OS大作业（救救共享）

克隆该仓库：

```
git clone https://gitcode.com/MOrtzz/zzu-cs-os-design.git
cd zzu-cs-os-design
```

第一题

①准备

```
mkdir score
touch score/1.txt score/2.txt score/3.txt
```

内容如下：

`1.txt`

```
李娜 郑州大学 2021级 1班 85
张伟 郑州大学 2021级 1班 72
刘波 郑州大学 2021级 1班 93
王芳 郑州大学 2021级 1班 68
陈丽 郑州大学 2021级 1班 78
杨洋 郑州大学 2021级 1班 91
赵静 郑州大学 2021级 1班 87
孙强 郑州大学 2021级 1班 58
周浩 郑州大学 2021级 1班 82
吴婷 郑州大学 2021级 1班 79
```

`2.txt`

```
萧燕 郑州大学 2021级 2班 88
方莹 郑州大学 2021级 2班 76
高翔 郑州大学 2021级 2班 95
何敏 郑州大学 2021级 2班 70
江燕 郑州大学 2021级 2班 81
孔涛 郑州大学 2021级 2班 90
李莉 郑州大学 2021级 2班 73
马超 郑州大学 2021级 2班 55
宁娜 郑州大学 2021级 2班 84
```

`3.txt`

```
彭辉 郑州大学 2021级 3班 92
齐东 郑州大学 2021级 3班 74
任梅 郑州大学 2021级 3班 89
沈磊 郑州大学 2021级 3班 52
陶杰 郑州大学 2021级 3班 80
田芳 郑州大学 2021级 3班 86
王维 郑州大学 2021级 3班 75
吴雯 郑州大学 2021级 3班 69
```

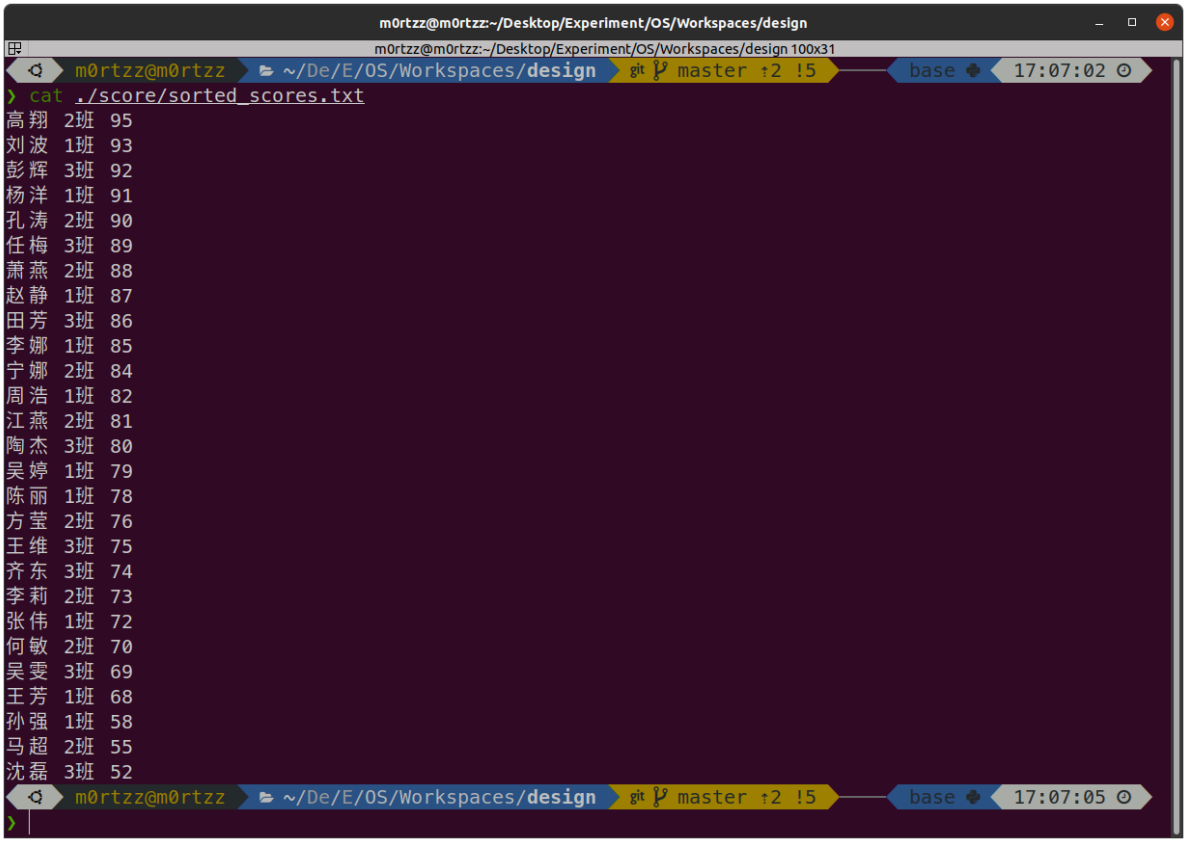
②代码

```zsh
#!/bin/zsh

# ----------------------------------------------------------------
# @file: score.sh
# @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
# @author: MOrtzz
# @date: 2024-04-30
# ----------------------------------------------------------------

rm -f ./score/sorted_scores.txt

# 按分数逆序排序
{
    for file in ./score/*.txt; do
        cat "${file}" | awk '{print $1, $4, $5}'
        # 文件之间添加换行符
        printf "\n"
    done
} | sort -k3 -rn >./score/sorted_scores.txt

# 输出年级排名前十
echo "年级排名前十: "
head -n 10 ./score/sorted_scores.txt

# 统计各分数区间人数
echo "60以下人数: "
awk '$3 < 60 {count++} END {print count}' ./score/sorted_scores.txt

echo "60-70人数: "
awk '$3 >= 60 && $3 < 70 {count++} END {print count}'
./score/sorted_scores.txt

echo "70-80人数: "
awk '$3 >= 70 && $3 < 80 {count++} END {print count}'
./score/sorted_scores.txt

echo "80-90人数: "
awk '$3 >= 80 && $3 < 90 {count++} END {print count}'
./score/sorted_scores.txt

echo "90-100人数: "
awk '$3 >= 90 && $3 <= 100 {count++} END {print count}'
./score/sorted_scores.txt

# 平均分
echo -n "平均分: "
awk '{sum+=$3; count++} END {print sum/count}' ./score/sorted_scores.txt

# 删除后三行的换行符
sed -i '$d' ./score/sorted_scores.txt
sed -i '$d' ./score/sorted_scores.txt
sed -i '$d' ./score/sorted_scores.txt
```

③运行

```
sudo chmod +x score.sh
./score.sh
```

```
cat ./score/sorted_scores.txt
```



第二题

①安装LLVM工具集（使用清华源）及依赖库

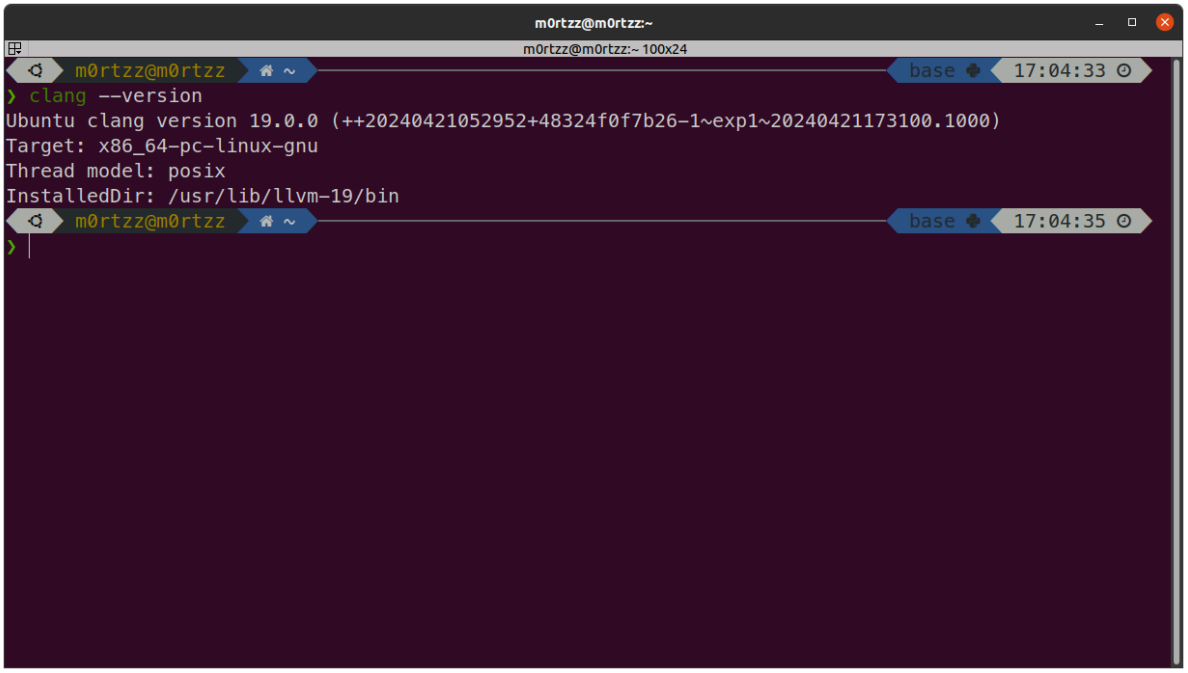
Reference: https://mirrors.tuna.tsinghua.edu.cn/help/llvm-apt/

