ICS Seminar Week10 Prep

王善上 倪嘉怡 许珈铭

2023.11.27

Rules

remainder <- ordinal number in WeChat Group % 4

for all questions do

    if question number % 4 == remainder then

        you should work on it

    end

end

Q1

10、下面关于非局部跳转的描述，正确的是（　　）

A. setjmp 可以和 siglongjmp 使用同一个 jmp_buf 变量

B. setjmp 必须放在 main()函数中调用

C. 虽然 longjmp 通常不会出错，但仍然需要对其返回值进行出错判断

D. 在同一个函数中既可以出现 setjmp，也可以出现 longjmp

A. sigsetjmp和siglongjmp由信号处理函数使用，且sigsetjmp与siglongjmp相互对应，setjmp与longjmp相互对应，不能混用同一个jmp_buf变量
B. setjmp和longjmp可以在任何函数中使用
C. longjmp可能会出错，所以需要对setjmp返回值作出错判断
D. 只要用不同jmp_buf变量即可

D

Q2

6. 一段程序中阻塞了 SIGCHLD 和 SIGUSR1 信号。接下来，向它按顺序发送 SIGCHLD，SIGUSR1，SIGCHLD 信号，当程序取消阻塞继续执行时，将处理这三个信号中的哪几个？

   A. 都不处理
   B. 处理一次 SIGCHLD
   C. 处理一次 SIGCHLD，一次 SIGUSR1
   D. 处理所有三个信号

同一时间同一特定类型的信号只会保留一个，其余直接被丢弃。
阻塞SIGCHLD和SIGUSR1时，发送SIGCHLD被丢弃，发送SIGUSR1被丢弃，发送SIGCHLD被丢弃，取消阻塞时恢复一次SIGCHLD一次SIGUSR1。

