linux基础——ubuntu

基础命令

- 命令行--终端打开ctrl+atl t
- cmd -options args
- 善于tab(补齐)
- 一切没有消息的消息就是最好的消息
- 命令---》已经写好的可执行文件
- 环境变量PATH--》路径:/bin/:/sbin/:/usr/bin/ .....
- history 查看登陆用户的历史命令
- 上下方向键切换历史命令
- cal month year查看日历

文件管理

- cd 切换工作路径
    - 路径：linux os文件系统从/开始的
        - 绝对路径：从/开始
        - 相对路径: 从当前.
- pwd 显示当前工作路径
- ls -[l/d/h/i/a] args
    - 列出目录的文件信息
    - -l 长格式显示
        - 类型
            - -普通文件
            - d目录文件
            - c字符设备文件
            - b块设备文件
            - s套接字文件
            - l链接文件
            - p管道文件
        - 权限
            - 作用对象
                - 拥有着 u
                - 所属组 g
                - 其他用户 o
            - 基本权限
                - r 读　目录：ls
                - w 写　目录：touch mkdir rm mv
                - x 执行 目录：cd 检索为
            - 修改权限
                - chmod u / g / o +- r / w/ x file
                    - chmod u+r file

        - 每一组转换为一个八进制数值
            - rw-rw-r------>664
    - 特殊权限
        - u+s--->not dir 可执行--->4
            - 使得执行拥有此权限的用户再期间拥有文件拥有着身份(提升权限)---》passwd
        - g+s--->dir---->2
            - 使得拥有此权限的目录的子文件继承其组
        - o+t--->dir---->1
            - 粘着位 使得所有用户再此目录下只能删改自己创建的文件
    - umask 文件屏蔽字
        - 防止产生权限过大文件
- 硬链接个数
    - 硬链接：同一个文件的多个名字 ln src newhard
    - 软链接(符号链接):快捷方式 ln -s src newsoft
    - 系统不允许为目录创建硬链接
        - . ..是目录硬链接
- 拥有者
- 所属组
- 字节大小
    - ！= 所占磁盘空间
    - stat path
        - blocks--->512k
    - du -sh path
- 最后一次修改时间
    - atime
        - 最后一次访问时间
    - ctime
        - 最后一次属性改变
    - mtime
        - 最后一次内容改变
- 文件名

- touch 创建空文件
- mkdir 创建目录
- cp [-r] src_path dest_path 复制文件
- mv src_path dest_path 剪切/重命名
- rm [-r] src_path
- cat src_path 显示文件的全部内容
- less src_path 按页查看
- head -3 src_path 查看文件的前3行 (默认10行)
- tail -3 src_path 产看文件的后3行(默认10行)
- grep "root" path 过滤文件

    - -n:显示行号
    - -i:忽略大小写
    - -v:反向筛选
    - -w:精确查找
- evince .pdf
- wc file 统计文件字节个数，单词个数 行数
    - -c 字节
    - -w 单词
    - -l 行

用户管理

- root有权限
- sudo 以root身份执行
- sudo useradd -m username 创建用户username
- sudo passwd username 给username用户创建密码
- sudo userdel -r username 删除用户
- su - username 切换用户并切换工作路径到用户的家目录
- exit 登出
- /etc/passwd--->用户信息 man 5 passwd
- /etc/group --->组信息 man 5 group
- /etc/shadow--->密码 man 5 shadow
- sudo gpasswd -a username group 将用户加入组内
- sudo gpasswd -d username group 将用户从组内移除
- id username 查看用户信息

文件查找

- which / wheris 查找可执行文件路径
- locate filename 查找数据库中文件
- find path -name/-type/-size/-mtime/-perm xx
    - -type:
        - f / d/ s/ l/ c/ b/ p
    - -size
        - c / K / M / G
    - -exec 执行
        - find path -name "hello" -exec cp {} . \;
    - -ok 询问执行

文件打包压缩

- 打包
    - tar -cf xxx.tar file1 dir
        - -c 创建包
        - -f 后面接包名
        - -t 查看包中内容
        - -r 追加新文件
- 压缩

    - gzip filepath gz格式
    - bzip2 filepath bz2格式
    - xz filepath xz格式
- 解压缩
    - gzip -d filepath / gunzip filepath
    - bzip2 -d filepath / bunzip2 filepath
    - xz -d filepath
- 打包并压缩
    - gz
        - tar -czvf xxx.tar.gz file dir
    - bz2
        - tar -cjvf xxx.tar.gz file dir
    - xz
        - tar -cJvf xxx.tar.gz file dir
- 解压缩包
    - tar -xvf xxx -C path
        - -C 指定解压路径

