调用这两个函数来获取所耗费的时间和内存，更新最大值，最后等用户程序运行完后，时间和内存最大值就是用户程序所耗费的时间和内存。

## 3.3 总控功能模块

总控功能模块主要是对整个在线评测系统内核模块进行逻辑控制，图 3-1 已经很好描述了总控功能模块的大体实现逻辑。在守护进程中，获取正常的编译结果后会初始化一个结果结构体，来存储本次提交的运行结果、耗费时间和内存信息。然后按照测试用例的组数来依次调用运行功能模块和对比功能模块，每次调用均会更新结果结构体。如果能正常跑完所有的测试用例，则返回正常的结果，否则返回对应错误结果和错误信息（如果有可输出的错误信息）。本文抛开对大体逻辑设计实现的讨论，专注讨论逻辑之外的部分棘手细节问题。

### 3.3.1 守护进程的持续性和可控性

守护进程的持续性很好理解，即只要本内核模块在运行，守护进程也将一直运行，以获取中心服务模块发到内核模块的判题请求。最简单的实现只需要用一个无限循环不停运行守护进程获取请求的代码段即可，但是这样导致的后果是无法对守护进程进行任何控制，如果需要退出则只能强行终止进程，这样会带来很大的安全隐患和系统的不稳定性。

守护进程的可控性即是本内核模块能响应外部的系统终止信号，在接收到终止信号后，停止一直在运行的守护进程，并清理当前内核模块运行时所产生的临时文件，按正常流程结束内核模块的运行。当发生通信文件描述符异常时，一样需要按上述流程来使内核模块正常结束。

在本系统中，我们使用了两个全局变量 terminated 和 socket_broken 来标识是否产生了需要结束内核模块的两个信号，将 SIGTERM 和 SIGPIPE 两个系统信号量的处理函数重定向至修改 terminated 和 socket_broken 值的函数。具体实现伪代码为：

```
...
int terminated = 0;
int socket_broken = 0;

void sigterm_handler(int signal) { terminated = 1; }
void sigpipe_handler(int signal) { socket_broken = 1; }

int main(int argc, char* argv[]) {
  ...
  // 重定向系统信号量处理函数
  installSignalHandler(SIGTERM, sigterm_handler);
  installSignalHandler(SIGPIPE, sigpipe_handler);
  ...
  // 如果没发生异常则持续运行守护进程
  while (!terminated && !socket_broken) {
    process(communicate_socket);
    ...
  }
  // 清理临时文件, 正常退出
}
```

### 3.3.2 非堵塞方法获取通信接口内容

在守护进程读取中心服务模块的判题请求时，如果使用普通的 read 函数，或者是在本系统经过重新封装过加入容错性的 socket_read 函数，在读取时均采用的是堵塞模式，即如果中心服务模块没有发来任何判题请求，则守护进程会一直堵塞在读取这一操作上，从而导致系统无法在响应系统信号量处理函数的行为后退出该无限循环。

本系统实现过程中在进行读取前使用 pselect 函数通过检查通信端口是否就绪来实现非堵塞读取模式。且由于在接收到判题请求后，其他的读取行为都必须是不间断的，所以只在获取判题请求时使用此方法。对 pselect 函数的详细说明请读者参见相关资料，本文中不详细讨论。相对 select 函数而言，pselect 提供了 sigmask 这样的一个指向 sigset_t 的结构体指针，可以在执行 pselect 操作的时候暂时堵塞 sigmask 集合中的信号，等 pselect 操作完成后再反馈给本程序，本系统此处实现即暂时堵塞了 SIGTERM 和 SIGPIPE 两个我们需要获取的信号量。

## 3.4 编译功能模块

编译功能模块需要从主控功能模块处获取需要编译的源文件名及路径，自动将其编译成同名可执行文件。如果用户源代码是非编译执行型语言（如 Java），则本模块不对其进行任何操作。