C

Q3

8. 下列说法正确的是：

A. SIGTSTP 信号既不能被捕获，也不能被忽略

B. 存在信号的默认处理行为是进程停止直到被 SIGCONT 信号重启

C. 系统调用不能被中断，因为那是操作系统的工作

D. 子进程能给父进程发送信号，但不能发送给兄弟进程

A. SIGSTOP SIGKILL既不能被捕获也不能被忽略
B. 正确
C. 信号可能中断系统调用
D. 可以通过kill函数发送信号给自己所在的进程组（pid=0）

B

Q4

10. 下列关于信号的说法不正确的是：

A. 在键盘上输入 Ctrl-C 会导致内核发送一个 SIGINT 信号到前台进程组中的每个进程

B. 每种类型最多只能有一个未处理的信号

C. SIGINT 的处理函数不能被另一个 SIGINT 信号中断

D. 进程可以通过使用 signal 函数修改和 SIGSTOP 相关联的默认行为

A. 正确
B. 正确
C. 信号处理函数默认阻塞相同类型的待处理信号
D. SIGSTOP默认行为终止不能被修改

D

Q5

5. 关于进程，以下说法正确的是：

A. 没有设置模式位时，进程运行在用户模式中，允许执行特权指令，例如发起 I/O 操作。

B. 调用 waitpid(-1, NULL, WNOHANG & WUNTRACED)会立即返回：如果调用进程的所有子进程都没有被停止或终止，则返回 0；如果有停止或终止的子进程，则返回其中一个的 ID。

C. execve 函数的第三个参数 envp 指向一个以 null 结尾的指针数组，其中每一个指针指向一个形如”name=value”的环境变量字符串。

D. 进程可以通过使用 signal 函数修改和信号相关联的默认行为，唯一的例外是 SIGKILL，它的默认行为是不能修改的。

A. 没有设置模式位时， 进程就运行在用户模式中，不允许执行特权指令
B. 应该是 或 而非 与
C. 正确
D. 不是唯一， 还有SIGSTOP

C

Q6

13. 对于 Linux 系统，下列说法错误的是：

A. 在单核 CPU 上，所有看似并行的逻辑流实际上是并发的。

B. 用户模式不可访问/proc 文件系统中包含的内核数据结构的内容。

C.在 execve 函数参数的 argv 数组加入"> 1.txt"，不能自动实现 IO 重定向。

D. 即便在信号处理程序中调用 printf 前阻塞所有信号，也不一定安全。

A. 并行 = 并发 + 不同处理核

B. 错误，用户模式只能通过系统调用接口间接访问

C. I/O 重定向由shell处理，使用特殊符号（`>`或`<`）指定输入和输出的来源或目标，调用相应的系统调用（如`open`、`dup2`）实现重定向。而`execve`是系统调用，在当前进程中加载并执行一个新的程序。将 `"> 1.txt"` 的字符串放入 `argv` 数组中不能实现I/O重定向，因为`execve`仅仅是执行指定的程序，而不会解释这个字符串作为shell命令。在 `execve` 的上下文中，字符串 `"> 1.txt"` 会被当作普通的命令行参数传递给新程序，不会被解释为I/O重定向操作。

D. printf是非异步安全的，使用了全局变量或状态，例如内部的文件锁、缓冲区。即便在调用 `printf` 之前阻塞了所有信号，其他线程或信号处理程序可能仍能在阻塞期间修改共享资源，导致竞争条件。

B

Q7

1、关于进程和异常控制流，以下说法正确的是：

A、调用 waitpid (-1, NULL, WNOHANG & WUNTRACED) 会立即返回：如果调用进程的所有子进程都没有被停止或终止，则返回 0；如果有停止或终止的子进程，则返回其中一个的 ID。

B、进程可以通过使用 signal 函数修改和信号相关联的默认行为，唯一的例外是 SIGKILL，它的默认行为是不能修改的。

C、从内核态转换到用户态有多种方法，例如设置程序状态字；从用户态转换到内核态的唯一途径是通过中断/异常/陷入机制。

D、中断一定是异步发生的，陷阱可能是同步发生的，也可能是异步发生的。

A. 应该是 或 而非 与
B. 不是唯一， 还有SIGSTOP
C. 正
D. 陷阱是同步的

C

Q8

1. 下列关于系统 I/O 的说法中，正确的是（）：
A. Linux shell 创建的每个进程开始时都有三个打开的文件：标准输入（描述符为 0），标准输出（描述符为 1），标准错误（描述符为 2），这使得程序始终不能使用保留的描述符 0,1,2 读写其他文件。
B. Unix I/O 的 read/write 函数是异步信号安全的，故可以在信号处理函数中使用。
C. RIO 函数包的健壮性保证了对于同一个文件描述符，任意顺序调用 RIO 包中的任意函数不会造成问题。
D. 使用 int fd1 = open("ICS.txt", O_RDWR); 打开 ICS.txt 文件后，再用 int fd2 = open("ICS.txt", O_RDWR); 再次打开文件，会使得 fd1 对应的打开文件表中的引用计数 refcnt 加一。

A.可通过重定向更改
C.带缓冲和不带缓冲的不能交叉使用
D.两次打开在打开文件表中对应两个不同的元素，第
打开不会影响第一次打开

B

Q9

19. 下列系统 I/O 的说法中，正确的是(　　)

A. C 语言中的标准 I/O 函数在不同操作系统中的实现代码一样

B. 对于同一个文件描述符，混用 RIO 包中的 rio_readnb 和 rio_readn 两个函数不会造成问题

C. C 语言中的标准 I/O 函数是异步线程安全的

D. 使用 I/O 缓冲区可以减少系统调用的次数，从而加快 I/O 的速度

A. linux:open , windows:winopen
B. 带缓冲和不带缓冲的不能交叉使用
C. printf是一个比较大的函数， 可能主程序printf执行了一半，被打断进入handler， 在handler调用可能会导致主程序出错