软件包管理

- 源中软件包 （软件和更新 选择源）
    - sudo apt install 包名 下载并安装
    - sudo apt remove 包名 卸载
    - sudo apt purge 包名 卸载包含配置文件
    - sudo apt update 更新
    - sudo apt upgrade 升级
    - sudo apt list 列出所有软件包
- 下载xxx.deb--->二进制
    - sudo dpkg -i xxx.deb 安装
    - sudo dpkg -r xxx 卸载
- 源码包
    - 解压解包
        - tar -xvf xxx.tar.bz2 -C xxx
    - 查看INSTALL/README
    - ./configue
    - make
    - make install

文件重定向和管道

- 标准输入 stdin 0
- 标准输出 stdout 1
- 标准错误输出 stderr 2
- 0< 或 < 标准输入重定向
- 1> 或 > 标准输出重定向
- 2> 标准错误输出重定向

- $>> file$追加
- 写管道|读管道　管道符

网络管理与进程管理

- ifconfig 显示所有网卡信息
  - ip mask
- ping ip / 域名 查看网络是否联通
- 启动网络服务
  - /etc/init.d/networking start / restart/ stop
- 防火墙
  - /etc/init.d/ufw stop/start/restart
- 静态显示进程
  - ps aux
    - USER PID TTY STATE COMMAND
  - ps axj
    - PPID PID PGID SID TTY STATE COMMAND
  - ps axm -L
    - 线程
- 动态显示进程
  - top

vim使用

- 一般模式(打开编辑文件时)
  - yy 复制
  - 3yy 复制3行
  - p 粘贴
  - P 向上粘贴
  - dd 剪切
  - 3dd 剪切3行
  - u 撤销
  - ctrl+r
  - x 删除光标所在字符（向后）
  - X 删除光标所在字符（向前）
  - D 删除光标所在字符及以后
  - r 替换
  - R 替换模式
  - ctrl+v 块操作
  - gg 首行
  - G 末行
  - /key
- 插入模式
  - 进入插入模式
    - i 光标所在字符前
    - I 光标所在行行首
    - a 光标所在字符后

        - A 光标所在行行末
        - o 光标所在行下一行
        - O 光标所在行上一行
        - s 删除光标所在字符并插入
        - S 删除光标所在行并插入
- 命令行模式
    - 在一般模式 :
    - :w 保存
    - :q 退出
    - :a 所有
    - :q! 强制退出
    - :1,5s/hello/world/g 所有1到5行中的hello替换为world g全部替换
    - :%s/old/new/g %整个文件
    - :5,$s/old/new/g $最后
    - :set nu 行号
    - :set ts=4 设置tab宽度
    - :set mouse=a
    - :set sw=4
    - :set cindent c风格缩进
    - 永久配置vim---》~/.bashrc
    - :vsp newfile 垂直分屏
    - :sp newfile 水平分屏
- vim -p file1 file 分页打开多文件
- vim -O file1 file2 垂直分屏打开
- vim -o file1 file2 水平分屏打开

git 基本使用

- 分布式版本控制---》linus
- 初始化仓库
    - 本地 git init
    - 远程git clone http:xxxx
- 工作区
    - git工作目录
- 暂存区
    - git add filename
- 版本库
    - git commit -m "解释说明"
- 退回上一个版本
    - git reset --hard HEAD^
- 退回指定版本
    - git log 查看版本号
    - git reset --hard GPL(版本号)
- 回到新版本
    - git reflog
    - git reset --hard GPL(版本号)
- 暂存区中的文件退回工作区
    - git rm --cached file

- 已经提交到版本库中的修改执行删除 rm
    - 后悔删除
        - git checkout -- file
    - 删除版本库中文件
        - git rm file
- 提交到远程仓库
    - git push
- 从远程仓库拉最近
    - git pull
- 提交跳过密码校验
    - git config --global credential.helper store

c语言

主要内容

1. 基本数据类型
2. 变量与常量
3. 运算符和表达式
4. 读入和输出
5. 控制流
    1. 控制语句
    2. 循环语句
6. 函数
    1. 函数的组成
    2. 变量的作用域和生存周期--》c存储空间布局
    3. 特殊函数
    4. 宏和预处理指令
7. 数组
    1. 一维数组
    2. 二维数组
    3. 字符数组及字符串
8. 指针
    1. 一级指针
    2. 多级指针
    3. 堆空间的开辟和释放
9. 高级函数数组指针
    1. 数组指针
    2. 指针数组
    3. 函数指针
    4. 函数指针数组
10. 复合类型
    1. 结构体
    2. 共用体
11. 标准io
    1. 打开

