1. **【Linux Science】Linux进程函数大全- -**

atexit（设置程序正常结束前调用的函数）    
相关函数  \_exit,exit,on\_exit  
  
表头文件  #include<stdlib.h>  
  
定义函数  int atexit (void (\*function)(void));  
  
函数说明  atexit()用来设置一个程序正常结束前调用的函数。当程序通过调用exit()  
或从main中返回时，参数function所指定的函数会先被调用，然后才真正由exit()结束  
程序。  
  
返回值  如果执行成功则返回0，否则返回-1，失败原因存于errno中。  
  
范例  #include<stdlib.h>  
void my\_exit(void)  
{  
printf(“before exit () !\n”);  
}  
main()  
{  
atexit (my\_exit);  
exit(0);  
}  
  
执行  before exit()!  
  
　   
  
  
  
  
execl（执行文件）    
相关函数  fork，execle，execlp，execv，execve，execvp  
  
表头文件  #include<unistd.h>  
  
定义函数  int execl(const char \* path,const char \* arg,....);  
  
函数说明  execl()用来执行参数path字符串所代表的文件路径，接下来的参数代表执行  
该文件时传递过去的argv(0)、argv[1]……，最后一个参数必须用空指针(NULL)作结  
束。  
  
返回值  如果执行成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno  
中。  
  
范例  #include<unistd.h>  
main()  
{  
execl(“/bin/ls”,”ls”,”-al”,”/etc/passwd”,(char \* )0);  
}  
  
执行  /\*执行/bin/ls -al /etc/passwd \*/  
-rw-r--r-- 1 root root 705 Sep 3 13 :52 /etc/passwd  
  
　   
  
  
  
  
execlp（从PATH 环境变量中查找文件并执行）    
相关函数  fork，execl，execle，execv，execve，execvp  
  
表头文件  #include<unistd.h>  
  
定义函数  int execlp(const char \* file,const char \* arg,……)；  
  
函数说明  execlp()会从PATH 环境变量所指的目录中查找符合参数file的文件名，找到  
后便执行该文件，然后将第二个以后的参数当做该文件的argv[0]、argv[1]……，最后  
一个参数必须用空指针(NULL)作结束。  
  
返回值  如果执行成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno   
中。  
  
错误代码  参考execve()。  
  
范例  /\* 执行ls -al /etc/passwd execlp()会依PATH 变量中的/bin找到/bin/ls \*/  
#include<unistd.h>  
main()  
{  
execlp(“ls”,”ls”,”-al”,”/etc/passwd”,(char \*)0);  
}  
  
执行  -rw-r--r-- 1 root root 705 Sep 3 13 :52 /etc/passwd  
  
　   
  
  
  
  
execv（执行文件）    
相关函数  fork，execl，execle，execlp，execve，execvp  
  
表头文件  #include<unistd.h>  
  
定义函数  int execv (const char \* path, char \* const argv[ ]);  
  
函数说明  execv()用来执行参数path字符串所代表的文件路径，与execl()不同的地方  
在于execve()只需两个参数，第二个参数利用数组指针来传递给执行文件。  
  
返回值  如果执行成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno   
中。  
  
错误代码  请参考execve（）。  
  
范例  /\* 执行/bin/ls -al /etc/passwd \*/  
#include<unistd.h>  
main()  
{  
char \* argv[ ]={“ls”,”-al”,”/etc/passwd”,(char\*) }};  
execv(“/bin/ls”,argv);  
}  
  
执行  -rw-r--r-- 1 root root 705 Sep 3 13 :52 /etc/passwd  
  
　   
  
  
  
  
execve（执行文件）    
相关函数  fork，execl，execle，execlp，execv，execvp  
  
表头文件  #include<unistd.h>  
  
定义函数  int execve(const char \* filename,char \* const argv[ ],char \* const   
envp[ ]);  
  
函数说明  execve()用来执行参数filename字符串所代表的文件路径，第二个参数系利  
用数组指针来传递给执行文件，最后一个参数则为传递给执行文件的新环境变量数组。  
  