D

Q10

15. 关于 IO 操作，以下说法中正确的是（）

A. 由于 RIO 包的健壮性，所以 RIO 中的函数都可以交叉调用

B. 成功调用 open 函数后，返回一个不小于 3 的文件描述符

C. 调用 Unix I/O 开销较大，标准 I/O 库使用缓冲区来加快 I/O 的速度

D. 和描述符表一样，每个进程拥有独立的打开文件表

A. 带缓冲和不带缓冲的不能交叉使用
B.close之后，也可以返回0/1/2
C.开销在于多次陷入系统内核，缓冲区降低陷入次数
D.每个进程共用一个

C

Q11

14. 以下关于文件 I/O 的说法中，正确的是：

A. 文件重定向（dup 和 dup2）操作仅仅改变了文件描述符的指向，不会改变打开文件表中的内容

B. 进程调用 fork()时，可能对文件描述符表和打开文件表采用写时拷贝（Copy on Write）机制

C. 对同一描述符，rio_readlineb 和 rio_readnb 可以任意交叉使用，rio_readn 和 rio_writen 也可以任意交叉使用

D. RIO 中包括无缓冲的输入输出函数和带缓冲的输入输出函数，使用带缓冲的输入输出函数时，要先声明一个 rio_t 类型变量并调用 rio_readinitb 函数

1.会改变refcnt
2.不会copy打开文件表
4.无带缓冲输出函数

C

Q12

6. 假设某进程有且仅有五个已打开的文件描述符：0~4，分别引用了五个不同的文件，尝试运行以下代码：

dup2(3, 2)； dup2(0, 3)； dup2(1, 10)； dup2(10, 4)； dup2(4, 0)；

关于得到的结果，说法正确的是：

A. 运行正常完成，现在有四个描述符引用同一个文件

B. 运行正常完成，现在进程共引用四个不同的文件

C. 由于试图从一个未打开的描述符进行复制，发生错误

D. 由于试图向一个未打开的描述符进行复制，发生错误

fd表依次如下：

0 1 2 3 4

0 1 3 3 4

0 1 3 0 4

0 1 3 0 4 ... 1

0 1 3 0 1 ... 1

1 1 3 0 1 ... 1

B.3个（0,1,3）

C.复制时oldfd都存在

D.不会发生错误

A

Q13

18.ICS.txt 中包含 3000 个字符，考虑如下代码段：

```
int main(int argc, char** argv) {
   int fd = open("ICS.txt", O_CREAT | O_RDWR, S_IRUSR |
S_IWUSR);
   write(fd, "ICS", 3);

   char buf[128];
   int i;
   for (i = 0; i < 10; i++) {
      int fd1 = open("ICS.txt", O_RDWR);
      int fd2 = dup(fd1);

      int cnt = read(fd1, buf, 128);
      write(fd2, buf, cnt);
   }
   return 0;
}
```

上述代码执行完后，ICS.txt 中包含多少个字符？(　　)（假设所有系统调用都成功）

A. 3　　B. 256　　C. 3000　　D. 3072

最多修改到256，远不到3000
所以只会原地覆盖，不会增加长度

C

Q14

14. 考虑以下代码，假设 result.txt 中的初始内容为“666666”

```
char *str1 = "6666";
char *str2 = "2333";
char *str3 = "hhhh";
int fd1, fd2, fd3, i;

fd1 = open("result.txt", O_RDWR);
fd2 = open("result.txt", O_RDWR);
dup2(fd1, fd2);

for (i = 0; i < 5; ++i) {
    fd3 = open("result.txt", O_RDWR);
    write(fd1, str1, 4);
    write(fd2, str2, 4);
    write(fd3, str3, 4);
    close(fd3);
}

close(fd1); close(fd2);
```

假设所有系统调用均成功，则这段代码执行结束后，result.txt 的内容中有（）个“6”

A. 6

B. 16

C. 20

D. 22

6666233366662333666623336666233366662333
hhhh
fd1和fd2指向同一个打开文件表的元素
fd3一直只覆盖0~3位置
故16个6

