1. **fork 系统调用**

在Linux系统中，用户创建一个进程的唯一方法就是使用系统调用fork。内核为完成系统调用fork要进行几步操作第一步，为新进程在进程表中分配一个表项。系统对一个用户可以同时运行的进程数是有限制的，对超级用户没有该限制，但也不能超过进程表的最大表项的数目。第二步，给子进程一个唯一的进程标识号（PID）。该进程标识号其实就是该表项在进程表中的索引号。第三步，复制一个父进程的进程表项的副本给子进程。内核初始化子进程的进程表项时，是从父进程处拷贝的。所以子进程拥有与父进程一样的uid、euid、gid、用于计算优先权的nice值、当前目录、当前根、用户文件描述符表等。第四步，把与父进程相连的文件表和索引节点表的引用数加1。这些文件自动地与该子进程相连。第五步，内核为子进程创建用户级上下文。内核为子进程的u区及辅助页表分配内存，并复制父进程的区内容。这样生成的是进程的静态部分。第六步，生成进程的动态部分，内核复制父进程的上下文的第一层，即寄存器上下文和内核栈，内核再为子进程虚设一个上下文层，这是为了子进程能“恢复”它的上下文。这时，该调用会对父进程返回子进程的pid，对子进程返回0。

系统调用fork和vfork的声明格式如下：

pid\_t fork(void);

在使用这个系统调用的程序中要加入以下头文件：

#include<sys/types.h>

#include<unistd.h>

1. **exec 系统调用**

系统调用exec是用来执行一个可执行文件来代替当前进程的执行映像。需要注意的是，该调用并没有生成新的进程，而是在原有进程的基础上，替换原有进程的正文，调用前后是同一个进程，进程号PID不变。但执行的程序变了（执行的指令序列改变了）。它有六种调用的形式，随着系统的不同并不完全与以下介绍的相同。它们的声明格式如下：

int execl( const char \*path, const char \*arg, ...);

int execlp( const char \*file, const char \*arg, ...);

int execle( const char \*path, const char \*arg , ..., char\* const envp[]);

1. **exit 系统调用**

系统调用exit的功能是终止发出调用的进程。它的声明格式如下：

void \_exit(int status);

在使用这个系统调用的程序中要加入以下头文件：

#include <stdlib.h>

系统调用\_exit立即终止发出调用的进程。所有属于该进程的文件描述符都关闭。该进程的所有子进程由进程1（进程init）接收，并对该进程的父进程发出一个SIGCHLD（子进程僵死）的信号。参数status作为退出的状态值返回父进程，该值可以通过系统调用wait来收集。返回状态码status只有最低一个字节有效。如果进程是一个控制终端进程，则SIGHUP信号将被送往该控制终端的前台进程。系统调用\_exit从不返回任何值给发出调用的进程；也不刷新I/O缓冲，如果要自动完成刷新，可以用函数调用exit。

1. **wait 系统调用**

系统调用wait的功能是发出调用的进程只要有子进程，就睡眠直到它们中的一个终止为止。该调用声明的格式如下：

pid\_t wait(int \*status)

pid\_t waitpid(pid\_t pid, int \*status, int options);

在使用这些系统调用的程序中要加入以下头文件

#include <sys/types.h>

#include <sys/wait.h>

发出wait调用的进程进入睡眠直到它的一个子进程退出时或收到一个不能被忽略的信号时被唤醒。如果调用发出时，已经有退出的子进程（这时子进程的状态是僵死状态），该调用立即返回。其中调用返回时参数status中包含子进程退出时的状态信息。

调用waitpid与调用wait的区别是waitpid等待由参数pid指定的子进程退出。其中参数pid的含义与取值方法如下：

参数pid < -1时，当退出的子进程满足下面条件时结束等待：该子进程的进程组ID(process group)等于绝对值的pid这个条件。

参数pid = 0时，等待任何满足下面条件的子进程退出：该子进程的进程组ID等于发出调用进程的进程组ID。

参数pid > 0时，等待进程ID等于参数pid的子进程退出。

参数pid = -1时，等待任何子进程退出，相当于调用wait。

对于调用waitpid中的参数options的取值及其含义如下：

WNOHANG：该选项要求如果没有子进程退出就立即返回。

WUNTRACED：对已经停止但未报告状态的子进程，该调用也从等待中返回和报告状态。如果status不是空，调用将使status指向该信息。下面的宏可以用来检查子进程的返回状态。前面三个用来判断退出的原因，后面三个是对应不同的原因返回状态值：

1. **sleep 函数调用**

函数调用sleep可以用来使进程挂起指定的秒数。该函数调用的声明格式如下：

unsigned int sleep(unsigned int seconds)

在使用这个系统调用的程序中要加入以下头文件

#include < unistd.h >

该函数调用使得进程挂起一个指定的时间，直到指定时间用完或者收到信号。系统的活动对指定的时间有一定的影响。Linux系统是用SIGALRM实现的，在Linux系统里，sleep函数不能和alarm()调用混用。

如果指定挂起的时间到了，该调用返回0；如果该函数调用被信号所打断，则返回剩余挂起的时间数（指定的时间减去已经挂起的时间）。

1. **int lockf(files,function,size)**

其中：files是文件描述符；function是锁定和解锁：1表示锁定，0表示解锁。size是锁定或解锁的字节数，为0，表示从文件的当前位置到文件尾。

1. **key\_t ftok( char \* fname, int id )**

系统建立IPC通讯（如信号量、消息队列、共享内存时）必须指定一个ID值。通常情况下，该id值通过ftok函数得到。

ftok原型如下：key\_t ftok( char \* fname, int id )

fname就时你指定的文件名(该文件必须是存在而且可以访问的)，id是子序号，虽然为int，但是只有8个比特被使用(0-255)。

当成功执行的时候，一个key\_t值将会被返回，否则 -1 被返回。

在一般的UNIX实现中，是将文件的索引节点号取出，前面加上子序号得到key\_t的返回值。如指定文件的索引节点号为65538，换算成16进制为 0x010002，而你指定的ID值为38，换算成16进制为0x26，则最后的key\_t返回值为0x26010002。

查询文件索引节点号的方法是： ls -i filename