返回值  如果执行成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno   
中。  
  
错误代码  EACCES  
1. 欲执行的文件不具有用户可执行的权限。  
2. 欲执行的文件所属的文件系统是以noexec 方式挂上。  
3.欲执行的文件或script翻译器非一般文件。  
EPERM  
1.进程处于被追踪模式，执行者并不具有root权限，欲执行的文件具有SUID 或SGID   
位。  
2.欲执行的文件所属的文件系统是以nosuid方式挂上，欲执行的文件具有SUID 或SGID   
位元，但执行者并不具有root权限。  
E2BIG 参数数组过大  
ENOEXEC 无法判断欲执行文件的执行文件格式，有可能是格式错误或无法在此平台执  
行。  
EFAULT 参数filename所指的字符串地址超出可存取空间范围。  
ENAMETOOLONG 参数filename所指的字符串太长。  
ENOENT 参数filename字符串所指定的文件不存在。  
ENOMEM 核心内存不足  
ENOTDIR 参数filename字符串所包含的目录路径并非有效目录  
EACCES 参数filename字符串所包含的目录路径无法存取，权限不足  
ELOOP 过多的符号连接  
ETXTBUSY 欲执行的文件已被其他进程打开而且正把数据写入该文件中  
EIO I/O 存取错误  
ENFILE 已达到系统所允许的打开文件总数。  
EMFILE 已达到系统所允许单一进程所能打开的文件总数。  
EINVAL 欲执行文件的ELF执行格式不只一个PT\_INTERP节区  
EISDIR ELF翻译器为一目录  
ELIBBAD ELF翻译器有问题。  
  
范例  #include<unistd.h>  
main()  
{  
char \* argv[ ]={“ls”,”-al”,”/etc/passwd”,(char \*)0};  
char \* envp[ ]={“PATH=/bin”,0}  
execve(“/bin/ls”,argv,envp);  
}  
  
执行  -rw-r--r-- 1 root root 705 Sep 3 13 :52 /etc/passwd  
  
　   
  
  
  
  
execvp（执行文件）    
相关函数  fork，execl，execle，execlp，execv，execve  
  
表头文件  #include<unistd.h>  
  
定义函数  int execvp(const char \*file ,char \* const argv []);  
  
函数说明  execvp()会从PATH 环境变量所指的目录中查找符合参数file 的文件名，找  
到后便执行该文件，然后将第二个参数argv传给该欲执行的文件。  
  
返回值  如果执行成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno  
中。  
  
错误代码  请参考execve（）。  
  
范例  /\*请与execlp（）范例对照\*/  
#include<unistd.h>  
main()  
{  
char \* argv[ ] ={ “ls”,”-al”,”/etc/passwd”,0};  
execvp(“ls”,argv);  
}  
  
执行  -rw-r--r-- 1 root root 705 Sep 3 13 :52 /etc/passwd  
  
　   
  
  
  
  
exit（正常结束进程）    
相关函数  \_exit，atexit，on\_exit  
  
表头文件  #include<stdlib.h>  
  
定义函数  void exit(int status);  
  
函数说明  exit()用来正常终结目前进程的执行，并把参数status返回给父进程，而进  
程所有的缓冲区数据会自动写回并关闭未关闭的文件。  
  
返回值    
  
范例  参考wait（）  
  
　   
  
  
  
\_    
exit（结束进程执行）    
相关函数  exit，wait，abort  
  
表头文件  #include<unistd.h>  
  
定义函数  void \_exit(int status);  
  
函数说明  \_exit()用来立刻结束目前进程的执行，并把参数status返回给父进程，并关  
闭未关闭的文件。此函数调用后不会返回，并且会传递SIGCHLD信号给父进程，父进程可  
以由wait函数取得子进程结束状态。  
  
返回值    
  
附加说明  \_exit（）不会处理标准I/O 缓冲区，如要更新缓冲区请使用exit（）。  
  
　   
  
  
  
  
vfork（建立一个新的进程）    
相关函数  wait，execve  
  
表头文件  #include<unistd.h>  
  
定义函数  pid\_t vfork(void);  
  
函数说明  vfork()会产生一个新的子进程，其子进程会复制父进程的数据与堆栈空间，  
并继承父进程的用户代码，组代码，环境变量、已打开的文件代码、工作目录和资源限  
制等。Linux 使用copy-on-write(COW)技术，只有当其中一进程试图修改欲复制的空间  
时才会做真正的复制动作，由于这些继承的信息是复制而来，并非指相同的内存空间，  
因此子进程对这些变量的修改和父进程并不会同步。此外，子进程不会继承父进程的文  
件锁定和未处理的信号。注意，Linux不保证子进程会比父进程先执行或晚执行，因此编  
写程序时要留意  
  
死锁或竞争条件的发生。    
  
返回值  如果vfork()成功则在父进程会返回新建立的子进程代码(PID)，而在新建立的  
子进程中则返回0。如果vfork 失败则直接返回-1，失败原因存于errno中。  
  
