1. 关于进程和异常控制流，以下说法正确的是：
2. 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。
3. 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。
4. 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。
5. 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。

答案：C

简单

A中option参数应该使用 | 运算结合。B中SIGKILL不是唯一的例外，例外共有两个：SIGKILL、SIGSTOP。D陷阱一定是同步发生的

2、异常可以分为四类：中断、陷阱、故障、终止。以下都属于中断的是：

A、I/O请求完成、系统定时器的信号

B、除零、缺页

C、系统调用、非法指令

D、键盘Ctrl+C、机器检查

答案：A

中等

1. 中断、中断
2. 故障、故障
3. 陷阱、终止
4. 中断、终止

P506

一. 异常控制流

**PART A.** Alice 想用两个进程来顺序输出奇数和偶数，请你帮她补全代码。她的做法是：父进程首先 fork 出两个子进程，之后子进程之间互相通信，顺序输出 1,2,…,N。请严格按照注释描述的功能填写代码，**假设兄弟进程的 pid 是相邻的**。

int N;

int nxt; //表示该子进程下一个需要输出的数

int pid\_1 = 0; //第一次fork出的子进程pid

int pid\_2 = 0; //第二次fork出的子进程pid

// All child process should do their work in handler 1

void handler1(int sig) {

printf("%d\n", nxt);

nxt += 2;

if (nxt & 1) kill(getpid()+1, SIGUSR1);

else {

assert(pid\_1 != 0);

\_\_\_\_\_\_\_\_A\_\_\_\_\_\_\_\_ // 通知兄弟进程

}

if (nxt > N) exit(0); // 子进程完成输出后退出

}

int main(int argc, char\* argv[]) {

\_\_\_\_\_\_\_\_B\_\_\_\_\_\_\_\_ //将handler1绑定到SIGUSR1上

N = atoi(argv[1]);

nxt = 1;

if ((pid\_1 = fork()) != 0) {

\_\_\_\_\_\_\_\_C\_\_\_\_\_\_\_; // 设置nxt的初值

if ((pid\_2 = fork()) != 0) { // 该进程是父进程

kill(pid\_1, SIGUSR1);

goto wait\_til\_end; //跳转并等待子进程结束

}

}

while (1) { sleep(1); } // 子进程会在此循环直到输出完成

wait\_til\_end:

int status;

while (\_\_\_\_\_\_\_D\_\_\_\_\_\_\_) // 用waitpid等待子进程结束，注意status

assert(WIFEXITED(status));

return 0;

}

（4分, 每空1分）

（难度：简单）A.

（难度：简单）B.

（难度：简单）C.

（难度：简单）D.

**PART B.** 阅读如下 C 代码，回答问题

int counter = 0;

int pid = 0;

int N = 2;

void handler1(int sig) {

counter++;

printf("%d", counter); fflush(stdout);

// Kill(pid, SIGUSR2);

}

void handler2(int sig) {

printf("R"); fflush(stdout);

}

int main() {

Signal(SIGUSR1, handler1);

Signal(SIGUSR2, handler2);

if ((pid = Fork()) == 0) { // child

for (int i = 0; i < N; ++i) {

printf("C"); fflush(stdout);

Kill(Getppid(), SIGUSR1);

}

} else { // parent

Wait(NULL);

}

return 0;

}

1.（难度：简单）进程在何时检查待处理信号，并调用相应的signal handler处理信号？\_\_\_\_\_\_\_\_（1分）

A.随时 B.用户态切换到内核态 C.内核态切换到用户态

2.（难度：简单）在两次检查并处理信号量的时间间隙中，如果进程接收到n个**相同**信号，那么进程实际会处理几个信号？\_\_\_\_\_\_\_\_（1分）

A.1 B.1到n之间的随机数值 C.n

3.（难度：中等）如果N=2，所有可能的输出为：\_\_\_\_\_\_\_\_

（全部选对得 2 分，部分选对得 1 分，选错不得分）

A.CC B.CC1 C. CC12 D. C1C2

4.（难度：难）如果 N=2，并且取消hanlder1中的注释（第7行），所有**不可能**的输出为：\_\_\_\_\_\_\_\_

（全部选对得 2 分，部分选对得 1 分，选错不得分）

A. CC B. CC1 C. CC1R2 D. C1RC2 E. C1CR2

答：

**PART A.**

A: kill(pid\_1, SIGUSR1);或 kill(getpid()−1, SIGUSR1);

B: signal(SIGUSR1, handler1);

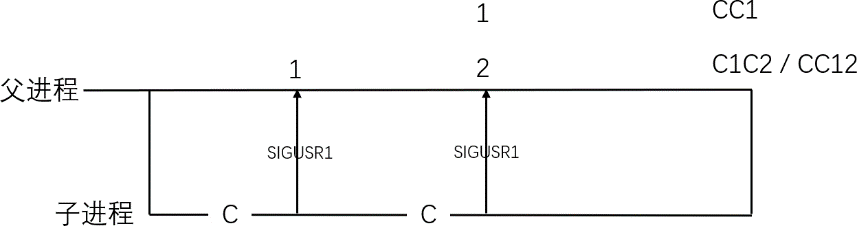
C: nxt = 2;

D: waitpid(0, &status, 0)> 0 或 waitpid(−1, &status, 0)> 0

**PART B.**

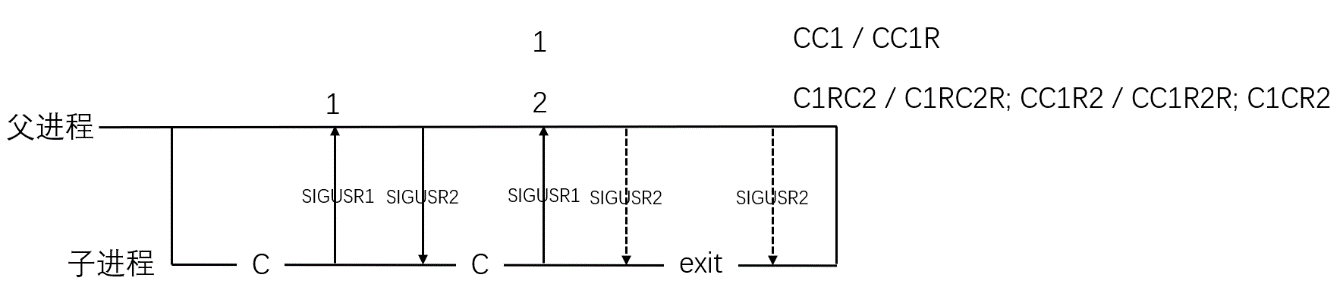
1.C

2.A

3.BCD: CC1, C1C2, CC12

4.A: CC

所有可能的输出为CC1, CC1R, CC1R2, CC1R2R, C1RC2, C1RC2R, C1CR2。

比如，CC1是可能的，因为子进程可以在接收到父进程的SIGUSR2之前退出。

解题要点：

1. 进程只在内核态切换到用户态时检测并处理信号

2. 在处理信号之前相同信号接收到多次，只按照一次计算

3. 注意子进程可以比父进程先结束，父进程将无法发送信号给子进程