2021~2022 学年第二学期《Linux 程序设计》试卷甲 A

1 Single choice（24’）

（1） Regarding the File *fp, which statement is **wrong**.

A) One cannot use fp to write the file which is opened by fp=fopen(…., “r”).
B) One can use fp++ or fp-- to move the file read/write position when fp is opened correctly.
C) When one uses fread(&record, sizeof(record), 1, fp) to read the record from the file, the system will optimize the reading process with fixed size regardless the size of the record.
D) When the program finishes running, the system will auto-invoke fclose(fp) to write file content back to the hard disk.

（2） Regarding the File descriptor fd, which statement is **wrong**.

A) The File descriptor fd is an integer greater than or equal to 0.
B) Typically, when a process is executing, file descriptor 0, 1, and 2 are referred to the standard input, standard output, and standard error output, respectively.
C) When one uses read(fd, record, sizeof(record)) to read the record from the file, the system will optimize the reading process with fixed size regardless the size of the record.
D) A file can be associated with multiple file descriptors when invoking system call dup(fd).

（3） Assume compiling foo.c needs the library file libfoo.so (in directory /home/stu/lib), then the correct complying command is:

A) gcc foo.c -I/home/stu/lib -lfoo -o foo
B) gcc foo.c -L/home/stu/lib -lfoo -o foo
C) gcc foo.c -I/home/stu/lib -ifoo -o foo
D) gcc foo.c -L/home/stu/lib -Ifoo -o foo

（4） Read the following code and determine how many lines of "abc" will be displayed on the screen:

```
int main()
{
        if(fork()==0){
                printf(“abc\n”);
        }
        else{
                printf(“abc\n”);
        }
        printf(“abc\n”);
}
```

A) 3　　B) 4　　C) 2　　D) 1

（5） If one runs the codes :

```
main(){
      int fd;
      fd = open("/tmp/log.file", O_RDWR|O_APPEND|O_CREAT, S_IRWXU);
      close(1);
      dup(fd);
      printf("log information\n");
}
```

what will happen?

A) nothing will output
B) the character string "log information" will be written into the file "/tmp/log.file"
C) the character string "log information" will be displayed on the current screen
D) the character string "log information" will be written into "/dev/null"

（6） Suppose someone is running an executable program "abc" as:

```
$abc 2>/dev/null
```

So, what is the meaning of "2>/dev/null"

A) discard standard output, so there is nothing to output
B) discard standard input
C) discard standard error output, so if running error there is no error information to output
D) create a file /dev/null and store information in it

（7） About POSIX semaphore and system V semaphore, which statement is **not true**:

A) The semaphore of System V is a semaphore set including multiple semaphores, and one operation (such as semop()), can operate multiple semaphores simultaneously.

B) The semaphore of POSIX is a single one. Normally, named semaphore supports inter-process communication and unnamed semaphore can be used for inter-thread communication.
C) The semaphore of System V must be destroyed by the programmer.
D) The semaphore of POSIX is auto-destroyed when the process ends.

（8） Regarding the Memory-mapped files (mmap(…)), which statement is **wrong**.
A) Memory-mapped files allow for multiple processes to share read-only access to a common file. That is, only one copy of the file needs to be loaded into physical memory, even if there are thousands of programs running.
B) In some cases, memory-mapped files simplify the logic of a program by using memory-mapped I/O. Rather than using fseek() multiple times to jump to random file locations, the data can be accessed directly by using an index into an array.
C) Memory-mapped files provide more efficient access for initial reads. When read() is used to access a file, the file contents are first copied from disk into the kernel’s buffer cache. Then, the data must be copied again into the process’s user-mode memory for access. Memory-mapped files bypass the buffer cache, and the data is copied directly into the user-mode portion of memory.
D) In contrast to message-passing forms of IPC (such as pipes), memory-mapped files create persistent IPC. Once the data is written to the shared region, it can be repeatedly accessed by other processes and will be written back to the file on disk automatically.