错误代码  EAGAIN 内存不足。ENOMEM 内存不足，无法配置核心所需的数据结构空间。  
  
范例  #include<unistd.h>  
main()  
{  
if(vfork() = =0)  
{  
printf(“This is the child process\n”);  
}else{  
printf(“This is the parent process\n”);  
}  
}  
  
执行  this is the parent process  
this is the child process  
  
　   
  
  
  
  
getpgid（取得进程组识别码）    
相关函数  setpgid，setpgrp，getpgrp  
  
表头文件  #include<unistd.h>  
  
定义函数  pid\_t getpgid( pid\_t pid);  
  
函数说明  getpgid()用来取得参数pid 指定进程所属的组识别码。如果参数pid为0，则  
会取得目前进程的组识别码。  
  
返回值  执行成功则返回组识别码，如果有错误则返回-1，错误原因存于errno中。  
  
错误代码  ESRCH 找不到符合参数pid 指定的进程。  
  
范例  /\*取得init 进程（pid＝1）的组识别码\*/  
#include<unistd.h>  
mian()  
{  
printf(“init gid = %d\n”,getpgid(1));  
}  
  
执行  init gid = 0  
  
　   
  
  
  
  
getpgrp（取得进程组识别码）    
相关函数  setpgid，getpgid，getpgrp  
  
表头文件  #include<unistd.h>  
  
定义函数  pid\_t getpgrp(void);  
  
函数说明  getpgrp()用来取得目前进程所属的组识别码。此函数相当于调用getpgid  
(0)；  
  
返回值  返回目前进程所属的组识别码。  
  
范例  #include<unistd.h>  
main()  
{  
printf(“my gid =%d\n”,getpgrp());  
}  
  
执行  my gid =29546  
  
　   
  
  
  
  
getpid（取得进程识别码）    
相关函数  fork，kill，getpid  
  
表头文件  #include<unistd.h>  
  
定义函数  pid\_t getpid(void);  
  
函数说明  getpid（）用来取得目前进程的进程识别码，许多程序利用取到的此值来建  
立临时文件，以避免临时文件相同带来的问题。  
  
返回值  目前进程的进程识别码  
  
范例  #include<unistd.h>  
main()  
{  
printf(“pid=%d\n”,getpid());  
}  
  
执行  pid=1494 /\*每次执行结果都不一定相同\*/  
  
　   
  
  
  
  
getppid（取得父进程的进程识别码）    
相关函数  fork，kill，getpid  
  
表头文件  #include<unistd.h>  
  
定义函数  pid\_t getppid(void);  
  
函数说明  getppid()用来取得目前进程的父进程识别码。  
  
返回值  目前进程的父进程识别码。  
  
范例  #include<unistd.h>  
main()  
{  
printf(“My parent ‘pid =%d\n”,getppid());  
}  
  
执行  My parent pid =463  
  
　   
  
  
  
  
getpriority（取得程序进程执行优先权）    
相关函数  setpriority，nice  
  
表头文件  #include<sys/time.h>  
#include<sys/resource.h>  
  
定义函数  int getpriority(int which,int who);  
  
函数说明  getpriority()可用来取得进程、进程组和用户的进程执行优先权。  
  
参数  which有三种数值，参数who 则依which值有不同定义  
which who 代表的意义  
PRIO\_PROCESS who 为进程识别码  
PRIO\_PGRP who 为进程的组识别码  
PRIO\_USER who 为用户识别码  
此函数返回的数值介于-20 至20之间，代表进程执行优先权，数值越低代表有较高的优  
先次序，执行会较频繁。  
  
返回值  返回进程执行优先权，如有错误发生返回值则为-1 且错误原因存于errno。  
  
附加说明  由于返回值有可能是-1，因此要同时检查errno是否存有错误原因。最好在调  
用次函数前先清除errno变量。  
  
错误代码  ESRCH 参数which或who 可能有错，而找不到符合的进程。EINVAL 参数  
which 值错误。  
  
　   
  
  
  
  
nice（改变进程优先顺序）    
相关函数  setpriority，getpriority  
  
表头文件  #include<unistd.h>  
  
定义函数  int nice(int inc);  
  
函数说明  nice()用来改变进程的进程执行优先顺序。参数inc数值越大则优先顺序排在  
越后面，即表示进程执行会越慢。只有超级用户才能使用负的inc 值，代表优先顺序排  
在前面，进程执行会较快。  
  
返回值  如果执行成功则返回0，否则返回-1，失败原因存于errno中。  
  
错误代码  EPERM 一般用户企图转用负的参数inc值改变进程优先顺序。  
  
　   
  
