

# OS大作业（救救共享）

## 第一题

### ①准备

```
1 | mkdir score && touch {score/1.txt, score/2.txt, score/3.txt}
```

内容如下：

#### 1.txt

1	李娜 郑州大学 2021级 1班 85
2	张伟 郑州大学 2021级 1班 72
3	刘波 郑州大学 2021级 1班 93
4	王芳 郑州大学 2021级 1班 68
5	陈丽 郑州大学 2021级 1班 78
6	杨洋 郑州大学 2021级 1班 91
7	赵静 郑州大学 2021级 1班 87
8	孙强 郑州大学 2021级 1班 58
9	周浩 郑州大学 2021级 1班 82
10	吴婷 郑州大学 2021级 1班 79

#### 2.txt

1	萧燕 郑州大学 2021级 2班 88
2	方莹 郑州大学 2021级 2班 76
3	高翔 郑州大学 2021级 2班 95
4	何敏 郑州大学 2021级 2班 70
5	江燕 郑州大学 2021级 2班 81
6	孔涛 郑州大学 2021级 2班 90
7	李莉 郑州大学 2021级 2班 73
8	马超 郑州大学 2021级 2班 55
9	宁娜 郑州大学 2021级 2班 84

#### 3.txt

1	彭辉 郑州大学 2021级 3班 92
2	齐东 郑州大学 2021级 3班 74
3	任梅 郑州大学 2021级 3班 89
4	沈磊 郑州大学 2021级 3班 52
5	陶杰 郑州大学 2021级 3班 80
6	田芳 郑州大学 2021级 3班 86
7	王维 郑州大学 2021级 3班 75
8	吴雯 郑州大学 2021级 3班 69

## ②代码

```

1 #!/bin/zsh
2
3 # -----
4 # @file: score.sh
5 # @brief: 合并分数文件，并输出年级排名前十、各分数区间人数、平均分
6 # @author: M0rtzz
7 # @date: 2024-04-30
8 # -----
9
10 rm -f ./score/sorted_scores.txt
11
12 # 按分数逆序排序
13 {
14     for file in ./score/*.txt; do
15         cat "${file}" | awk '{print $1, $4, $5}'
16         # 文件之间添加换行符
17         printf "\n"
18     done
19 } | sort -k3 -rn >./score/sorted_scores.txt
20
21 # 输出年级排名前十
22 echo "年级排名前十："
23 head -n 10 ./score/sorted_scores.txt
24
25 # 统计各分数区间人数
26 echo "60以下人数："
27 awk '$3 < 60 {count++} END {print count}' ./score/sorted_scores.txt
28
29 echo "60-70人数："
30 awk '$3 >= 60 && $3 < 70 {count++} END {print count}'
31     ./score/sorted_scores.txt
32
33 echo "70-80人数："
34 awk '$3 >= 70 && $3 < 80 {count++} END {print count}'
35     ./score/sorted_scores.txt
36
37 echo "80-90人数："
38 awk '$3 >= 80 && $3 < 90 {count++} END {print count}'
39     ./score/sorted_scores.txt
40
41 # 平均分
42 echo -n "平均分："
43 awk '{sum+=$3; count++} END {print sum/count}' ./score/sorted_scores.txt
44
45 # 删除后三行的换行符
46 sed -i '$d' ./score/sorted_scores.txt
47 sed -i '$d' ./score/sorted_scores.txt
48 sed -i '$d' ./score/sorted_scores.txt

```

## ③运行

```
1 | sudo chmod +x score.sh
2 | ./score.sh
```

The terminal window shows the command `./score.sh` being run. The output displays student names, their scores, and class information. It also shows the average score and the number of students in different score ranges (60-70, 70-80, 80-90, 90-100). Finally, it lists the files generated by the script: `1.txt`, `2.txt`, `3.txt`, and `sorted_scores.txt`.

```
m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design
m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design 100x29
base 20:52:59
> ./score.sh
年级排名前十:
高翔 2班 95
刘波 1班 93
彭辉 3班 92
杨洋 1班 91
孔涛 2班 90
任梅 3班 89
萧燕 2班 88
赵静 1班 87
田芳 3班 86
李娜 1班 85
60以下人数:
6
60-70人数:
2
70-80人数:
8
80-90人数:
9
90-100人数:
5
平均分: 70.7333
base 20:53:40
> ls score
1.txt 2.txt 3.txt sorted_scores.txt
base 20:53:47
> |
```

## 第二题

### ①安装LLVM工具集（使用清华源）及依赖库

Reference: <https://mirrors.tuna.tsinghua.edu.cn/help/llvm-apt/>

```
1 | wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key | sudo apt-key add -
```

```
1 | echo -e "deb [arch=amd64] https://mirrors.tuna.tsinghua.edu.cn/llvm-apt/focal/
  llvm-toolchain-focal main\n# deb-src [arch=amd64]
  https://mirrors.tuna.tsinghua.edu.cn/llvm-apt/focal/ llvm-toolchain-focal
  main" | sudo tee /etc/apt/sources.list.d/llvm-apt.list > /dev/null && sudo cp
  /etc/apt/sources.list.d/llvm-apt.list /etc/apt/sources.list.d/llvm-
  apt.list.save
```

```
1 | sudo apt update -y && sudo apt upgrade -y && sudo apt install clang clangd
  llvm clang-format liblldb-19-dev
```

```
1 | clang --version
```

```
m0rtzz@m0rtzz:~$ clang --version
Ubuntu clang version 19.0.0 (++20240414053049+b8d0cba14bcf-1~exp1~20240414173206.993)
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/lib/llvm-19/bin
m0rtzz@m0rtzz:~/mnt/e/M实/大三下/操作系统/kernel_source$
```

## ②构建工具

```

1 CC = clang
2 CFLAGS = -std=gnu11 -Wall -g -O2 -pthread # 避免使用-std=c11, 否则将无法使用一些必要的函数和类型
3 SRC_FILES := $(wildcard *.c)
4 OUT_DIR := out
5 OUT_FILES := $(patsubst %.c,$(OUT_DIR)/%.out,$(SRC_FILES))
6
7 all: $(OUT_FILES)
8
9 $(OUT_DIR)/%.out: %.c | $(OUT_DIR)
10      $(CC) $< -o $@ $(CFLAGS)
11
12 $(OUT_DIR):
13      mkdir -p $(OUT_DIR)
14
15 .PHONY: clean
16
17 clean:
18      rm -rf $(OUT_DIR)
```

## ③代码

```

1 /**
2 * @file pv.c
3 * @brief 生产者、计算者、消费者问题
4 * @author M0rtzz E-mail : m0rtzz@outlook.com
5 * @version 1.0
6 * @date 2024-04-30
7 *
8 */
9
10 #include <time.h>
11 #include <stdio.h>
12 #include <ctype.h>
```

```

13 #include <unistd.h>
14 #include <stdlib.h>
15 #include <stdbool.h>
16 #include <pthread.h>
17 #include <semaphore.h>
18
19 #define BUFFER_SIZE 5
20 #define THREAD_NUM 3
21
22 /**
23 * @brief 两缓冲区
24 */
25 char buffer_1[BUFFER_SIZE];
26 char buffer_2[BUFFER_SIZE];
27
28 /**
29 * @brief empty_i 表示 buffer_i 中空闲位置信号量, full_i 表示 buffer_i 中填充数据
30 信号量
31 */
32 sem_t empty_1, full_1, empty_2, full_2;
33
34 /**
35 * @brief
36 * @param arg
37 * @return void*
38 */
39 void *producerFunc(void *arg)
40 {
41     while (true)
42     {
43         // 初始化随机数种子
44         srand((unsigned int)time(NULL));
45         // 生成随机小写字母
46         char item = 'a' + rand() % 26;
47
48         // P
49         sem_wait(&empty_1);
50         // 放入 buffer_1
51         for (int i = 0; i < BUFFER_SIZE; i++)
52         {
53             if (buffer_1[i] == '\0')
54             {
55                 buffer_1[i] = item;
56                 break;
57             }
58
59             // V
60             sem_post(&full_1);
61
62             sleep(1);
63         }
64
65         return NULL;
66     }
67