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

```
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
```

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

```
clang --version
```



②构建工具

```
CC = clang
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
SRC_FILES := $(wildcard *.c)
OUT_DIR := out
OUT_FILES := $(patsubst %.c,$(OUT_DIR)/%.out,$(SRC_FILES))

all: $(OUT_FILES)

$(OUT_DIR)/%.out: %.c | $(OUT_DIR)
    $(CC) $< -o $@ $(CFLAGS)

$(OUT_DIR):
    mkdir -p $(OUT_DIR)

.PHONY: clean

clean:
    rm -rf $(OUT_DIR)
```

③代码

```
/**
 * @file pv.c
 * @brief 生产者、计算者、消费者问题
 * @author M0rtzz E-mail : m0rtzz@outlook.com
 * @version 1.0
 * @date 2024-04-30
 *
 */
```

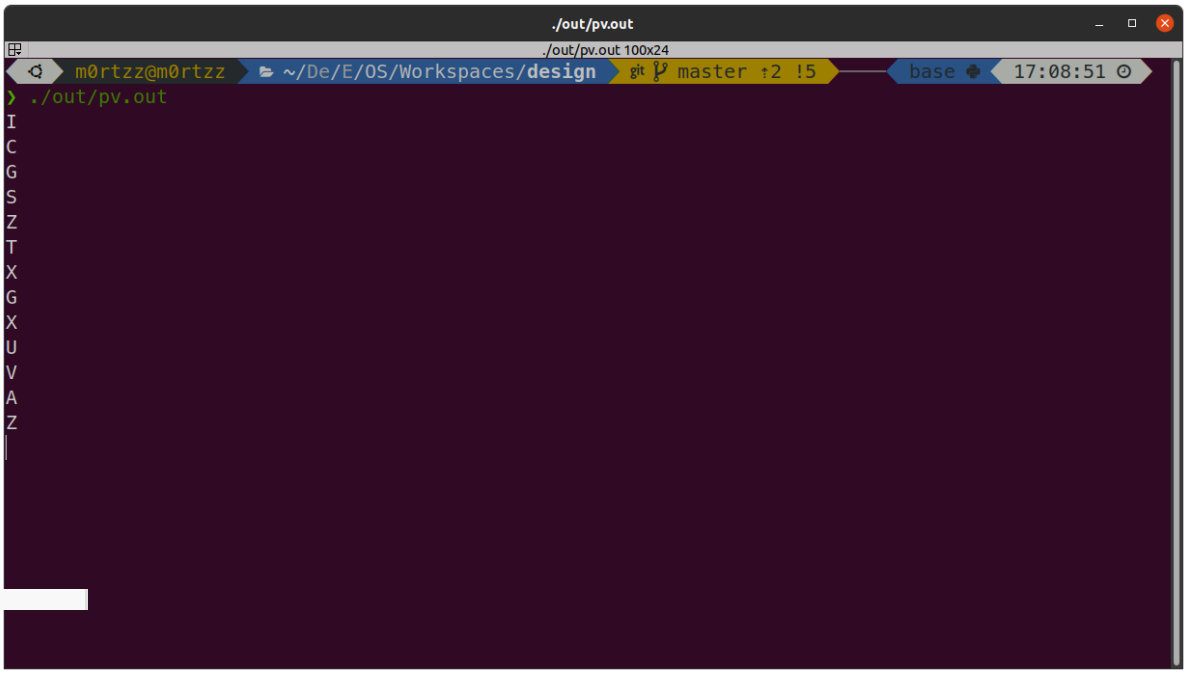
```
9
10  #include <time.h>
11  #include <stdio.h>
12  #include <ctype.h>
13  #include <unistd.h>
14  #include <stdlib.h>
15  #include <stdbool.h>
16  #include <pthread.h>
17  #include <semaphore.h>
18
19  #define BUFFER_SIZE 5
20  #define THREAD_NUM 3
21
22  /**
23   * @brief 两缓冲区
24   */
25  char buffer_1[BUFFER_SIZE];
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
    信号量
30   */
31  sem_t empty_1, full_1, empty_2, full_2;
32
33  /**
34   * @brief 生产者
35   * @param arg
36   * @return void*
37   */
38  void *producerFunc(void *arg)
39  {
40      while (true)
41      {
42          // 初始化随机数种子
43          srand((unsigned int)time(NULL));
44          // 生成随机小写字母
45          char item = 'a' + rand() % 26;
46
47          // P
48          sem_wait(&empty_1);
49          // 放入 buffer_1
50          for (int i = 0; i < BUFFER_SIZE; i++)
51          {
52              if (buffer_1[i] == '\0')
53              {
54                  buffer_1[i] = item;
55                  break;
56              }
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             }
```

```
        }

        // V
        sem_post(&empty_2);

        sleep(1);
    }

    return NULL;
}

signed main()
{
    pthread_t threads[THREAD_NUM];

    // 初始化信号量
    sem_init(&empty_1, 0, BUFFER_SIZE);
    sem_init(&full_1, 0, 0);
    sem_init(&empty_2, 0, BUFFER_SIZE);
    sem_init(&full_2, 0, 0);

    // 创建线程
    pthread_create(&threads[0], NULL, producerFunc, NULL);
    pthread_create(&threads[1], NULL, calculatorFunc, NULL);
    pthread_create(&threads[2], NULL, consumerFunc, NULL);

    // 等待线程结束
    for (int i = 0; i < THREAD_NUM; ++i)
        pthread_join(threads[i], NULL);

    // 销毁信号量
    sem_destroy(&empty_1);
    sem_destroy(&full_1);
    sem_destroy(&empty_2);
    sem_destroy(&full_2);

    return EXIT_SUCCESS;
}
```

④运行

```
make all
./out/pv.out
```



第三题

①代码（构建工具同上）

```c
/**
 * @file socket_server.c
 * @brief  服务器
 * @author M0rtzz E-mail : m0rtzz@outlook.com
 * @version 1.0
 * @date 2024-05-01
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define MAX_LINK_NUM 5 // 最大连接数

// 用已连接客户端套接字
int client_sockfd[MAX_LINK_NUM];
// 服务器套接字
int server_sockfd;
// 当前连接数
int cur_link = 0;
// 同步服务器连接数的信号量
sem_t mutex;
```

```
// 答案
int secret_num = 0;

/**
 * @brief 服务器与客户端的收发通信函数，n为连接数组序号
 * @param n
 */
void rcv_snd(int n)
{
    int retval;
    char recv_buf[1024];
    char send_buf[1024];
    pthread_t tid;
    tid = pthread_self();

    printf("服务器线程id=%lu使用套接字%d与客户机对话开始...\n", tid,
client_sockfd[n]);

    // 发送随机数给客户端
    // sprintf(send_buf, "答案是%d", secret_number);
    // write(client_sockfd[n], send_buf, strlen(send_buf));

    do
    {
        memset(recv_buf, 0, 1024);
        int rcv_num = read(client_sockfd[n], recv_buf, sizeof(recv_buf));
        if (rcv_num > 0)
        {
            int guess = atoi(recv_buf); // 将客户端的输入转换为整数

            if (guess < secret_num)
            {
                strcpy(send_buf, "小了");
            }
            else if (guess > secret_num)
            {
                strcpy(send_buf, "大了");
            }
            else
            {
                strcpy(send_buf, "猜对了");
                write(client_sockfd[n], send_buf, strlen(send_buf));
                break;
            }

            write(client_sockfd[n], send_buf, strlen(send_buf));
        }
    } while (true);

    printf("服务器线程id=%lu与客户机对话结束\n", tid);
    close(client_sockfd[n]);
    // 关闭服务器监听套接字
    close(server_sockfd);
    client_sockfd[n] = -1;
    cur_link--;
    printf("当前连接数为: %d\n", cur_link);
```

```
    sem_post(&mutex);
    pthread_exit(&retval);
}

signed main()
{
    // 初始化随机数生成器
    srand(time(NULL));

    // 生成1到100的随机数
    secret_num = rand() % 100 + 1;

    printf("答案是: %d\n", secret_num);

    socklen_t client_len = 0;
    // 服务器端协议地址
    struct sockaddr_in server_addr;
    // 客户端协议地址
    struct sockaddr_in client_addr;
    // 连接套接字数组循环变量
    int i = 0;
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    // 指定网络套接字
    server_addr.sin_family = AF_INET;
    // 接受所有IP地址的连接
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    // 绑定到9736端口
    server_addr.sin_port = htons(9736);
    bind(server_sockfd, (struct sockaddr *)&server_addr,
sizeof(server_addr)); // 协议套接字命名为server_sockfd
    printf("1、服务器开始listen...\n");
    // 创建连接数最大为MAX_LINK_NUM的套接字队列，监听命名套接字，listen不会阻塞，它向
内核报告套接字和最大连接数
    listen(server_sockfd, MAX_LINK_NUM);
    // 忽略子进程停止或退出信号
    signal(SIGCHLD, SIG_IGN);

    for (i = 0; i < MAX_LINK_NUM; i++)
        client_sockfd[i] = -1; // 初始化连接队列

    sem_init(&mutex, 0, MAX_LINK_NUM); // 信号量mutex初始化为连接数

    while (true)
    {
        // 搜寻空闲连接
        for (i = 0; i < MAX_LINK_NUM; i++)
            if (client_sockfd[i] == -1)
                break;

        // 如果达到最大连接数，则客户等待
        if (i == MAX_LINK_NUM)
        {
            printf("已经达到最大连接数%d,请等待其它客户释放连接...\n",
MAX_LINK_NUM);
            // 阻塞等待空闲连接
            sem_wait(&mutex);
```

```
            // 被唤醒后继续监测是否有空闲连接
            continue;
        }

        client_len = sizeof(client_addr);
        printf("2、服务器开始accept...i=%d\n", i);
        client_sockfd[i] = accept(server_sockfd, (struct sockaddr *)&client_addr, &client_len);
        // 当前连接数增1
        cur_link++;
        // 可用连接数信号量mutex减1
        sem_wait(&mutex);
        printf("当前连接数为: %d(<=%d)\n", cur_link, MAX_LINK_NUM);
        // 输出客户端地址信息
        printf("连接来自:连接套接字号=%d,IP地址=%s,端口号=%d\n", client_sockfd[i], inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
        pthread_create(malloc(sizeof(pthread_t)), NULL, (void *)(&rcv_snd), (void *)i);
    }

    return EXIT_SUCCESS;
}
```