  
  
  
on\_exit（设置程序正常结束前调用的函数）    
相关函数  \_exit，atexit，exit  
  
表头文件  #include<stdlib.h>  
  
定义函数  int on\_exit(void (\* function)(int, void\*),void \*arg);  
  
函数说明  on\_exit()用来设置一个程序正常结束前调用的函数。当程序通过调用exit()  
或从main中返回时，参数function所指定的函数会先被调用，然后才真正由exit()结束  
程序。参数arg指针会传给参数function函数，详细情况请见范例。  
  
返回值  如果执行成功则返回0，否则返回-1，失败原因存于errno中。  
  
附加说明    
  
范例  #include<stdlib.h>  
void my\_exit(int status,void \*arg)  
{  
printf(“before exit()!\n”);  
printf(“exit (%d)\n”,status);  
printf(“arg = %s\n”,(char\*)arg);  
}  
main()  
{  
char \* str=”test”;  
on\_exit(my\_exit,(void \*)str);  
exit(1234);  
}  
  
执行  before exit()!  
exit (1234)  
arg = test  
  
　   
  
  
  
  
setpgid（设置进程组识别码）    
相关函数  getpgid，setpgrp，getpgrp  
  
表头文件  #include<unistd.h>  
  
定义函数  int setpgid(pid\_t pid,pid\_t pgid);  
  
函数说明  setpgid()将参数pid 指定进程所属的组识别码设为参数pgid 指定的组识别  
码。如果参数pid 为0，则会用来设置目前进程的组识别码，如果参数pgid为0，则会以  
目前进程的进程识别码来取代。  
  
返回值  执行成功则返回组识别码，如果有错误则返回-1，错误原因存于errno中。  
  
错误代码  EINVAL 参数pgid小于0。  
EPERM 进程权限不足，无法完成调用。  
ESRCH 找不到符合参数pid指定的进程。  
  
　   
  
  
  
  
setpgrp（设置进程组识别码）    
相关函数  getpgid，setpgid，getpgrp  
  
表头文件  #include<unistd.h>  
  
定义函数  int setpgrp(void);  
  
函数说明  setpgrp()将目前进程所属的组识别码设为目前进程的进程识别码。此函数相  
当于调用setpgid(0,0)。  
  
返回值  执行成功则返回组识别码，如果有错误则返回-1，错误原因存于errno中。  
  
　   
  
  
  
  
setpriority（设置程序进程执行优先权）    
相关函数  getpriority，nice  
  
表头文件  #include<sys/time.h>  
#include<sys/resource.h>  
  
定义函数  int setpriority(int which,int who, int prio);  
  
函数说明  setpriority()可用来设置进程、进程组和用户的进程执行优先权。参数  
which有三种数值，参数who 则依which值有不同定义  
which who 代表的意义  
PRIO\_PROCESS who为进程识别码  
PRIO\_PGRP who 为进程的组识别码  
PRIO\_USER who为用户识别码  
参数prio介于-20 至20 之间。代表进程执行优先权，数值越低代表有较高的优先次序，  
执行会较频繁。此优先权默认是0，而只有超级用户（root）允许降低此值。  
  
返回值  执行成功则返回0，如果有错误发生返回值则为-1，错误原因存于errno。  
ESRCH 参数which或who 可能有错，而找不到符合的进程  
EINVAL 参数which值错误。  
EPERM 权限不够，无法完成设置  
EACCES 一般用户无法降低优先权  
  
　   
  
  
  
  
system（执行shell 命令）    
相关函数  fork，execve，waitpid，popen  
  
表头文件  #include<stdlib.h>  
  
定义函数  int system(const char \* string);  
  
函数说明  system()会调用fork()产生子进程，由子进程来调用/bin/sh-c string来执  
行参数string字符串所代表的命令，此命令执行完后随即返回原调用的进程。在调用  
system()期间SIGCHLD 信号会被暂时搁置，SIGINT和SIGQUIT 信号则会被忽略。  
  
返回值  如果system()在调用/bin/sh时失败则返回127，其他失败原因返回-1。若参数  
string为空指针(NULL)，则返回非零值。如果system()调用成功则最后会返回执行shell  
命令后的返回值，但是此返回值也有可能为system()调用/bin/sh失败所返回的127，因  
此最好能再检查errno 来确认执行成功。  
  
