2020~2021 学年第二学期《Linux 操作系统与程序设计》试卷甲 A

1 Single choice（30’）

(1) Observe the output of ‘ls – l’ command, the last 3 bits of the first column represents:
A) file type B) permissions for the file owner
C) permissions for the the group user D) permissions for other users

(2) If chmod 744 fido is executed successfully, then permission of fido is:
A) -rwxr-xr-x B) -rwxr--r-- C) -r--r--r-- D) -r-xr-x—x

(3) About command ‘mv abc xyz’, which statement is wrong:
A) If abc is a normal file and xyz doesn't exist, then this command will change the file name abc to xyz.
B) If abc is a normal file and xyz is a directory, this command will move file abc into directory xyz.
C) If abc is a directory and xyz is also a directory, this command will make abc to be the subdirectory of xyz.
D) If abc is a directory and xyz is a normal file, this command will overwrite xyz, i.e. xyz will lost.

(4) File wang.tar.gz in the current directory. The correct tar command to extract this file is:
A) tar zcvf wang.tar.gz B) tar jcvf wang.tar.gz C)tar zxvf wang.tar.gz D)tar jxvf wang.tar.gz

(5) Which technique is used in the command‘ls -l >tmp’
A) Input redirection B) Output redirection C) Pipe

(6) Which command can be used to mount 'Windows C: disk (hda1)' in the /winsys directory:
A) #mount hda1 /winsys B) #mount /winsys /dev/hda1 C) #mount winsys /dev/hda1 D) #mount /dev/hda1 /winsys

(7) Users 'user1' and 'user2' both belong to the users' group. The file 'file1' belongs to 'user1'. If 'user2' wants to edit this file, which permission of file1 can guarantee the successful modification.
A) 744 B) 664 C) 646 D) 746

(8) The command to copy all files under directory 'abc' (including subdirectories) to the directory '/usr/xyz' is
A) cp –y abc /usr/xyz B) cp –f abc /usr/xyz C) cp –r abc /usr/xyz D) cp –p abc /usr/xyz

(9) How to execute the executable file 'abc' mandatorily in the current directory
A) ./abc B) ../abc C) .\abc D) ..\abc

(10) The following is the output of command ‘ls –l’, which one is the symbolic link file:
A) -rw------- 2 hel users 56 Sep 09 11:05 goodbye
B) drwx----- 1 hel users 1024 Sep 10 08:10 zhang
C) lrwx----- 1 hel users 2024 Sep 12 08:12 cheng->goodbye
D) -rw------- 1 hel users 56 Sep 09 11:05 hello

D B D C B D B C A C

(11) The following is the output of command 'ls –l', which one is a directory file (folder):

A) -rw------- 1 hel users 56 Sep 09 11:05 hello

B) -rw------- 2 hel users 56 Sep 09 11:05 goodbye

C) drwx----- 1 hel users 1024 Sep 10 08:10 zhang

D) lrwx----- 1 hel users 2024 Sep 12 08:12 cheng->goodbye

(12) The following is the output of 'ls –l', which one is a hard link file:

A) -rw------- 1 hel users 56 Sep 09 11:05 hello

B) -rw------- 2 hel users 56 Sep 09 11:05 goodbye

C) drwx----- 1 hel users 1024 Sep 10 08:10 zhang

D) lrwx----- 1 hel users 2024 Sep 12 08:12 cheng->goodbye

(13) The following is the output of command 'ps –t tty1', which one is the zombie process.

```
        UID       PID   PPID   C STIME TTY        TIME CMD
A)      root      2258  2199   0 10:09 tty1     00:00:00 -bash
B)      root      2339  2258   0 10:13 tty1     00:00:00 ./zomb
C)      root      2340  2339   0 10:13 tty1     00:00:00 [zomb] <defunct>
```

(14) The following is the output of the command 'ps -t tty1', which one is the parent process of './zomb'.

```
       UID          PID   PPID   C STIME TTY            TIME CMD
A)      root      2258  2199   0 10:09 tty1     00:00:00 -bash
B)      root      2339  2258   0 10:13 tty1     00:00:00 ./zomb
C)      root      2340  2339   0 10:13 tty1     00:00:00 [zomb] <defunct>
```

(15) The correct combined commands for finding function "func" in all the .c file under the directory "/usr/src" is

A) find /usr/src –name *.c –print|grep "func("

B) find /usr/src –path *.c –print|xargs grep "func("

C) find /usr/src –name *.c –print|xargs grep "func("

D) find /usr/src –path *.c –print|grep "func("

(16) The directory structure of the Linux file system is an upside-down tree, with files placed in related directories. If there is an external device file, in which directory one should place.