编译有关的背景描述已经在 2.2.1 节中详细说明，本处只说明模块实现逻辑。

为了让编译功能模块的代码清晰易维护，具体编译的语句以 Shell 脚本的形式独立存在于本内核模块的脚本目录中。当编译功能模块需要编译程序时，创建新进程运行该 Shell 脚本并将需要编译的源文件名和路径以参数形式给出，由该脚本调用相关命令将文件编译成可执行文件。该 Shell 脚本先将工作目录转换至源代码所在目录，然后去除源代码文件名后缀形成可执行文件名，最后依据源代码文件名后缀来识别源代码类别并执行相关的编译命令。脚本核心代码为：

```
// 依次是 C, C++, Pascal, $bin 为可执行程序文件名, $src 为源文件名
gcc -o $bin -ansi -fno-asm -O2 -Wall -lm -static -s -DONLINE_JUDGE $src
>/dev/null
g++ -o $bin -ansi -fno-asm -O2 -Wall -lm -static -s -DONLINE_JUDGE $src
>/dev/null
fpc -o$bin -Fe"/proc/self/fd/2" -Sd -dONLINE_JUDGE -O2 $src >/dev/null
```

其中编译开关和参数优化等参照 ACM/ICPC 竞赛官方标准。

创建新进程执行该 Shell 脚本时，创建一个管道，获取该进程的标准错误输出。当进程结束后，调用 waitpid 来获取结束状态，通过 WEXITSTATUS 分析该进程是否正常结束，如果是非正常结束，则返回 COMPILE_ERROR，并将该进程的标准错误输出返回给中心服务模块作为错误信息展示给用户。

## 3.5 运行功能模块

当总控模块调用运行功能模块时，会将用户程序的可执行文件路径、对应题目的资源限制通知运行功能模块，运行功能模块将相关资源限制及对应题目的输入作为标准输入文件来构造进程创建信息结构体，并设置监控位为真。用此进程创建信息结构体创建新进程并运行用户程序，然后转入运行模块的监控函数，一直监控到用户程序运行结束。