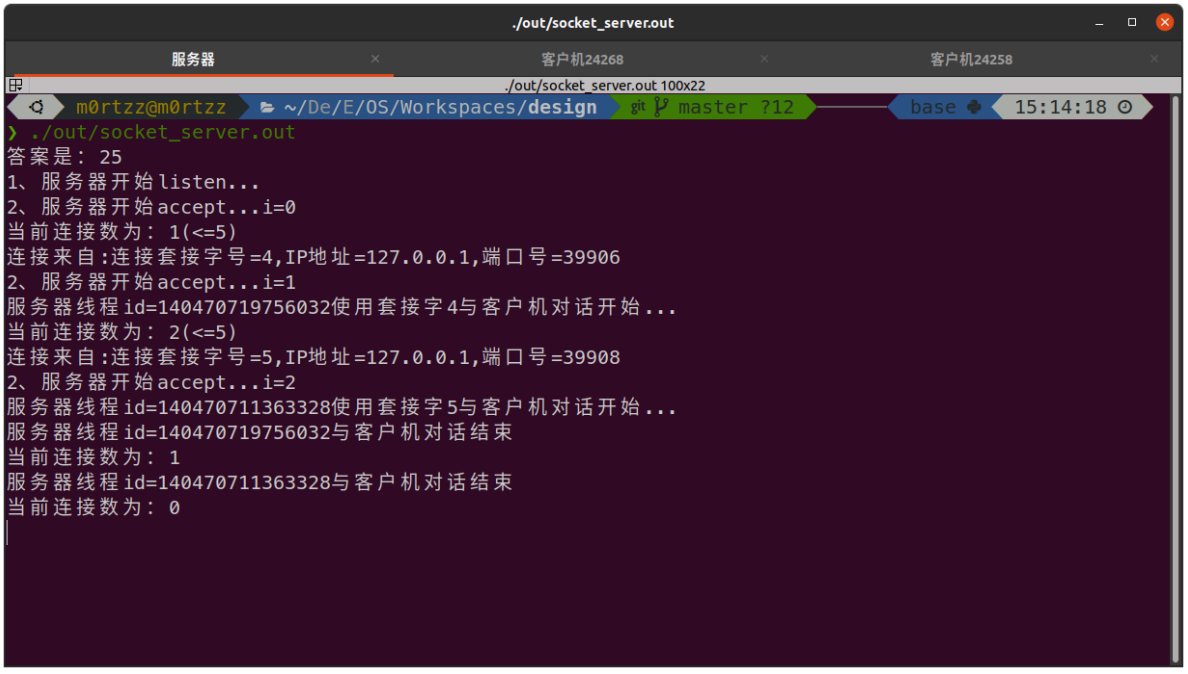
```
/**
 * @file socket_client.c
 * @brief  客户机
 * @author MOrtzz E-mail : mOrtzz@outlook.com
 * @version 1.0
 * @date 2024-05-01
 *
 */

#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

signed main()
{
    // 客户端套接字描述符
    int sockfd;
    // 地址结构体的长度
    int len = 0;
    // 套接字协议地址
    struct sockaddr_in address;
    // 发送消息缓冲区
```

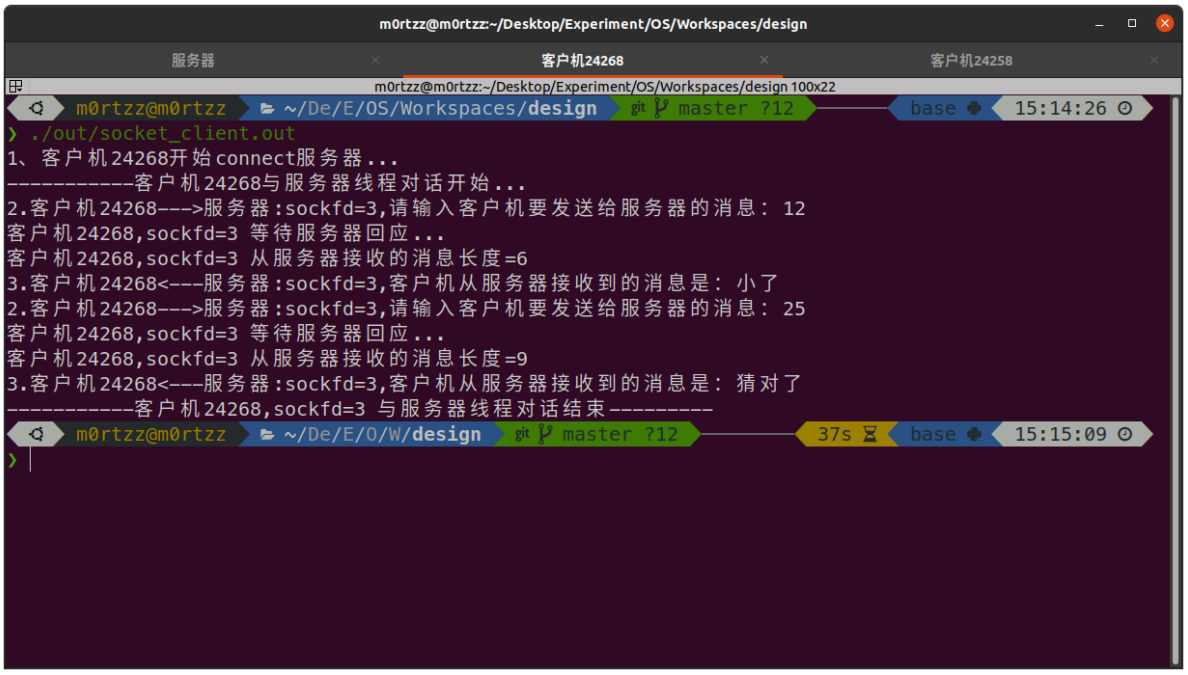
```
    char snd_buf[1024];
    // 接收消息缓冲区
    char rcv_buf[1024];
    // connect函数调用的结果
    int result;
    // 接收消息长度
    int rcv_num;
    // 客户进程标识符
    pid_t cpid;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    if (sockfd < 0)
    {
        perror("客户端创建套接字失败! \n");
        return EXIT_FAILURE;
    }

    // 使用网络套接字
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("127.0.0.1"); // 服务器地址
    // 服务器所监听的端口
    address.sin_port = htons(9736);

    if (inet_aton("127.0.0.1", &address.sin_addr) < 0)
    {
        printf("inet_aton error.\n");
        return -EXIT_FAILURE;
    }

    len = sizeof(address);
    // 获取客户进程标识符
    cpid = getpid();
    printf("1、客户机%d开始connect服务器...\n", cpid);
    result = connect(sockfd, (struct sockaddr *)&address, len);

    if (result == -1)
    {
        perror("客户机connect服务器失败!\n");
        exit(EXIT_FAILURE);
    }

    printf("-----------客户机%d与服务器线程对话开始...\n", cpid);

    // 客户机与服务器循环发送接收消息
    do
    {

        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);
        // 发送缓冲区清零
        memset(snd_buf, 0, 1024);
        scanf("%s", snd_buf); // 键盘输入欲发送给服务器的消息字符串
        // 将消息发送到套接字
        write(sockfd, snd_buf, sizeof(snd_buf));

        if (strncmp(snd_buf, "quit", 2) == 0)
```