```

```

68 /**
69 * @brief 计算者
70 * @param arg
71 * @return void*
72 */
73 void *calculatorFunc(void *arg)
74 {
75     while (true)
76     {
77         // P
78         sem_wait(&full_1);
79         // 从 buffer_1 中取出字母，转换为大写字母后放入 buffer_2
80         for (int i = 0; i < BUFFER_SIZE; i++)
81         {
82             if (buffer_1[i] != '\0')
83             {
84                 buffer_2[i] = toupper(buffer_1[i]);
85                 buffer_1[i] = '\0';
86                 break;
87             }
88         }
89
90         // V
91         sem_post(&empty_1);
92         sem_post(&full_2);
93
94         sleep(1);
95     }
96
97     return NULL;
98 }
99
100 /**
101 * @brief 消费者
102 * @param arg
103 * @return void*
104 */
105 void *consumerFunc(void *arg)
106 {
107     while (true)
108     {
109         // P
110         sem_wait(&full_2);
111         // 从 buffer_2 中取出字符并打印到屏幕上
112         for (int i = 0; i < BUFFER_SIZE; i++)
113         {
114             if (buffer_2[i] != '\0')
115             {
116                 printf("%c\n", buffer_2[i]);
117                 buffer_2[i] = '\0';
118                 break;
119             }
120         }
121
122         // V
123         sem_post(&empty_2);

```

```

124         sleep(1);
125     }
126
127     return NULL;
128 }
129
130
131 signed main()
132 {
133     pthread_t threads[THREAD_NUM];
134
135     // 初始化信号量
136     sem_init(&empty_1, 0, BUFFER_SIZE);
137     sem_init(&full_1, 0, 0);
138     sem_init(&empty_2, 0, BUFFER_SIZE);
139     sem_init(&full_2, 0, 0);
140
141     // 创建线程
142     pthread_create(&threads[0], NULL, producerFunc, NULL);
143     pthread_create(&threads[1], NULL, calculatorFunc, NULL);
144     pthread_create(&threads[2], NULL, consumerFunc, NULL);
145
146     // 等待线程结束
147     for (int i = 0; i < THREAD_NUM; ++i)
148         pthread_join(threads[i], NULL);
149
150     // 销毁信号量
151     sem_destroy(&empty_1);
152     sem_destroy(&full_1);
153     sem_destroy(&empty_2);
154     sem_destroy(&full_2);
155
156     return EXIT_SUCCESS;
157 }
```

## ④运行

```

1 | make all
2 | ./out/pv.out
```

---

## 第三题

---

### ①代码（构建工具同上）

```

1 /**
2 * @file socket_server.c
3 * @brief 服务器
4 * @author M0rtzz E-mail : m0rtzz@outlook.com
5 * @version 1.0
6 * @date 2024-05-01
7 *
8 */
9
```

```

10 #include <stdio.h>
11 #include <string.h>
12 #include <stdbool.h>
13 #include <stdlib.h>
14 #include <signal.h>
15 #include <unistd.h>
16 #include <pthread.h>
17 #include <semaphore.h>
18 #include <arpa/inet.h>
19 #include <sys/types.h>
20 #include <sys/socket.h>
21 #include <netinet/in.h>
22 #define LinkNum 5 // 连接数
23
24 // 分别记录服务器端的套接字与连接的多个客户端的套接字
25 int client_sockfd[LinkNum];
26 // 命名套接字
27 int server_sockfd = -1;
28 // 当前连接数
29 int curLink = 0;
30 // 表示连接数的资源信号量
31 sem_t mutex;
32 // 服务器端发送消息缓冲区
33 char stopmsg[100];
34
35 int success_client_num = 0;
36
37 int secret_num = 0;
38
39 /**
40 * @brief 服务器与客户端的收发通信函数，n为连接数组序号
41 * @param n
42 */
43 void rcv_snd(int n)
44 {
45     int retval;
46     char recv_buf[1024];
47     char send_buf[1024];
48     pthread_t tid;
49     tid = pthread_self();
50
51     printf("服务器线程id=%lu使用套接字%d与客户机对话开始...\n", tid,
52           client_sockfd[n]);
53
54     // 发送随机数给客户端
55     // sprintf(send_buf, "答案是%d", secret_number);
56     // write(client_sockfd[n], send_buf, strlen(send_buf));
57
58     do
59     {
60         memset(recv_buf, 0, 1024);
61         int rcv_num = read(client_sockfd[n], recv_buf, sizeof(recv_buf));
62         if (rcv_num > 0)
63         {
64             int guess = atoi(recv_buf); // 将客户端的输入转换为整数

```

```

65     if (guess < secret_num)
66     {
67         strcpy(send_buf, "小了");
68     }
69     else if (guess > secret_num)
70     {
71         strcpy(send_buf, "大了");
72     }
73     else
74     {
75         strcpy(send_buf, "猜对了");
76         ++success_client_num;
77         write(client_sockfd[n], send_buf, strlen(send_buf));
78         break;
79     }
80
81     write(client_sockfd[n], send_buf, strlen(send_buf));
82 }
83 } while (true);
84
85 printf("服务器线程id=%lu与客户机对话结束\n", tid);
86 close(client_sockfd[n]);
87 // 关闭服务器监听套接字
88 close(server_sockfd);
89 client_sockfd[n] = -1;
90 curLink--;
91 printf("当前连接数为: %d\n", curLink);
92 sem_post(&mutex);
93 pthread_exit(&retval);
94 }
95
96 signed main()
97 {
98     // 初始化随机数生成器
99     srand(time(NULL));
100
101    // 生成1到100的随机数
102    secret_num = rand() % 100 + 1;
103
104    printf("答案是: %d\n", secret_num);
105
106    socklen_t client_len = 0;
107    // 服务器端协议地址
108    struct sockaddr_in server_addr;
109    // 客户端协议地址
110    struct sockaddr_in client_addr;
111    // 连接套接字数组循环变量
112    int i = 0;
113    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
114    // 指定网络套接字
115    server_addr.sin_family = AF_INET;
116    // 接受所有IP地址的连接
117    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
118    // 绑定到9736端口
119    server_addr.sin_port = htons(9736);