B

Q15

15. 以下程序执行完成后，ICS.txt 文件中的内容是：

```
int main(int argc, char** argv) {
    int fd1 = open("ICS.txt", O_CREAT|O_RDWR,
                               S_IRUSR|S_IWUSR);
    write(fd1, "ics ", 4);
    int fd2 = fd1;
    int fd3 = dup(fd2);
    int fd4 = open("ICS.txt", O_APPEND|O_RDWR);
    write(fd2, "segmentation fault ", 19);
    write(fd4, "tao", 3);
    int fd5 = fd4;
    dup2(fd3, fd5);
    write(fd4, "lab", 3);
    close(fd1);
    return 0;
}
```

A. ics segmentation fault tao
B. ics segmentation fault lab
C. ics taomentation fault lab
D. tao segmentation fault lab

只有B是一个正确的句子，故选B（x

fd1,fd2,fd3指向同一个
fd4不同，但有append模式
执行完write tao，文件内容为：
ics segmentation fault tao
此时fd1的文件位置在fault后面

后执行dup2，fd4也被fd1同化
此时从fault后面开始写
所以为
ics segmentation fault lab

B

Q16

2. 考虑以下代码，假设 ICS.txt 中的初始内容为"ICS!!!ics!!!"：

```
int fd = open("ICS.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
for (int i = 0; i < 2; ++i){
    int fd1 = open("ICS.txt", O_RDWR | O_APPEND);
    int fd2 = open("ICS.txt", O_RDWR);
    write(fd2, "!!!!!!", 6);
    write(fd1, "ICS", 3);
    write(fd, "ics", 3);
}
```

假设所有系统调用均成功，则这段代码执行结束后，ICS.txt 的内容为（）：

A．ICSics

B．!!!icsICS

C．!!!icsics!!!ICSICS

D．!!!icsICSICS

依次写入：

!!!!!!

ICS

ics

!!!!!!

ICS

ics

于是最后写入的：

!!!icsICSICS

D

Q17

Part II

请阅读以下程序，然后回答问题（假设程序中的函数调用都可以正确执行，且每条语句都是原子动作）：

```
pid_t pid;
int even = 0;
int counter1 = 0;
int counter2 = 1;
void handler1(int sig) {
    if (even % 2 == 0) {
        printf("%d\n", counter1);
        counter1 = ____A_____ ;
        } else {
            printf("%d\n", counter2);
            counter2 = ____B____ ;
        }
        even = even+___C_____ ;
    }
void handler2(int sig) {
        if (_____D_____) {
        counter1 = even*even;
        } else {
        counter2 = even*even;
        }
}
```

```
int main() {
    signal(SIGUSR1, handler1);
    signal(SIGUSR2, handler2);
   if ((pid = fork()) == 0) {
      while (1) {};
   }
      while (even < 30) {
         kill(pid, ___E___);
         sleep(1);
         kill(pid, ___F___);
         sleep(1);
         even = even+___G____ ;
      }
      kill(pid, SIGKILL);
      exit(0);
   }
```

（1）完成程序，使得程序在输出的数字为以下 Q 队列的前 30 项，Q 队列定义如下：

$$Q_0 = 0,\ Q_1 = 1,\ Q_{n+2} = \begin{cases} Q_n + 1, & \text{n\%2} = 0 \\ Q_n \times 2, & \text{n\%2} \neq 0 \end{cases} \quad (n = 0,1,2,3,\ldots)$$