A) /bin   B) /etc   C) /dev   D) /lib

(17) The combined command "ls - l | grep -v '^d'" means:

A) Display all directory files in a long list in the current directory.

B) Display all files in a long list in the current directory except directory files.

C) Display all files in a long list in the current directory

D) Display all device files in a long list in the current directory.

(18) Assume an ordinary user 'user1' has a file 'abc' under the current directory, user1 want to execute this file. When he type 'abc' in the "$" prompt, the system reports an error, so the possible reason for this error is:

① The value of the system variable 'PATH' of user1 does not include the current directory, so the system cannot find the file.

② User user1 does not have executable permissions of 'abc'.

③ The current directory of user1 does not have executable permissions.

A) ①   B) ①②   C) ①②③   D) ②③)

(19) In the command mode of vi, which character can be used to append the text after the current cursor on the same line.

A) i   B) o   C) a   D) A

(20) How to cut and paste a line in vi.

A) yy;p   B) xx;p   C) dd;p   D) jj;p

C B C A C   C B B C C

(21) In vi, how to repeat the last command
A) up arrow B) down arrow C) . D) *

(22) In vi, how to delete the character under the current cursor
A) d B) x C) q D) p

(23) In vi, how to substitute str1 with str2 in the whole text:
A) 1,$s/str1/str2 B) 1,$s/str1/str2/p C) 1,$s/str1/str2/g D) 1,$/str1/str2/p

(24) In vi, how to undo the character you just typed
A) ESC;u B) ESC;U C) ESC;x D) ESC;X

(25) The first character of command 'ls -l' indicates the file type. Which statement is wrong:
A) '-' indicates a regular file.
B) 'd' indicates a directory file.
C) 'c' indicates a character text file.
D) 'b' indicates a block device file.

(26) The following is a shell program, read the code and answer the question:(comments: "tar rvf …" is used to append the file into an archive)

```
for i in `find $1 –name *.c -print`
do
    tar rvf back.c $i 2>/dev/null
done
```

① What is the value of $1
A) all the files with the suffix .c in the system
B) The argument specified in the command line indicates the starting directory for the searching
C) all the files with the suffix .c in /dev/null
D) The argument specified in the command line to determine the file type for the searching

(27) ② What is the value of $i
A) all the files with the suffix .c in the system
B) all the files with the suffix .c in the directory specified by $1
C) all the directories searched
D) none of the above

(28) ③ what is the correct form of using this program? (assume this file name is 'myfind')
A) ./myfind usr/src B) ./myfind *.c C) ./myfind /usr/src D)./myfind /usr/src/*.c

(29) ④ What is the role of '2>/dev/null'
A) output the result into /dev/null
B) compress the result into /dev/null
C) hide the error message
D) none of the above

(30) 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

C B C A C   B B C C B

2 Program reading and analysis

Codes list 2-1:

```
// omit header files

int main()
{
    int fd;
    int new_fd1, new_fd2;

    if((fd = open("myoutput",O_WRONLY|O_CREAT,0644)) == -1) {
          perror("file open fail!");
          exit(1);
    }
    printf("fd = %d\n",fd);
    new_fd1 = dup(fd);
    printf("new_fd1 = %d\n",new_fd1);
    close(1);
    new_fd2 = dup(fd);
    printf("new_fd2 = %d\n",new_fd2);
    printf("It is an output test\n");
}
```

(1) Determine the value of fd and explain why.
(2) Determine the value of new_fd1 and explain why.
(3) Determine the value of new_fd2 and explain why.
(4) Will the last printf be outputted to the screen? Why?

1. 2 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. 2 分
dup(fd)会从未使用过的文件描述符中选择最小的文件描述符作为返回值，并指向原先的 fd 所指向的文件。所以 new_fd1=dup(fd)执行成功后，new_fd1 的值等于 4。
3. 3 分
因为再一次执行 dup(fd)前，执行了 close（1），所以最小的未使用过的文件描述符是 1。所以 new_fd2=dup(fd)执行成功后，new_fd2 的值等于 1。
4. 3 分
Printf 的输出内容指向文件描述符 1 指向的文件，现在 1 指向的 fd 指向的文件，即 myoutput，所以输出内容会写到文件 myoutput 中。

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. 3 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. 3 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: /*receiver and sender by message queue.*/