```

```

120     bind(server_sockfd, (struct sockaddr *)&server_addr,
121     sizeof(server_addr)); // 协议套接字命名为server_sockfd
122     printf("1、服务器开始listen...\n");
123     /*创建连接数最大为LinkNum的套接字队列，监听命名套接字，listen不会阻塞，它向内核报告
124     套接字和最大连接数*/
125     listen(server_sockfd, LinkNum);
126     // 忽略子进程停止或退出信号
127     signal(SIGCHLD, SIG_IGN);
128
129     for (i = 0; i < LinkNum; i++)
130         client_sockfd[i] = -1; // 初始化连接队列
131
132     sem_init(&mutex, 0, LinkNum); // 信号量mutex初始化为连接数
133
134     while (true)
135     {
136         // 搜索空闲连接
137         for (i = 0; i < LinkNum; i++)
138             if (client_sockfd[i] == -1)
139                 break;
140
141         // 如果达到最大连接数，则客户等待
142         if (i == LinkNum)
143         {
144             printf("已经达到最大连接数%d,请等待其它客户释放连接...\n", LinkNum);
145             // 阻塞等待空闲连接
146             sem_wait(&mutex);
147             // 被唤醒后继续监测是否有空闲连接
148             continue;
149         }
150
151         client_len = sizeof(client_addr);
152         printf("2、服务器开始accept...i=%d\n", i);
153         client_sockfd[i] = accept(server_sockfd, (struct sockaddr
154 *) &client_addr, &client_len);
155         // 当前连接数增1
156         curLink++;
157         // 可用连接数信号量mutex减1
158         sem_wait(&mutex);
159         printf("当前连接数为: %d(<=%d)\n", curLink, LinkNum);
160         // 输出客户端地址信息
161         printf("连接来自:连接套接字=%d,IP地址=%s,端口号=%d\n",
162             client_sockfd[i], inet_ntoa(client_addr.sin_addr),
163             ntohs(client_addr.sin_port));
164         pthread_create(malloc(sizeof(pthread_t)), NULL, (void *)(&recv_send),
165             (void *)i);
166     }
167
168     return EXIT_SUCCESS;
169 }
```

```

1 /**
2 * @file socket_client.c
3 * @brief 客户机
4 * @author M0rtzz E-mail : m0rtzz@outlook.com
```

```
5 * @version 1.0
6 * @date 2024-05-01
7 *
8 */
9
10 #include <stdio.h>
11 #include <string.h>
12 #include <stdbool.h>
13 #include <stdlib.h>
14 #include <signal.h>
15 #include <unistd.h>
16 #include <pthread.h>
17 #include <semaphore.h>
18 #include <arpa/inet.h>
19 #include <sys/types.h>
20 #include <sys/socket.h>
21 #include <netinet/in.h>
22
23 signed main()
24 {
25     // 客户端套接字描述符
26     int sockfd;
27     int len = 0;
28     struct sockaddr_in address; // 套接字协议地址
29     // 发送消息缓冲区
30     char snd_buf[1024];
31     // 接收消息缓冲区
32     char rcv_buf[1024];
33     int result;
34     // 接收消息长度
35     int rcv_num;
36     // 客户进程标识符
37     pid_t cpid;
38     sockfd = socket(AF_INET, SOCK_STREAM, 0);
39
40     if (sockfd < 0)
41     {
42         perror("客户端创建套接字失败! \n");
43         return EXIT_FAILURE;
44     }
45
46     // 使用网络套接字
47     address.sin_family = AF_INET;
48     address.sin_addr.s_addr = inet_addr("127.0.0.1"); // 服务器地址
49     // 服务器所监听的端口
50     address.sin_port = htons(9736);
51
52     if (inet_aton("127.0.0.1", &address.sin_addr) < 0)
53     {
54         printf("inet_aton error.\n");
55         return -EXIT_FAILURE;
56     }
57
58     len = sizeof(address);
59     // 获取客户进程标识符
60     cpid = getpid();
```

```

61     printf("1. 客户机%d开始connect服务器...\\n", cpid);
62     result = connect(sockfd, (struct sockaddr *)&address, len);
63
64     if (result == -1)
65     {
66         perror("客户机connect服务器失败!\\n");
67         exit(EXIT_FAILURE);
68     }
69
70     printf("-----客户机%d与服务器线程对话开始...\\n", cpid);
71
72     // 客户机与服务器循环发送接收消息
73     do
74     {
75
76         printf("2.客户机%d--->服务器:sockfd=%d,请输入客户机要发送给服务器的消息: ", cpid, sockfd);
77         // 发送缓冲区清零
78         memset(snd_buf, 0, 1024);
79         scanf("%s", snd_buf); // 键盘输入欲发送给服务器的消息字符串
80         // 将消息发送到套接字
81         write(sockfd, snd_buf, sizeof(snd_buf));
82
83         if (strncmp(snd_buf, "quit", 2) == 0)
84             break; // 若发送"quit", 则结束循环, 通信结束
85
86         // 接收缓冲区清零
87         memset(rcv_buf, 0, 1024);
88         printf("客户机%d,sockfd=%d 等待服务器回应...\\n", cpid, sockfd);
89         rcv_num = read(sockfd, rcv_buf, sizeof(rcv_buf));
90         printf("客户机%d,sockfd=%d 从服务器接收的消息长度=%lu\\n", cpid, sockfd,
91         strlen(rcv_buf));
92         // 输出客户机从服务器接收的消息
93         printf("3.客户机%d<---服务器:sockfd=%d,客户机从服务器接收到的消息是:
94         %s\\n", cpid, sockfd, rcv_buf);
95
96     } while (strncmp(rcv_buf, "猜对了", 2) != 0); // 如果收到"!q", 则结束循环,
97     // 通信结束
98     printf("-----客户机%d,sockfd=%d 与服务器线程对话结束-----\\n",
99     cpid, sockfd);
100    // 关闭客户机套接字
101    close(sockfd);
102
103    return EXIT_SUCCESS;
104 }
```

## ②运行

```

1 | make all
2 | ./out/socket_server.out
3 | ./out/socket_client.out # 不多于5个
```

# README

```
./out/socket_server.out
服务器 x 客户机24268 x 客户机24258 x
m0rtzz@m0rtzz ~/De/E/OS/Workspaces/design git master ?12 base 15:14:18 ⊞
> ./out/socket_server.out
答案是：25
1、服务器开始listen...
2、服务器开始accept...i=0
当前连接数为：1(<=5)
连接来自：连接套接字号=4,IP地址=127.0.0.1,端口号=39906
2、服务器开始accept...i=1
服务器线程id=140470719756032使用套接字4与客户机对话开始...
当前连接数为：2(<=5)
连接来自：连接套接字号=5,IP地址=127.0.0.1,端口号=39908
2、服务器开始accept...i=2
服务器线程id=140470711363328使用套接字5与客户机对话开始...
服务器线程id=140470719756032与客户机对话结束
当前连接数为：1
服务器线程id=140470711363328与客户机对话结束
当前连接数为：0
```

```
m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design
服务器 x 客户机24268 x 客户机24258 x
m0rtzz@m0rtzz ~/De/E/OS/Workspaces/design git master ?12 base 15:14:26 ⊞
> ./out/socket_client.out
1、客户机24268开始connect服务器...
-----客户机24268与服务器线程对话开始...
2.客户机24268-->服务器:sockfd=3,请输入客户机要发送给服务器的消息：12
客户机24268,sockfd=3 等待服务器回应...
客户机24268,sockfd=3 从服务器接收的消息长度=6
3.客户机24268<--服务器:sockfd=3,客户机从服务器接收到的消息是：小了
2.客户机24268-->服务器:sockfd=3,请输入客户机要发送给服务器的消息：25
客户机24268,sockfd=3 等待服务器回应...
客户机24268,sockfd=3 从服务器接收的消息长度=9
3.客户机24268<--服务器:sockfd=3,客户机从服务器接收到的消息是：猜对了
-----客户机24268,sockfd=3 与服务器线程对话结束-----
m0rtzz@m0rtzz ~/De/E/0/W/design git master ?12 → 37s ✘ base 15:15:09 ⊞
```

```
m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design
服务器 x 客户机24268 x 客户机24258 x
m0rtzz@m0rtzz ~/De/E/OS/Workspaces/design git master ?12 base 15:14:27 ⊞
> ./out/socket_client.out
1、客户机24258开始connect服务器...
-----客户机24258与服务器线程对话开始...
2.客户机24258-->服务器:sockfd=3,请输入客户机要发送给服务器的消息：55
客户机24258,sockfd=3 等待服务器回应...
客户机24258,sockfd=3 从服务器接收的消息长度=6
3.客户机24258<--服务器:sockfd=3,客户机从服务器接收到的消息是：大了
2.客户机24258-->服务器:sockfd=3,请输入客户机要发送给服务器的消息：24
客户机24258,sockfd=3 等待服务器回应...
客户机24258,sockfd=3 从服务器接收的消息长度=6
3.客户机24258<--服务器:sockfd=3,客户机从服务器接收到的消息是：小了
2.客户机24258-->服务器:sockfd=3,请输入客户机要发送给服务器的消息：25
客户机24258,sockfd=3 等待服务器回应...
客户机24258,sockfd=3 从服务器接收的消息长度=9
3.客户机24258<--服务器:sockfd=3,客户机从服务器接收到的消息是：猜对了
-----客户机24258,sockfd=3 与服务器线程对话结束-----
m0rtzz@m0rtzz ~/De/E/0/W/design git master ?12 → 36s ✘ base 15:15:06 ⊞
```