运行功能模块的监控函数主要是进行逻辑控制，核心的监控机制将在监控功能模块中实现。逻辑控制流程图如下图 3-2 所示：

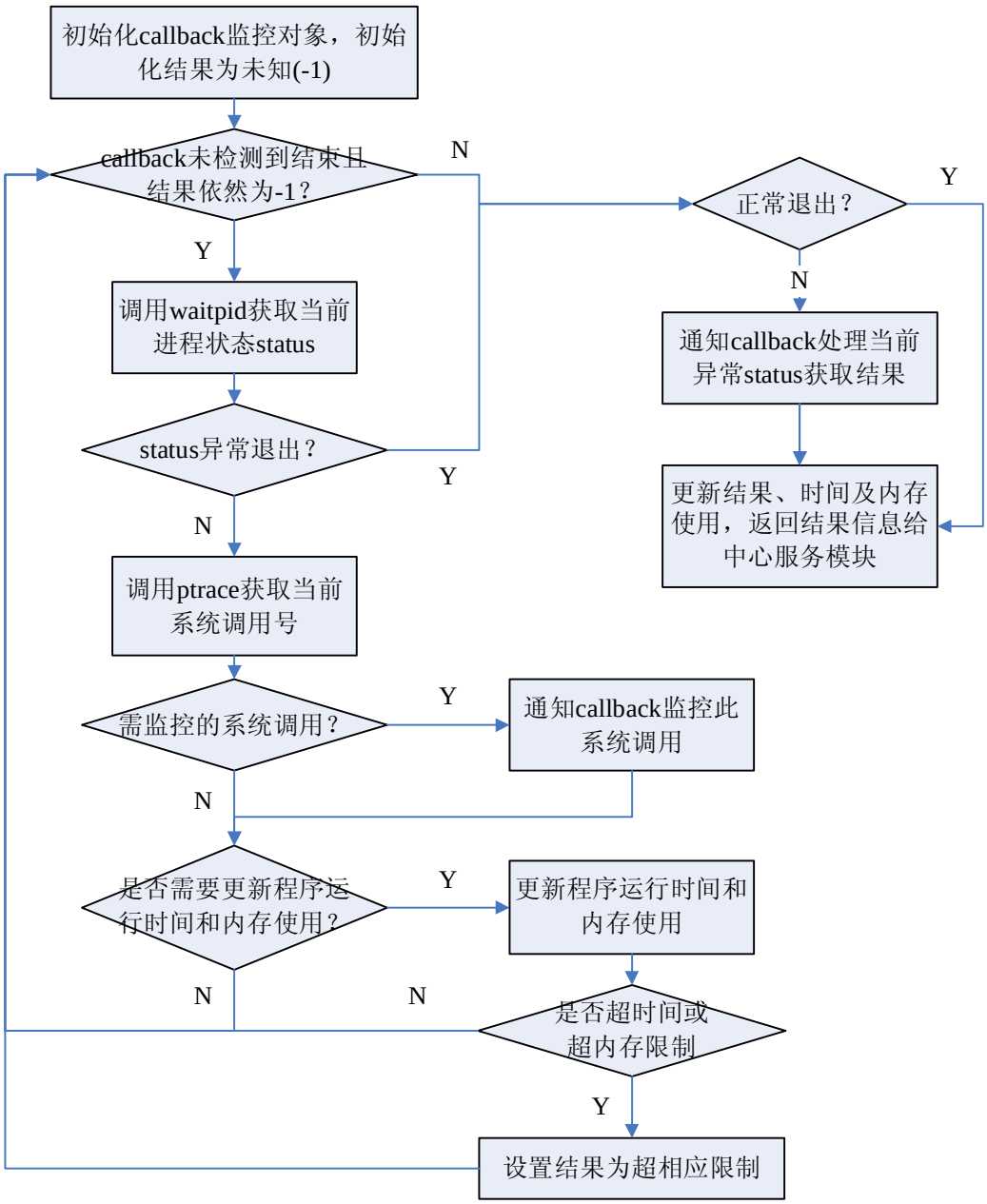


图 3-2 运行功能模块流程图

## 3.6 对比功能模块

当总控功能模块调用运行功能模块时，需要对用户的结果输出和对应题目的标准输出做比较来判断结果正确与否。根据题目结果的唯一性与否，对比功能模块将调用普通文本比较函数或特殊判断程序比较函数来获取最终结果。

文本比较的标准在 2.1.2 节中已经给出，这里给出具体实现的判断逻辑如下图 3-3 所示：

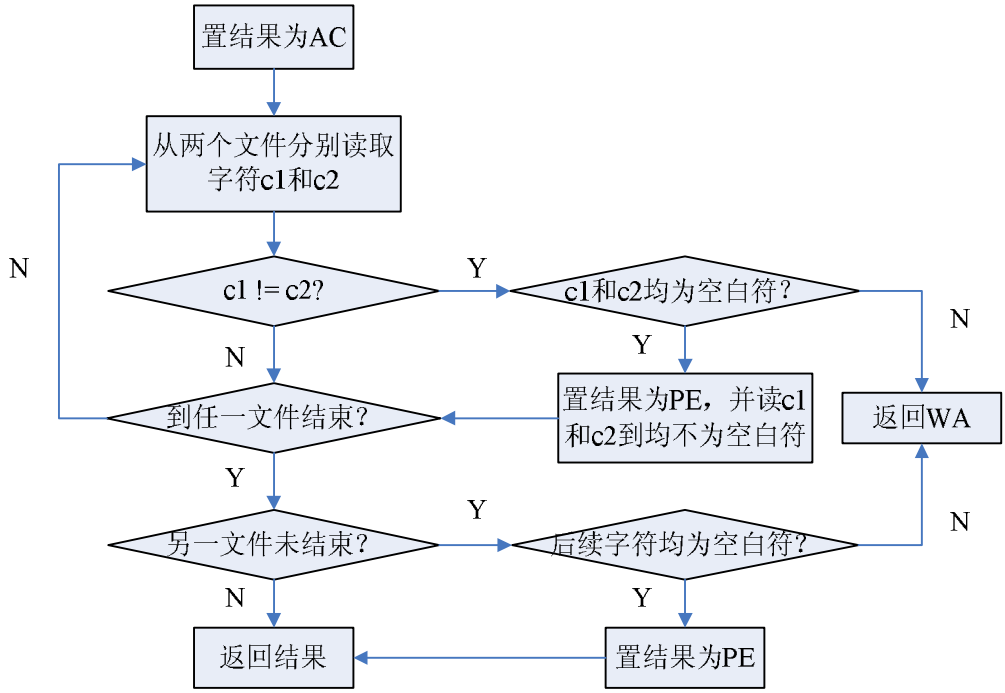


图 3-3 普通文本比较流程图

特殊判断程序比较则构造进程创建信息结构体，将相应题目标准输入、标准输出、用户程序输出文件作为参数，目标程序为特殊判断程序，创建新进程运行特殊判断程序来比较结果，最终按照该特殊判断程序的运行结果并转换为 AC/WA/PE 返回给中心服务模块。类似运行功能模块除去了程序监控和时间内存获取的简化实现，且去除了程序异常终止处理。