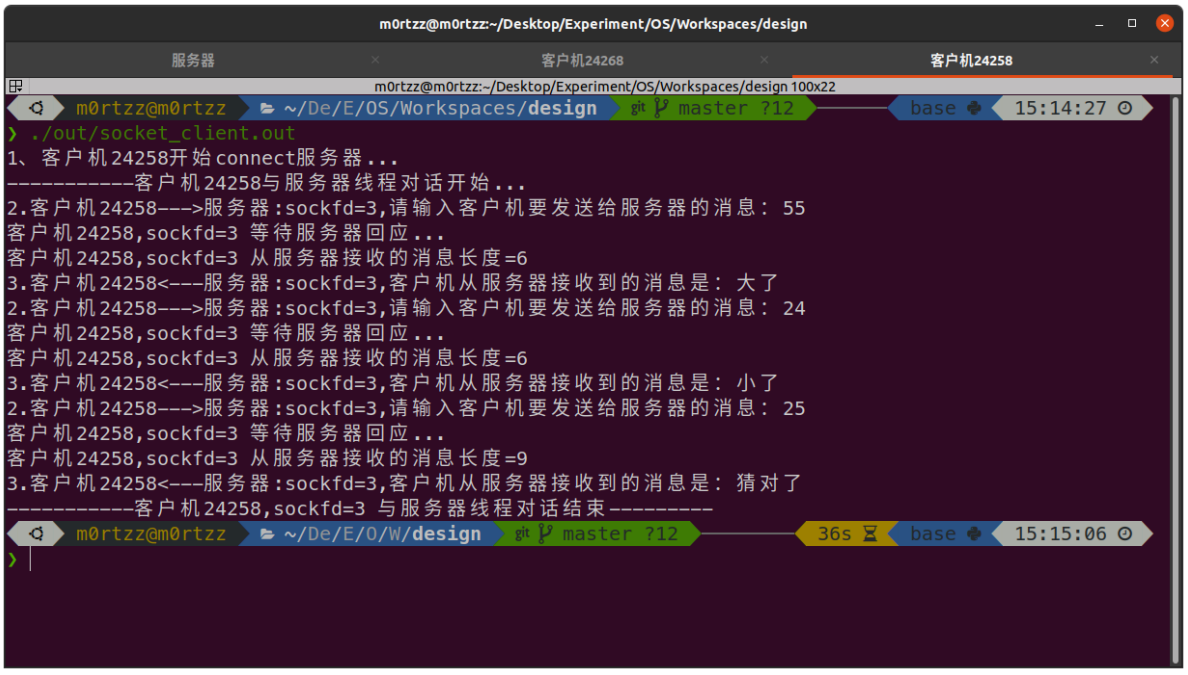
```c
            break; // 若发送"quit"，则结束循环，通信结束

        // 接收缓冲区清零
        memset(rcv_buf, 0, 1024);
        printf("客户机%d,sockfd=%d 等待服务器回应...\n", cpid, sockfd);
        rcv_num = read(sockfd, rcv_buf, sizeof(rcv_buf));
        printf("客户机%d,sockfd=%d 从服务器接收的消息长度=%lu\n", cpid, sockfd, strlen(rcv_buf));
        // 输出客户机从服务器接收的消息
        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:%s\n", cpid, sockfd, rcv_buf);

        sleep(1);

    } while (strncmp(rcv_buf, "猜对了", 2) != 0); // 如果收到"!q"，则结束循环，通信结束
    printf("-----------客户机%d,sockfd=%d 与服务器线程对话结束---------\n", cpid, sockfd);
    // 关闭客户机套接字
    close(sockfd);

    return EXIT_SUCCESS;
}
```

②运行

```
make all
./out/socket_server.out
./out/socket_client.out # 不多于5个
```







第四题

①源码编译安装gdb版Bochs（x86模拟器）

安装版本为Bochs-2.2.6，原因是：

https://github.com/oldlinux-web/oldlinux-files/blob/master/bochs/README_FIRST

```
When running the Linux kernel 0.1x in Bochs environment with version number
newer or equal to 2.2.6, you may encounter an problem at the bootstrap period
with the error message "HD controller not ready". There are two general
methods you can use to fix this problem.
```

因不想使用官方推荐的`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，所以我还是使用了`apt`：

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

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

```
cd linux/src/code/
./setup.sh
```

> **Note**
>
> 以下为最后运行Bochs时出现BUG后经过code spelunking后解决问题的过程【本仓库上传的源码鄙人已解决BUG:)】：
>
> 1) `linux/src/code//bochs-2.2.6/gdbstub.cc`:
>
> ```
> // @brief: add
> // @line: 492-494
> else if (last_stop_reason == GDBSTUB_STOP_NO_REASON)
> {
>   write_signal(&buf[1], SIGSEGV);
> }
> ```
>
> ```
> if (last_stop_reason == GDBSTUB_EXECUTION_BREAKPOINT ||
>     last_stop_reason == GDBSTUB_TRACE)
> {
>   write_signal(&buf[1], SIGTRAP);
> }
> else if (last_stop_reason == GDBSTUB_STOP_NO_REASON)
> {
>   write_signal(&buf[1], SIGSEGV);
> }
> else
> {
>   write_signal(&buf[1], 0);
> }
> put_reply(buf);
> break;
> }
> ```
>
> 2) `linux/src/code/bochs-2.2.6/cpu/cpu.cc`:
>
> ```
> // @brief: comment
> // @line: 143-147
> // #if BX_GDBSTUB
> //     if (bx_dbg.gdbstub_enabled) {
> //       return;
> //     }
> // #endif
> ```

```
// BUG: fix bug
// #if BX_GDBSTUB
//     if (bx_dbg.gdbstub_enabled) {
//         return;
//     }
// #endif
  }
```

②改写内核源码过程（`linux/linux-0.12/`）

本仓库中的`linux/linux-0.12`鄙人已修改过源码，未修改过的源码tarball包为`linux/linux-0.12-pure-unmodified.tar.gz`。

1）`include/unistd.h`

```
// @brief: add
// @line: 149, 150
#define __NR_m0rtzz 87
#define __NR_ashore 88
```

```
#define __NR_uselib 86
#define __NR_m0rtzz 87
#define __NR_ashore 88

#define _syscall0(type,name) \
```

2）`include/linux/sys.h`

```
// @brief: add
// @line: 92
extern int sys_m0rtzz();
extern int sys_ashore();

fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,
sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,
sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,
sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,
sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,
sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,
sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,
sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,
sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,
sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,
sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,
sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,
sys_setreuid,sys_setregid, sys_sigsuspend, sys_sigpending, sys_sethostname,
sys_setrlimit, sys_getrlimit, sys_getrusage, sys_gettimeofday,
```

```
sys_settimeofday, sys_getgroups, sys_setgroups, sys_select, sys_symlink,
sys_lstat, sys_readlink, sys_uselib, sys_m0rtzz, sys_ashore};
```

```
extern int sys_uselib();
extern int sys_m0rtzz();
extern int sys_ashore();

fn_ptr sys_call_table[] = { sys_setup, sys_exit, sys_fork, sys_read,
sys_write, sys_open, sys_close, sys_waitpid, sys_creat, sys_link,
sys_unlink, sys_execve, sys_chdir, sys_time, sys_mknod, sys_chmod,
sys_chown, sys_break, sys_stat, sys_lseek, sys_getpid, sys_mount,
sys_umount, sys_setuid, sys_getuid, sys_stime, sys_ptrace, sys_alarm,
sys_fstat, sys_pause, sys_utime, sys_stty, sys_gtty, sys_access,
sys_nice, sys_ftime, sys_sync, sys_kill, sys_rename, sys_mkdir,
sys_rmdir, sys_dup, sys_pipe, sys_times, sys_prof, sys_brk, sys_setgid,
sys_getgid, sys_signal, sys_geteuid, sys_getegid, sys_acct, sys_phys,
sys_lock, sys_ioctl, sys_fcntl, sys_mpx, sys_setpgid, sys_ulimit,
sys_uname, sys_umask, sys_chroot, sys_ustat, sys_dup2, sys_getppid,
sys_getpgrp, sys_setsid, sys_sigaction, sys_sgetmask, sys_ssetmask,
sys_setreuid,sys_setregid, sys_sigsuspend, sys_sigpending, sys_sethostname,
sys_setrlimit, sys_getrlimit, sys_getrusage, sys_gettimeofday,
sys_settimeofday, sys_getgroups, sys_setgroups, sys_select, sys_symlink,
sys_lstat, sys_readlink, sys_uselib, sys_m0rtzz, sys_ashore};
```

4) kernel/ststem_call.s

```
# @brief: change
# @line: 63
nr_system_calls = 84
```

```
nr_system_calls = 84
```