2. 读文件
3. 写文件
4. 设置文件读写位置
5. 关闭文件

第一个c程序

```
#include <stdio.h>
```

#开头的指令---》预处理指令

main入口函数

```
int main(void)
{
    正确缩进;
    return 0;
}
```

printf(3);向终端输出 "原样输出"-->变量格式占位符

格式占位符

- %d int
- %c char
- %ld long
- %s
- %o 八进制
- %x 十六进制

编译: gcc

- 预处理　gcc -E hello.c -o hello.i
  - 将文件中所有的#开头的
  - #include 包含头文件（函数的声明，类型的定义）
- 编译 gcc -S hello.i -o hello.s
  - 将c程序翻译位汇编
- 汇编 gcc -c hello.s -o hello.o
  - 生成机器码
- 链接 gcc hello.o -o hello
  - 指定动态链接的地址,生成可执行文件

c基本类型

变量：值可以随时改变

常量：只读

类型:

- 整型
  - short 2
  - int   4
  - long   8

    - long long 8
- 实型
    - float 4
    - double 8
    - long double 16
- 字符类型
    - char 1
        - ''--->man ascii
- 空类型
    - void
- 有符号和无符号
    - signed(默认)
    - unsigned

变量定义

- type name;
- 标识符:变量名，函数名，类型名
    - 命名规范:由字母、数字、下划线组成，数字不开头,避开c关键字，区分大小写
    - "顾名思义"

初始化

- type name = value; // 函数内变量未初始化，值是随机

运算符

- 单目
    - () + - ~ ! (type) & * [] -> . sizeof
- 算术运算符
    - +， -， *， /， %
- 位运算符
    - << >> & | ^ ~
- 关系运算符
    - == != >= <= > <
- 逻辑运算符
    - && | | !
- 条件运算符
    - ?:
- 赋值运算符
    - = += -= *= /= &= |= %=
- 逗号运算符
    - ,
- 自增自减运算符
    - ++ --

控制语句

条件语句

- if

```
if (变量/表达式) {
    // 循环体多条语句，一定要由{}
}
```

- if ... else

```
if (变量/表达式) {
    // 条件为真
} else {
    // 条件为假
}
```

- if... else if ....else if... else
- ```
  if (变量/表达式) {

  } else if (变量/表达式) {

  } else {

  }
  ```

- switch
- ```
  switch(变量/常量表达式) {
      case 常量值:
          break;
      case 常量值:
          break;
      default:
          break;
  }
  ```

循环语句

- for
- ```
  for (表达式1; 表达式2; 表达式3) {
      循环体语句4;
  }
  表达式1:循环变量初始化,只有进入循环执行一次
  表达式2:循环条件
  表达式3:循环变量的改变
  循环:1 2 4 3 2 4 3
  都可缺省
  ```

- while
- ```
  while (条件) {
      循环体语句;
  }
  ```

- do .. while
- ```
  do {
      循环体;
  }while(条件);
  多用于错误校验
  ```
- continue和break
  - continue 终止本次循环，继续下一次
  - break 终止最近循环
- goto 无条件跳转
  - 标号ERROR:

函数

基本组成

- 定义:返回值类型 函数名(参数列表)
  - 函数名---》标识符 顾名思义
  - 返回值类型--》默认是int
  - 如果有返回值，return
- 声明
  - 返回值类型 函数名(参数列表类型);
- 调用
  - 函数名(参数)
  - 函数的传参---》值传递过程