## 3.7 监控功能模块

监控功能模块用一个 Callback 类来实现需要监控的功能，且为保证监控的可靠性，本系统采用的单例来实现 Callback 对象，即同一时刻只能有一个。Callback 类能为外部调用的处理函数主要有以下两个：

```
// 处理监控父进程通过 waitpid 获取的 status
void processResult(int status);
// 处理监控父进程捕获的子进程 pid 需要判断的系统调用 syscall
void processSyscall(pid_t pid, int syscall);
```

其中 processResult 主要使用一个 switch 语句来判断传入的 status 经 WTERMSIG 宏处理后，判断是由什么系统信号量来终止的监控子进程，设置对应结果。对应表如下：

```
SIGXCPU ->  TIME_LIMIT_EXCEEDED
SIGALRM ->  TIME_LIMIT_EXCEEDED
SIGXFSZ ->  OUTPUT_LIMIT_EXCEEDED
SIGSEGV ->  RUNTIME_ERROR_SIGSEGV
SIGFPE  ->  RUNTIME_ERROR_SIGFPE
SIGBUS  ->  RUNTIME_ERROR_SIGBUS
SIGKILL ->  RUNTIME_ERROR_SIGABRT
SIGABRT ->  RUNTIME_ERROR_SIGABRT
```

processSyscall 函数通过一个 switch 来判断输入的 syscall 是属于什么类型，从而进行相应的监控行为。为什么监控这些系统调用以及如何监控的详细分析将在第 4 章中进行，以下是函数功能实现的伪代码：

```
switch (syscall) {
  case SYS_exit :
```

```
  case SYS_exit_group :
    // 程序退出系统调用, 通知 callback 此进程正常结束
  case SYS_brk :
    // 检查内存申请状况是否超内存限制
  case SYS_clone :
  case SYS_fork :
  case SYS_vfork :
    // 申请创建新进程, 检查是否超进程数限制
  case SYS_execve :
    // 申请转换程序空间, 检查是否是监控父进程的第一次转换
  case SYS_open :
  case SYS_access :
    // 文件操作, 读内存里要访问的文件路径, 检查是否是允许访问的文件
}
```

# 第4章　在线评测系统内核的安全机制

作为一个评测系统，在执行用户程序的时候，要考虑用户程序刻意或者无意中对系统造成的安全隐患，这是本系统内核模块面临的最大挑战。

## 4.1 运行用户程序时需要监控的资源分析

在本系统内核模块中，经分析，在运行用户程序时，需要监控的资源和原因如下：

**1. 打开文件的控制**

评测系统应严格限制用户打开的文件，一般来说只有运行程序所必须的系统文件和作为输入的文件可以被用户程序打开使用。

**2. 运行时间的限制**

评测系统一般有用户程序运行时间的限制，这样可以在一定程度上要求用户必须采用合适的算法来解决评测系统上的特定题目，这样才能在评测系统上得到通过的结果，同时也可以避免用户程序意外的进入死锁或者僵死态。

**3. 使用内存的限制**

评测系统必须保证用户程序所使用的内存在一个可控范围内，这样才不会破坏评测系统内核模块所在宿主机的正常运行，且某些题目也会要求用户必须在限制的内存内解决，迫使用户使用合适的算法。

**4. 输出大小的限制**

为了避免用户程序进入异常状态无限制的输出而破坏评测系统内核模块所在宿主机的正常运行，评测系统必须保证用户程序不能无限制的输出。

**5. 用户程序的工作文件夹**

评测系统应该限制用户程序运行在一个封闭的目录里，这样用户程序就不会破坏宿主机的操作系统，在发生异常或者用户提交恶意程序攻击系统时可以防备。

**6. 用户程序的运行用户和运行用户组**

评测系统应该使用一个低权限的用户及用户组来运行提交上来的用户程序，这样也是为了防备在用户程序或者用户提交恶意程序攻击系统。

**7. 用户程序的进程数**

由于评测系统是为程序设计语言、数据结构和算法等基础课程的学习训练和程序设计竞赛的训练或者比赛服务的，所以要求用户程序只能使用单进程模式运行，继而严格要求用户的基础能力和尽可能的保护宿主机系统，而且可以保证用户程序是一直处于被监控状态的。

**8. 系统管理操作**

如对磁盘的加载、卸载，创建文件连接等。

除开上面提及的特殊要求外，运行用户程序并没有什么不同。只是需要由评测系统通过 fork() 来生成子进程后，在子进程里设置好相关限制，并通知评测系统来监控生成的此子进程，再调用 execv 来将子进程的程序空间和数据空间转换成用户程序。在执行过程中不断通过 ptrace 来监控用户程序，防止恶意行为。