5) kernel/m0rtzz.c

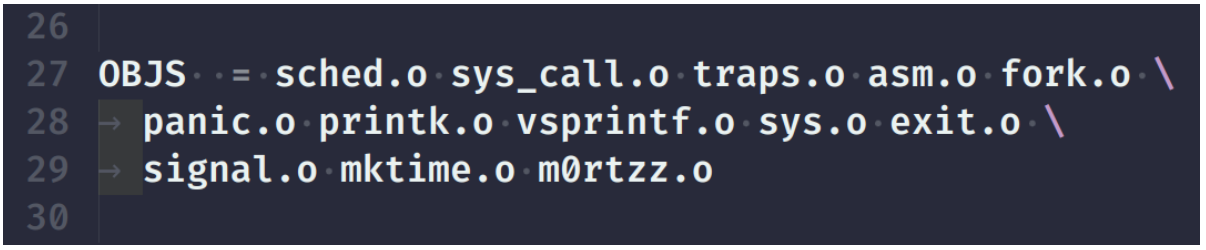
```
/**
 * @file m0rtzz.c
 * @brief 实现自定义的系统调用函数
 * @author M0rtzz
 * @version 1.0
 * @date 2024-04-30
 */

#include <errno.h>
#include <string.h>
#include <asm/segment.h>

char msg[30]; // 全局变量，用于存储用户传递的消息

/**
 * @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
 * @param str 用户提供的字符串指针
 * @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
 */
int sys_m0rtzz(const char *str)
```

```
{
    int i;
    char tmp[40]; // 临时缓冲区，用于存储从用户空间读取的字符串

    // 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
    for (i = 0; i < 40; i++)
    {
        // 从用户空间读取一个字符并存储到临时缓冲区中
        tmp[i] = get_fs_byte(str + i);

        if (tmp[i] == '\0') // 如果读取到字符串结束符，则退出循环
            break;
    }

    // 统计读取的字符个数
    i = 0;
    while (i < 40 && tmp[i] != '\0')
        i++;

    int len = i;

    // 如果读取的字符个数超过30个，则返回错误码
    if (len > 30)
        return -(EINVAL);

    // 将读取的字符串拷贝到全局变量msg中
    strcpy(msg, tmp);

    // 返回拷贝的字符个数
    return i;
}

/**
 * @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
 * @param str 用户提供的缓冲区指针，用于存储消息
 * @param size 缓冲区的大小
 * @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
 */
int sys_ashore(char *str, unsigned int size)
{
    int len = 0;

    // 统计全局变量msg中的字符个数
    for (; msg[len] != '\0'; len++)
        ;

    // 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
    if (len > size)
        return -(EINVAL);

    // 将全局变量msg中的消息拷贝到用户提供的缓冲区中
    int i;
    for (i = 0; i < size; i++)
    {
        put_fs_byte(msg[i], str + i); // 将字符逐个写入用户空间

```

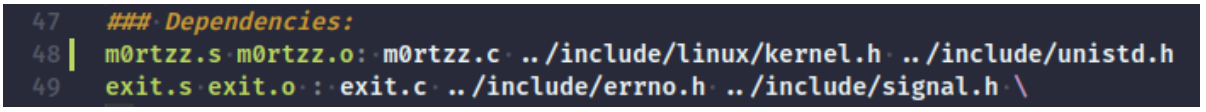
```
        if (msg[i] == '\0') // 如果遇到字符串结束符，则退出循环
            break;
    }

    // 返回拷贝的字符个数
    return i;
}
```

6）`kernel/Makefile`

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




```
# @brief: add
# @line: 48
m0rtzz.s m0rtzz.o: m0rtzz.c ../include/asm/segment.h ../include/string.h
../include/errno.h
```



③编译linux-0.12

```
cd linux/oslab/
code run.sh # 修改脚本
```

```
#!/bin/sh

lock_file="./hdc.img.lock"

if [ -f "${lock_file}" ]; then
    rm "${lock_file}"
fi

export OSLAB_PATH=$(dirname `which $0`)

if [ ! -e "hdc.img" ]; then
tar -xvJf hdc.tar.xz
fi

if [ "$1" ] && [ "$1" = "-m" ]
then
```

```
(cd ../linux-0.12; make clean; make; cp Image ../oslab/Image)
elif [ "$1" ] && [ "$1" = "-g" ]
then
${OSLAB_PATH}/bochs/bochs-gdb -q -f ${OSLAB_PATH}/bochs/bochsrc-gdb.bxrc & \
gdb -x ${OSLAB_PATH}/bochs/.gdbrc ../linux-0.12/tools/system
else
bochs -q -f ${OSLAB_PATH}/bochs/bochsrc.bxrc
fi
```

```
./run.sh -m # 编译内核源码
```

④编写代码

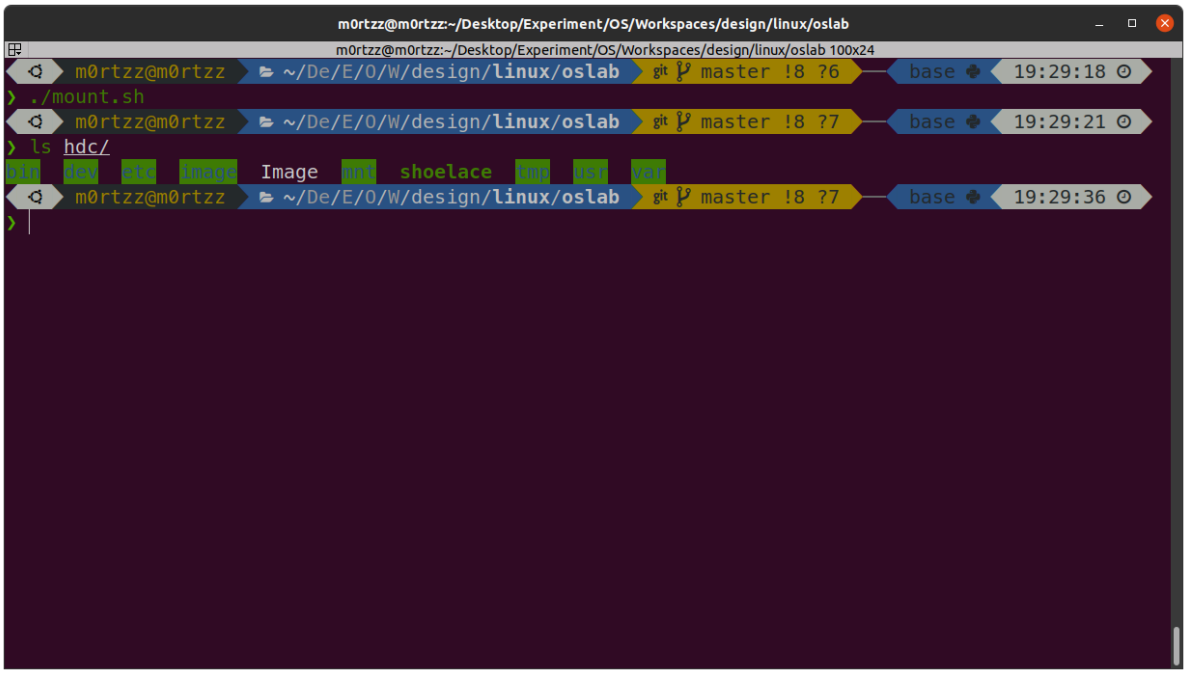
```
cd linux/oslab/
touch mount.sh umount.sh
```

```
#!/usr/bin/sudo /bin/zsh

# ------------------------------------------------------------------
# @file: mount.sh
# @brief: 挂载文件系统，此文件系统是linux-0.11的文件系统映像，但不影响使用，在此不过多赘述
# @author: MOrtzz
# @date: 2024-05-01
# ------------------------------------------------------------------

mount_folder="hdc"

if [ ! -d "${mount_folder}" ]; then
    mkdir "${mount_folder}"
fi

export OSLAB_PATH=$(cd $(dirname "${BASH_SOURCE[0]}") >/dev/null && pwd)
mount -t minix -o loop,offset=1024 ${OSLAB_PATH}/hdc.img ${OSLAB_PATH}/hdc
```

```
#!/usr/bin/sudo /bin/zsh

# ------------------------------------------------------------------
# @file: umount.sh
# @brief: 卸载文件系统
# @author: MOrtzz
# @date: 2024-05-01
# ------------------------------------------------------------------

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：

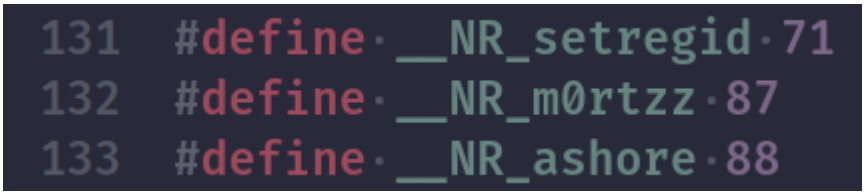
```
sudo chmod +x mount.sh
./mount.sh
```



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:

```
code hdc/
```

```
// @file: hdc/usr/include/unistd.h
// @brief: add
// @line: 132, 133
#define __NR_m0rtzz 87
#define __NR_ashore 88
```




```
/**
 * @file hdc/usr/root/m0rtzz_1.c
 * @brief 使用系统调用 `__NR_m0rtzz`
 * @author M0rtzz E-mail : m0rtzz@outlook.com
 * @version 1.0
 * @date 2024-05-01
 *
 */

/**
 * @brief 启用系统调用
 */
#define __LIBRARY__
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>

```

```c
_syscall1(int, m0rtzz, const char *, str);

int main(int argc, char **argv)
{
    m0rtzz(argv[1]);

    return EXIT_SUCCESS;
}
```

```c
/**
 * @file hdc/usr/root/m0rtzz_2.c
 * @brief 使用系统调用 `__NR_ashore`
 * @author M0rtzz E-mail : m0rtzz@outlook.com
 * @version 1.0
 * @date 2024-05-01
 *
 */

/**
 * @brief 启用系统调用
 */
#define __LIBRARY__
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>