变量类型

- 局部变量:在函数体内定义的
  - 作用域与生存周期:从定义开始到函数结束
  - 未初始化值随机
- 全局变量:在函数体外定义的
  - 作用域和生存周期:整个文件
  - 未初始化值0
- 修饰变量关键字
  - 全局变量--->extern(默认) static
  - 局部变量---->auto(默认) static const register volatile
    - static:
      - 修饰全局变量，限制其作用域在本文件中
      - 修饰函数，限制作用域在本文件中调用
      - 修饰局部变量，局部静态变量，延长生命周期到整个文件,但作用域不变
        - 只初始化一次
        - 未初始化值未0
    - const
      - read-only 变量
    - register
      - 寄存器变量
  ```

        - volatile
            - 易失变量：防止编译优化
        - extern
            - 外部变量

c存储空间布局

- 栈
    - 局部变量，函数的形参
    - 未初始化，值随机值
    - 生命周期随着函数调用的结束而结束
- 堆
    - 用户自行申请和释放
- bss
    - 未初始化的局部静态变量和全局变量
    - 初始值为0
- 数据段
    - 已经初始化的局部静态和全局变量
    - 生命周期都是从定义开始到进程结束
- 文本段

特殊的函数

- 递归函数
    - 在函数体内调用函数本身
        - 找到递归的终止条件
        - 找到递归点
    - 递归用于解决复杂问题
        - 汉诺塔
        - 迷宫
        - 树
- 变参函数
    - ...
    - printf(3) / scanf(3)

宏

- #define NAME value
- 宏在gcc预处理阶段纯替换
- 宏函数
    - 参数加括号
    - 末尾无分号
    - 建议do {}while(0)
    - 必须一行
    - 太长续行
    - #x 将x替换为字符串
    - '##' 连接
- #if 常量 / 常量表达式 #endif
- #ifdef 常量 #endif

- #ifndef xxx #endif

数组

基本知识

1. 定义：相同类型元素的集合
    - type name[n];
2. 赋值
    - 每一个成员变量通过下标值，从0开始
    - for (i = 0; i < n ; i++)
3. 初始化
    - type name[n] = {}; 全部初始化为0
    - type name[n] = {1}; 第一个元素为1，其他为0
    - type name[] = {1,2,3}; 成员个数可缺省，大小取决于给定的值

数组名

- 数组首地址---》首元素的地址
- 常量值
- type *
- 类型决定了步长+取值字节大小(sizeof(type))

运算符

- a[b]--->*(a+b)--->b[a]
- *
- 取数组中的元素
    - name[index]
    - *(name+index)
- 元素地址
    - name + index
    - &name[index];// &*(name+i)
    - *&抵消
    - 数组索引值[0,n-1],一定不要越界！！！！

排序

1. 冒泡排序
    - 相邻元素两两比较,不符合大小顺序则交换
    - 每比较一整趟，使得一个元素有序。如果n元素，则比较n-1趟
        - for (i = 0; i < n-1; i++)
    - 每一趟比较的都是无序序列中的所有元素
        - for (j = 0; j < n-i-1; j++)
2. 选择
    - 依次选择待排序的位置，该位置的元素与所有无序序列中的而元素一一比较，不符合大小关系则交换
    - 选
        - for (i = 0; i < n-1; i++)

- 比
  - for (j = i+1; j < n; j++)

3. 直接插入排序
   - 将所有元素分有序区和无序区，从无序序列中依次选择每一个元素，向有序序列中插入
   - 无序区中选带插入的元素
     - for (i = 1; i < n; i++)
   - 有序区待比较的元素
     - for (j = i-1; j >= 0; j --)

查找

有序数组---》二分查找(折半查找)

产生随机数

rand(3)

srand(3)

提供种子：time(2)---》获取时间戳 / getpid(2)

字符数组和字符串

c语言中没有字符串类型，用字符数组来存储字符串

1. 字符串:多个字符组成并由'\0'作为结束标志
2. 字符串初始化数组
   - char str[3] = "hi";
   - char str[] = "hi";
   - char str[3] = {'h', 'i', '\0'};

多维数组

二维数组：理解为多个一维数组组成

1. 定义

```
type arr[一维数组个数][每一个一维数组的成员个数];
int arr[2][3];由两个一维数组组成，每一个一维数组中有3个整形变量组成
arr--->int[3] *
arr[0]-->int *
```

2. 初始化

```
int arr[2][3] = {{1,2,3}, {4,5,6}};
```

指针

存储地址的变量

定义

1. type *name;

类型

1. 步长 取值大小
2. 存地址---》变量大小8字节

用途

1. 形参通过得到实参地址---》改变实参
2. 可以通过参数返回值---》回填
3. 数组，函数可以作为形参

二级指针

```
1. 存储一级指针变量的地址
```

2. 用途
    1. 存储指针数组首地址
    2. 参数列表返回地址(堆)

数组指针

1. int (*p)[3]; 存储的3个地址连续整型数的地址
2. ```
   int arr[2][3];
   p = arr;
   ```

函数指针

1. 存储函数地址(函数名)
2. 返回值类型 (*p)(参数列表类型);