## 4.2 运行用户程序时的监控机制实现

对 4.1 节中分析的要监控资源，其中运行时间、输出大小、工作文件夹、运行用户和用户组限制可以完全由创建进程时所设置的资源限制进行控制，使用内存和用户程序的进程数限制可以部分由创建进程时所设置的资源限制进行控制，具体在创建进程时如果进行资源限制请查看本文 3.2.2 节中的相关说明。

对其他的资源监控，均通过 ptrace 来实现。在创建进程时就通知父进程需要监控本子进程，在父进程中不断获取子进程的系统调用，然后通过查看系统调用过滤表来判断是否需要监控本次系统调用，以达到限制资源的目的。对系统管理操作所进行的系统调用一律在过滤表中设置为禁止。其他需要监控的系统调用在过滤表中设置为监控，检测到需监控的系统调用时执行 3.7 节中描述的过程来进行监控。详细的系统调用过滤表及设置注释请参见本系统实现源代码中相关文件。

为满足监控使用内存的限制，通过监控 brk 系统调用实现。当发生 brk 系统调用时，检测是否新申请了内存，如果有新申请内存空间的操作，则判断子进程的总内存空间是否超过限制大小。

为满足监控进程数的限制，通过监控 fork 和 vfork 系统调用来防止新建进程，以及相关的 execve 等系统调用防止转换程序空间以达到一个进程运行多个程序的目的。

如果是打开文件或者创建文件的系统请求，首先读取相关寄存器获取请求的文件名的内存地址以及请求标识位，通过直接读取内存获取具体文件名及路径，判断请求的文件是否是属于被限制访问的文件。

# 第5章　总结和展望

本课题欲实现的可分布式部署的在线评测系统目前已基本完成，并运行在武汉大学 ACM/ICPC 集训队机房的内网服务器上，在测试微调完成后将迁移至武汉大学在线评测系统对外服务器上，为整个互联网提供服务。

## 5.1 本课题成果总结

对本文提出的目前各在线评测系统单服务器模式带来的各功能单元间高度耦合，从而带来更新维护不便的问题，本系统开发人员通过将整个在线评测系统分割成若干大的服务模块，模块间通过自定义网络协议通信的设计，对设计完整正确实现后，达到了软件工程中追求的高内聚、低耦合的目标，降低了系统潜在问题的可能性，并让未来的维护人员能清晰明了的理解系统结构和逻辑流程，降低维护成本，且对未来的继续开发提供了可扩展的基础。

对本文提出的目前各在线评测系统单服务器模式在负载高峰时服务压力过大导致无法及时响应的问题，本系统将整个服务拆分为服务集群模式的设计已经在理论上较好的解决了这一问题。在实验室中通过将评测内核和 Web 服务模块分布部署于多台普通 PC 机上进行测试，在面对用户爆发性访问和用户程序爆发性提交的压力测试，完美的做到了高效的页面响应和评测响应。

本文的关注焦点是评测系统内核，通过本文详细论述的设计实现后，最终实现版本在面对全面的单元测试和大量的用户盲测，稳定高效的通过了各种正确、不正确及攻击性用户程序的测试，并且保证了系统安全，达到了预期的目标。

## 5.2 本课题前景展望

受开发时间的限制，本系统内核模块依然存在较大的改进空间，如加入更多的语言支持，主要是解释执行类语言，对系统调用过滤表的进一步细化和明确，以进一步完善开发文档，为以后的维护和继续开发提供更便捷的方式。在安全机制上，现有的通过 ptrace 实现的安全机制依然存在一定瑕疵，最彻底的方法是通过 Linux 内核编程来达到完全监控系统调用从而完全保护系统，且对 Java，还有更多需要考虑的安全细节。

总之，在线评测系统是为程序设计竞赛及教学服务的一个辅助平台，无论本系统如何改进，课题灵魂思想还是希望能尽可能的服务用户，让用户提高其程序设计能力，从而在相关领域获取更大的竞争力。本系统将延续武汉大学当前的在线评测系统在对武汉大学相关教学及 ACM/ICPC 竞赛训练中发挥的巨大作用，并力争以后能更好的发挥作用。

# 参考文献


[1] (美)W.Richard Stevens, Stephen A. Rago 著, 尤晋元、张亚英、戚正伟译，UNIX 环境高级编程（第二版）[M], 机械工业出版社, 2006

