linux系统和网络编程

L01-06 文件系统与目录操作

L01 标准库函数和系统调用

- 标准库函数
  - fopen — FILE *fopen(const char *pathname, const char *mode)
  - fgetc — int fgetc(FILE *stream)
  - fputc — int fputc(int char *), FILE *stream)
  - fclose — int fclose(FILE *stream)
- 标准库函数在用户空间的缓冲区
  - 全缓冲
    - 缓冲区满了触发调用系统调用
    - 缓冲区满了，或者遇到换行，调用系统调用
  - 行缓冲
    - 标准输出是行缓冲
  - 无缓冲
    - stderr是无缓冲
  - 测试：file/setbuf.c
- 程序启动自动打开三个文件
  - STDIN_FILENO — 0
  - STDOUT_FILENO — 1
  - STDERR_FILENO — 2

L02 文件与IO

- 系统调用函数
  - open — int open(const char *pathname, int flags);
    - 练习：file/file4.c
  - close — int close(int fd);
  - read — ssize_t read(int fd, void *buf, size_t count);
    - 练习：file/read.c、readdir.dc.c
  - write — ssize_t write(int fd, const void *buf, size_t count);
    - 练习：file/nonblockread.c
  - umask
- 阻塞与非阻塞
  - 状态
    - 阻塞状态 — sleep
    - 运行状态 — 正在被调度执行
    - 就绪状态
    - 非阻塞状态
  - fcntl — int fcntl(int fd, int cmd, ...); /* arg */ }
    - 改变一个已打开文件的阻塞非阻塞
    - 练习：file/nonblockread.c
  - 练习：file/nonblockread.c
- 标准库函数
  - fopen — FILE *fopen(const char *pathname, const char *mode);
  - fclose — FILE *fclose(int fd, const char *mode);
  - stdin,stdout,stderr
  - fgetc — int fgetc(FILE *stream);
  - fputc — int fputc(int c, FILE *stream);
  - fclose — int fclose(FILE *stream);
  - fseek — int fseek(FILE *stream, long offset, int whence); — whence取值：offset偏移量
    - long ftell(FILE *stream);
    - void rewind(FILE *stream);
    - od -tx1 -c hello.txt — 查看文件内容，各不同定制以2
  - 练习file2cpf./file.cpf.c、file2cpf./myrp.c、file2cpf./myrp.c
- 系统调用函数
  - lseek — off_t lseek(int fd, off_t offset, int whence);
    - 内核态的文件读写位置，不是用户区
    - 练习：file/cpf./lseek.c
  - fcntl — int fcntl(int fd, int cmd, ...); /* arg */ }
    - 改变一个已打开文件的属性，比如阻塞、阻塞等
    - 练习：file/cpf./fcntl.c — 改变打开文件的阻塞非阻塞
    - 练习：file/cpf./fcntlGetFlag.c
      - 取得文件标记符的状态，包括读写、阻塞等
      - ./a.out 0
      - ./a.out +dbx/tTV
      - ./a.out +good.txt
  - ioctl — int ioctl(int fd, unsigned long request, ...);
    - 练习：file/cpf./ioctlget.c — 获取终端的窗口尺寸
    - void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
  - mmap
    - 练习：file/cpf./mmap.c

L03-04 文件与IO-高级控制

- ext2文件系统
  - 启动块 — 1KB，由PC引导程序，用来存储硬盘分区信息和启动信息，任何文件系统都不能使用该块
  - 超级块 — 描述整个分区文件系统信息，例如大小、文件系统信息等，上次mount时间等等，超级块每个块组的开头都有一份拷贝
  - 块组描述符表 — 由很多块组描述符组成，整个分区分成多少个块组就对应有多少个块组描述符，每组描述符包括：块位图块、索引节点位图、块组内的索引表、空闲块的数目、目录的数目等，数据块从索引开始，数据块从最末开始，空闲的inode表明索引节点是否空闲，块位图表示哪些块是空闲块或是被一些索引
  - 块位图 — 用来描述整个块组中哪些块已经使用，哪些空闲
  - inode位图 — 标记一个inode是否空闲可用
  - inode表 — 文件类型（常规、目录、符号链接等），权限，文件大小、创建、修改、访问时间等信息，每个文件都有一个inode
  - 数据块
    - 常规文件 — 文件的数据存储在数据块中
    - 目录 — 该目录下所有文件和目录的名称和它们的索引节点号，文件名保存在它所在的目录的数据块中，其他信息都保存在该文件的inode中。
    - 符号链接 — 如果目标路径名比较短则直接保存在inode中，否则分配一个数据块保存
    - 设备文件、FIFO、和socket等特殊文件 — 没有数据块，设备文件的主设备号和辅设号保存在inode中
  -

