5.1 main函数

实际上程序的真正启动点是系统提供的一个初启函数

5.2命令行参数

是在启动程序执行的shell命令中给出的以空格分开的字符串

5.2.1 命令行参数的语法约定

5.2.2 扫描命令行中的选项

用getopt()函数来读取其中的选项

5.3 环境变量

环境变量用于定义不经常改变的或者许多进程共享的信息

用户可以通过环境变量赋值命令env和export命令来设置

5.3.1 环境表

记录进程的所有环境变量及其值。使用环境指针前，必须得先加以说明

Extern char \*\* environ

5.3.2 访问环境

应用程序一般在需要访问整个环境或者向其他程序递交环境的情况下才使用environ，当只需要访问某个特定的环境变量时，使用getenv()和putenv()函数

char\* getenv(const char\* name)返回一个字符串给出环境变量name定义 int putenv(char\* str)设置环境变量或者去掉环境变量定义.

Putenv设置的是进程自己的环境变量，而不是shell的环境变量。子进程虽然继承父进程的环境变量，但不传播自己的环境变量给父进程。

5.4 终止进程

正常终止有main返回，或者直接调用exit()或者\_exit()而终止。异常终止通过调用abort()或者由于信息而终止exit()通过调用\_exit()回到内核，并且在调用之前还须完成某些清理工作

void exit(int status)

5.4.1 出口状态

exit，\_exit都需要传递一个整数作为出口状态，用于向父进程报告程序运行是否成功。进程的出口状态是主程序main()的返回值

5.4.2 终止前的清理

在初启函数中调用atexit()指明程序终止时执行这些清理函数，使得清理工作对应透明。

Int atexit(void (\*func) (void))

5.4.3 流产程序

异常终止程序常用的方法是调用abort()函数。该函数会在当前工作目录中生成一个内存转储文件(core) void abort(void) 它不执行用atexit()注册的清理函数。所有的异常终止程序的原因都是由信号造成的。

5.5 进程的存储空间

UNIX不允许用户进程直接访问存储器的物理地址，进程看的都是有操作系统控制的虚拟空间地址，由内核管理，进行虚拟空间至实地址空间的转化。用malloc()调用系统调用sbrk()进行扩张虚拟地址空间的大小。

5.5.1 进程的地址空间(虚拟空间)

当进程用exec()装入一个新程序的时候，内核变为这个新进程建立地址空间。地址空间由很多段组成，每个段是一片连续存储单元。

有正文段(代码)：是只读共享的

初始化数据段和未初始化数据段。

栈：存放函数内的局部变量，临时变量以及函数的调用环境(函数的返回地址，函数的栈帧指针，函数的入口参数等……)。

堆：用于动态申请的存储空间。

5.5.2 动态存储分配与释放

malloc calloc realloc memallign valloc.

5.5.3 释放存储单元

free()

5.6 setjmp() longjmp()函数

实现从一个函数直接跳到另一个函数，这种控制转移称为非局部转移

int setjmp(jmp\_buf env) void longjmp(jmp\_buf env, int val)

两个函数结合使用

5.7 进程资源

5.7.1 查看与设置资源限制

函数 int getrlimit(int resource, struct rlimit \*rlptr)

Int setrlimit(int resource, const struct rlimit \*rlptr)

调用这两个函数都必须用resource指定一个资源并给出指向下述结构的指针

struct rlimit{ rlim\_t rlim\_cur(软限制值); rlim\_t rlim\_max(硬限制值)}; rlim\_t是长整型

5.7.2 资源使用统计

了解进程实际运行啦多少cpu时间，内存最大存储空间大小，进程被切换出内存的次数，读和写硬盘的次数，收到的信号个数，进程间通信量统计，页缺失次数，累积的各种存储段(正文 数据 栈)等等内容

使用函数 int getrusage(int who, struct rusage \*rusage)