[2] (美) Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides 著，李英军、马晓星、蔡敏、刘建中等译，设计模式：可复用面向对象软件的基础[M]，机械工业出版社，2000

[3] (美)W.Richard Stevens, Bill Fenner, Andrew M.Rudoff 著，杨继张译，UNIX 网络编程（第一卷：套接口 API）（第三版）[M]，清华大学出版社，2005

[4] (美)Samuel P.Harison III, Guy L. Steele Jr. 著，邱仲潘等译，C 语言参考手册[M]，机械工业出版社，2003

[5] 王欢、杨树青著，Linux 环境下 C 编程指南[M]，清华大学出版社，2007

[6] 王腾、姚丹霖，Online Judge 系统的设计开发[J]，计算机应用与软件，2006(12)

[7] 周高嶔，基于白箱测试的源代码在线评测系统[D]，北京化工大学，2005

[8] 苑文会，基于黑箱测试的源代码在线评测[D]，北京化工大学，2005

[9] 李文新、郭炜，北京大学程序在线评测系统及其应用[J]，吉林大学学报(信息科学版)，2005(S2)

[10] 唐俊武、南理勇、左强，在线考试系统开发中的几个问题及解决方法[J]，计算机与数字工程，2005(08)

[11] 毛华、罗朝盛，基于 Web 的在线考试系统的设计与实现[J]，计算机时代，2005(11)

[12] Pradeep Padala， Playing with ptrace, Part I[OL]， Linux Journal， 2002(11)，http://www.linuxjournal.com/article/6100 (访问时间：2008/3/2)

[13] Pradeep Padala， Playing with ptrace, Part II[OL]， Linux Journal， 2002(12)，http://www.linuxjournal.com/article/6210 (访问时间：2008/3/2)

# 致　　谢

在我们完成毕业设计的过程中，有非常多给予我们帮助的师长，同学，以及众多网络上素未谋面的朋友。

感谢我们毕业设计的指导老师董文永老师，在完成毕业设计的过程中给予了我们大量专业知识和求学处事的指导。并且身为武汉大学 ACM/ICPC 集训队的总教练，在本课题组的三个人参加 ACM/ICPC 竞赛的过程中也给予了大量的帮助和支持，对我们的学习生涯有重要的指导意义，并让我们在完成毕业设计的过程中发扬武汉大学 ACM/ICPC 集训队快速高效能吃苦的精神，在短期内非常好的完成了这一复杂的毕业设计的设计实现和测试。

感谢武汉大学在线评测系统（noah 及 oak）的开发人员：杨传辉、吴永坚、刘高杰、杨宝奎、董超、高双星，有他们的工作才奠定了本课题最原始的理论和实践基础，他们也本毕业设计实现的过程中给出了大量目的明确、效果明显的建议和帮助。

感谢武汉大学 ACM/ICPC 集训队，给我们提供了开发的环境和平台，让我们能完成毕业设计并能将其部署以提供服务，感谢集训队的同学们给我们大量有重要价值的用户反馈和改进建议，让我们将毕业设计做的更好。

感谢网络上那些通过邮件、聊天、技术博客等方式提供帮助的朋友：浙江大学的 xuchuan、oldbig，北京大学的 frkstyc、XieDi，华中科技大学的 Sempr、MagicD，中国地质大学（武汉）的 ZmStone，谷歌上海工程研究院的实习生同事郑欣杰。特别感谢目前就职于谷歌中国工程研究院的 xuchuan，在我设计实现本系统评测内核的过程中给予了难以言表的大量指导和帮助。

感谢武汉大学，感谢计算机学院，给我在这个美丽的校园学习四年的机会。

感谢在我大学四年中遇到的各位老师，是你们辛勤的培养和教诲，才能有我们今天的知识储备和专业修养，能在面对未来的学习和工作时能有的放矢、得心应手。

感谢我的父母，在我成长求学路上无言的关心、鼓励和支持。

感谢刘潜、马陈吉尔，在进行毕业设计的几个月里，我们互相鼓励和奋斗才最终完成这一系统。

在本毕业设计实现的过程中还有许多给予了我们帮助的人，受篇幅限制，无法一一感谢。在此，感谢上述所有的人，祝你们身体健康、学习工作顺利。

叶文

二〇〇八年五月

于武汉大学 ACM/ICPC 集训队训练基地