（注：若某个位置中的程序内容，对本次程序执行结果没有影响，请在相应位置填写“无关”）

A: _____

B: _____

C: _____

D: _____

E: _____

F: _____

G: _____

Q17

Part II

请阅读以下程序，然后回答问题（假设程序中的函数调用都可以正确执行，且每条语句都是原子动作）：

```
pid_t pid;
int even = 0;
int counter1 = 0;
int counter2 = 1;
void handler1(int sig) {
    if (even % 2 == 0) {
        printf("%d\n", counter1);
        counter1 = ____A____ ;
        } else {
            printf("%d\n", counter2);
            counter2 = ____B____ ;
        }
        even = even+___C____ ;
    }
void handler2(int sig) {
        if (____D____) {
        counter1 = even*even;
        } else {
        counter2 = even*even;
        }
}
```

```
int main() {
    signal(SIGUSR1, handler1);
    signal(SIGUSR2, handler2);
    if ((pid = fork()) == 0) {
        while (1) {};
    }
        while (even < 30) {
            kill(pid, ___E___);
            sleep(1);
            kill(pid, ___F___);
            sleep(1);
            even = even+___G____ ;
        }
        kill(pid, SIGKILL);
        exit(0);
}
```

EF：由队列迭代公式，奇数和偶数分别基于上一个值加一或乘二得到新值，所以都应调用handler1，handler2为干扰项；注意SIGUSRx是信号，handlerx是信号处理函数

AB：由公式，n%2==0时加一，否则乘2；根据Q0和Q1也可判断

C：子进程接收到信号，每次+1

G：父进程，+2

（1）完成程序，使得程序在输出的数字为以下 Q 队列的前 30 项，Q 队列定义如下：

$$Q_0 = 0,\ Q_1 = 1,\ Q_{n+2} = \begin{cases} Q_n + 1, & n\%2 = 0 \\ Q_n \times 2, & n\%2 \neq 0 \end{cases} \quad (n = 0,1,2,3,\ldots)$$

（注：若某个位置中的程序内容，对本次程序执行结果没有影响，请在相应位置填写“无关”）

A: _____

B: _____

C: _____

D: _____

E: _____

F: _____

G: _____

counter1 + 1
counter2 * 2
1
无关
SIGUSR1
SIGUSR1
2

Q18

Part II

请阅读以下程序，然后回答问题（假设程序中的函数调用都可以正确执行，且每条语句都是原子动作）：

```
pid_t pid;
int even = 0;
int counter1 = 0;
int counter2 = 1;
void handler1(int sig) {
    if (even % 2 == 0) {
        printf("%d\n", counter1);
        counter1 = ____A_____ ;
        } else {
            printf("%d\n", counter2);
            counter2 = ____B____ ;
        }
        even = even+___C_____ ;
    }
void handler2(int sig) {
        if (_____D_____) {
        counter1 = even*even;
        } else {
        counter2 = even*even;
        }
}
```

```
int main() {
    signal(SIGUSR1, handler1);
    signal(SIGUSR2, handler2);

    if ((pid = fork()) == 0) {
        while (1) {};
    }
        while (even < 30) {
            kill(pid, ___E___);
            sleep(1);
            kill(pid, ___F___);
            sleep(1);
            even = even+___G____ ;
        }
        kill(pid, SIGKILL);
        exit(0);
    }
```

（1）完成程序，使得程序在输出的数字为以下 Q 队列的前 30 项，Q 队列定义如下：

$$Q_0 = 0,\ Q_1 = 1,\ Q_{n+2} = \begin{cases} Q_n + 1, & \text{n\%2} = 0 \\ Q_n \times 2, & \text{n\%2} \neq 0 \end{cases} \quad (n = 0,1,2,3,\ldots)$$

