实　　验　　报　　告

学　　号	2103003 1009	姓　　名	惠欣宇	专业班级	计算机 4 班
课程名称	**操作系统**			学期	**2023 年秋季学期**
任课教师	窦金凤	完成日期	2023.12.15	上机课时间	周五 3-6
实 验 名 称	**Linux 进程控制—wait、exit**				

一、实验目的及要求

1.了解进程与程序的区别，加深对进程概念的理解；

2.进一步认识进程并发执行的原理，理解进程并发执行的特点，区别进程并发执行与顺序执行；

3.分析进程争用临界资源的现象，学习解决进程互斥的方法。

4.了解 fork()系统调用的返回值，掌握用 fork()创建进程的方法；

5.熟悉 wait、exit 等系统调用。

二、实验内容

修改程序，在父、子进程中分别使用 wait、exit 等系统调用“实现”其同步推进，多次反复运行改进后的程序，观察并记录运行结果。

三、实验源码

wait:

```c
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
int main() {
    pid_t pid;
    int status, i;
    if(fork() == 0){
        printf("I am son. My pid is %d.\n", getpid());  //输出子进程的打印信息及PID
        exit(11);  //这里的11作为exit函数返回的状态码  这里的状态码可以自行指定
    } else {
        printf("I am father. Waiting for my son.\n");
        pid = wait(&status);  //wait会返回子进程结束状态值并将子进程的PID赋值给pid
        i = WEXITSTATUS(status);  //从exit那里取出状态码赋值给i
        printf("Son’s pid is %d. exit status is %d.\n", pid, i);  //输出子进程的信息以及返回值
    }
    usleep(1000);
    return 0;
}
```

exit：

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    printf("exit enter\n");
    for(int i = 0; i < 10; i++) {
        if (i != 8) {
            printf("loop %d, no exit\n", i);
        } else {
            printf("i = %d, exit!\n", i);
            exit(0);
        }
    }
    return 0;
}
```

四、实验结果

输出结果如图 1、2。

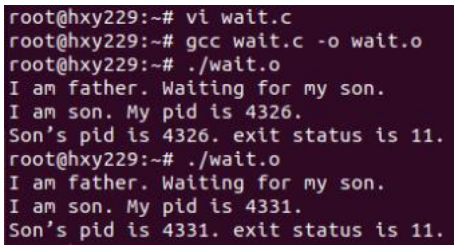

图 1 wait 程序的执行结果

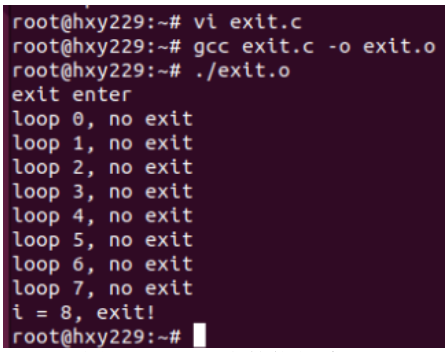

图 2 exit 程序的执行结果

五、结果分析

wait 函数功能是：父进程一旦调用了 wait 就立即阻塞自己，由 wait 自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个僵尸子进程，wait 就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait 就会一直阻塞在这里，直到有一个出现为止。

当父进程没有使用 wait 函数等待已终止的子进程时，子进程就会进入一种无父进程的状态，此时子进程就是实验一自己补充部分提到的孤儿进程。

如果任何进程有多个子进程，则在调用 wait()之后，如果没有子进程终止，则父进程必须处于 wait

状态。

如果只有一个子进程被终止，那么 wait() 返回被终止的子进程的进程 ID。

如果多个子进程被终止，那么 wait() 将获取任意子进程并返回该子进程的进程 ID。

wait 的目的之一是通知父进程子进程结束运行了，它的第二个目的是告诉父进程子进程是如何结束的。wait 返回结束的子进程的 PID 给父进程。父进程如何知道子进程是以何种方式退出的呢？

答案在传给 wait 的参数之中。父进程调用 wait 时传一个整型变量地址给函数。内核将子进程的退出状态保存在这个变量中。如果子进程调用 exit 退出，那么内核把 exit 的返回值存放到这个整数变量中；如果进程是被杀死的，那么内核将信号序号存放在这个变量中。这个整数由 3 部分组成，8 个 bit 记录子进程 exit 值，7 个 bit 记录信号序号，另一个 bit 用来指明是否发生错误。

wait 函数是线程同步中的一个重要方法，它用于暂停当前线程，直到某个特定的条件满足后再继续执行。 当一个线程调用 wait 函数时，它会释放持有的锁，并进入等待状态。 只有当其他线程调用相应的 notify 或 notify_all 函数时，等待线程才会被唤醒并继续执行。

wait()要与 fork()配套出现，如果在使用 fork()之前调用 wait()，wait()的返回值则为-1，正常情况下 wait()的返回值为子进程的 PID。

如果先终止父进程，子进程将继续正常进行，只是它将由 init 进程，也就是实验一中提到的那个进程继承，当子进程终止时，init 进程捕获这个状态，与实验一的补充部分相同。

参数 status 用来保存被收集进程退出时的一些状态，它是一个指向 int 类型的指针。但如果我们对这个子进程是如何死掉毫不在意，只想把这个僵尸进程消灭掉，（事实上绝大多数情况下，我们都会这样想），我们就可以设定这个参数为 NULL，就像下面这样：

```
1  pid = wait(NULL);
```

如果成功，wait 会返回被收集的子进程的进程 ID，如果调用进程没有子进程，调用就会失败，此时 wait 返回-1，同时 errno 被置为 ECHILD。

如果参数 status 的值不是 NULL，wait 就会把子进程退出时的状态取出并存入其中，这是一个 int 值，指出了子进程是正常退出还是被非正常结束的，以及正常结束时的返回值，或被哪一个信号结束的等信息。因为这些信息被存放在一个整数的二进制位中，因此有一套专门的宏来完成信息的读取：

1，WIFEXITED(status)；用来指出子进程是否为正常退出的，如果是，它会返回一个非零值。

2，WEXITSTATUS(status)； 当 WIFEXITED 返回非零值时，我们可以用这个宏来提取子进程的返回值，如果子进程调用 exit(5)退出，WEXITSTATUS(status) 就会返回 5；如果子进程调用 exit(7)，WEXITSTATUS(status)就会返回 7。如果进程不是正常退出的，WIFEXITED 返回 0，此时无意义。

使用 wait 函数将立即终止一个进程，并把它的状态值返回。由于进程是非正常结束；所有当进程死亡时会通知父进程发出一个信号，这个 SIGCHLD 信号将告知系统回收该进程的资源，并且退出时不刷新缓冲区，若有打开的文件也不会进行关闭操作，所以可能会造成数据丢失。

参数 status 进程退出时的状态值，即在使用时给它一个无符号的整型数，并且要在 0-255 范围内，否则将自动默认为未定义退出状态值。

进程调用 exit 函数会立即退出该进程，并像上述返回一些参数。最简单的例子就是在循环中达到条件后**退出循环且退出进程，**退出状态码也可以自行设定，只需要在 exit 的参数中给出即可。

下面分析一下 exit()函数的退出状态码：

0：命令成功完成；

1：通常的未知错误；

2：误用 shell 命令；

126：命令无法执行；

127：没有找到命令；

128：无效的退出参数；

128+x：使用 Linux 信号 x 的致命错误；

130：使用 Ctrl + C 终止的命令；

255：规范外的退出状态。