1）Main主线程中初始化init：

1. 创建4个mysql数据库的连接；
2. 用pthread\_create创建4个线程，创建成功后，执行各自的thread\_work函数，把线程tid作为参数传给每个线程的thread\_work函数，这些线程就并行执行，分别处理用户提交的题目；
3. pthread\_mutex\_init初始化互斥锁
4. sem\_init(&bin\_sem,0,0)初始化信号量bin\_sem，第二个参数为0表示当前进程的所有线程共享；第三个参数给出了信号量的初始值
5. pinfo[i].have=false;2000个题目信息都没有从数据库获取过~
6. 1.初始化系统调用数组为0,2.数组中序号为安全的系统调用号则设置为1，在之后ptrace子进程时，kill掉调用不安全的系统调用~

2）回到主线程，select数据库中最新提交的前10的waiting状态的记录，若有waiting状态的记录，存到submit结构体中，然后把submit添加到队列中，并且使信号量+1，信号量的值为waiting待处理的记录数(资源个数)，这两步要加互斥锁，是临界资源。

3）各子线程与主线程也都分别独立并行工作

**子线程：**

While死循环处理submit数据，循环中：

1. sem\_wait(&bin\_sem); 使信号量-1,使线程访问顺序能够同步，信号量的值 == 资源个数，如果信号量==0，则当前线程阻塞，直到有可访问的资源~
2. get\_queue(&submit)从队列中获取submit数据, 这里要给临界资源加互斥锁~在使用互斥资源时使用，互斥量值只能为0/1，使用后解锁。
3. 设置submit为compiling，如果submit->problem\_id的题目信息没有从数据库获取过，则查询数据库，然后保存到pinfo数据结构中,
4. compile(getid,submit)&&judger(getid,submit);编译和评判submit的结果，得到结果后保存结果到数据库中。

**4）compile**

1.从数据库中得到submit->problem\_id的源代码，给读到./tmp/%d.cpp文件中，格式转换为linux上的，用system函数（执行shell 命令）执行“g++ ./tmp/%d.cpp -o ./tmp/%d -o0 -ansi -w 2> ./err/%d”编译cpp源文件，把编译好的目标文件保存到./tmp/%d下，如果system返回0,代表运行成功，木有语法问题，否则submit->result=CE;

**5）judger**

Vfork()一个子进程，这是父进程阻塞，直到子进程执行execl(path,NULL,NULL);，唤醒父进程

子进程执行：

1. 重定向"./data/%d/data.in"为标准输入，重定向"./tmp/%d.out"为标准输出，使用setrlimit进行内存和运行时间的限制，ptrace(PTRACE\_TRACEME,0,NULL,NULL);//本进程被其父进程所跟踪。execl(path,NULL,NULL);开个子进程运行"./tmp/%d"目标文件，

当进程调用一种exec函数时，该进程完全由新程序代换，而新程序则从其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并未改变。exec只是用另一个新程序替换了当前进程的正文、数据、堆和栈段。

1. 1）此时唤醒父进程，wait4(pid,&status,0,&usage);//阻塞当前进程，直到子进程状态改变，可获取子进程状态转变的信息，还可以获得子进程的资源使用信息，资源统计信息包括用户CPU时间总量、系统CPU时间总量、页面出错次数、接收到信号的次数等。

宏测试WIFEXITED(status)；返回非0值，会正常退出~

WIFSIGNALED(status))//返回为真，则为异常退出~

sig=WTERMSIG(status);若成功返回被终止的子进程的信号值。

switch (sig)

{

case SIGXCPU : submit->result=TLE;break;

case SIGFPE :

case SIGSEGV :submit->result=RE;break;

default: submit->result=MLE;

}

break;

2）父进程与子进程同时执行，父进程在跟踪子进程的系统调用~~

syscallID=ptrace(PTRACE\_PEEKUSER,pid,4\*ORIG\_EAX,NULL);从寄存器中读取系统调用号，

csafecall[syscallID]==0，则submit->result=RF;//RF结果代表调用了不安全的系统调用，此时杀掉子进程，使它退出。

/通过/proc/%d/statm，是取第二个数，代表物理内存大小，获得当前线程内的vfork新开的进程(可是是进程号)使用的物理内存~如果这个内存大于要求的内存，submit->result=MLE，此时杀掉子进程，使它退出。

不然把当前运行的内存设置为得到的物理内存大小，然后使子进程重新运行，忽略引起调试进程中止的信号.

如果submit->result==AC，则比较标准输出和"./data/%d/data.out"文件，返回值是1则AC，是-1，则WA，0则PE（输出格式错误）

通过rusage 统计用户CPU时间总量、系统CPU时间总量计算得到submit总的运行时间，如果大于要求的时间，submit->result=TLE;最后在数据库中保存submit运行结果。

**系统安全性：**

**内存时间,系统调用的限制：**

**内存：**我使用了rlimit进行控制，同时也是方便在运行结束后获得内存使用情况的数据，不过有一个缺点就是如果声明了超大的空间但是从未访问过就会不被统计进来，但是观察到很多ACM或者OI比赛也都是这样处理的，所有这也不算是一个问题。

**时间：**首先同样也是使用rlimit进行CPU时间控制，注意它只能控制CPU时间，不能控制实际运行时间，所以像是sleep或者IO阻塞之类的情况是没有办法的，所以还在额外添加了一个alarm来进行实际的限制。按照大部分比赛的管理，最终统计的时间是 CPU 时间。

1. **系统调用：**由于在线评测系统只要是为了对用户程序的算法优劣进行评测，因而不需要使用创建目录、 文件、 建立 Socket 等在普通应用程序中比较常见的函数，只需要通过在之后ptrace子进程时，kill掉调用不安全的系统调用~

由于在线评测系统是直接编译链接用户提交的程序然后执行，有可能会出现用户编写恶意程序来影响性能，可能出现的恶意情况如下:

( 1) 使用malloc不断地分配大内存，耗尽系统可用内存;

( 2) 使用无意义的while( 1) 死循环，使CPU一直忙碌;

( 3) 不停地向硬盘写大文件，耗尽硬盘存储空间;

( 4) 不停地调用 fork 产生子进程， 消耗大量的系统资源;

针对以上恶意情况，需要对各种资源进行相应限制。Linux 内核提供的 setrlimit函数，可以对进程使用的各种资源进行限制设置

Fork()与vfock()都是创建一个进程，那他们有什么区别呢？总结有以下三点区别：

1. fork()：子进程拷贝父进程的数据段，代码段

Vfork()：子进程与父进程共享数据段

2. fork()父子进程的执行次序不确定

vfork 保证子进程先运行，在调用exec 或exit 之前与父进程数据是共享的,在它调用exec或exit 之后父进程才可能被调度运行。

3. vfork()保证子进程先运行，在她调用exec 或exit 之后父进程才可能被调度运行。如果在调用这两个函数之前子进程依赖于父进程的进一步动作，则会导致死锁。