（注：若某个位置中的程序内容，对本次程序执行结果没有影响，请在相应位置填写“无关”）

A: _____

B: _____

C: _____

D: _____

E: _____

F: _____

G: _____

Q18

Part II

请阅读以下程序，然后回答问题（假设程序中的函数调用都可以正确执行，且每条语句都是原子动作）：

```
pid_t pid;
int even = 0;
int counter1 = 0;
int counter2 = 1;
void handler1(int sig) {
    if (even % 2 == 0) {
        printf("%d\n", counter1);
        counter1 = ____A____ ;
        } else {
            printf("%d\n", counter2);
            counter2 = ____B____ ;
        }
        even = even+___C____ ;
    }
void handler2(int sig) {
        if (____D____) {
        counter1 = even*even;
        } else {
        counter2 = even*even;
        }
}
```

```
int main() {
    signal(SIGUSR1, handler1);
    signal(SIGUSR2, handler2);
    if ((pid = fork()) == 0) {
        while (1) {};
    }
        while (even < 30) {
            kill(pid, ___E___);
            sleep(1);
            kill(pid, ___F___);
            sleep(1);
            even = even+___G____ ;
        }
        kill(pid, SIGKILL);
        exit(0);
}
```

（1）完成程序，使得程序在输出的数字为以下 Q 队列的前 30 项，Q 队列定义如下：

$$Q_0 = 0,\ Q_1 = 1,\ Q_{n+2} = \begin{cases} Q_n + 1, & \text{n\%2} = 0 \\ Q_n \times 2, & \text{n\%2} \neq 0 \end{cases} \quad (n = 0,1,2,3,\ldots)$$

（注：若某个位置中的程序内容，对本次程序执行结果没有影响，请在相应位置填写“无关”）

A: _____

B: _____

C: _____

D: _____

E: _____

F: _____

G: _____

counter1 + 1
counter2 * 2
1
无关
SIGUSR1
SIGUSR1
2

Q19

第四题（10 分）

分析以下C程序，其中f1.txt和f2.txt为已有用户有读写的文件，初始文件内容为空。

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <sys/types.h>
4.  #include <sys/stat.h>
5.  #include <fcntl.h>
6.  #include <unistd.h>
7.  #include <sys/types.h>
8.  #include <sys/wait.h>
9.
10. int main()
11. {
12.   int fd1,fd2,fd3,fd4;
13.   int pid;
14.   int c=1;
15.   fd1=open("./f1.txt",O_WRONLY,0);
16.   fd2=open("./f1.txt",O_WRONLY,0);
17.
18.   printf("fd1=%d,fd2=%d;\n",fd1,fd2);
19.
20.   write(fd1,"EECSPKU",7);
21.   write(fd2,"2019",4);
22.
23.   close(fd2);
24.
25.   fd3=open("./f2.txt",O_WRONLY,0);
26.   fd4=dup(fd3);
27.
28.   printf("fd3=%d,fd4=%d;\n",fd3,fd4);
29.
30.   pid=fork();
31.   if((pid==0)) {
32.     c--;
33.     write(fd3,"PKU",3);
34.     write(fd4,"ICS",3);
35.     printf("c= %d\n",c);
36.   }
37.   else {
38.     waitpid(-1,NULL,0);
39.     c++;
40.     write(fd3,"2019",4);
41.     close(fd3);
42.     close(fd4);
43.     printf("c= %d\n",c);
44.   }
45.
46.   if(c)
47.    write(fd1,"CS",2);
48.   c++;
49.   close(fd1);
50.   printf("c=%d\n",c);
51. }
```

当程序正确运行后，填写输出结果：

（1） 程序第 18 行：fd1=__①__，fd2=__②__；

（2） 程序第 28 行：fd3=__①__，fd4=__②__；

（3） 程序第35、43、50行输出c的值依次分别为：_____；

（4） 文件 f1.txt 中的内容为：_____；

（5） 文件 f2.txt 中的内容为：_____。

Q19

第四题（10 分）

分析以下C程序，其中f1.txt和f2.txt为已有用户有读写的文件，初始文件内容为空。

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <sys/types.h>
4.  #include <sys/stat.h>
5.  #include <fcntl.h>
6.  #include <unistd.h>
7.  #include <sys/types.h>
8.  #include <sys/wait.h>
9.
10. int main()
11. {
12.   int fd1,fd2,fd3,fd4;
13.   int pid;
14.   int c=1;
15.   fd1=open("./f1.txt",O_WRONLY,0);
16.   fd2=open("./f1.txt",O_WRONLY,0);
17.
18.   printf("fd1=%d,fd2=%d;\n",fd1,fd2);
19.
20.   write(fd1,"EECSPKU",7);
21.   write(fd2,"2019",4);
22.
23.   close(fd2);
24.
25.   fd3=open("./f2.txt",O_WRONLY,0);
26.   fd4=dup(fd3);
27.
28.   printf("fd3=%d,fd4=%d;\n",fd3,fd4);
29.
30.   pid=fork();
31.   if((pid==0)) {
32.     c--;
33.     write(fd3,"PKU",3);
34.     write(fd4,"ICS",3);
35.     printf("c= %d\n",c);
36.   }
37.   else {
38.     waitpid(-1,NULL,0);
39.     c++;
40.     write(fd3,"2019",4);
41.     close(fd3);
42.     close(fd4);
43.     printf("c= %d\n",c);
44.   }
45.
46.   if(c)
47.    write(fd1,"CS",2);
48.   c++;
49.   close(fd1);
50.   printf("c=%d\n",c);
51. }
```

