第一章 unix基础知识

关于本书相关辅助参考，有info libc。

C编译器

历史上，cc(C Compiler)是C编译器。在配置了GUN C编译系统的系统中，C编译器是gcc。其中cc通常链接到gcc。

```
[root@core ~]# whereis cc          查探该系统中的C编译器
cc: /usr/bin/cc
[root@core ~]# ls -l /usr/bin/cc
lrwxrwxrwx. 1 root root 3 May 31  2012 /usr/bin/cc -> gcc
```

```
localhost curl-7.29.0 # ./configure 2> /dev/null | egrep 'compiler is|Compiler:'
checking if compiler is icc (to build with PIC)... no   icc for integrated circuit compiler
checking if compiler is DEC/Compaq/HP C... no
checking if compiler is HP-UX C... no   至少存在于当前的
checking if compiler is IBM C... no     C编译器
checking if compiler is Intel C... no
checking if compiler is clang... no
checking if compiler is GNU C... yes
checking if compiler is LCC... no
checking if compiler is SGI MIPSpro C... no
checking if compiler is SGI MIPS C... no
checking if compiler is SunPro C... no
checking if compiler is Tiny C... no
checking if compiler is Watcom C... no
  Compiler:         gcc      编译该软件将会用到的编译器
```

输入输出

文件描述符：内核用来标识一个已打开文件的符号，通常是一个非负整数。

标准【输入、输出、错误】：在头文件<unistd.h>(类unix标准，posix标准符号常量)中定义了常量STDIN_FILENO、STDOUT_FILENO和STDERR_FILENO。

```
localhost ~ # grep \#define$'\t'STD /usr/include/unistd.h
#define STDIN_FILENO    0       /* Standard input.  */
#define STDOUT_FILENO   1       /* Standard output.  */
#define STDERR_FILENO   2       /* Standard error output.  */
```

系统调用IO

由POSIX来定义的系统调用，IO相关函数(open,read,write,lseek,close)，它们没有提供缓冲机制(相应的缓冲器需要由程序员自己来设计和实现)，并且其操作的对象都是以文件描述符为基础。

标准IO

标准IO由ISO C来定义，在此当中IO相关的函数有(fopen,fread,fwrite,fseek,fclose)，这些函数封装了系统调用IO函数，提供出了带缓冲机制的接口，使使用者操作起来更为方便。

note

通常一个函数的返回值类型一般都用int来标记，如下列各式:

```
int main() {
   return 0;
}
```

那么，这里的int，为什么不用一个char来代替？难道返回值会大于255?

其实这里的返回值类型并非由返回值的大小来决定，而是在返回的值当中，可能会存在负数。这样，char类型就无法表示了。所以需要一个有符号的数据类型来接收返回值。比如EOF控

制字符，它所对应的值就是-1。

程序和进程

程序是存放于磁盘中的可执行二进制文件，使用exec函数，由内核将其读入存储器，并使其执行。

进程和进程id

程序的执行实例被称为进程，在系统中标识进程的唯一数字，称为进程id。

进程控制

控制进程的主要函数有三个：fork、exec、waitpid。

进程和线程id

线程存在于进程中，同一个进程中的所有线程，共用该进程中的所有资源(地址空间、文件描述符、栈、进程相关属性)。为了区别进程中的每个线程，会给每个线程分配一个在该进程中唯一的id。

出错处理

头文件<errno.h>定义了符号errno，以及可以赋予它的各种常量。每个线程都有属于它自己的局部errno，以避免一个线程干扰另一个线程。

errno的两条规则：一、如果没有出错，则其值不会被另一个例程清除；二、任一函数都不会将errno值置为0。

C标准定义了两个函数，帮助打印出错信息

```
#include <string.h>
char *strerror(int errnum);            返回值：指向消息字符串的指针
```

```
#include <stdio.h>
void perror(const char *msg);        在打印出错信息前，先打印指针参数所指的字符串
```

时间值

当度量一个进程的执行时间时，UNIX系统使用三个进程时间值：

- 时钟时间
- 用户CPU时间

```
即除系统调用外的指令，下面这段代码执行的时间为用户CPU时间。
int i;
for (i = 0; i < 10000; i++) {
}
```

-
- 系统CPU时间

```
执行于内核中的指令，系统调用的逻辑一般都运行在内核中。下面函数执行的时间都被算在系统CPU时间。
getpid();
fork();
gethostname();
```

时钟时间又称墙上时钟时间。它是进程运行的时间总量。

用户CPU时间是执行用户指令所用时间。系统CPU时间是为该进程执行内核程序所经历的时

间。进程中系统调用函数(read、write……)所执行的时间，将被记入该进程的系统CPU时间。用户CPU时间加系统CPU时间被称为CPU时间。

习题

2、在执行该程序的两次期间，有其它两个进程被运行过。所以会从进程851之间跳到854。

```
mail cpp # cat 1-4.cpp
#include <stdio.h>       // For printf();
#include <stdlib.h>      // For exit();
#include <unistd.h>      // For getpid();
int main(int argc, const char *argv[]) {
    printf("This process ID is %d\n", getpid());
    exit(0);
}
mail cpp # gcc 1-4.cpp -o 1-4
mail cpp # ./1-4
This process ID is 9332
mail cpp # ./1-4
This process ID is 9333
mail cpp # ls
1-4  1-4.cpp
mail cpp # ./1-4
This process ID is 9335
```

源码

编译出可执行文件

系统默认是按顺序分配进程ID的

执行该ls命令时，占用了PID 9334

3、strerror的参数是一个数值，传递的时候是对原值进程复制，再传进来的，所以它和源值没任何关系，即对它的修改更动不了原值。但是perror传的是一个指针，如果不限定它，函数里面可以直接对该指针所指的位置内容做修改。

4、因为erron是该进程的全局变量，随时会变，也许就在该函数执行的过程当中被改变，这样，就会产生错误的日志记录。

5、有符号的32位整形所能保存的最大数字为2147483647，换算下来，该区域到2038年该会溢出。一般应用(不论服务器或客户端)在10年间，都会被更新替换，更新的软件一般会和目标平台相配套。在当前64位系统中，time_t已经是64位大小。所以，不必为此而担心。

```
mail cpp # echo $((2#11111111111111111111111111111111))
2147483647
mail cpp # date -d@$((2#11111111111111111111111111111111))
Tue Jan 19 11:14:07 CST 2038
```

二进制的31个1转换成十进制的数值大小

这是该值的机器时间

6、约249天后溢出。

```
mail cpp # date -d@$((2#11111111111111111111111111111111/100)) +%j
249
```

换算下来为，这一年的第249天