Receiver:

```
//omit header files

struct my_msg_st {
        long int my_msg_type;
        char some_text[BUFSIZ];
};

int main()
{
        int running = 1;
        int msgid;
        struct my_msg_st some_data;
        long int msg_to_receive = 0;

        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

        if (msgid == -1) {
                fprintf(stderr, "msgget failed with error: %d\n", errno);
                exit(EXIT_FAILURE);
        }
        while(running) {
                if (msgrcv(msgid, (void *)&some_data, BUFSIZ, msg_to_receive, 0) == -1) {
                        fprintf(stderr, "msgrcv failed with error: %d\n", errno);
                        exit(EXIT_FAILURE);
                }
                printf("You wrote: %s", some_data.some_text);
                if (strncmp(some_data.some_text, "end", 3) == 0) {
                        running = 0;
                }
        }

        if (msgctl(msgid, IPC_RMID, 0) == -1) {
                fprintf(stderr, "msgctl(IPC_RMID) failed\n");
                exit(EXIT_FAILURE);
        }

        exit(EXIT_SUCCESS);
}
```

Sender:

```
//omit header files

#define MAX_TEXT 512

struct my_msg_st {
        long int my_msg_type;
        char some_text[MAX_TEXT];
};

int main()
```

```
{
        int running = 1;
        struct my_msg_st some_data;
        int msgid;
        char buffer[BUFSIZ];

        msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

        if (msgid == -1) {
                fprintf(stderr, "msgget failed with error: %d\n", errno);
                exit(EXIT_FAILURE);
        }

        while(running) {
                printf("Enter some text: ");
                fgets(buffer, BUFSIZ, stdin);
                some_data.my_msg_type = 1;
                strcpy(some_data.some_text, buffer);

                if (msgsnd(msgid, (void *)&some_data, MAX_TEXT, 0) == -1) {
                        fprintf(stderr, "msgsnd failed\n");
                        exit(EXIT_FAILURE);
                }
                if (strncmp(buffer, "end", 3) == 0) {
                        running = 0;
                }
        }

        exit(EXIT_SUCCESS);
}
```

(1) Describe how the program works.

(2) All the error handling in the program uses fprintf(stderr,…), can one replace it with printf()? If that is OK, would you recommend that replacement? Why?

(3) The receiver invokes msgctl(msgid, IPC_RMID, 0) to delete the message queue. Is it possible letting the system automatically delete the message queue at the end of the program's execution instead of invoking these statements directly?

(1) 5 分

两个进程通过消息队列发送/接受数据。Sender 进程从键盘读取数据，然后把读取的数据发送到消息队列里，Receiver 从消息队列里读取数据，然后把读到的数据显示在当前终端。当 Sender 从键盘读取的数据为 end，Sender 和 Receiver 结束循环（通过整形变量 running 控制），然后 Receiver 调用 msgctl(msgid, IPC_RMID, 0)删除消息队列，程序结束。

(2) 5 分

stderr 指的是文件描述符为 2，printf 使用的文件描述符为 1，默认情况他们都指向当前终端的显示器，所以对于当前程序来说，将 fprintf 替换为 printf 不会有大的影响。但是我们不建议这么做，因为将正常输出（使用文件描述符 1）和错误输出分开（使用文件描述符 2），便于程序执行时错误的跟踪和定位（比如可以使用"$... 2>logfile"）

(3) 5 分

消息队列是一种全局资源，操作系统没有办法判断哪些进程使用或将要使用这一资源，所以只能交由应用程序主动调用 msgctl(msgid, IPC_RMID, 0)删除，回收这一资源。

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

```
//omit header files

void *thread_function(void *arg);
int x;
int main()
{
	int res;
	pthread_t a_thread;
	void *thread_result;

	res = pthread_create(&a_thread,NULL,thread_function,NULL);
	if(res != 0){
		perror("Thread creation failed");
		exit(EXIT_FAILURE);
	}
	for(;;){
		x = 1;
		printf("x = %d\n", x);
		sleep(1);
	}
}
void *thread_function(void *arg){
	for(;;){
		x = -1;
		printf("x = %d\n", x);
		sleep(1);
	}
}
```

(1) Read and analyse the above code, describe how it works.
(2) Use POSIX semaphore to rewrite this program
(3) Variable x can be regarded as the resource shared by threads. Using this instance to explain how to implement shared/private resources of threads via C language.

