6.1：进程标识

UNIX中每一个进程有一个唯一的称为进程标识的正整数与之相连，该整数称为进程ID。内核需要知道PID才能像这个进程发送信号。PID为1的进程是init进程，PID为0的是调度进程。表示PID的类型是pid\_t，它是有符号整数类型。函数getpid()可以获得进程的PID，getppid()可以获得父进程的PID，返回值都是pid\_t。

6.2：进程创建

pid\_t fork(void)几乎完全是对父进程的复制，父子进程共享此程序的代码段。调用成功的话，父进程返回的是子进程的PID，子进程返回的是0.若创建失败，fork返回-1，并置error指出失败原因。为咯区分父子进程并使用不同的程序做不同的事，fork的返回值总是作为if语句的分支条件。If语句的一个分支包含父进程的代码，另一个分支则包含子进程的代码。创建进程成功后，父子进程的执行先后顺序是不知道的，由内核来决定

vfork()创建的子进程共享父进程的地址空间，直到子进程调用啦exit或者exec为止。

vfork()在出借父进程的地址空间给子进程的同时阻塞父进程，父进程被悬挂，直到调用了exec()或\_exec()。vfork有点危险，因为它允许一个进程使用甚至修改另一个进程的地址空间子进程不能从调用vfork的那个函数返回。这样做会使父进程的控制感到非常迷惑。

6.3：执行一个新程序

这些函数统称为exec函数

int execl (const char \*path, const char \*arg0, …, (char\*) 0);

int execlp (const char \*file, const char \*arg0, …, (char\*) 0);

int execle (const char \*path, const char \*arg0, …, (char\*) 0, const char \*envp[]);

int execv (const char \*path, const char \*argv[]);

int execvp (const char \*file, const char \*argv[]);

int execv (const char \* file, const char \*argv[], const char \*envp[]);

file参数可以是相对路径名，path参数则必须是绝对路径名

envp是个以空指针结尾的字符串指针数组，这些字符串构成了新程序映像的环境变量表。

如果调用进程的是vfork创建的，则exec函数返回老的空间给父进程，否则它将释放老的地址空间。只有调用不成功的时候exec才会返回，其返回值为-1并设置error

exec()用新程序的地址空间覆盖当前程序的地址空间时，它只覆盖正文段，数据段和栈段，原进程的几乎所有内部执行环境变量仍保持不变。这里我们应当理解为进程仍保持原样，只是与它相连接的程序被替换。因此，凡是与进程有关的属性仍保持不变，变化的只是程序映像有关的属性。

6.4：等待进程完成

我们希望父进程常常需要等待子进程执行完后才能继续执行。

用函数wait waitpid来等待子进程的终止和获得子进程的出口状态

pid\_t wait( int \*stat\_loc) pid\_t waitpid( pid\_t pid, int \*stat\_loc, int options);

wait返回已终止的子进程的PID，并将孩子进程的终止状态存储在stat\_loc所存储单元，异常将会返回-1.参数stat\_loc是一个指向整数的指针。

我们可以通过一些宏来查看子进程的终止状态，这些宏的参数stat\_val是wait存放在stat\_loc中返回的整型值

waitpid可以等待一个特定的子进程或一组子进程

pid == 0：等待进程组ID与调用进程的进程组ID相同的任意子进程

pid == -1：等待任意子进程，同wait一样啦

pid > 0：等待进程ID为pid的子进程

pid < -1：等待进程组ID等于pid绝对值的任意子进程

参数options是一个位串，用于对waitpid的行为进行修改，其值或为0，或是宏WNOHANG或WUNTRACED的按位或

如果想让父进程检查一个特定子进程是否已经终止，可以使用如下调用waitpid(child\_pid, (int \*)0, WNOHANG);如果进程还没有终止，他将会返回0，否则返回child\_pid指定的PID或者发现错误的时候返回-1并设置error