_syscall2(int, ashore, char *, str, unsigned, size);

int main()
{
    char s[30];
    ashore(s, 30);
    printf("The string is: %s\n", s);

    return EXIT_SUCCESS;
}
```

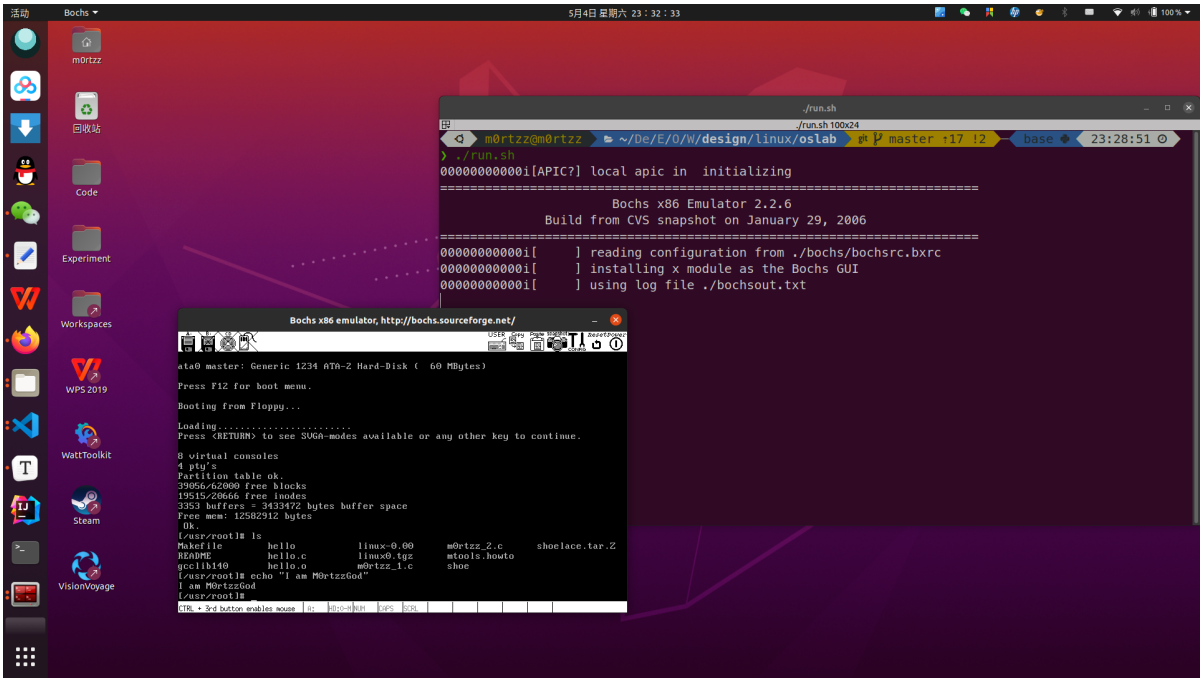
```makefile
# @file: hdc/usr/root/Makefile

