2020/3/5 - 17:00 - 63分钟

1. 自我介绍

2. gdb单步执行的原理

（简历上有一个关于程序运行的项目，本质是gdb

我一开始以为是中断的原理。。。然后就说是软件断点，汇编的话是`int 3`指令，程序执行到该指令处，就会触发中断，程序中止，回到调试程序中。

后面才意识到，问的是单步执行的原理。（这个我还真不知道。。。）就说，单步执行应该也是这个原理。。。

2. gdb内部实现

我记得Linux上有个ptrace指令（这里说错了，是有个ptrace函数，指令应该是strace。。。），gdb内部就是用这个去实现的，具体的实现我不太清楚。。。

3. 那你们是怎么调用这个gdb程序的呢

我们一开始是在gdb中写了一个socket服务器与后端进程通信的，后来发现有外国人写过调用gdb的python包，于是就用了他的，这个包本质是通过管道进行进程通信的。

4. 那这个gdb程序是一直存在于后台吗

不是，有了响应（这里又说错了，是请求。。。）才启动gdb程序的。

5. 那这个gdb程序什么时候关闭呢

我们通信使用的是WebSocket协议，WebSocket关闭的时候，会调用回调函数，我们就在那个回调函数中杀死gdb进程的。

6. 说一下ELF文件结构

包含数据段和代码段，数据段里面的话，又包含`.data`、`.bss`、`.debug`、`.symtab`等等这些。

7. 说一下.bss段是用来做什么的

用来存放未初始化的静态和全部变量。

8. 为什么未初始化还要保存呢

因为是全局的，不保存下来的话，程序就会找不到这些变量。

9. 讲一下程序加载过程

好像有一个libc_start函数，会先为程序准备内存空间，然后再将eip指向main函数入口地址，开始执行程序。前面还会其它函数，但我忘了。。。

（这里有点忘了，全靠记忆瞎说。。。其实不应该讲函数。。。

10. 了解unordered_map吗？说一下原理

我记得普通map是红黑树实现的，unordered_map应该是哈希表实现的。

11. 讲讲哈希表

通过哈希函数，将key映射到一个内存地址，然后找到value的值。

12. 那怎么解决冲突

有一种是，通过链表的方式，如果映射到同一个地址，就将值加到该地址对应链表中。

还有一种是建立公共缓冲区。

13. 知道有什么哈希函数吗

第一秒想到RAS加密，显然不是。。。然后，说不记得了。

14. 知道vector和list吗？什么时候用vector，什么时候用list

假如插入删除比较多的话，就用list。

那什么时候都用list不行吗？

vector索引比较快，而list查找不太方便。

15. vector为什么插入比较慢

因为插入一个就要往后平移。

那尾部插入是不是没问题？

尾部插入相对快一点。

16. 假如有一百万个数据追加到一个vector中，系统还是会很慢，知道为什么吗（有点记不清了

因为vector有个最大容量，如果超过这个最大容量时，会重新分配空间，可能是因为这个问题导致很慢。

那重新分配空间的大小是多少呢？

应该是原来的两倍。

那怎么解决这个问题呢？

重新分配空间的时候，直接分配十倍或者一百倍这样。

但是你不能改vector源码啊。

我记得有个resize()函数，好像可以直接改变vector的容量。

17. MySQL原理知道吗

B+树

为什么要用B+树来实现呢?

因为数据比较大，如果用普通平衡二叉树的话，会导致高度很大，而B+树则不会。（这里答偏了。。。

那为什么stl中的map用红黑树树，而不用B+树?

（支支吾吾。。。

你对数据库还不怎么熟吗?

不太熟，因为我们这学期才开始学数据库。

那你说说数据库的四大特性是什么?

一致性、隔离性、持久性（原子性一直没想起来。。。

你们不是开始学了吗?

我们这学期还没开课。。。

18. 计算机网络学过吗

也是这学期学。但我之前看过书，了解一些应用层和运输层的知识。

19. HTTP了解吗？怎么获取一个文件

用户先请求，然后会把文件放在响应的数据体里面。

20. 那HTTP断点续传知道吗

啥？不知道。。。

那你HTTP知道什么?

请求报文格式，响应报文格式什么的。

21. 那你说说HTTP请求响应的报文格式

请求报文的话，包含三部分，最开始一行包括请求方法、HTTP版本、URL。然后中间的首部行包括一些首部字段，如：cookie、host、user agent。最后是请求体，如果是POST的话，就会有数据在里面。

响应也包含三部分，最开始一行包括响应状态码、HTTP版本。然后中间也是首部的一些字段。最后是返回的数据体。

（有些名词可能有偏差。。。记不太清了当时。。。

22. 说一下有那些状态码

状态码有1、2、3、4、5开头的。2开头表示成功，3开头表示重定向，4开头表示客户端错误，5开头表示服务器错误。（漏了1。。。

23. 知道长连接吗

知道。HTTP1.0是每个请求完成后，就关闭TCP连接，假如同时有多个请求的话，会消耗一定的服务器资源。然后，HTTP1.1 改成了长连接，一段时间内的多个请求，都可以使用同一个TCP连接。

那可以并发请求吗?