B C B B B C D D

2 Program reading and analysis

Codes list 2-1:

```
//omit header files

int main()
{
        char buffer[1024];

        int nread, out;
        nread = read(0, buffer, 1024);
        out = open("file.out", O_WRONLY|O_CREAT, S_IRUSR|S_IWUSR);
        write(out, buffer, nread);

        exit(0);
}
```

(1) What is most likely value of out? Why?
(2) If execute kb1<drf.txt, what is the result? (drf.txt is a text file)
(3) What are the permissions of the owner, group and other users to manipulate the file.out?
(4) Is the code style good? Do you have any suggestions?

(1) 4 分

Linux 程序执行时，默认打开文件描述符 0、1、2，分别指向标准输入（键盘），标准输出和标准错误输出（当前终端显示器）。当打开（创建）一个新的文件时，文件描述符会选择最小的未被使用的文件描述符，所以 fd=3。

(2) 4 分

因为使用了输入重定向，所以 file.out 会是 drf.txt 前 1024 字节的内容（如果 drf.txt 不超过 1024 字节，那么 file.out 就是 drf.txt）。

(3) 4 分

拥有者（owner）对文件的权限是可读可写，组用户（group）和其他用户（other users）对文件 file.out 没有权限。

(4) 4 分

代码风格不好，应该判断主要的系统（函数）调用的返回值。

Codes list 2-2:

```
//omit header files

int main()
{
       int n;
       printf("fork example\n");
       if ((pid=fork())<0) {
             perror("fork error");
             exit(0);
       }
      if(pid == 0){
             n=3;
           //execlp("ls", "ls", "-l", 0);
      }
       else n=2;
       for(; n>0; n--) printf("fork example\n");
}
```

(1) How many lines of "fork example" will be outputted on the screen? Why?
(2) If open "execlp("ls", "ls", "-l", 0)", how many lines of "fork example" will be outputted on the screen? Why?
(3) Some lines of "fork example" will be outputted after prompt $. Could you explain this? Could you modify the program to guarantee all the "fork example" be outputted before the last line of prompt symbol $?

(1)　4 分

共输出 6 行"fork example"，fork（）之前输出一行，fork（）执行成功后，子进程输出 3 行，父进程输出 2 行。

(2)　4 分

输出 3 行，子进程调用 execlp 后，替换了原来的子进程映像，原来的 printf 语句被覆盖了。

(3)　4 分

父进程先执行结束，返回 shell，子进程再执行，所以子进程的输出内容在$的后面。要想保证所有输出内容都在$之前，只要在父进程中调用 wait()或 waitpid()即可。

Codes list 2-3:

```
//omit header files

     int x = 0 ;                                   int x = 0 ;
     …;                                            …;
     If(fork()==0) {                                pthread_create(&a_thread,NULL,fun,NULL);
             x++;                                  x++;
      }                                            …;
      else {                                       void *fun(void *arg) {
             x--;                                        x--;
      }                                            }
```

(1) The above two pieces of code exploit process and thread technique, respectively. Compare and discuss the difference between these two techniques in terms of the change of the variable x.

（1）8分
x变量在父进程和子进程中是独立的副本；x变量在线程中是同一个副本（两个线程共享x这一资源），所以在线程中x变量最终的值取决于x++

或者x--，哪一个后执行（可以通过pthread_join控制执行顺序）

Codes list 2-4: /*socket*/

```
    …;
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("127.0.0.1");
    address.sin_port = htons(9734);
    len = sizeof(address);
    …;
```

(1) What type socket and what port number used in this program? What does the htons() do?

```
    …;
    result = connect(sockfd, (struct sockaddr *)&address, len);
    if(result == -1) {
        perror("connect fail!");
        exit(1);
    }
    msg="hello";
    write(sockfd, msg, strlen(msg)+1);
    …;
```