CC = gcc

all:m0rtzz_1.out m0rtzz_2.out

%.out: %.c
    $(CC) $< -o $@

.PHONY: clean

clean:
    rm -f *.out
```

⑤进入linux-0.12编译并运行代码

1）以普通模式进入linux-0.12

```
cd linux/oslab/
./run.sh # 以普通模式进入linux-0.12
```

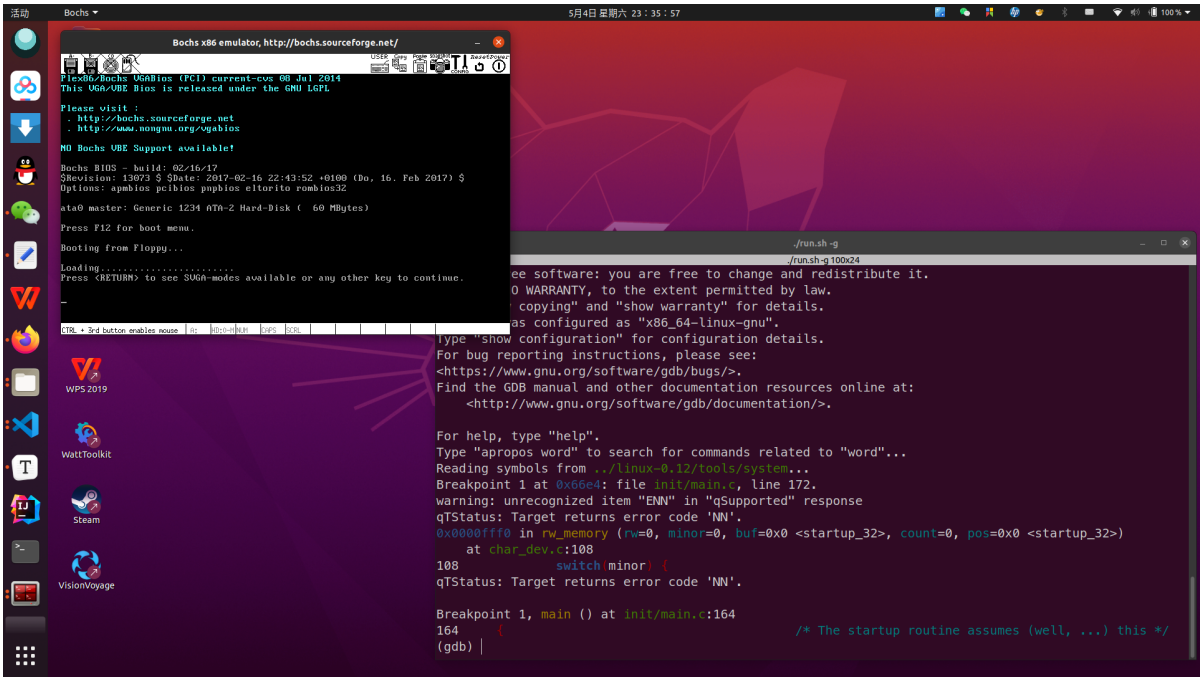
进入系统之后输入基础命令发现正常使用，无BUG：



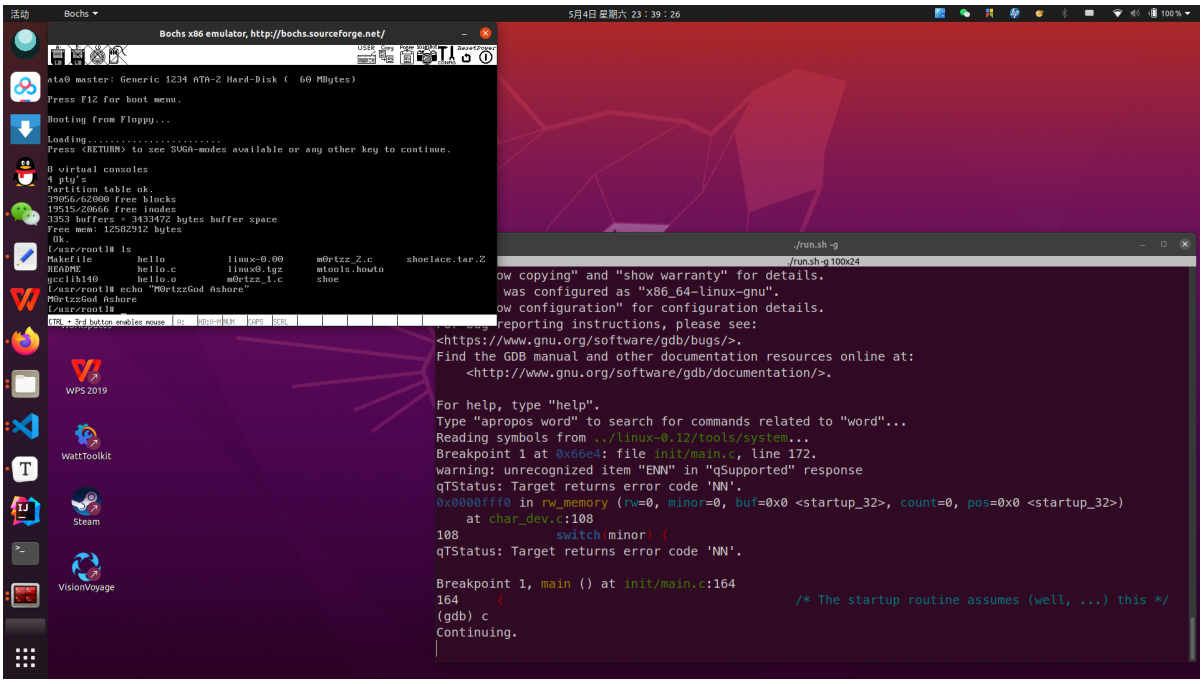
2）以gdb模式进入linux-0.12

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

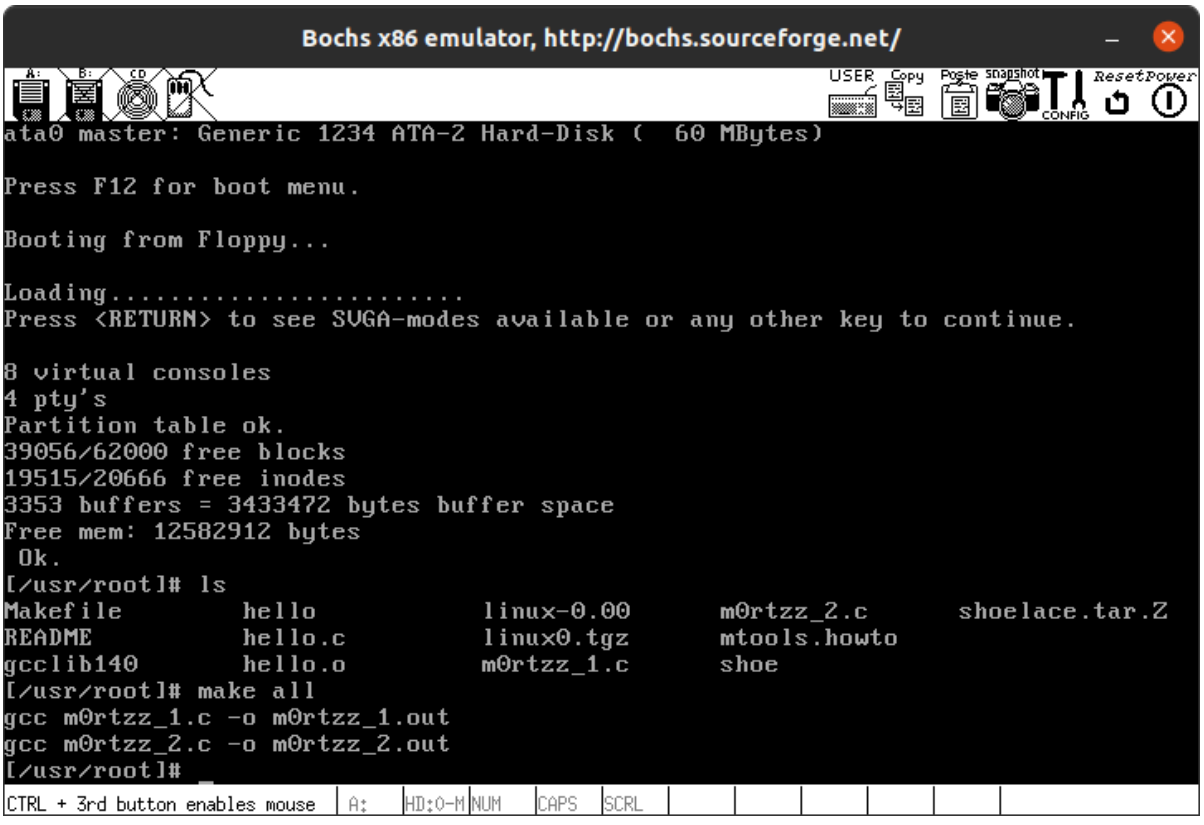
一开始本地终端进入gdb模式但模拟器终端没有进入文件系统：



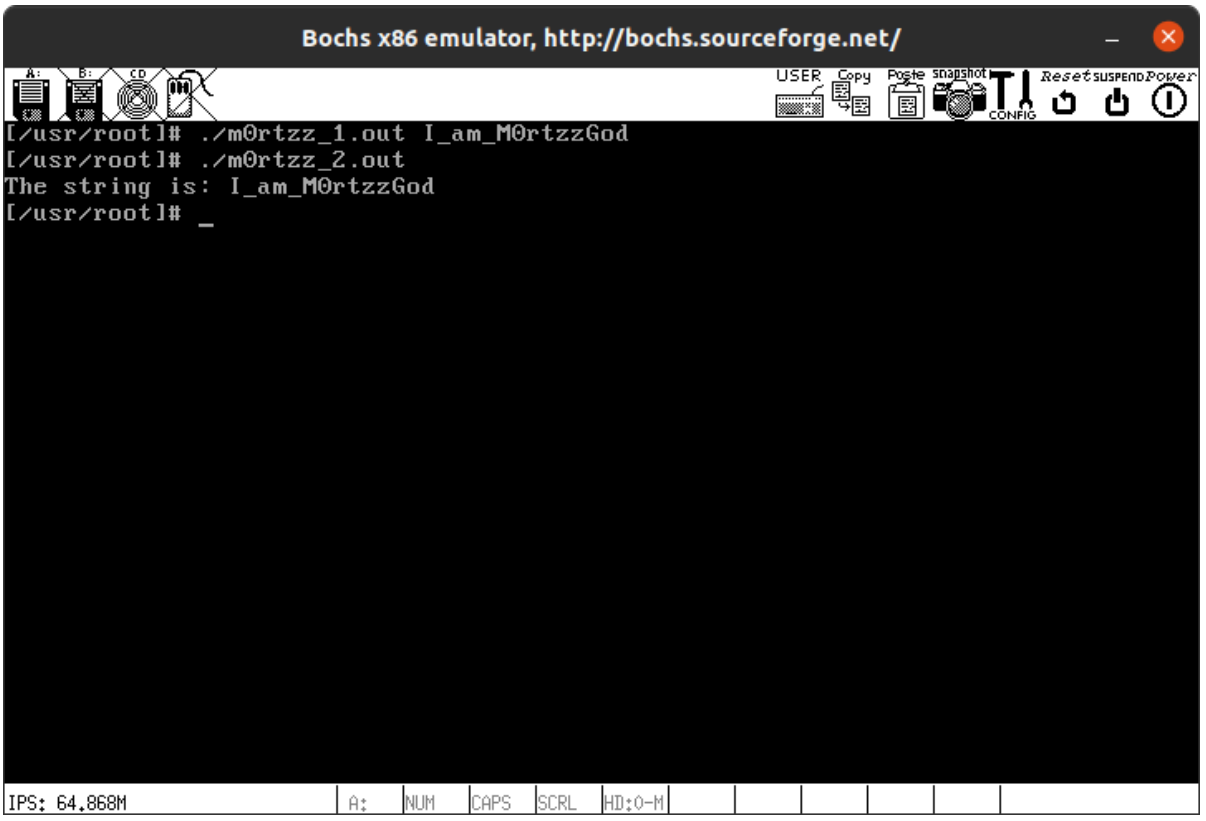
我们需要在本地终端按`c`键（continue）并回车进入文件系统后就可以正常使用命令了：




