# 实验报告四 “病毒”程序

## 实验内容：

在Unix/Linux系统中，基于进程系统函数，使用C/C++完成程序的自动书写、编译和执行等相关操作。具体内容见下要求：

## 实验要求：

1. 编写一个C/C++程序，要求：在不使用读文件功能情况下，程序能够将自身打印到新的文件；
2. 如果上述输出文件是.txt，将上述程序内将文件类型改为.c或.cpp文件，并在该程序基础上，使用fork、exec等程序，实现对该文件的编译、执行等功能；需要注意的是，由于该程序会自身不断繁殖，需要通过读取文件内容等方式对程序的繁殖次数做出限制；
3. 在2的基础上加入信号操作，signal(SIGXXX,sig\_fun)，使得2中无条件的繁殖变为有条件繁殖，即：如果进程收到信号SIGXXX，那么主进程调用fork，在子进程中执行文件生成等相关操作，并调用exec函数执行，在父进程中，则终止进程。

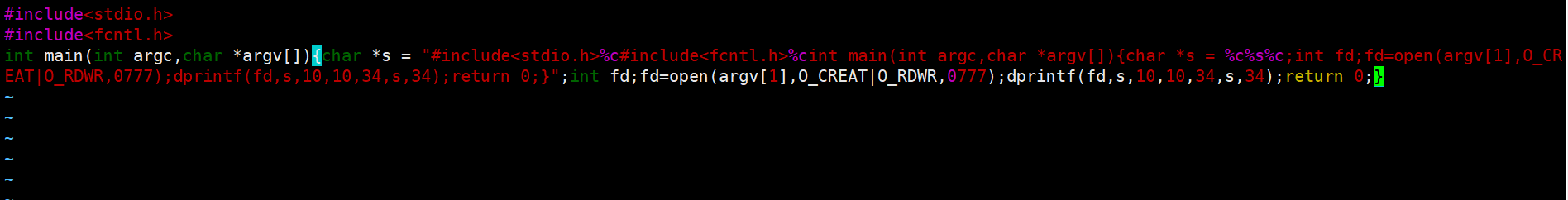
## 实验报告：

1. 解释如何实现程序的打印自身，并说明利用程序打印自身与直接打开元文件读取内容再打印有什么差别？

回答：

首先要澄清，什么是“打印”，什么是“自身”，以及程序执行环境的限制。“打印”可以有各种理解，比如向屏幕输出，向磁盘输出，向打印机输出，向内存输出，等等，但是本质上都是一样的，因此不做限制。  
    “自身”的一种理解是源程序，另一理解是机器码。提到“机器码”这个名词，你可能立刻就想到了“病毒”了吧，对，病毒就是最典型的自我打印程序，病毒的起源“磁心大战”就是各个程序在存储器中复制自己抢地盘，发展到现在更是丰富多彩，从传统的文件到时髦的Email，无所不用其极。  
    但是各种病毒型的自打印程序都有一个限制，即需要宿主系统提供的服务才能完成关键的“得到自己”的动作。如果自身代码在内存中，如何才能知道其起始地址呢？问操作系统吧。或者读取指令指针的值再做调整，这其实也是利用了执行环境之一的“指令指针”。又如邮件病毒，甚至不需要知道自己在哪里，只要调用软件的“转发”接口就行了。作为病毒当然是不错的构思，但是要作为“打印自身”的题解，显然就是耍赖皮了。推向极端，操作系统提供一个“把我自己打出来”的API，程序调用一下不就完了？  
    因此，限定程序的执行环境是讨论问题的必要前提。现代计算机和操作系统的“外部环境”实在太复杂了，搬出图灵机理论在这里也不太适合，干脆就这么定义命题：使用某种高级语言（比如说C语言），除了屏幕输出(如printf函数)以外不得使用其他系统相关函数和IO函数，打印出程序自身的源代码。

