# 【系统api】

int fd = open("abc.txt", O\_RDONLY);

int fd = open("newfile", O\_CREAT | O\_WRONLY, 0664);

int nb = read(fd, buf, sizeof(buf));

int ret = write(fd, buf, count);

ulink("temp.txt"); //删除文件目录项当所有进程使用完该文件后该文件被释放

close(fd);

perror("open error");

exit(1);

sleep(1);

time(NULL);

srand(time(NULL));

rand();

int ret = lseek(fd, 0, SEEK\_END); //移动文件指针偏移SEEK\_END+0，并指向下一个字节位置，返回SEEK\_END+0大小（文件字节不够，产生空洞文件填补）

int ret = ftruncate(fd, 5); //空洞文件5Byte

struct stat buf ; int ret = stat("abc.txt", &buf); //查看文件属性，指令：stat 文件名

pid\_t pid = fork(); //0：子进程，pid号：父进程

printf("I'm child: pid = %u, ppid = %u\n", getpid(), getppid());

execlp("ls", "ls","-l", NULL); //调用系统程序

execl("/bin/ls", "-l", NULL); //调用自己的程序, 调用路径+程序名

dup2(fd, STDOUT\_FILENO); //重定向STDOUT\_FILENO到fd（标准输出将会输出到fd对应的文件）

pid\_t wpid = wait(&status); //阻塞回收子进程

if(WIFEXITED(status)){ //正常结束

printf("child exit with %d\n", WEXITSTATUS(status));

}

if(WIFSIGNALED(status)){ //异常结束

printf("child killed by %d\n", WTERMSIG(status));

}

int wp = waitpid(-1, NULL, WNOHANG);

int fd[2]; int ret = pipe(fd); //创建管道

char \* p = mmap(NULL, 4, PROT\_READ|PROT\_WRITE, MAP\_SHARED, fd, 0);

int ret = munmap(p, 4); //关闭映射区

p = mmap(NULL, 4, PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_ANONYMOUS, -1, 0); // MAP\_ANONYMOUS匿名映射，p = MAP\_FAILED创建失败

kill(num3\_pid, SIGKILL); //给进程发信号，SIGKILL为杀死信号

int ret = alarm(10); //定时10s，返回剩余时长，alarm(0);取消定时

int ret = setitimer(ITIMER\_REAL, &it, &oldit); //us级定时

sighandler\_t hd = signal(SIGALRM, myfun); // //注册SIGALRM信号的回调函数

sigset\_t myset; sigemptyset(&myset); // myset信号集值0

sigaddset(&myset, SIGQUIT); //加入要屏蔽的信号SIGQUIT

int ret = sigprocmask(SIG\_BLOCK, &myset, &oldset); //屏蔽信号（操作自定义信号集去影响阻塞信号集）

sigpending(&ped); //获取未决信号集

if(sigismember(&ped, i) == 1) //判断某信号是否在信号集ped里

struct sigaction act; int ret = sigaction(SIGINT, &act, &old\_act); //注册SIGINT及回调

pause(); //挂起当前进程，等待唤醒

sigset\_t suspmask; sigsuspend(&suspmask); //挂起当前进程，等待唤醒

getpgrp(); //组id 等价于 getpgid(0)

setpgid(getpid(), getppid()); //将父进程加入到进程组

getsid(0); //会议id

setsid(); //以自己的id设置进程组id，也是设置会话id

umask(0002); //指定后续创建的文件的权限002

chdir("/home/gj"); //改变进程工作目录

printf("thread id:%lu ", pthread\_self()); //主控线程id

pthread\_t tid; ret = pthread\_create(&tid, NULL, thrd1\_fun, (void \*)i); //创建线程并把参数i传递给线程thrd1\_fun

pthread\_exit(NULL); //线程退出

pthread\_join(tid, NULL); //阻塞等待线程退出

pthread\_detach(tid[1]); //线程分离，结束后自动回收

pthread\_cancel(tid[2]); //杀死线程

pthread\_testcancel(); //定义取消点，配合cancel杀死进程

pthread\_mutex\_t mutex1; //互斥锁

pthread\_mutex\_init(&mutex1, NULL);

pthread\_mutex\_lock(&mutex1);

pthread\_mutex\_unlock(&mutex1);