3, 4
4, 5
0, 1, 2, 3
2019PKUCS
PKUICS2019

当程序正确运行后，填写输出结果：

（1） 程序第 18 行：fd1=__①__，fd2=__②__；

（2） 程序第 28 行：fd3=__①__，fd4=__②__；

（3） 程序第 35、43、50 行输出 c 的值依次分别为：_____；

（4） 文件 f1.txt 中的内容为：_____；

（5） 文件 f2.txt 中的内容为：_____。

Q20

第四题（10 分）

分析以下C程序，其中f1.txt和f2.txt为已有用户有读写的文件，初始文件内容为空。

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <sys/types.h>
4.  #include <sys/stat.h>
5.  #include <fcntl.h>
6.  #include <unistd.h>
7.  #include <sys/types.h>
8.  #include <sys/wait.h>
9.
10. int main()
11. {
12.   int fd1,fd2,fd3,fd4;
13.   int pid;
14.   int c=1;
15.   fd1=open("./f1.txt",O_WRONLY,0);
16.   fd2=open("./f1.txt",O_WRONLY,0);
17.
18.   printf("fd1=%d,fd2=%d;\n",fd1,fd2);
19.
20.   write(fd1,"EECSPKU",7);
21.   write(fd2,"2019",4);
22.
23.   close(fd2);
24.
25.   fd3=open("./f2.txt",O_WRONLY,0);
26.   fd4=dup(fd3);
27.
28.   printf("fd3=%d,fd4=%d;\n",fd3,fd4);
29.
30.   pid=fork();
31.   if((pid==0)) {
32.     c--;
33.     write(fd3,"PKU",3);
34.     write(fd4,"ICS",3);
35.     printf("c= %d\n",c);
36.   }
37.   else {
38.     waitpid(-1,NULL,0);
39.     c++;
40.     write(fd3,"2019",4);
41.     close(fd3);
42.     close(fd4);
43.     printf("c= %d\n",c);
44.   }
45.
46.   if(c)
47.    write(fd1,"CS",2);
48.   c++;
49.   close(fd1);
50.   printf("c=%d\n",c);
51. }
```

当程序正确运行后，填写输出结果：

（1） 程序第 18 行：fd1=__①__，fd2=__②__；

（2） 程序第 28 行：fd3=__①__，fd4=__②__；

（3） 程序第35、43、50行输出c的值依次分别为：_____；

（4） 文件 f1.txt 中的内容为：_____；

（5） 文件 f2.txt 中的内容为：_____。

Q20

第四题（10 分）

分析以下C程序，其中f1.txt和f2.txt为已有用户有读写的文件，初始文件内容为空。

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <sys/types.h>
4.  #include <sys/stat.h>
5.  #include <fcntl.h>
6.  #include <unistd.h>
7.  #include <sys/types.h>
8.  #include <sys/wait.h>
9.
10. int main()
11. {
12.   int fd1,fd2,fd3,fd4;
13.   int pid;
14.   int c=1;
15.   fd1=open("./f1.txt",O_WRONLY,0);
16.   fd2=open("./f1.txt",O_WRONLY,0);
17.
18.   printf("fd1=%d,fd2=%d;\n",fd1,fd2);
19.
20.   write(fd1,"EECSPKU",7);
21.   write(fd2,"2019",4);
22.
23.   close(fd2);
24.
25.   fd3=open("./f2.txt",O_WRONLY,0);
26.   fd4=dup(fd3);
27.
28.   printf("fd3=%d,fd4=%d;\n",fd3,fd4);
29.
30.   pid=fork();
31.   if((pid==0)) {
32.     c--;
33.     write(fd3,"PKU",3);
34.     write(fd4,"ICS",3);
35.     printf("c= %d\n",c);
36.   }
37.   else {
38.     waitpid(-1,NULL,0);
39.     c++;
40.     write(fd3,"2019",4);
41.     close(fd3);
42.     close(fd4);
43.     printf("c= %d\n",c);
44.   }
45.
46.   if(c)
47.    write(fd1,"CS",2);
48.   c++;
49.   close(fd1);
50.   printf("c=%d\n",c);
51. }
```