L05-06 文件系统与目录操作

- 系统调用
  - stat
    - 返回文件信息
    - int stat(const char *pathname, struct stat *statbuf)
    - int fstat(int fd, struct stat *statbuf);
    - int lstat(const char *pathname, struct stat *statbuf);
    - 命令行可以直接查看一个文件的信息 — stat test.txt
    - 练习：sfu/stat/stat.c
  - 其他函数
    - access
    - chmod、fchmod
    - chown、fchown、lchown
    - utime
    - truncate、ftruncate
    - link
    - unlink
    - rename
    - readlink
    - mkdir
    - rmdir
  - opendir/readdir/closedir — 遍历目录信息
    - 练习：sfu/stat/dirOpt.c
  - chdir/fchdir — 修改程序的工作目录
    - 练习：sfu/dup/dupOpt.c
- 作业1：实现ls -al

L07-08 进程休系与进程管理

- 目录
  - 进程控制块PCB
  - 进程控制fork
  - exec函数族
  - wait和waitpid
  - 进程间通信
- fork
  - 根据一个现有的进程复制出一个新的进程
  - 查看进程 — ps aux、pstree
  - pid_t fork(void);
  - getpid — 获取进程id
  - getppid — 获取父进程id
  - 练习：process/forkOpt.c、multiProc.c
- exec函数族
  - 替代当前进程
  - int execl(const char *pathname, const char *arg, ... /* (char *) NULL */);
  - int execlp(const char *file, const char *arg, ... /* (char *) NULL */);
  - int execle(const char *pathname, const char *arg, ... /* (char *) NULL, char *const envp[] */);
  - int execv(const char *pathname, char *const argv[]);
  - int execvp(const char *file, char *const argv[];
  - int execve(const char *filename, char *const argv[], char *const envp[]);
  - char *getenv(const char *name);
  - 练习：process/execOpt.c、execOpt.c、execOpt2.c
  - 重定向
    - 重定向输入输出
    - 练习 process/redirect/callback.c
- wait、wait.pid
  - 一个进程结束以后会发送所有父进程进程，释放其用户空间分配的内存，但它的PCB还保留着，内核在其中保存了一些信息如果父亲想之前保存着退出状态，发送信号给父进程保存信息，父进程可能要？
  - pid_t wait(int *wstatus);
  - pid_t waitpid(pid_t pid, int *wstatus, int options);
  - 练习：process/combs.c、waitOpt.c

L09-10 进程间通信

- 一种最古老的IPC机制，管道是通信信道
  - int pipe(int pipefd[2]);
  - 4种特殊情况
    - 1.如果所有管道写端都关闭了，有读端，read返回0，就像文件末尾一样
    - 2.如果写端没关闭，没有写数据，读端read会阻塞，指导有数据读取才返回
    - 3.所有读端都关了，写端write会异常终止，signpe信号
    - 4.如果读端没有关闭，写端一直写，写满后write会阻塞
  - 练习：process/pipeCom/unnamed.pipe/pipe3.c
- 关闭管道
  - 作业2：实现cat
- 实现简单的Shell

  用神话的各种C函数实现一个简单的交互式Shell，要求：
  1、给出提示符，让用户输入一行命令，识别程序名和参数并调用适当的exec函数执行程序，等执行完成后再次给出提示符。
  2、识别和处理以下符号：
  简单的标准输入输出重定向（<和>），对管道描述符，然后fork出两个子进程，一个子进程关闭读端，调用dup2把写端赋给标准输出，另一个子进程关闭写端，调用dup2把读端赋给标准输入，两个子进程分别调用exec执行程序，而Shell进程把管道的两端都关闭，调用wait等待两个子进程终止。