(1) 5 分

两个线程通过调用 sleep（1）交替执行，一个线程输出 1，一个输出-1。

(2) 10 分 （注：下面的示例代码在 Ctrl+C 的信号处理函数中删除信号量资源，使用其他的方式删除信号量资源也可以）

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <pthread.h>
#include <semaphore.h>
void *thread_function(void *arg);
int x;
sem_t empty, full;
```

```
void foo(int c){
        sem_destroy(&empty);
        sem_destroy(&full);
        signal(SIGINT, SIG_DFL);
}
int main(){
        int res;
        pthread_t a_thread;
        void *thread_result;

        signal(SIGINT, foo);
        sem_init(&empty, 0, 1);
        sem_init(&full, 0, 0);

        res = pthread_create(&a_thread,NULL,thread_function,NULL);
        if(res != 0){
                perror("Thread creation failed");
                exit(EXIT_FAILURE);
        }
        for(;;){
                sem_wait(&empty);
                x = 1;
                printf("x = %d\n", x);
                sem_post(&full);
        }
}

void *thread_function(void *arg){
        for(;;){
                sem_wait(&full);
                x = -1;
                printf("x = %d\n", x);
                sem_post(&empty);
        }
}
```

(3)　5 分

多个线程共享进程内的资源，这些资源可以是全局变量或者是堆上通过动态分配（malloc）的变量；线程的私有资源一般是线程函数内的局部变量。

The following description of semaphore functions are excerpted from Linux manual pages.

int sem_init(sem_t *sem, int pshared, unsigned int value);

DESCRIPTION

sem_init() initializes the unnamed semaphore at the address pointed to by sem. The value argument specifies the initial value for the semaphore.

The pshared argument indicates whether this semaphore is to be shared between the threads of a process, or between processes.

If pshared has the value 0, then the semaphore is shared between the threads of a process, and should be located at some address that is visible to all threads (e.g., a global variable, or a variable allocated dynamically on the heap).

RETURN VALUE

sem_init() returns 0 on success; on error, -1 is returned, and errno is set to indicate the error.

int sem_wait(sem_t *sem);

DESCRIPTION

sem_wait() decrements (locks) the semaphore pointed to by sem. If the semaphore's value is greater than zero, then the decrement proceeds, and the function returns, immediately. If the semaphore currently has the value zero, then the call blocks until either it becomes possible to perform the decrement (i.e., the semaphore value rises above zero), or a signal handler interrupts the call.

RETURN VALUE

All of these functions return 0 on success; on error, the value of the semaphore is left unchanged, -1 is returned, and errno is set to indicate the error.

int sem_post(sem_t *sem);

DESCRIPTION

sem_post() increments (unlocks) the semaphore pointed to by sem. If the semaphore's value consequently becomes greater than zero, then another process or thread blocked in a sem_wait(3) call will be woken up and proceed to lock the semaphore.

RETURN VALUE

sem_post() returns 0 on success; on error, the value of the semaphore is left unchanged, -1 is returned, and errno is set to indicate the error.

int sem_destroy(sem_t *sem);

DESCRIPTION

sem_destroy() destroys the unnamed semaphore at the address pointed to by sem. Only a semaphore that has been initialized by sem_init(3) should be destroyed using sem_destroy().

RETURN VALUE

sem_destroy() returns 0 on success; on error, -1 is returned, and errno is set to indicate the error.

以下是对 semaphore 函数的描述，节选自 Linux 手册页面。

int sem_init(sem_t *sem, int pshared, unsigned int value);

描述

sem_init()在 sem 所指向的地址处初始化未命名的信号灯。 value 参数指定了 semaphore 的初始值。

pshared 参数表示这个信号是在一个进程的线程之间，还是在进程之间共享。

如果 pshared 的值为 0，那么该信号在进程的线程之间共享，并且应该位于所有线程都能看到的某个地址（例如，全局变量，或者在堆上动态分配的变量）。

返回值

sem_init()成功时返回 0；错误时返回-1，并设置 errno 以指示错误。

int sem_wait(sem_t *sem);

描述

sem_wait()递减(锁定)sem 所指向的 semaphore。如果 semaphore 的值大于 0，那么递减将继续进行，并且函数立即返回。 如果 semaphore 当前的值为 0，那么调用将被阻止，直到有可能执行递减（即 semaphore 的值高于 0），或者有信号处理程序中断调用。

返回值

所有这些函数在成功时返回 0；在错误时，信号灯的值保持不变，返回-1，并设置 errno 以指示错误。

int sem_post(sem_t *sem);

描述

sem_post()增加(解锁)sem 所指向的 semaphore。如果 semaphore 的值因此而大于 0，那么在 sem_wait(3)调用中被阻塞的另一个进程或线程将被唤醒，并继续锁定该 semaphore。

返回值

sem_post()在成功时返回 0；在错误时，semaphore 的值保持不变，返回-1，并设置 errno 来表示错误。

int sem_destroy(sem_t *sem);

描述

sem_destroy()销毁了 sem 所指向的地址上的未命名的信号灯。只有通过 sem_init(3)初始化的信号灯才可以使用 sem_destroy()进行销毁。

返回值

sem_destroy()函数成功时返回 0；错误时返回-1，并设置 errno 来表示错误。