还有wait3和wait4函数

6.5：进程终止和僵死进程

前面讲到的进程的终止有正常终止和异常终止

进程执行完一系列代码后这个终止进程即进入僵死状态。当进程调用exit或者\_exit而正常终止时，进程可以给它传递一个参数作为出口状态，在异常终止的情况下，则有内核生成终止状态指出异常终止的原因

如果子进程正常终止，父进程可以获得它的出口状态

init进程将会成为其父进程已终止的任何子进程的父进程，也称为init继承啦已终止进程的子进程

另一方面，当子进程先于父进程终止，此时成为僵死进程，还会和父进程保持连接，直到父进程调用wait为止

程序正常结束时系统会自动释放它和所有子进程的proc结构。但是如果父进程是异常终止的，则该子进程将自动的以init作为父进程，于是这个进程是一个不在运行的僵死进程，且被init所继承，故将保持在进程表中直到下一次重启init清除进程表为止。这时候这个进程不能用kill来杀死，因为它已经死啦，且会占用proc结构，从而减少系统中可创建的最大进程个数。常常通过调用二次fork函数来避免僵死进程留在系统中。这种是第二个子进程的父进程先于它终止，使得它被init继承，且这时候它是活跃进程，当它终止的时候，init调用wait释放其proc结构，因而也不会成为僵死进程。

6.6：system函数

典型的，一个进程执行另一个程序需要有三步骤：派生一个子进程，由子进程调用exec加载程序执行，等待子进程执行完毕。ANSI C提供啦一个便捷的函数，system将这些步骤集成为一体，使得程序可以如shell命令一般方便的执行另一个程序。

6.7：进程组

在shell命令中可以用管道操作‘|’可以连接好几个程序来运行多个程序。另一方面，即使只运行一个程序，它也能在内部使用多个进程。另外，shell也将用括号括住，以分号分隔的多条命令视为单条shell命令处理，这种命令也连接了好几个进程。属于shell单条命令的这些进程称为一个进程组或者作业。

进程组内的其他进程从它的父进程继承进程组ID

获取和设置进程的进程组ID使用下面的函数

pid\_t getpgrp(void) int setpgid(pid\_t pid, pid\_t pgid) 改变参数pid指定的进程组ID为pgid，如果两个参数相同，则由pid指定的进程成为一个新的进程组的组长

利用进程组机制，UNIX可以对一个组内的所有进程进行操作，如强制终止命令。

6.8：会晤期

会晤期是一至多个进程组的集合，有进程组，前台进程组，后台进程组，每一个会晤期有一个会晤期主席，它是创建会晤期的那个进程，进程通过函数pid\_t setsid(void)来建立一个新的会晤期

调用进程不能是进程组组长否则函数返回-1并设置errno，成功返回调用进程组ID。通常，新会晤期是由系统的注册程序所创建的，并且会晤期主席就是运行用户注册shell的进程。因此，会晤期主席是一个会晤期内所创建的所有进程的祖先。会晤期主席也是进程组组长，当会晤期主席终止时，它会结束会晤期。

6.9：控制终端

每一个会晤期可以有一个控制终端，进程可以通过控制终端进行输入，输出和控制作业。

1控制终端由fork函数调用时候继承。2会晤期也可以没有控制终端。3与控制终端连接的进程组叫做前台进程组.4那些在会晤期主席已经终止后仍继续运行的进程组将标志为孤儿进程组。这时候这些进程会收到终止信号，一般就会终止，没有终止的也不能访问终端

函数 char \*ctermid( char \*ptr)用于确定控制终端的文件名

函数 pid\_t tcgetpgrp(int fd) int tcsetpgrp(int fd, pid\_t pgrpid)用于读与终端相连的前台进程组的进程组ID和设置终端的前台进程组

参数fd是一个与终端对应的文件描述字调用进程必须时与pgrpid同一个会晤期的进程，fd引用的必须是会晤期控制终端。tcsetpgrp调用成功返回0.

6.10：作业控制