附加说明  在编写具有SUID/SGID权限的程序时请勿使用system()，system()会继承环境  
变量，通过环境变量可能会造成系统安全的问题。  
  
范例  #include<stdlib.h>  
main()  
{  
system(“ls -al /etc/passwd /etc/shadow”);  
}  
  
执行  -rw-r--r-- 1 root root 705 Sep 3 13 :52 /etc/passwd  
-r--------- 1 root root 572 Sep 2 15 :34 /etc/shadow  
  
　   
  
  
  
  
wait（等待子进程中断或结束）    
相关函数  waitpid，fork  
  
表头文件  #include<sys/types.h>  
#include<sys/wait.h>  
  
定义函数  pid\_t wait (int \* status);  
  
函数说明  wait()会暂时停止目前进程的执行，直到有信号来到或子进程结束。如果在  
调用wait()时子进程已经结束，则wait()会立即返回子进程结束状态值。子进程的结束  
状态值会由参数status 返回，而子进程的进程识别码也会一快返回。如果不在意结束状  
态值，则  
  
参数  status可以设成NULL。子进程的结束状态值请参考waitpid()。  
  
返回值  如果执行成功则返回子进程识别码(PID)，如果有错误发生则返回-1。失败原因  
存于errno中。  
  
附加说明    
  
范例  #include<stdlib.h>  
#include<unistd.h>  
#include<sys/types.h>  
#include<sys/wait.h>  
main()  
{  
pid\_t pid;  
int status,i;  
if(fork()= =0){  
printf(“This is the child process .pid =%d\n”,getpid());  
exit(5);  
}else{  
sleep(1);  
printf(“This is the parent process ,wait for child...\n”;  
pid=wait(&status);  
i=WEXITSTATUS(status);  
printf(“child’s pid =%d .exit status=^d\n”,pid,i);  
}  
}  
  
执行  This is the child process.pid=1501  
This is the parent process .wait for child...  
child’s pid =1501,exit status =5  
  
　   
  
  
  
  
waitpid（等待子进程中断或结束）    
相关函数  wait，fork  
  
表头文件  #include<sys/types.h>  
#include<sys/wait.h>  
  
定义函数  pid\_t waitpid(pid\_t pid,int \* status,int options);  
  
函数说明  waitpid()会暂时停止目前进程的执行，直到有信号来到或子进程结束。如果  
在调用wait()时子进程已经结束，则wait()会立即返回子进程结束状态值。子进程的结  
束状态值会由参数status返回，而子进程的进程识别码也会一快返回。如果不在意结束  
状态值，则参数status可以设成NULL。参数pid为欲等待的子进程识别码，其他数值意义  
如下:  
pid<-1 等待进程组识别码为pid绝对值的任何子进程。  
pid=-1 等待任何子进程，相当于wait()。  
pid=0 等待进程组识别码与目前进程相同的任何子进程。  
pid>0 等待任何子进程识别码为pid的子进程。  
参数option可以为0 或下面的OR 组合  
WNOHANG 如果没有任何已经结束的子进程则马上返回，不予以等待。  
WUNTRACED 如果子进程进入暂停执行情况则马上返回，但结束状态不予以理会。  
子进程的结束状态返回后存于status，底下有几个宏可判别结束情况  
WIFEXITED(status)如果子进程正常结束则为非0值。  
WEXITSTATUS(status)取得子进程exit()返回的结束代码，一般会先用WIFEXITED 来判断  
是否正常结束才能使用此宏。  
WIFSIGNALED(status)如果子进程是因为信号而结束则此宏值为真  
WTERMSIG(status)取得子进程因信号而中止的信号代码，一般会先用WIFSIGNALED 来判  
断后才使用此宏。  
WIFSTOPPED(status)如果子进程处于暂停执行情况则此宏值为真。一般只有使用  
WUNTRACED 时才会有此情况。  
WSTOPSIG(status)取得引发子进程暂停的信号代码，一般会先用WIFSTOPPED 来判断后才  
使用此宏。  
  
返回值  如果执行成功则返回子进程识别码(PID)，如果有错误发生则返回-1。失败原因  
存于errno中。  
  
范例  参考wait()。  
  
　   
  
  
  
  
fprintf（格式化输出数据至文件）    
相关函数  printf，fscanf，vfprintf  
  
表头文件  #include<stdio.h>  
  
定义函数  int fprintf(FILE \* stream, const char \* format,.......);  
  
函数说明  fprintf()会根据参数format字符串来转换并格式化数据，然后将结果输出到  
参数stream指定的文件中，直到出现字符串结束('\0')为止。  
  