(1) 5 分

使用的是 TCP 套接字，端口号是 9734，函数 htons（）的作用是把主机字节序转换为网络字节序。

(2) What is role of function connect()? What is data being sent?

```
   listen(server_sockfd, 5);
    while(1) {
        printf("server waiting\n");
        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_address, &client_len);

        if(fork() == 0) {
            memset(msg, 0, 256);
            read(client_sockfd, msg, sizeof(msg));
            printf("client say: %s\n", msg);
            write(client_sockfd, toupper(msg), sizeof(msg));    /* toupper(msg) converts msg to uppercase characters */
            close(client_sockfd);
            exit(0);
        }
        else {
            close(client_sockfd);
        }
```

(2) 5 分

connect（）函数由客户端执行，向服务器发送连接请求，建立与指定 socket 的连接。发送的数据为“hello”。

(3) What is role of function accept()? Why use fork()? What information does the program send to the client?

(3) 5 分

accept（）接受监听套接字的等待连接队列中第一个连接请求，创建一个新的套接字，并返回指向该套接字的文件描述符用于和客户端的通信。fork（）的作用是创建子进程，然后在子进程中和客户端通信。服务器使用 toupper(msg)将 msg，也就是字符串“hello”转换为大写的“HELLO”，发送给客户端。

3 Programming

The below are two programs named as receiver and sender respectively. Could you write a program exploiting pipe programming technique to conduct the communication between these two programs? When the communication is done, the screen will display: “receiver: sender send message”.

```
//omit header files

Codes list 3-1:    /*receiver*/
main()
{
        char buffer[1024];

        fgets(stdin, buffer);
        printf(“receiver: %s\n”, buffer);

        exit(0);
}

Codes list 3-2:    /*sender*/
main()
{
          printf(“sender send message\n”) ;
}
```

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
main(){
                static char process1[]="receiver",process2[]="sender";
                int fd[2];

                pipe(fd);
                if (fork()==0){
                        close(fd[0]);
                        close(1);
                        dup(fd[1]);
                        close(fd[1]);
                        execl(process2, process2, 0);
                }
                else{
                        close(fd[1]);
                        close(0);
                        dup(fd[0]);
                        close(fd[0]);
                        execl(process1, process1, 0);
                }
          exit(0);
}
```

The following descriptions of some system calls, which you may use in programming, are excerpted from Linux manual pages.

(1) int pipe(int pipefd[2]);

DESCRIPTION

pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array pipefd is used to return two file descriptors referring to the ends of the pipe. pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and errno is set appropriately.

(2) int dup(int oldfd);

DESCRIPTION

These system calls create a copy of the file descriptor oldfd.

dup() uses the lowest-numbered unused descriptor for the new descriptor.

….

After a successful return from one of these system calls, the old and new file descriptors may be used interchangeably. They refer to the same open file description (see open(2)) and thus share file offset and file status flags; for example, if the file offset is modified by using lseek(2) on one of the descriptors, the offset is also changed for the other.

RETURN VALUE

On success, these system calls return the new descriptor. On error, -1 is returned, and errno is set appropriately.

以下是一些系统调用的描述，你在编程中可能会用到这些调用，这些描述摘自 Linux 手册页面。

(1) int pipe(int pipefd[2]);

描述

pipe()创建一个管道，一个单向的数据通道，可用于进程间通信。数组 pipefd 用来返回两个文件描述符，指的是管道的两端。pipefd[0]指的是管道的读端。 pipefd[1]指的是管道的写端。写入管道的写端数据由内核缓冲，直到从管道的读端读取。

返回值

成功时，返回 0。出错时，返回-1，并适当地设置 errno。

(2) int dup(int oldfd);

描述

这些系统调用创建了一个文件描述符 oldfd 的副本。

dup()使用最低编号的未使用的描述符作为新的描述符。

....

从这些系统调用之一成功返回后，新旧文件描述符可以交替使用。它们指的是同一个打开的文件描述（见 open(2)），因此共享文件偏移量和文件状态标志；例如，如果在其中一个描述符上使用 lseek(2)修改了文件偏移量，另一个描述符的偏移量也会发生变化。

返回值

成功时，这些系统调用返回新的描述符。出错时，返回-1，并适当地设置 errno。