## 第四题

安装编译工具链：

```
1 | sudo apt install bin86 gcc-multilib
```

### ①下载内核源码

```
1 | git clone https://kkgithub.com/JackeyLea/Linux-0.12.git linux-0.12
```

### ②源码编译安装gdb版Bochs (x86模拟器)

因不想使用`src/code/setup.sh`中的`aptitude`工具（此工具一般用于解决依赖问题，它会`autoremove`系统中的软件包），但此系统使用`apt`安装时没有遇见依赖问题，所以我将它换成了`apt`：

```
1 | # @file: src/code/setup.sh
2 | # @line: 80
3 | sudo apt update && sudo apt install libgtk2.0-dev
```

```
79 | . . . . . if [ "$1" . ] && [ "$1" . = . "-d" . ]; then
80 | . . . . . # sudo apt-get install aptitude && sudo apt install libgtk2.0-dev
81 | . . . . . sudo apt update && sudo apt install libgtk2.0-dev
82 | . . . . . ./configure --enable-gdb-stub --enable-disasm
```

```
1 | cd src/code/
2 | ./setup.sh
3 | # ./setup.sh会提示你先修改Bochs源码(oslab/bochs-2.6.9/gdbstub.cc) 在运行此脚本，修改
   | 方法在`oslab/README.txt`，按照要求修改即可
4 | ./setup.sh
```

打开(O) README.txt 保存(S) /Desktop/Experiment/OS/Workspaces/design/linux/oslab

```

5 *.bxrc 文件中的变量不要用`{}`包起来, 例如要用`${OSLAB_PATH}`, 不要用`${${OSLAB_PATH}}`。否则会找不到变量。
6
7 2. 如何使bochs、gdb调试忽略page fault信号?
8
9 Created a patch "gdbstub.cc.patch" against bochs (version CVS 20080110)
10 Bochs always tries to find out the reason of an exception, so that it can generate the
11 right signal for gdb.
12 If it fails to find a reason, bochs assigns a value GDBSTUB_STOP_NO_REASON (see bochs.h),
13 which causes
14 debug_loop() (see gdbstub.cc) to generate a signal of number 0.
15 Signal 0 is problematic to gdb, as gdb doesn't allow us to ignore it.
16 Somehow when we simulate linux, we get tons of signal 0's that seem to be caused by page
17 faults.
18 This patch makes bochs send SIGSEGV instead of signal 0, so that we can ignore it in gdb.
19
20 *** gdbstub.cc.orig Thu Oct 18 18:44:38 2007
21 --- gdbstub.cc Sat Jan 12 17:25:22 2008
22 **** static void debug_loop(void)
23 *** 489,494 ****
24 --- 489,498 ---
25 {
26     write_signal(&buf[1], SIGTRAP);
27 }
28+ else if (last_stop_reason == GDBSTUB_STOP_NO_REASON)
29+
30+ {
31+     write_signal(&buf[1], SIGSEGV);
32+
33     {
34         write_signal(&buf[1], 0);

```

纯文本 ▾ 制表符宽度: 4 ▾ 第 34 行, 第 38 列 ▾ 插入

```

487 if (last_stop_reason == GDBSTUB_EXECUTION_BREAKPOINT || 
488     last_stop_reason == GDBSTUB_TRACE)
489 {
490     write_signal(&buf[1], SIGTRAP);
491 }
492 else if (last_stop_reason == GDBSTUB_STOP_NO_REASON)
493 {
494     write_signal(&buf[1], SIGSEGV);
495 }
496 else
497 {
498     write_signal(&buf[1], 0);
499 }
500 put_reply(buf);
501 break;
502 }

```

### ③改写内核源码 (Linux-0.12/)

#### 1) include/unistd.h

```

1 // @line: 149, 150
2 #define __NR_m0rtzz 87
3 #define __NR_ashore 88

```

```

148 #define __NR_uselib 86
149 #define __NR_m0rtzz 87
150 #define __NR_ashore 88
151
152 #define __syscall0(type, name) \

```

## 2) include/linux/sys.h

```

1 // @line: 92
2 extern int sys_m0rtzz();
3 extern int sys_ashore();
4
5 fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,
6 sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,
7 sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,
8 sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,
9 sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,
10 sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,
11 sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,
12 sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,
13 sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,
14 sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,
15 sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,
16 sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,
17 sys_setreuid, sys_setregid, sys_sigsuspend, sys_sigpending, sys_sethostname,
18 sys_setrlimit, sys_getrlimit, sys_getrusage, sys_gettimeofday,
19 sys_settimeofday, sys_getgroups, sys_setgroups, sys_select, sys_symlink,
20 sys_lstat, sys_readlink, sys_uselib, sys_m0rtzz, sys_ashore};

```

```

91 extern int sys_uselib();
92 extern int sys_m0rtzz();
93 extern int sys_ashore();
94
95 fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,
96 sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,
97 sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,
98 sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,
99 sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,
100 sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,
101 sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,
102 sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,
103 sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,
104 sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,
105 sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,
106 sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,
107 sys_setreuid, sys_setregid, sys_sigsuspend, sys_sigpending, sys_sethostname,
108 sys_setrlimit, sys_getrlimit, sys_getrusage, sys_gettimeofday,
109 sys_settimeofday, sys_getgroups, sys_setgroups, sys_select, sys_symlink,
110 sys_lstat, sys_readlink, sys_uselib, sys_m0rtzz, sys_ashore};

```

## 4) kernel/system\_calls.s

```

1 # @line: 63
2 nr_system_calls = 84

```

```

62
63 |    movl    $84, %eax
64

```

## 5) kernel/m0rtzz.c

```

1 /**
2 * @file m0rtzz.c
3 * @brief 实现自定义的系统调用函数
4 * @author M0rtzz
5 * @version 1.0
6 * @date 2024-04-30
7 */
8
9 #include <errno.h>
10 #include <string.h>
11 #include <asm/segment.h>
12
13 char msg[30]; // 全局变量，用于存储用户传递的消息
14
15 /**
16 * @brief 实现 `sys_m0rtzz` 系统调用函数，将用户提供的字符串拷贝到内核空间
17 * @param str 用户提供的字符串指针
18 * @return 返回拷贝的字符个数，如果超过30个字符，则返回负值错误码
19 */
20 int sys_m0rtzz(const char *str)
21 {
22     int i;
23     char tmp[40]; // 临时缓冲区，用于存储从用户空间读取的字符串
24
25     // 从用户空间逐个字符读取，直到遇到结束符或者缓冲区满为止
26     for (i = 0; i < 40; i++)
27     {
28         // 从用户空间读取一个字符并存储到临时缓冲区中
29         tmp[i] = get_fs_byte(str + i);
30
31         if (tmp[i] == '\0') // 如果读取到字符串结束符，则退出循环
32             break;
33     }
34
35     // 统计读取的字符个数
36     i = 0;
37     while (i < 40 && tmp[i] != '\0')
38         i++;
39
40     int len = i;
41
42     // 如果读取的字符个数超过30个，则返回错误码