  你的程序应该可以处理以下命令：
  ○ls△-l△-R○>○file1○
  ○cat○<○file1○|○wc△-p○>○file1○
  ○表示零个或多个空格，△表示一个或多个空格

  子主题
- 关闭管道popen和pclose
  - 创建一个管道，fork一个子进程，关闭管道的不使用端，exec一个shell命令，等待命令终止
  - FILE *popen(const char *command, const char *type);
  - int pclose(FILE *stream);
  - 练习：process/pipeCom/unnamed.pipe/popenOpt.c popenOpt.c
- 命名管道FIFO
  - mkfifo /tube命令创建一个有名管道
  - 练习：process/pipeCom/named.pipe/mkfifo.c writefifo.c
- 共享内存
  - 共享内存允许两个或多个进程共享公用这段内存空间
  - key_t ftok(const char *pathname, int proj_id);
  - int shmget(key_t key, size_t size, int shmflg);
  - void *shmat(int shmid, const void *shmaddr, int shmflg);
  - void *shmat(int shmid, const void *shmaddr, int shmflg);
  - int shmctl(int shmid, int cmd, struct shmid_ds *buf);
  - 使用命令行 ipcs -m/-q/-s命令查看，使用命令行ipcrm -m/-q/-s删除
  - 练习：process/pipeCom/shared.memory/shmget.c
- 消息队列
  - 系统内核维护了一个存放消息的队列，不同用户可以访问队列中发送消息，或者从队列中接收消息
  - int msgget(key_t key, int msgflg); //创建或获取消息队列
  - int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
  - ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,int msgflg);
  - 练习：process/pipeCom/msgqueue/getMsg.c sendMsg.c
- 信号
  - 目录
  - kill -l命令查看信号
  - man 7 signal查看信号说明
  - 信号发送方式总结
  - 常见信号
  - 简单信号举例
    - kill
      - kill函数给一个指定的进程发送指定信号
      - int kill(pid_t pid, int sig);
      - 练习：signal/killOpt.c
    - raise
      - raise函数给当前进程发送指定的信号
      - int raise(int sig);
      - 练习：signal/raiseOpt.c
    - abort
      - abort函数向当前进程发送SIGABRT信号并异常终止
      - void abort(void);
      - 练习：signal/raiseOpt.c
    - 管道破裂信号
      - 练习：signal/pipeSignal.c
    - 闹钟信号
      - 练习：alarmOpt.c
    - sigchld信号
      - 子进程在终止时会给它父进程发SIGCHLD信号，该信号默认处理动作是忽略，父进程可以自定义SIGCHLD信号处理函数，函数中调wait处理，不需要主动等待调用wait,waitpid函数阻塞等待
      - 练习：signal/sigchld.c
  - 信号相关函数
    - 信号集操作函数
      - int sigempty(sigset_t *set);
      - int sigfillset(sigset_t *set);
      - int sigaddset(sigset_t *set, int signum);
      - int sigdelset(sigset_t *set, int signum);
      - int sigismember(const sigset_t *set, int signum);
    - 信号阻塞
      - int sigprocmask(int how, const sigset_t *set, sigset_t *oldset); — 信号阻塞字集写函数
      - 练习：signal/sigMask.c
    - 信号未决集
      - sigsuspend函数挂起进程的未决信号集，通过set参数传出
      - int sigpending(sigset_t *set);
      - 练习：signal/sigMask.c
    - 捕获信号
      - 如果信号的处理动作是用户自定义函数，在信号递达时调用该函数，成为捕获信号
      - int sigaction(int signum, const struct sigaction *act,struct sigaction *oldact);
      - 练习：signal/sighandler.c
      - pause函数使调用进程挂起等待信号递达。
      - int pause(void);
      - 练习 signal/mykill.c
    - 竞态条件与sigsuspend函数
      - 多子操作，临时设置sigmask屏蔽字，并挂起进程，等待信号唤醒
      - 练习：signal/suspend.c
  - 作业3：编写一个程序，实现父进程创建两个子进程，两个父进程通过sigqueue向对端发送数据（为了方便，传送字符串即可），父进程分别向两个子进程发送SIGUSR1和SIGUSR2，两个子进程收到信号后分别打印：
    "child 1 is killed by parent !!" "child 2 is killed by parent!!"