返回值  关于参数format字符串的格式请参考printf()。成功则返回实际输出的字符  
数，失败则返回-1，错误原因存于errno中。  
  
范例  #include<stdio.h>  
main()  
{  
int i = 150;  
int j = -100;  
double k = 3.14159;  
fprintf(stdout,”%d %f %x \n”,j,k,i);  
fprintf(stdout,”%2d %\*d\n”,i,2,i);  
}  
  
执行  -100 3.141590 96  
150 150  
  
　   
  
  
  
  
fscanf（格式化字符串输入）    
相关函数  scanf，sscanf  
  
表头文件  #include<stdio.h>  
  
定义函数  int fscanf(FILE \* stream ,const char \*format,....);  
  
函数说明  fscanf()会自参数stream的文件流中读取字符串，再根据参数format字符串  
来转换并格式化数据。格式转换形式请参考scanf()。转换后的结构存于对应的参数内。  
  
返回值  成功则返回参数数目，失败则返回-1，错误原因存于errno中。  
  
附加说明    
  
范例  #include<stdio.h>  
main()  
{  
int i;  
unsigned int j;  
char s[5];  
fscanf(stdin,”%d %x %5[a-z] %\*s %f”,&i,&j,s,s);  
printf(“%d %d %s \n”,i,j,s);  
}  
  
执行  10 0x1b aaaaaaaaa bbbbbbbbbb /\*从键盘输入\*/  
10 27 aaaaa  
  
　   
  
  
  
  
printf（格式化输出数据）    
相关函数  scanf，snprintf  
  
表头文件  #include<stdio.h>  
  
定义函数  int printf(const char \* format,.............);  
  
函数说明  printf()会根据参数format字符串来转换并格式化数据，然后将结果写出到  
标准输出设备，直到出现字符串结束('\0')为止。参数format字符串可包含下列三种字  
符类型  
1.一般文本，伴随直接输出。  
2.ASCII控制字符，如\t、\n等。  
3.格式转换字符。  
格式转换为一个百分比符号(％)及其后的格式字符所组成。一般而言，每个％符号在其  
后都必需有一printf()的参数与之相呼应（只有当％％转换字符出现时会直接输出％字  
符），而欲输出的数据类型必须与其相对应的转换字符类型相同。  
Printf()格式转换的一般形式如下  
％(flags)(width)(.prec)type  
以中括号括起来的参数为选择性参数，而％与type则是必要的。底下先介绍type的几种  
形式  
整数  
％d 整数的参数会被转成一有符号的十进制数字  
％u 整数的参数会被转成一无符号的十进制数字  
％o 整数的参数会被转成一无符号的八进制数字  
％x 整数的参数会被转成一无符号的十六进制数字，并以小写abcdef表示  
％X 整数的参数会被转成一无符号的十六进制数字，并以大写ABCDEF表示浮点型数  
％f double 型的参数会被转成十进制数字，并取到小数点以下六位，四舍五入。  
％e double型的参数以指数形式打印，有一个数字会在小数点前，六位数字在小数点  
后，而在指数部分会以小写的e来表示。  
％E 与％e作用相同，唯一区别是指数部分将以大写的E 来表示。  
％g double 型的参数会自动选择以％f 或％e 的格式来打印，其标准是根据欲打印的数  
值及所设置的有效位数来决定。  
％G 与％g 作用相同，唯一区别在以指数形态打印时会选择％E 格式。  
字符及字符串  
％c 整型数的参数会被转成unsigned char型打印出。  
％s 指向字符串的参数会被逐字输出，直到出现NULL字符为止  
％p 如果是参数是“void \*”型指针则使用十六进制格式显示。  
prec 有几种情况  
1. 正整数的最小位数。  
2.在浮点型数中代表小数位数  
3.在％g 格式代表有效位数的最大值。  
4.在％s格式代表字符串的最大长度。  
5.若为×符号则代表下个参数值为最大长度。  
width为参数的最小长度，若此栏并非数值，而是\*符号，则表示以下一个参数当做参数  
长度。  
flags 有下列几种情况  
#NAME?  
+ 一般在打印负数时，printf（）会加印一个负号，整数则不加任何负号。此旗标会使  
得在打印正数前多一个正号（+）。  
# 此旗标会根据其后转换字符的不同而有不同含义。当在类型为o 之前（如％#o），则  
会在打印八进制数值前多印一个o。  
而在类型为x 之前（％#x）则会在打印十六进制数前多印’0x’，在型态为e、E、f、g  
或G 之前则会强迫数值打印小数点。在类型为g 或G之前时则同时保留小数点及小数位数  
末尾的零。  
0 当有指定参数时，无数字的参数将补上0。默认是关闭此旗标，所以一般会打印出空白  
字符。  
  