```

```

43     if (len > 30)
44         return -(EINVAL);
45
46     // 将读取的字符串拷贝到全局变量msg中
47     strcpy(msg, tmp);
48
49     // 返回拷贝的字符个数
50     return i;
51 }
52
53 /**
54 * @brief 实现 `sys_ashore` 系统调用函数，将内核空间中的消息拷贝到用户提供的缓冲区中
55 * @param str 用户提供的缓冲区指针，用于存储消息
56 * @param size 缓冲区的大小
57 * @return 返回拷贝的字符个数，如果缓冲区大小不足，则返回负值错误码
58 */
59 int sys_ashore(char *str, unsigned int size)
60 {
61     int len = 0;
62
63     // 统计全局变量msg中的字符个数
64     for (; msg[len] != '\0'; len++)
65     ;
66
67     // 如果全局变量msg中的字符个数超过了缓冲区的大小，则返回错误码
68     if (len > size)
69         return -(EINVAL);
70
71     // 将全局变量msg中的消息拷贝到用户提供的缓冲区中
72     int i;
73     for (i = 0; i < size; i++)
74     {
75         put_fs_byte(msg[i], str + i); // 将字符逐个写入用户空间
76
77         if (msg[i] == '\0') // 如果遇到字符串结束符，则退出循环
78             break;
79     }
80
81     // 返回拷贝的字符个数
82     return i;
83 }
```

## 6) kernel/Makefile

```

1 # @line: 29
2 OBJS  = sched.o sys_call.o traps.o asm.o fork.o \
3         panic.o printk.o vsprintf.o sys.o exit.o \
4         signal.o mktime.o m0rtzz.o
```

```

26
27 OBJS...= sched.o sys_call.o traps.o asm.o fork.o \
28 → panic.o printk.o vsprintf.o sys.o exit.o \
29 → signal.o mktime.o m0rtzz.o
30

```

```

1 # @line: 48
2 m0rtzz.s m0rtzz.o: m0rtzz.c ../include/asm/segment.h ../include/string.h
  ..../include/errno.h

```

```

47 ##### Dependencies:
48 | m0rtzz.s m0rtzz.o: m0rtzz.c ../include/linux/kernel.h ../include/unistd.h
49 | exit.s exit.o: exit.c ../include/errno.h ../include/signal.h \

```

## ⑤编译linux-0.12

```

1 cd oslab/
2 code run.sh # 修改脚本

```

```

1#!/bin/sh
2
3 lock_file="./hdc.img.lock"
4
5 if [ -f "${lock_file}" ]; then
6     rm "${lock_file}"
7 fi
8
9 export OSLAB_PATH=$(dirname `which $0`)
10
11 if [ ! -e "hdc.img" ]; then
12     tar -xvJf hdc.tar.xz
13 fi
14
15 if [ "$1" ] && [ "$1" = "-m" ]
16 then
17     (cd ../linux-0.12; make clean; make; cp Image ../oslab/Image)
18 elif [ "$1" ] && [ "$1" = "-g" ]
19 then
20     ${OSLAB_PATH}/bochs/bochs-gdb -q -f ${OSLAB_PATH}/bochs/bochsrc-gdb.bxrc & \
21     gdb -x ${OSLAB_PATH}/bochs/.gdbrc ../linux-0.12/tools/system
22 else
23     bochs -q -f ${OSLAB_PATH}/bochs/bochsrc.bxrc
24 fi

```

```

1 ./run.sh -m # 编译内核源码

```

## ⑤编写代码

```

1 cd oslab/
2 touch {mount.sh, umount.sh}

```

```

1 #!/usr/bin/sudo /bin/zsh
2
3 # -----
4 # @file: mount.sh
5 # @brief: 挂载文件系统, 此文件系统是linux-0.11的文件系统影像, 但不影响使用, 在此不过多赘述
6 # @author: M0rtzz
7 # @date: 2024-05-01
8 # -----
9
10 mount_folder="hdc"
11
12 if [ ! -d "${mount_folder}" ]; then
13     mkdir "${mount_folder}"
14 fi
15
16 export OSLAB_PATH=$(cd $(dirname "${BASH_SOURCE[0]}") >/dev/null && pwd)
17 mount -t minix -o loop,offset=1024 ${OSLAB_PATH}/hdc.img ${OSLAB_PATH}/hdc

```

```

1 #!/usr/bin/sudo /bin/zsh
2
3 # -----
4 # @file: umount.sh
5 # @brief: 卸载文件系统
6 # @author: M0rtzz
7 # @date: 2024-05-01
8 # -----
9
10 umount ./hdc

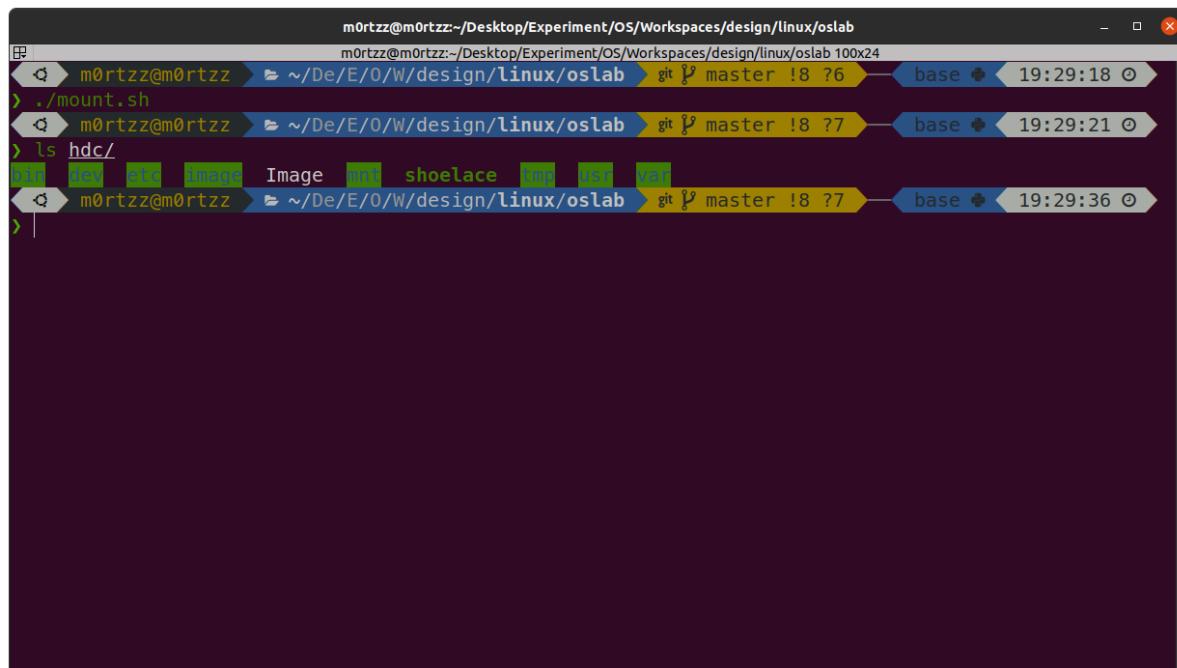
```