pthread\_mutex\_destroy(&mutex1); //摧毁锁

pthread\_rwlock\_t rwlock1; //读写锁, 读时共享，写时独占，并行阻塞写锁优先

pthread\_rwlock\_init(&rwlock1, NULL);

pthread\_rwlock\_rdlock(&rwlock1); //读加锁

pthread\_rwlock\_wrlock(&rwlock1); //写加锁

pthread\_rwlock\_unlock(&rwlock1);

pthread\_rwlock\_destroy(&rwlock1);

pthread\_cond\_t cond1; //条件变量

pthread\_cond\_wait(&cond1, &mutex1); //等待条件变量满足，解锁

pthread\_cond\_signal(&cond1); //唤醒某个阻塞在条件变量上的进程

pthread\_cond\_broadcast(&cond1); //唤醒所有阻塞在条件变量上的进程

pthread\_cond\_destroy(&cond1); //摧毁条件变量

sem\_t blank\_number; //信号量值

sem\_wait(&blank\_number); //信号量-1，为0时阻塞

sem\_post(&product\_number); //信号量+1

sem\_init(&blank\_number, 0, NUM); //信号量的值初始化为NUM

sem\_destroy(&blank\_number);

pthread\_mutexattr\_t mutexattr1; //锁属性

pthread\_mutexattr\_init(&mutexattr1);

pthread\_mutexattr\_setpshared(&mutexattr1, PTHREAD\_PROCESS\_SHARED);//将锁属性改为进程使用

pthread\_mutexattr\_destroy(&mutexattr1); //销毁锁属性对象

struct flock myflock; // fcntl文件锁（进程间同步）

myflock.l\_type = F\_WRLCK; fcntl(fd, F\_SETLKW, &myflock); //写加锁

myflock.l\_type = F\_RDLCK; fcntl(fd, F\_SETLKW, &myflock); //读加锁

myflock.l\_type = F\_UNLCK; fcntl(fd, F\_SETLKW, &myflock); //解锁

# 【网络api】

int listenfd = socket(AF\_INET, SOCK\_STREAM, 0); //设置网络套接字

struct sockaddr\_in saddr; bzero(&saddr, sizeof(saddr)); //清空

memset(saddr, 0, sizeof(saddr)); //置为0

inet\_pton(AF\_INET,IP, &(saddr.sin\_addr.s\_addr)); //将字符串IP地址转换为网络字节序

inet\_ntop(AF\_INET, &saddr.sin\_addr.s\_addr, client\_ip, sizeof(client\_ip)) ; //逆函数

ntohs(saddr.sin\_port); //网络字节序转化为%d

htons(PORT); // PORT转化为网络字节序

int ret = bind(listenfd, (struct sockaddr\*)&saddr, sizeof(saddr)); //绑定ip+端口

listen(listenfd, 128); //允许监听的数量

cfd = accept(listenfd, (struct sockaddr\*)&caddr, &caddrlen); //阻塞接受连接请求

nb= read(cfd, buf, sizeof(buf));

buf[i] = toupper(buf[i]); //小写转大写字母

fgets(buf, sizeof(buf), stdin); //从标准输入获取字符串，"hello\n\0"

write(cfd\_fd, buf, nb);

int ret = connect(cfd, (struct sockaddr\*)&saddr, saddrlen);

int opt = 1; setsockopt(listenfd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));

//允许端口复用

fd\_set rset;

FD\_ZERO(&rset); //文件描述符清0

FD\_SET(lfd, &rset); //将lfd加入描述符集合

int nready = select(lfd+1, &rset, NULL, NULL, NULL); //监听连接请求

if (FD\_ISSET(lfd, &rset)); // lfd是否在rset集合里

struct pollfd client[1024]; nready = poll(client, 1, -1); //第二个参数：数组元素有多少个描述符需要监听

struct epoll\_event evts, outevts[1024];

efd = epoll\_create(1024);

int res = epoll\_ctl(efd, EPOLL\_CTL\_ADD, lfd, & evts);

nready = epoll\_wait(efd, outevts, 1024, -1);

int flag = fcntl(cfd, F\_GETFL); //设置描述符非阻塞读

flag |= O\_NONBLOCK;

fcntl(cfd, F\_SETFL, flag);