5.8 用户信息

UNIX是一个多用户的操作系统

5.8.1：用户名

通过getlogin()得到注册用户名 char \* getlogin(void)，当用户已退出系统使得进程的控制终端不存在时，该函数将会返回NULL并置errno为ENXIO

5.8.2 用户数据库

保存所有用户的信息。/etc/passwd(口令数据库文件，所有用户都可以访问)和/etc/shadow(口令影子库文件，只有特权用户可以访问)都属于用户数据库。

口令数据库文件是一个ASCII文件，其中每一行由冒号分隔的七个域组成，描述一个独立的用户。每个域分别是：用户名，口令域，用户ID，组ID，用户名以及其他个人信息，用户的初始目录或称为主目录，用户的默认shell或者注册时运行的初始程序

给定了用户名或者用户ID，可以用函数struct passwd \*getpwuid(uid\_t uid)和struct passwd \*getpwname(const char \*name)。或者直接扫描整个口令数据文件，一次一个用户的读取，struct passwd \*getpwent(void); void setpwent(void); 打开用户数据库void endpwent(void); 关闭用户数据库

5.8.3 组数据库

类似于用户数据库。 有四个域，分别是：组名，组的口令，组ID，组成员。函数类似于用户数据库。

用户的组指定在/etc/group文件中，其中一个组是基本组，其他的都是附加组，基本组ID由/etc/passwd文件指定，当用户注册的时候，系统便将它放置在基本组内。

如果需要访问其他组的文件，就要用newgrp命令切换至其他组。newgrp groupname.如果命令执行成功的话，用户的有效ID便改为新组的ID。还可以切换到不属于这个组的用户切换到该组，这种情况下必须给出组命令。

使用附加组的优点是用户不必明显的改变组关系，这方便啦那些需要同时工作于多个组之间的用户。进程可以调用int getgroups(int gidsetsize, gid\_t grouplist[])

5.9 进程的身份凭证

进程可以访问属于他的一切资源，如果允许，也可以访问同组或其他用户的资源。

用户ID和组ID(包括附加组ID)是进程最基本的身份，也是系统判断一个进程是否有权进行某种操作或访问某个文件的依凭。

UNIX系统设置咯调整用户ID和调整组ID机制允许进程调整身份。所有就有了实际用户(组)ID，有效用户(附加组)iD，调整用户(组)ID，这些ID统称为进程身份凭证。

实际用户： 由login(1)程序在用户注册时设置，就是登陆的用户名。实际组ID标识该进程的真实用户所在组，

有效用户：是进程当前起作用的ID。exec()加载执行文件时，有效用户和实际用户设置为一样。有效用户ID和有效组ID影响文件的创建和访问，文件创建期间，内核设置文件的拥有者为创建进程的有效UID和GID。文件访问期间，内核使用进程的有效UID和有效GID来判断它是否能够访问文件。

保存的调整用户：是由exec()从有效用户ID和有效用户组ID复制而来

附加组ID：进程的附加组ID在进程被创建的时候设置为其父进程的附加组ID

有效用户ID是进程当前起作用的ID，它可以在保存的调整用户ID和实际用户ID之间进行切换，从而使得进程可以以两个身份来工作

文件创建期间，内核设置文件的拥有者为创建进程的有效UID和有效GID。文件访问期间，内核使用进程的有效UID和有效GID来判定它是否能够访问一个文件。

下面的函数可以读取进程的各种ID：

uid\_t getuid(void); uid\_t geteuid(void); gid\_t getgid(void); gid\_t getegid(void); gid\_t getgroups(int count, gid\_t groups)

5.10：调整进程身份

进程总是可以调整它的有效用户ID回到其实际ID，用函数：int setuid(uid\_t uid); int setgid(gid\_t gid); int setreuid(uid\_t ruid, uid\_t euid); int setregid(uid\_t rgid, uid\_t egid); int seteuid(uid\_t uid); int setegid(uid\_t gid);

利用函数char \*cuserid(char \*string)可以获得当前有效用户ID对应的用户名