如何实现：

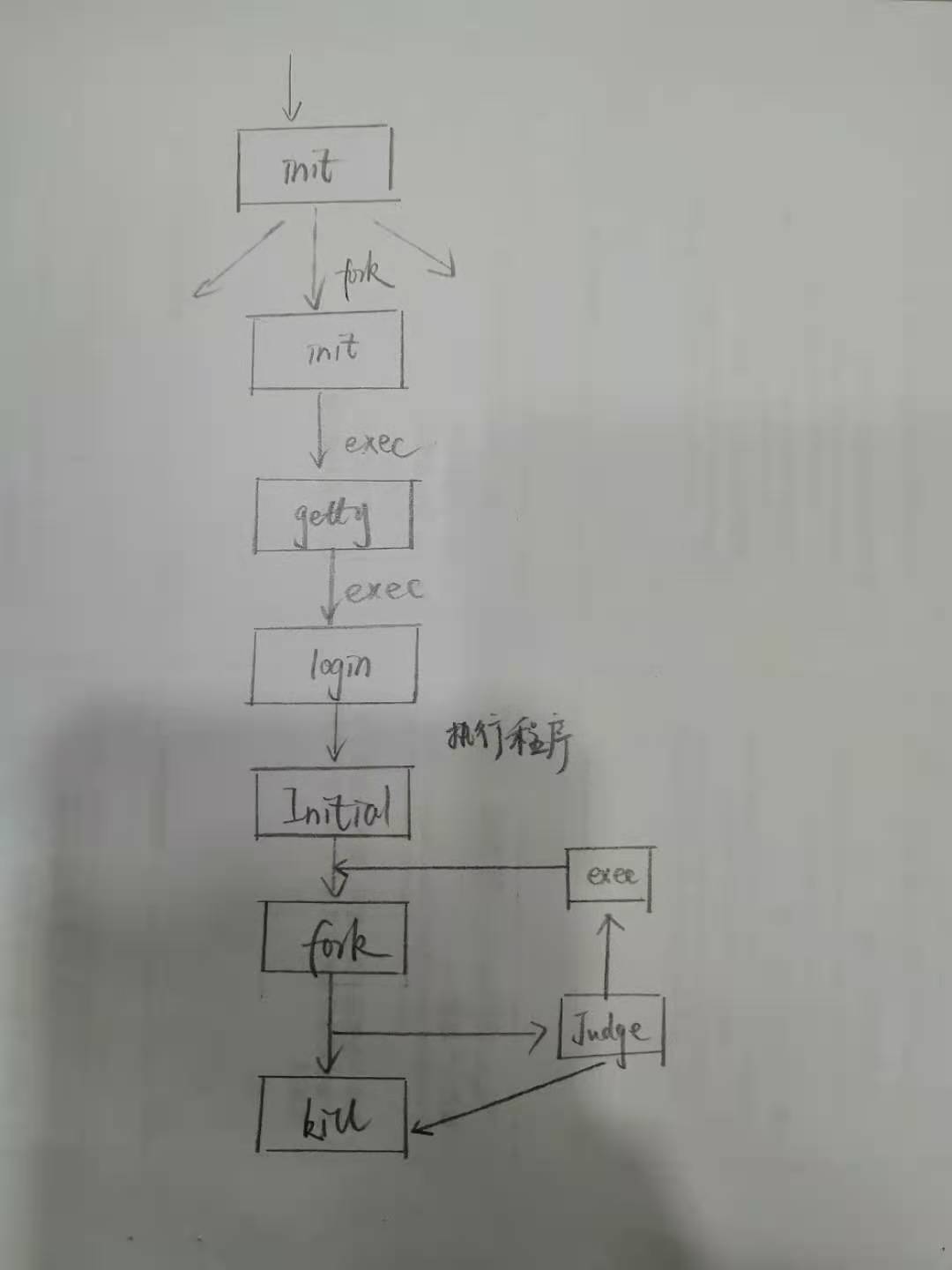


差别：

打印自身不受文件存在与否的限制，只要给出打印的地址就可以，文件或者打印机等等；而直接打开源文件再打印则需要源文件必须存在，其次是执行相应的读写字符串操作。

1. 参考系统进程关系（从init到执行用户进程），画出2的进程关系图，标注主要函数和主要参数，并说明其作用。

回答：

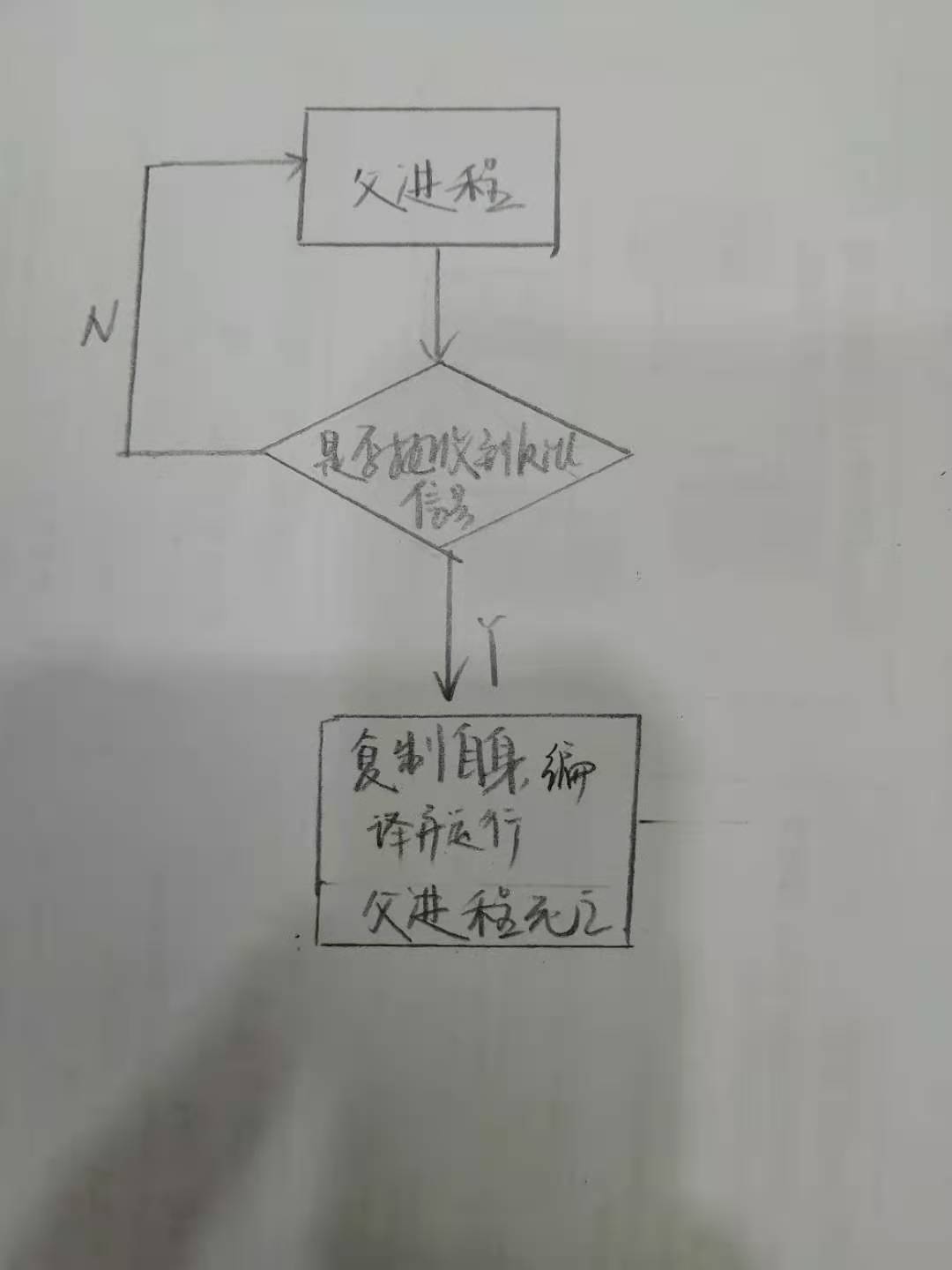


函数说明：

|  |  |
| --- | --- |
| 函数 | 功能说明 |
| void Initial(); | 判断是否创建count.txt。  如果没有则创建计数文件，并写入初始数值10，用于判断是否停止复制。反之则跳过 |
| int Judge(); | 每一次fork前调用此函数，将count.txt中数值减一；  如果为0则停止复制，反之继续 |

1. 参考系统进程关系，画出3的进程关系图，标注主要函数和主要参数，并说明其作用？

回答：



函数说明：

|  |  |
| --- | --- |
| 函数 | 功能 |
| void Initial(); | 判断是否创建count.txt。  如果没有则创建计数文件，并写入初始数值10，用于判断是否停止复制。反之则跳过 |
| int Judge(); | 每一次fork前调用此函数，将count.txt中数值减一；  如果为0则停止复制，反之继续 |
| void RandStr(char \*, int); | 用来生成文件，生成随机字符串名 |
| void catch\_signal(int); | 当捕获到 SIGINT ，执行操作：  用Judge（）判断能否继续  复制自身，编译并运行  父进程结束 |

1. 本次实验进程的自我复制和繁殖，请问从原理上阐述如何实现进程的随机变异？

回答：

多样化写信号捕捉后的操作函数Catch\_Signal(),每当一个信号的处理，fork()一个子进程时，执行不同的操作，从而实现进程的随机变异；