L11-13 信号

- 概念：由于同一个进程的多个线程共享同一地址空间，因此Text Segment、Data Segment都是共享的，如果定义一个函数，在各线程中都可以调用的，如果定义一个全局变量，在各线程中都可以访问，修改之类，各线程共享以下进程资源和环境：
  1.文件描述符表
  2.每种信号的处理方式
  3.当前进程工作目录
  4.用户id和组id
  线程可以拥有属于线程自己的：
  1.线程id
  2.上下文，包括各种寄存器的值，程序计数器和堆栈指针
  3.栈空间
  4.errno变量
  5.信号屏蔽字
  6.调度优先级
- 线程控制
  - 创建线程
    - int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);
    - pthread_t pthread_self(void); — 获取当前线程id
  - 终止线程
    - 1.从线程函数return，这样方式对主线程不适用，从main函数return相当于调用exit
    - int pthread_cancel(pthread_t thread); — 2.一个线程可以调用pthread_cancel终止同一个进程中的另一个线程
    - void pthread_exit(void *retval); — 3.线程可以调用pthread_exit终止自己
    - 练习：thread/threadCreat.c creatThread.c
  - 回收线程
    - 调用pthread join函数的线程将挂起等待，直到对thread指定的线程终止 — 线程挂？？
    - 练习：thread/testThread.c exitThread.c
    - 练习：thread/testdetach.thread.c
  - 互斥锁
    - 练习：thread/testdetach.thread.c
    - #include <pthread.h>
      pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;
      pthread_mutex_t recursivemutex = PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP;
      pthread_mutex_t errormutex = PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP;
      int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr);
      int pthread_mutex_lock(pthread_mutex_t *mutex);
      int pthread_mutex_trylock(pthread_mutex_t *mutex);
      int pthread_mutex_unlock(pthread_mutex_t *mutex);
      int pthread_mutex_destroy(pthread_mutex_t *mutex);
    - 练习：thread/testdetach.thread.c
  - 线程间同步
    - 条件变量
      - #include <pthread.h>
        pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
        int pthread_cond_init(pthread_cond_t *cond,pthread_condattr_t *cond_attr);
        int pthread_cond_signal(pthread_cond_t *cond);
        int pthread_cond_broadcast(pthread_cond_t *cond);
        int pthread_cond_wait(pthread_cond_t *cthread,mu=tex_t *mutex);
        int pthread_cond_timedwait(pthread_cond_t *cond,pthread_mutex_t *mutex,const struct timespec *abs-time);
        int pthread_cond_destroy(pthread_cond_t *cond);
      - 练习：thread/condv.c
    - 信号量
      - int sem_init(sem_t *sem, int pshared, unsigned int value);
      - int sem_wait(sem_t *sem);
      - int sem_trywait(sem_t *sem);
      - int sem_timedwait(sem_t *sem,const struct timespec *abs_timeout);
      - int sem_post(sem_t *sem);
      - int sem_destroy(sem_t *sem);
      - 练习：thread/sem/
  - 作业4：哲学家吃饭子
- 作业5：多进程和多线程的比较，并写出结论

L14-15 线程

- 文件锁
  - 练习：summary/flock.c write.lock
  - int flock(int fd, int operation);
- 管道
  - 练习：summary/FIFOpt.c
  - sfw和管道的区别在于管道没有名字，pipe() 或者调用socketpair()不进行创建文件，而是通过内核直接分配内存空间来实现。
  - 对于管道（无NAME），只有一个进程可以访问文件的两端一种情况下通信，并多个进程的进程可以共享管道，但是只有一个进程可以访问管道的写端，而其他进程只能读取，因此两个进程可以共享管道，但是两个进程共享管道的情况下，只能一个进程写入，其他进程只能读取，而不能同时写入。
  - 对于命名管道，文件系统中会创建一个特殊文件，这个文件的内容是管道的名字，这样一个进程可以通过这个文件的名字来访问管道，这样一个进程可以通过这个文件的名字来访问管道，这样一个进程可以通过这个文件的名字来访问管道