

## ICS 第八章

### 【异常的基本概念】

#### 1. 区分各个异常（打√）

| 异常的种类           | 是同步（Sync）的吗？ | 可能的行为？ |        |        |
|-----------------|--------------|--------|--------|--------|
|                 |              | 重复当前指令 | 执行下条指令 | 结束进程运行 |
| 中断<br>Interrupt |              |        |        |        |
| 陷入<br>Trap      |              |        |        |        |
| 故障<br>Fault     |              |        |        |        |
| 终止<br>Abort     |              |        |        |        |

| 行为  | 中<br>断 | 陷<br>入 | 故<br>障 | 终<br>止 |
|---|--------|--------|--------|--------|
| 执行指令 <code>mov \$57, %eax; syscall</code>     |        |        |        |        |
| 程序执行过程中，发现它所使用的物理内存损坏了                        |        |        |        |        |
| 程序执行过程中，试图往 main 函数的内存中写入数据                   |        |        |        |        |
| 按下键盘  |        |        |        |        |
| 磁盘读出了一块数据                                     |        |        |        |        |
| 用户程序执行了指令 <code>lgdt</code> ，但是这个指令只能在内核模式下执行 |        |        |        |        |

### 【fork】

#### 2. 阅读下列程序

```
int main() {
    char c = 'A';
    printf("%c", c); fflush(stdout);
    if (fork() == 0) {
        c++;
        printf("%c", c); fflush(stdout);
    } else {
        printf("%c", c); fflush(stdout);
        fork();
    }
    c++;
    printf("%c", c); fflush(stdout);
    return 0;
}
```

假设系统调用成功，所有子进程都正常运行。判断下列哪些输出是可能的：

- (1) (     ) AABBBBC    (2) (     ) ABCABB    (3) (     ) ABBABC  
 (4) (     ) AACBBBC    (5) (     ) ABABCB    (6) (     ) ABCBAB

### 【wait】

#### 3. 阅读下列程序

```
int main() {
    int child_status;
    char c = 'A';
    printf("%c", c); fflush(stdout);
    c++;
    if (fork() == 0) {
        printf("%c", c); fflush(stdout);
        c++;
        fork();
    } else {
        printf("%c", c); fflush(stdout);
        c += 2;
        wait(&child_status);
    }
    printf("%c", c); fflush(stdout);
    exit(0);
}
```

假设系统调用成功，所有子进程都正常运行。判断下列哪些输出是可能的：

- (1) (     ) ABBCDD    (2) (     ) ABBCDC    (3) (     ) ABBDCC  
(4) (     ) ABDBCC    (5) (     ) ABCDBC    (6) (     ) ABCDCB

### 【信号】

#### 4. 阅读下列程序

```
void handler() {
    printf("D\n");
    return;
}

int main() {
    signal(SIGCHLD, handler);
    if (fork() > 0) {
        /* parent */
        printf("A\n");
    } else {
        printf("B\n");
    }
    printf("C\n");
    exit(0);
}
```

假设系统调用成功，所有子进程都正常运行。判断下列哪些输出是可能的（忽略换行）：

- (1) (     ) ACBC      (2) (     ) ABCCD    (3) (     ) ACBDC  
(4) (     ) ABDCC    (5) (     ) BCDAC    (6) (     ) ABCC

5. 在 2018 年的 ICS 课堂上，老师给同学布置了一个作业，在 LINUX 上写出一份代码，运行它以后，输出能创建的进程的最大数目。下面是几位同学的答案。

**PART A.** Alice 同学的答案是：

```
int main() {
    int pid;
    int count = 1;
    while ((pid = fork()) != 0){
        // parent process
        count++;
    }
    If (pid == 0) {
        // child process
        exit(0);
    }
    printf("max = %d ", count);
}
```

这段代码不能够正确运行，原因在于对 fork 的返回值处理得不正确。请修改至多一处代码，使得程序正确运行。

**PART B.** Bob 同学对 Alice 同学修改过后的正确代码发出了疑问。Bob 同学认为，由于进程的调度时间和顺序都是不确定的，因此有的时候会调度到子进程，子进程执行 `exit(0)` 以后就结束了，因此父进程可以创建更多的进程，所以 Alice 的代码输出的答案大于真实的上限。

请问，Bob 的说法正确吗？如果正确，请指出 Alice 应当如何修改代码，以避免 Bob 提到的问题。如果 Bob 的说法错误，请指出他错在何处。

**PART C.** Carol 同学的答案是：

```
int main() {
    int pid;
    int count=1;
    while ((pid = fork()) > 0){
        // parent process
        count++;
    }
    If (pid == 0) {
        // child process
        while(1)
            sleep(1);
    }
    printf("max = %d ", count);
}
```

运行 Carol 同学的答案两次，发现结果分别如下：

```
linux $ ./test
max = 1795 linux $ ./test
max = 1
```

(1) 解释为什么会发生这种情况。

(2) 为了解决第一次运行后的遗留问题，可以不修改代码，而直接在 Linux 终端中使用指令来解决。假设在第一次程序运行完以后，使用 ps 指令，得到的列表前几项如下：

```
linux $ ./test
max = 1795 linux $ ps
22698 pts/0    00:00:00 bash
22725 pts/0    00:00:00 test
22726 pts/0    00:00:00 test
22727 pts/0    00:00:00 test
.....
```

再假设，test 程序开始运行后，没有任何新的进程被创建，并且所有进程号均按照顺序分配。

输入下列的指令，就可以让第二次运行得到正确的结果。其中-9 表示 SIGKILL。请填入正确的值。

```
linux $ kill -9 _____
```

A. 22725      B. 22724      C. -22725      D. -22724

**PART D.** Dave 同学修改了 Carol 同学的答案。他将 Carol 的最后一句 printf 改为如下代码：

```
if (pid < 0) {
    printf("max = %d ", count);
    kill(0,SIGKILL);
}
```

这段代码有时无法输出任何答案。Dave 想了一想，将 printf 中的字符串做了些修改，这样这段代码就能正确运行了。他修改了什么？