现在可以在本地直接访问Linux文件系统而不需要在模拟器终端中访问:

```

1 sudo chmod +x mount.sh
2 ./mount.sh

```



The screenshot shows a terminal window with the following session:

```

m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design/linux/oslab
m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design/linux/oslab 100x24
> ./mount.sh
> m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design/linux/oslab git [master] 18 ?6 base 19:29:18 ⓘ
> ls hdc/
bin dev etc image Image mnt shoelace tmp usr var
> m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design/linux/oslab git [master] 18 ?7 base 19:29:21 ⓘ
>
> m0rtzz@m0rtzz:~/Desktop/Experiment/OS/Workspaces/design/linux/oslab git [master] 18 ?7 base 19:29:36 ⓘ
>

```

之后需要修改此文件系统下的 `/usr/include/unistd.h` 并在此系统编写我们的上层C语言代码来调用我们之前编写进内核的系统调用函数：

```
1 | code hdc/
```

```
1 | // @file: hdc/usr/include/unistd.h
2 | // @line: 132, 133
3 | #define __NR_m0rtzz 87
4 | #define __NR_ashore 88
```

```
131 #define __NR_setregid 71
132 #define __NR_m0rtzz 87
133 #define __NR_ashore 88
```

```
1 /**
2 * @file hdc/usr/root/m0rtzz_1.c
3 * @brief 使用系统调用 `__NR_m0rtzz`
4 * @author M0rtzz E-mail : m0rtzz@outlook.com
5 * @version 1.0
6 * @date 2024-05-01
7 *
8 */
9
10 /**
11 * @brief 启用系统调用
12 */
13 #define __LIBRARY__
14 #include <stdio.h>
15 #include <errno.h>
16 #include <stdlib.h>
17 #include <unistd.h>
18
19 _syscall1(int, m0rtzz, const char *, str);
20
21 int main(int argc, char **argv)
22 {
23     m0rtzz(argv[1]);
24
25     return EXIT_SUCCESS;
26 }
```

```
1 /**
2 * @file hdc/usr/root/m0rtzz_2.c
3 * @brief 用系统调用 `__NR_ashore`
4 * @author M0rtzz E-mail : m0rtzz@outlook.com
5 * @version 1.0
6 * @date 2024-05-01
7 *
8 */
9
10 /**
```

```

11 * @brief 启用系统调用
12 */
13 #define __LIBRARY__
14 #include <stdio.h>
15 #include <errno.h>
16 #include <stdlib.h>
17 #include <unistd.h>
18
19 _syscall2(int, ashore, char *, str, unsigned, size);
20
21 int main()
22 {
23     char s[30];
24     ashore(s, 30);
25     printf("The string is: %s\n", s);
26
27     return EXIT_SUCCESS;
28 }
```

```

1 # @file: hdc/usr/root/Makefile
2
3 CC = gcc
4
5 all:m0rtzz_1.out m0rtzz_2.out
6
7 %.out: %.c
8     $(CC) $< -o $@
9
10 .PHONY: clean
11
12 clean:
13     rm -f *.out
```

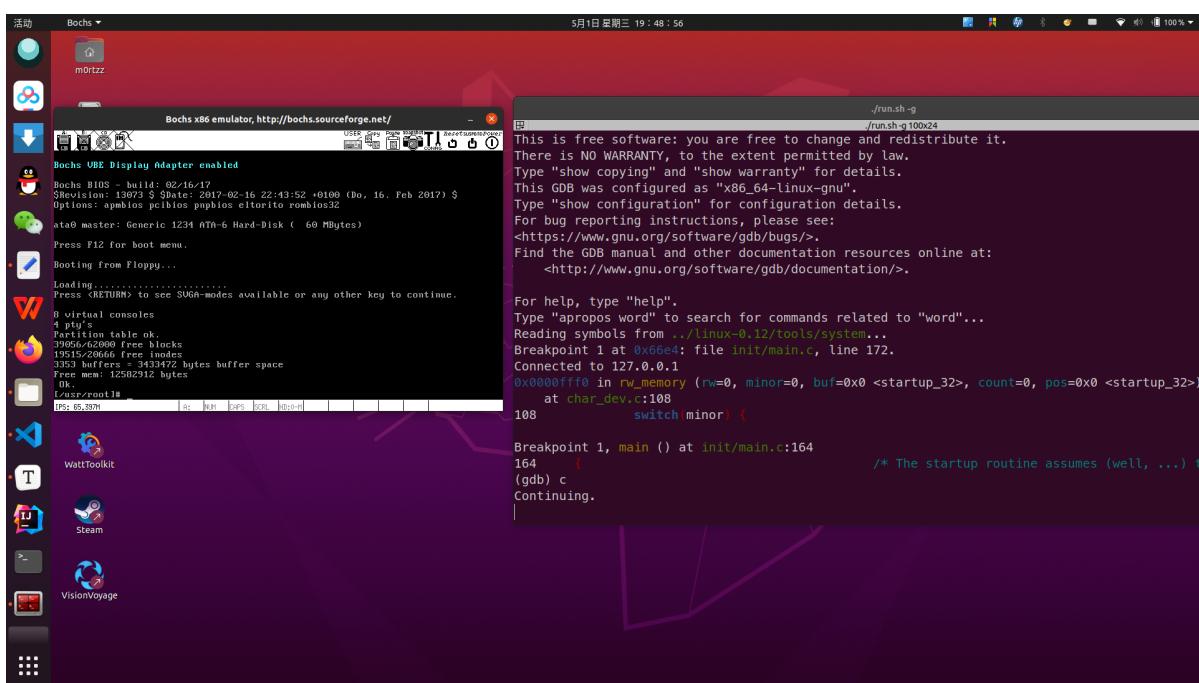
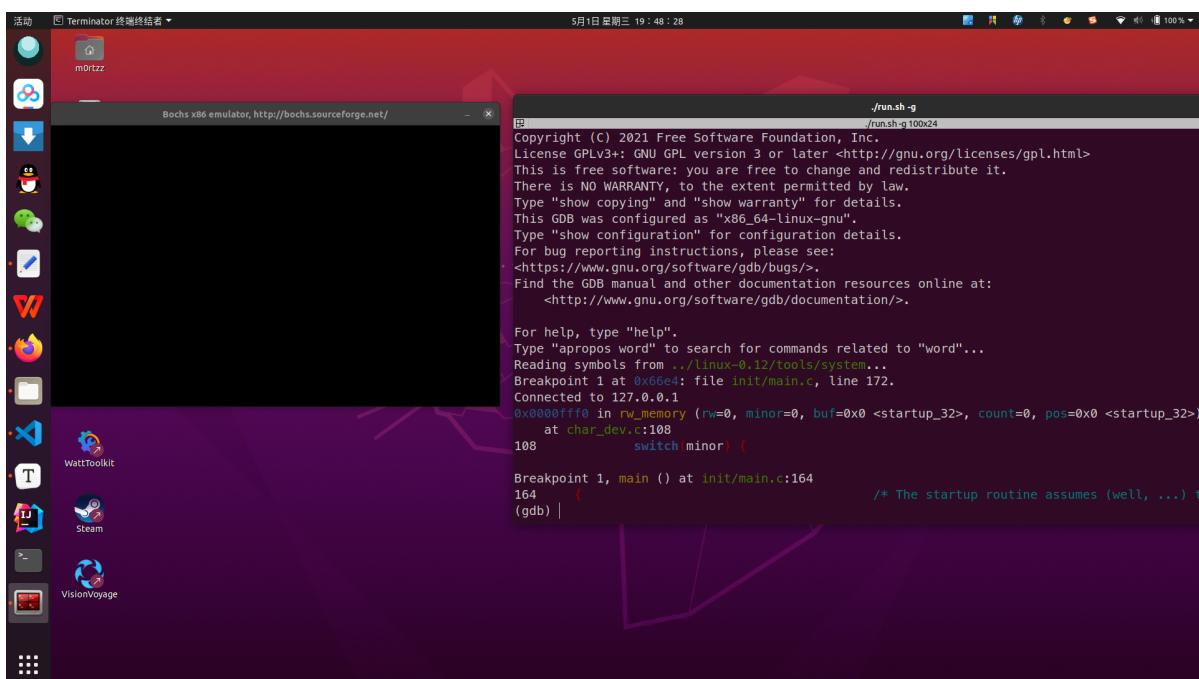
## ⑥进入linux-0.12编译并运行代码

```

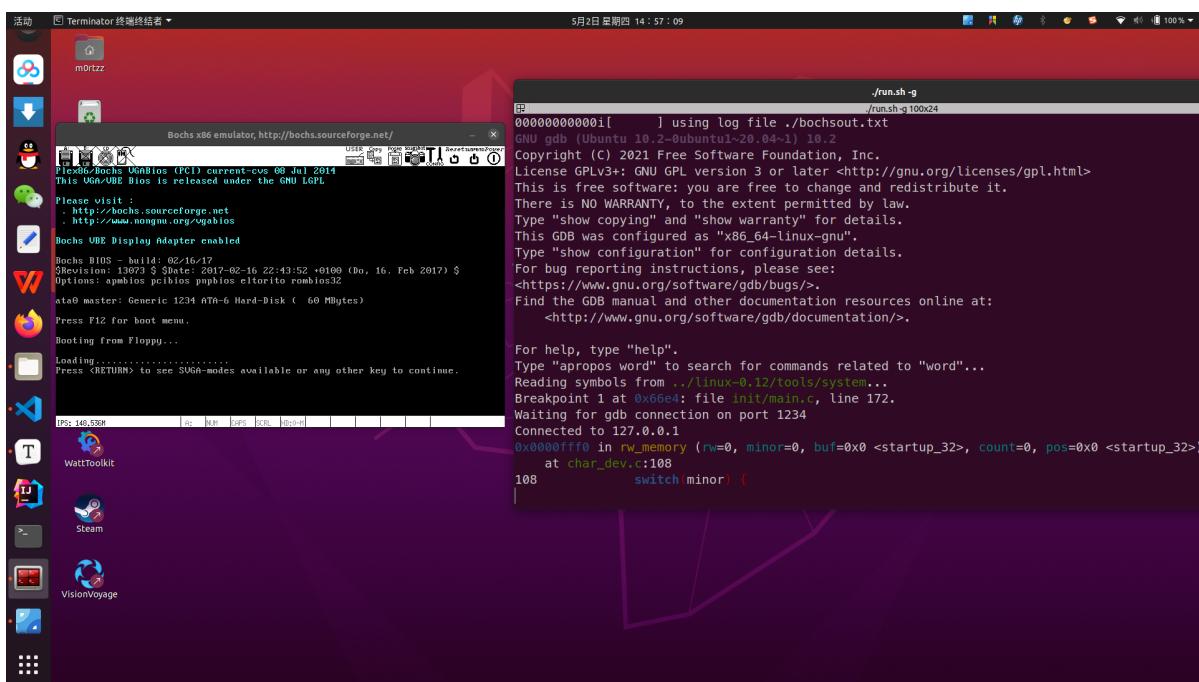
1 cd oslab/
2 ./run.sh -g # 以gdb模式进入linux-0.12
```