```
ls
make all
```



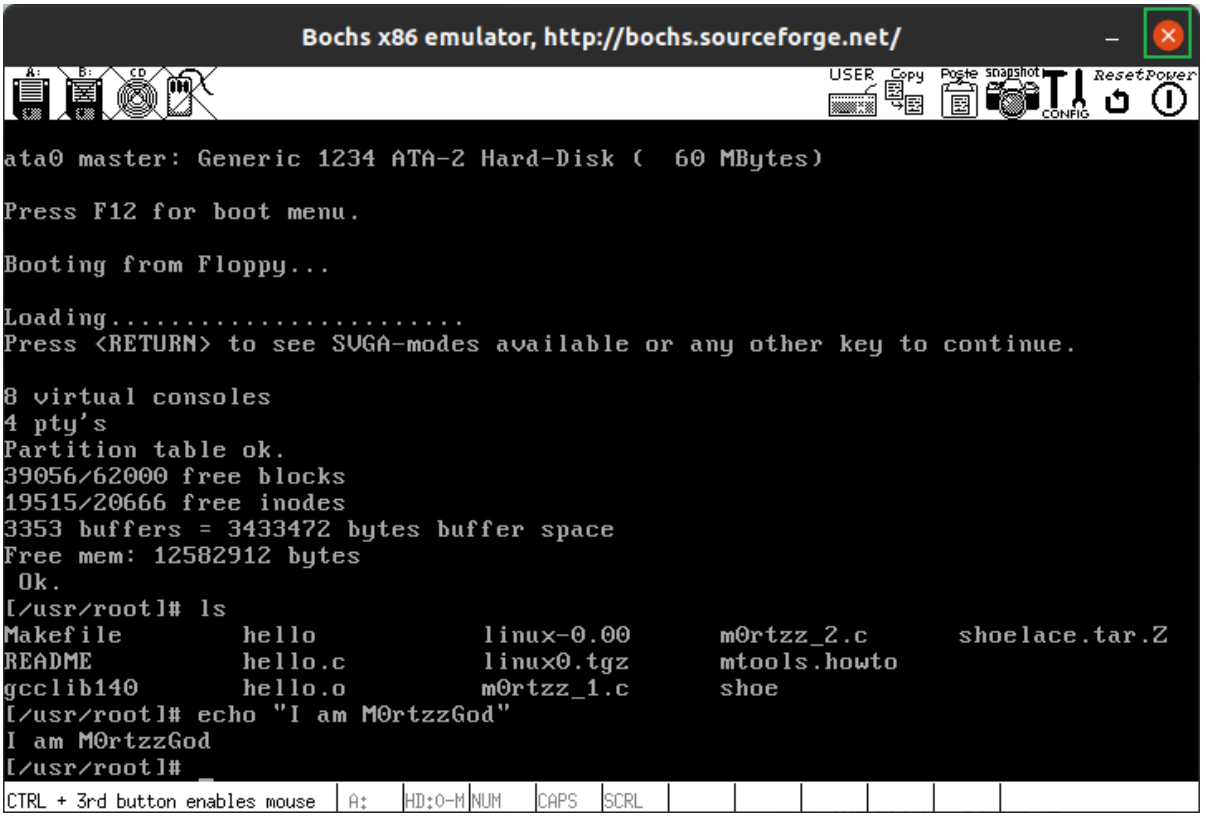

```
./m0rtzz_1.out str
./m0rtzz_2.out
```

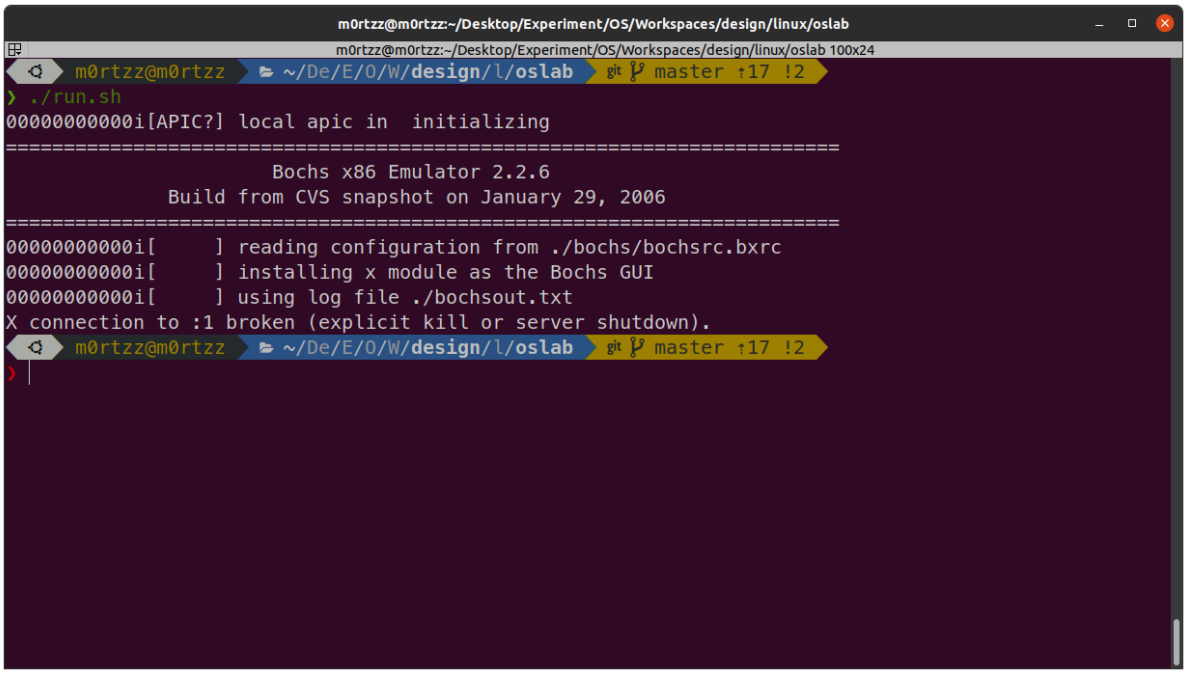


⑥退出linux-0.12

1）普通模式退出linux-0.12

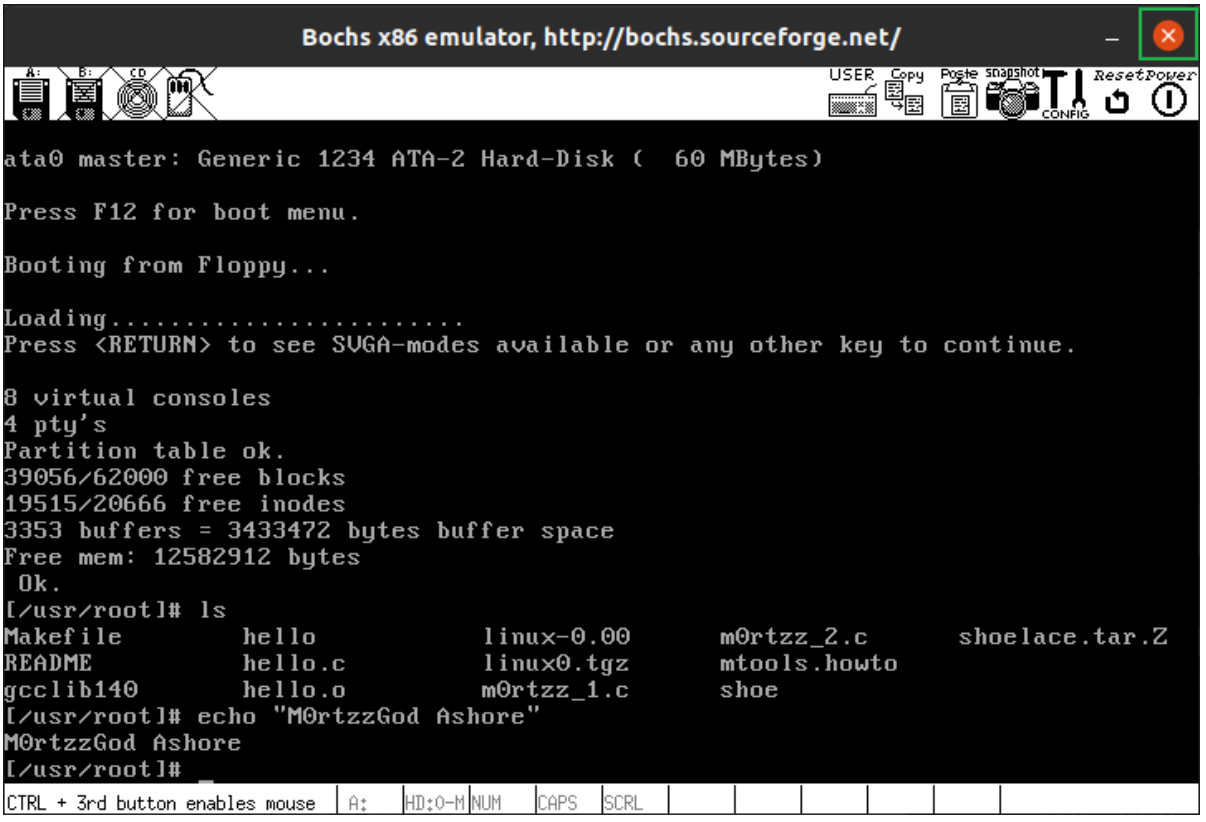
直接点击模拟器终端右上角的 × 即可退出：

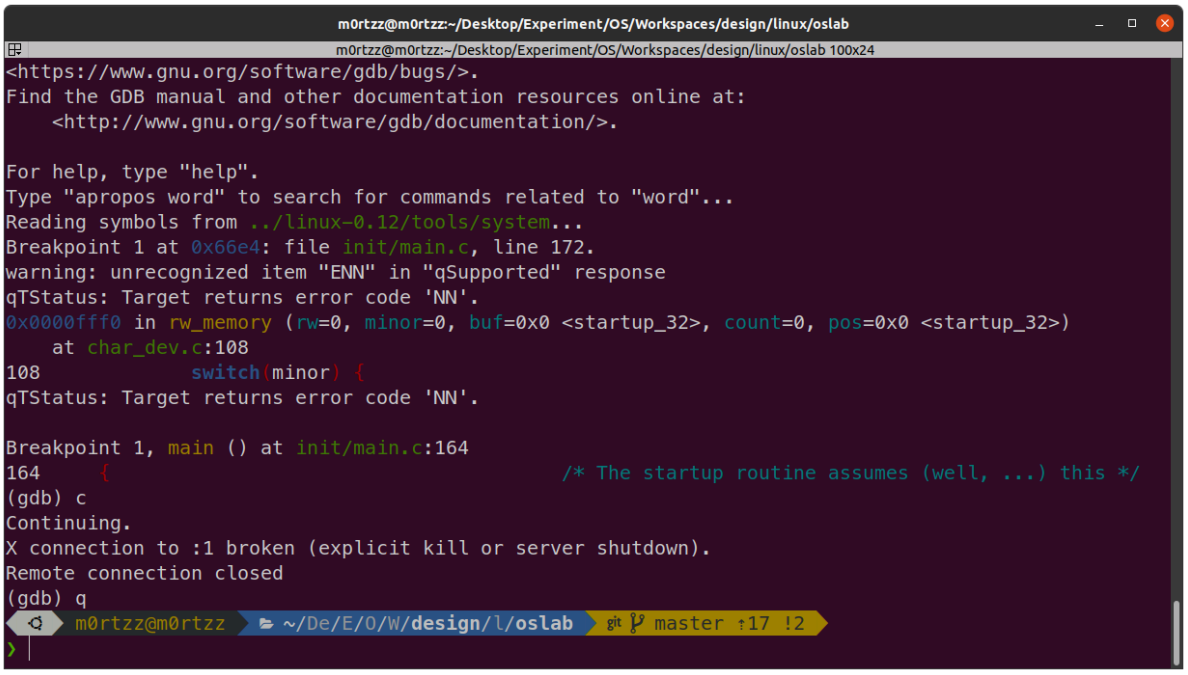




2）gdb模式退出linux-0.12

点击模拟器终端右上角的 `×` 后在本地终端输入 `q` 键并回车即可退出：





⑦卸载文件系统

```
sudo chmod +x umount.sh
./umount.sh
```