HTTP2.0 中好像新增了并发请求。

你计网看的是哪本书?

《计算机网络——自顶向下方法》

24. 做题

接下来开始做题，你参加过算法竞赛吗?

没有，只参加过学校内的，只拿了个二等奖。

第一题

> 给定m个不重复的字符 [a，b，c，d]，以及一个长度为n的字符串tbcacbdata，问能否在这个字符串中找到一个长度为m的连续子串，使得这个子串刚好由上面m个字符组成，顺序无所谓，返回任意满足条件的一个子串的起始位置，未找到返回-1。比如上面这个例子，acbd，3。

看了一下，说：这个可以用map来做，集满m个就算成功，遇到map中重复的，就重新开始。

那你这个时间复杂度是多少?

O(m*n)

那可以优化吗?

可以用滑动窗口算法来做，就是前后两个指针。如果后一个指针遇到m中存在且map中不存在的，就加入map，向后移动；如果遇到map中存在的，就移动前指针，直至遇到重复字符；如果遇到m中不存在的，就从当前位置重新开始。

你是做过类似的题?

好像做过，但题目不太一样。

那我把这道题改进一下：

> 给定m个可能重复的字符 [a，c，c，d]呢?

想了一会会，说：还是用map和滑动窗口来做。map的key还是字符，但value改为一个结构体，一个属性记录可重复的最大次数，一个属性记录当前重复的次数。

第二题

那我给你出道难点的题：

> 给定一个二进制数组，找到含有相同数量的 0 和 1 的最长连续子数组（的长度）。

一开始，我以为还是用滑动窗口，但发现不对，马上改口，然后继续想。

想着想着，他突然说，我看时间不够了，给你换道简单一点的。

第三题

给定一个由 '1'（陆地）和 '0'（水）组成的的二维网格，计算岛屿的数量。一个岛被水包围，并且它是通过水平方向或垂直方向上相邻的陆地连接而成的。你可以假设网格的四个边均被水包围。

```
5 6
1 1 1 1 0 0
1 1 0 1 0 0
1 1 0 0 1 1
0 0 0 0 0 1
0 0 0 0 1 1
输出: 2

输入:
11000
11000
00100
00011
输出: 3
```

这道题我看了一下说，可以用BFS做，然后大致讲了一下思路。

他说，OK，那你觉得第一和第三道题哪道容易一些，选一题写代码。

我说，第一题吧，BFS要判断方向，感觉代码写起来有点多。

那你就做第三题吧，他说完还在笑。。。

我。。。

这样，我给你20分钟写一下，我先去开个会。

写代码

```cpp
#include <cstdio>
#include <vector>
#include <queue>

using namespace std;

struct Pos {
    int i;
    int j;
    Pos(int a, int b): i(a), j(b) {}
};

int main() {
    int m,n;
    scanf("%d%d", &m, &n);
    vector<vector<int>> graph;
    for (int i=0; i<m; i++) {
        vector<int> temp;
        for (int j=0; j<n; j++) {
            int num;
            scanf("%d", &num);
            temp.push_back(num);
        }
        graph.push_back(temp);
    }
```

```
    vector<vector<bool>> flag;
    for (int i=0; i<m; i++) {
        vector<bool> temp;
        for (int j=0; j<n; j++) {
            temp.push_back(false);
        }
        flag.push_back(temp);
    }

    int ans = 0;
    for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++) {
            if (graph[i][j] == 1 && flag[i][j] == false) {
                ans++;
                queue<Pos> q;
                q.push(Pos(i,j));
                while (!q.empty()) {
                    Pos front = q.front();
                    printf("front.i:%d, front.j:%d\n", front.i, front.j);
                    q.pop();
                    // 向下找
                    if (front.i+1 < m && graph[front.i+1][front.j] == 1 &&
flag[front.i+1][front.j] == false) {
                        flag[front.i+1][front.j] = true;
                        q.push(Pos(front.i+1, front.j));
                    }
                    // 向右找
                    if (front.j+1 < n && graph[front.i][front.j+1] == 1 &&
flag[front.i][front.j+1] == false) {
                        flag[front.i][front.j+1] = true;
                        q.push(Pos(front.i, front.j+1));
                    }
                }
            }
        }
    }

    printf("%d\n", ans);
}
```

写完刚运行第一次，发现结果不对。刚准备调试，他就回来了，说，“时间差不多了，今天就到这吧。”。

我说，“写是写好了，但还有点问题。”

他说，“没事，我回去看一下思路。”

然后，我就提交了，面试结束。

面完自己调试了一下，发现是少考虑了向左走的情况（以为可以免去向上向左的寻找。。。