一开始模拟器黑屏，我们需要在终端按 **c** 键（continue）并回车进入文件系统：

# README

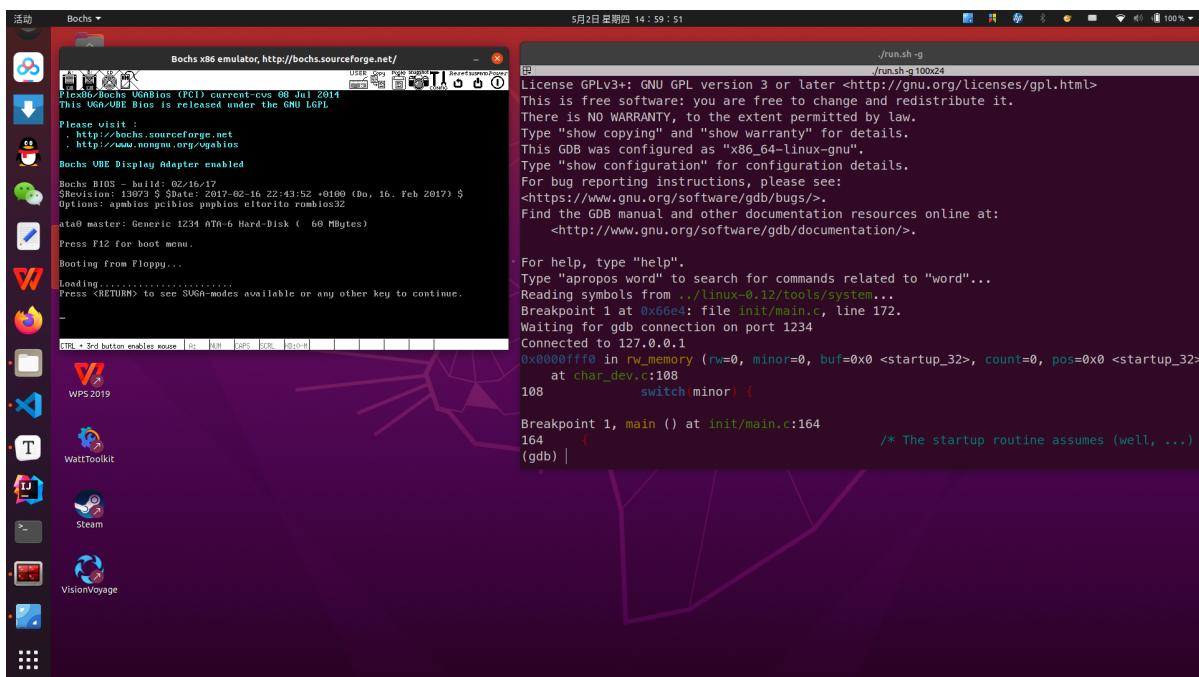


或者是这种情况（本地终端没有进入gdb模式且模拟器终端没有进入文件系统）：

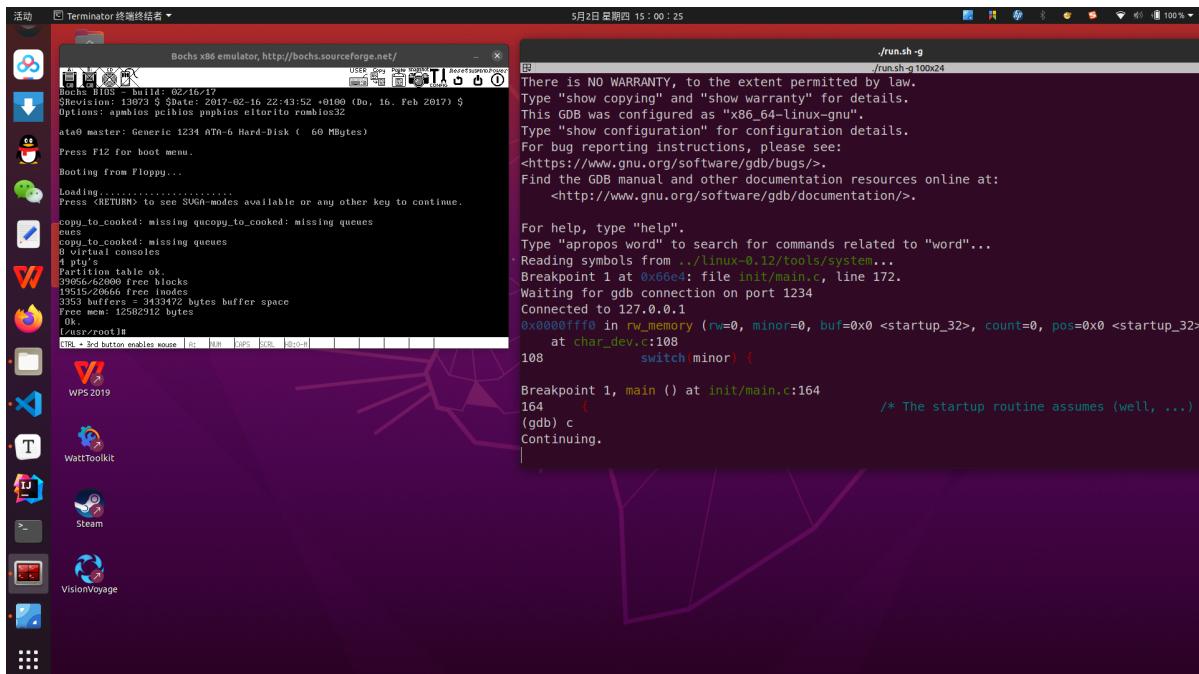


这种情况在模拟器中输入 **c** 键（此时本地终端进入gdb模式），再在本地终端输出**c**键并回车即可：

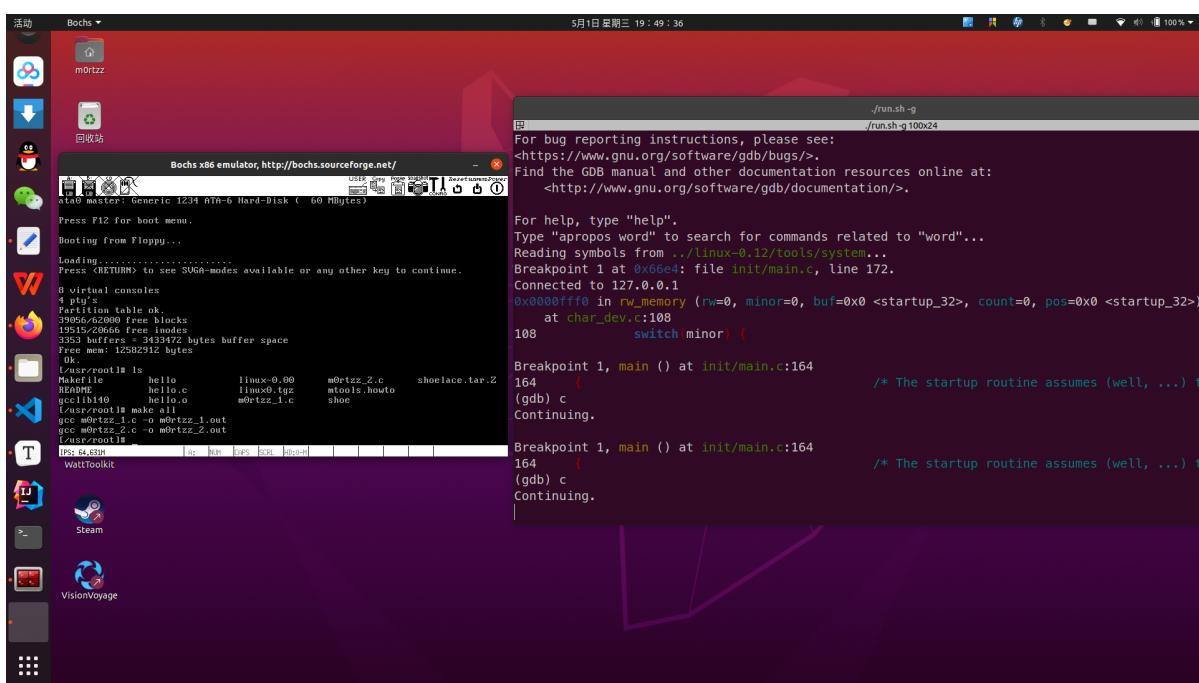
模拟器中输入 **c** 键后，本地终端进入gdb模式：