(1)(2)fd1,fd2,fd3,fd4都是直接选最小可用的数（
fd2被close掉了，所以fd3用的是4)
(3)c初值为1，先运行子进程，32句--c，于是3
c=0，然后48句++c，于是50句c=1，然后运行
程，39句++c，43句c=2，48句++c，50句c=4
(4)依次输出：
EECSPKU
2019
CS
(5)依次输出：
PKU
ICS
2019

3, 4
4, 5
0, 1, 2, 3
2019PKUCS
PKUICS2019

当程序正确运行后，填写输出结果：

（1） 程序第 18 行：fd1=__①__，fd2=__②__；

（2） 程序第 28 行：fd3=__①__，fd4=__②__；

（3） 程序第 35、43、50 行输出 c 的值依次分别为：_____；

（4） 文件 f1.txt 中的内容为：_____；

（5） 文件 f2.txt 中的内容为：_____。

Q21

6. Y86 指令 rmmovl 的 SEQ 实现如下图所示，其中①和②分别为：

	rmmovl rA, D(rB)
Fetch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_4[PC+2] valP ← ①
Decode	valA ← R[rA] valB ← R[rB]
Execute	valE ← ②
Memory	M_4[valE] ← valA
Write back	
PC update	PC ← valP

A．PC + 4，valB + 4　　B．PC + 4，valB + valC

C．PC + 6，valB + 4　　D．PC + 6，valB + valC

valC是M4[PC+2]，valP是从PC+6起的地址
valE是D(rB)的内存位置，valB—rB，valC—D

D

Q22

2. 在本课程的 PIPE 流水线中，下列情况会出现数据冒险的是：
   A. 当前指令会改变下一条指令的目的操作数
   B. 当前指令会改变下一条指令的源操作数
   C. 下一条指令会改变当前指令的目的操作数
   D. 下一条指令会改变当前指令的源操作数

只有（对于有数据转发的情况下）同一个操作数，当前改变并且下一条调用时会产生数据冒险

B

Q23

3． 缓冲区溢出会带来程序风险，下列避免方法中错误的是：
A．在栈中存放特殊字段用于检测是否发生缓冲区溢出
B．避免使用有风险的库函数，如 gets 等
C．随机设置栈的偏移地址
D．分配尽可能大的缓冲区数组

A.金丝雀
D.只要输入够多，照样可以溢出

D

Q24

4. 现有四级指令流水线，分别完成取指、取数、运算、传送结果 4 步操作。若完成上述操作的时间依次为 9ns、10ns、6ns、8ns，则流水线的操作周期应设计为 _____ ns。

A. 6　　B. 8　　C. 9　　D. 10

取max

D