返回值  成功则返回实际输出的字符数，失败则返回-1，错误原因存于errno中。  
  
范例  #include<stdio.h>  
main()  
{  
int i = 150;  
int j = -100;  
double k = 3.14159;  
printf(“%d %f %x\n”,j,k,i);  
printf(“%2d %\*d\n”,i,2,i); /\*参数2 会代入格式\*中，而与%2d同意义\*/  
}  
  
执行  -100 3.14159 96  
150 150  
  
　   
  
  
  
  
sacnf（格式化字符串输入）    
相关函数  fscanf，snprintf  
  
表头文件  #include<stdio.h>  
  
定义函数  int scanf(const char \* format,.......);  
  
函数说明  scanf()会将输入的数据根据参数format字符串来转换并格式化数据。Scanf  
()格式转换的一般形式如下  
％[\*][size][l][h]type  
以中括号括起来的参数为选择性参数，而％与type则是必要的。  
\* 代表该对应的参数数据忽略不保存。  
size 为允许参数输入的数据长度。  
l 输入的数据数值以long int 或double型保存。  
h 输入的数据数值以short int 型保存。  
底下介绍type的几种形式  
％d 输入的数据会被转成一有符号的十进制数字（int）。  
％i 输入的数据会被转成一有符号的十进制数字，若输入数据以“0x”或“0X”开头代  
表转换十六进制数字，若以“0”开头则转换八进制数字，其他情况代表十进制。  
％0 输入的数据会被转换成一无符号的八进制数字。  
％u 输入的数据会被转换成一无符号的正整数。  
％x 输入的数据为无符号的十六进制数字，转换后存于unsigned int型变量。  
％X 同％x  
％f 输入的数据为有符号的浮点型数，转换后存于float型变量。  
％e 同％f  
％E 同％f  
％g 同％f  
％s 输入数据为以空格字符为终止的字符串。  
％c 输入数据为单一字符。  
[] 读取数据但只允许括号内的字符。如[a-z]。  
[^] 读取数据但不允许中括号的^符号后的字符出现，如[^0-9].  
  
返回值  成功则返回参数数目，失败则返回-1，错误原因存于errno中。  
  
范例  #include <stdio.h>  
main()  
{  
int i;  
unsigned int j;  
char s[5];  
scanf(“%d %x %5[a-z] %\*s %f”,&i,&j,s,s);  
printf(“%d %d %s\n”,i,j,s);  
}  
  
执行  10 0x1b aaaaaaaaaa bbbbbbbbbb  
10 27 aaaaa  
  
　   
  
  
  
  
sprintf（格式化字符串复制）    
相关函数  printf，sprintf  
  
表头文件  #include<stdio.h>  
  
定义函数  int sprintf( char \*str,const char \* format,.........);  
  
函数说明  sprintf()会根据参数format字符串来转换并格式化数据，然后将结果复制到  
参数str所指的字符串数组，直到出现字符串结束(’\0’)为止。关于参数format字符串  
的格式请参考printf()。  
  
返回值  成功则返回参数str字符串长度，失败则返回-1，错误原因存于errno中。  
  
附加说明  使用此函数得留意堆栈溢出，或改用snprintf（）。  
  
范例  #include<stdio.h>  
main()  
{  
char \* a=”This is string A!”;  
char buf[80];  
sprintf(buf,”>>> %s<<<\n”,a);  
printf(“%s”.buf);  
}  
  
执行  >>>This is string A!<<<  
  
　   
  
  
  
  
sscanf（格式化字符串输入）    
相关函数  scanf，fscanf  
  
表头文件  #include<stdio.h>  
  
定义函数  int sscanf (const char \*str,const char \* format,........);  
  
函数说明  sscanf()会将参数str的字符串根据参数format字符串来转换并格式化数据。  
格式转换形式请参考scanf()。转换后的结果存于对应的参数内。  
  
返回值  成功则返回参数数目，失败则返回-1，错误原因存于errno中。  
  
范例  #include<stdio.h>  
main()  
{  
int i;  
unsigned int j;  
char input[ ]=”10 0x1b aaaaaaaa bbbbbbbb”;  
char s[5];  
sscanf(input,”%d %x %5[a-z] %\*s %f”,&i,&j,s,s);  
printf(“%d %d %s\n”,i,j,s);  
}  
  