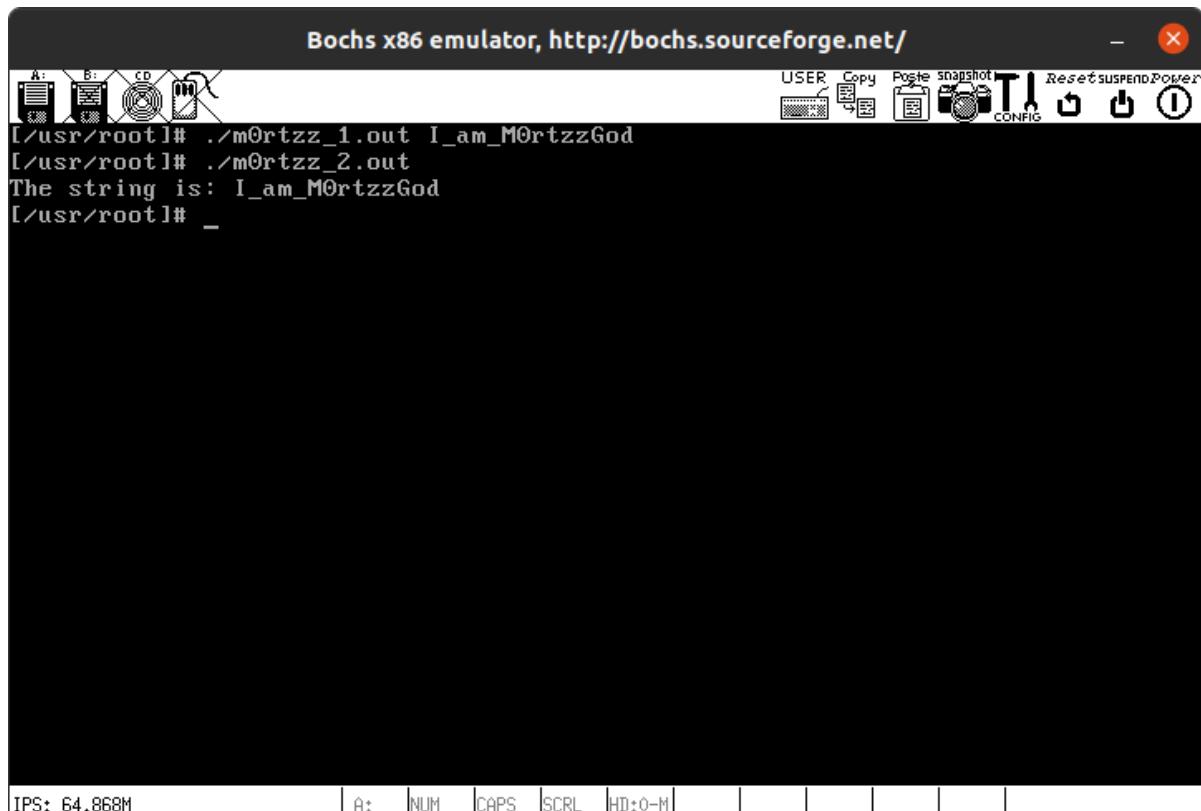
再在本地终端输出 **c** 键并回车后成功进入文件系统：



```
1 | ls
2 | make all # make的时候如果卡住不动可以在本地终端再输入 `c` 键并回车
```



```
1 | ./m0rtzz_1.out str
2 | ./m0rtzz_2.out
```



## ⑦卸载文件系统

```

1 # `ctrl + c` 并在本地终端输入 `q` 键并回车退出模拟器后再执行以下操作
2 sudo chmod +x umount.sh
3 ./umount.sh

```