执行  10 27 aaaaa  
  
　   
  
  
  
  
vfprintf（格式化输出数据至文件）    
相关函数  printf，fscanf，fprintf  
  
表头文件  #include<stdio.h>  
#include<stdarg.h>  
  
定义函数  int vfprintf(FILE \*stream,const char \* format,va\_list ap);  
  
函数说明  vfprintf()会根据参数format字符串来转换并格式化数据，然后将结果输出  
到参数stream指定的文件中，直到出现字符串结束(’\0’)为止。关于参数format字符  
串的格式请参考printf()。va\_list用法请参考附录C或vprintf()范例。  
  
返回值  成功则返回实际输出的字符数，失败则返回-1，错误原因存于errno中。  
  
范例  参考fprintf()及vprintf()。  
  
　   
  
  
  
  
vfscanf（格式化字符串输入）    
相关函数  scanf，sscanf，fscanf  
  
表头文件  #include<stdio.h>  
  
定义函数  int vfscanf(FILE \* stream,const char \* format ,va\_list ap);  
  
函数说明  vfscanf()会自参数stream 的文件流中读取字符串，再根据参数format字符  
串来转换并格式化数据。格式转换形式请参考scanf()。转换后的结果存于对应的参数  
内。va\_list用法请参考附录C 或vprintf()。  
  
返回值  成功则返回参数数目，失败则返回-1，错误原因存于errno中。  
  
范例  参考fscanf()及vprintf()。  
  
　   
  
  
  
  
vprintf（格式化输出数据）    
相关函数  printf，vfprintf，vsprintf  
  
表头文件  #include<stdio.h>  
#include<stdarg.h>  
  
定义函数  int vprintf(const char \* format,va\_list ap);  
  
函数说明  vprintf()作用和printf()相同，参数format格式也相同。va\_list为不定个  
数的参数列，用法及范例请参考附录C。  
  
返回值  成功则返回实际输出的字符数，失败则返回-1，错误原因存于errno中。  
  
范例  #include<stdio.h>  
#include<stdarg.h>  
int my\_printf( const char \*format,……)  
{  
va\_list ap;  
int retval;  
va\_start(ap,format);  
printf(“my\_printf( ):”);  
retval = vprintf(format,ap);  
va\_end(ap);  
return retval;  
}  
main()  
{  
int i = 150,j = -100;  
double k = 3.14159;  
my\_printf(“%d %f %x\n”,j,k,i);  
my\_printf(“%2d %\*d\n”,i,2,i);  
}  
  
执行  my\_printf() : -100 3.14159 96  
my\_printf() : 150 150  
  
　   
  
  
  
  
vscanf（格式化字符串输入）    
相关函数  vsscanf，vfscanf  
  
表头文件  #include<stdio.h>  
#include<stdarg.h>  
  
定义函数  int vscanf( const char \* format,va\_list ap);  
  
函数说明  vscanf()会将输入的数据根据参数format字符串来转换并格式化数据。格式  
转换形式请参考scanf()。转换后的结果存于对应的参数内。va\_list用法请参考附录C或  
vprintf()范例。  
  
返回值  成功则返回参数数目，失败则返回-1，错误原因存于errno中。  
  
范例  请参考scanf()及vprintf()。  
  
　   
  
  
  
  
vsprintf（格式化字符串复制）    
相关函数  vnsprintf，vprintf，snprintf  
  
表头文件  #include<stdio.h>  
  
定义函数  int vsprintf( char \* str,const char \* format,va\_list ap);  
  
函数说明  vsprintf()会根据参数format字符串来转换并格式化数据，然后将结果复制  
到参数str所指的字符串数组，直到出现字符串结束(’\0’)为止。关于参数format字符  
串的格式请参考printf()。va\_list用法请参考附录C或vprintf()范例。  
  
返回值  成功则返回参数str字符串长度，失败则返回-1，错误原因存于errno中。  
  
范例  请参考vprintf()及vsprintf()。  
  
　   
  
  
  
  
vsscanf（格式化字符串输入）    
相关函数  vscanf，vfscanf  
  
表头文件  #include<stdio.h>  
  
定义函数  int vsscanf(const char \* str,const char \* format,va\_list ap);  
  
函数说明  vsscanf()会将参数str的字符串根据参数format字符串来转换并格式化数  
据。格式转换形式请参考附录C 或vprintf()范例。  
  
返回值  成功则返回参数数目，失败则返回-1，错误原因存于errno中。  
  
范例  请参考sscanf()及vprintf()。