动态开辟

1. malloc(3)
2. calloc(3) 空间初始化为0
3. realloc(3) 再开辟
4. free(3)
5. 谁申请谁释放

指针数组

```
int main(int argc, char *argv[])
argv就是命令行每一个参数存储所在指针数组的首地址
char *arr[3];
```

typedef

1. 定义类型
    1. typedef int a;
    2. typedef int *p;
2. 与define差别
```

1. typedef 定义新类型
2. define 纯替换

重点函数

1. 字符串操作函数
    1. strlen(3)
    2. strncpy(3)
    3. strncat(3)
    4. strstr(3)
    5. strchr(3)
    6. strtok(3) / strsep(3)
    7. getline(3)
2. 存储空间
    1. memcpy(3)
    2. memmove(3)
    3. memset(3) / bzero(3)

数据结构

集合

线性

元素和元素之间一对一的对应关系

- 地址连续-->顺序存储结构-->顺序表
- 地址不连续--->指针维系一对一关系---》链表
    - 一个指针---》单链表
    - 两个指针---》双链表
        - 内核链表
- 特殊线性结构
    - 再表同一端插入删除--->栈
        - 插入删除的那一端称栈顶，插入-->入栈，删除-->出栈
        - LIFO
        - 应用
            - 汉诺塔
            - 进制间转换
            - 表达式转换--->逆波兰序
    - 在表一端插入，另一端删除--->队列
        - 插入--》队尾 --->入队 删除--->队头--->出队
        - FIFO
        - 应用
            - 排队系统
            - 球钟问题

动态库

- gcc -fPIC -shared -o libxxxx.so xxx.c
- gcc -o a.out main.c -lxxxx -Lpath
- ldd a.out 查看所有链接库的地址
- vim ~/.bashrc将动态库所在路径加入
  - LD_LIBRARY_PATH=$(LD_LIBRARY_PATH):
  - export LD_LIBRARY_PATH
- 或写入/etc/ld.so.conf
- source ~/.bashrc

静态库

- gcc -c -o xxx.o xxx.c
- ar -cr libxxx.a xxx.o
- gcc -o a.out main.c -lxxx -Lpath

makefile

工程管理

- vim makefile/Makefile
- $(NAME) 取变量
- $@ 取目标
- $^ 取所有依赖文件
- LDFLAGS-->指定动态库
- CFLAGS--->头文件路径

树型

1. 基本概念
   1. 结点、树 结点的度 树的度 树的深度 叶子结点（终端结点） 根
   2. 二叉树
2. 二叉树的遍历
   1. 先序 根--》左--》右
   2. 中序 左--》根--》右
   3. 后序 左--》右--》根
3. 根据给定先+中 或中+后确定唯一二叉树
4. 二叉排序树实现
   1. 增删改查
5. 平衡二叉排序树

图型

unix环境高级编程

io

标准io

1. 文件流
    1. FILE(pos, 缓存区，出错状态，文件描述符)
    2. 文件打开标志
    2. fopen(3)--->FILE *
    3. 打开方式
    - r
    - r+
    - w
    - w+
    - a
    - a+
    3. 读写
    4. 字节
    - fgetc(3) / getc(3) / getchar(3)
    - fputc(3) / putc(3) / putchar(3)
    2. 行
    - fgets(3)
    - fputs(3)
    3. 二进制
    - fread(3)
    - fwrite(3)
    4. 格式化
    - scanf(3)
    - printf(3) / fprintf(3) / snprintf(3)
    4. 缓存区
    5. 标准io在文件打开的时候创建缓存区
        2. 缓存类型
    - 行缓存:stdin stdout
    - 无缓存:stderr
    - 全缓存(4096bytes)
    3. 刷新
    - 行缓存遇到'\n'
    - fflush(3)
    - fclose(3)
    - 进程结束
    5. 定位流
    6. fseek(3) / ftell(3) / rewind(3)
    7. 关闭
    8. fclose(3)

文件io

1. 文件描述符
    1. 非负整型数
    2. 文件打开标志
    3. 使用当前可用最小
2. 基本函数
    1. open(2)
        1. flags打开方式
        - O_RDONLY/O_WRONLY/O_RDWR
        - O_CREAT

        - O_TRUNC
        - O_APPEND
    2. mode--->O_CREAT
    3. 返回值
  2. errno
    1. 全局变量
    2. 值---》出错原因字符串
      1. strerror(3)
      2. perror(3)
  3. read(2) / write(2)
  4. close(2)
  5. lseek(2)
3. 文件共享
  1. 进程表
  2. 文件表
  3. v结点表
4. dup2(2) / dup(2)
5. 原子操作
  1. 功能由一个接口函数完成的

文件和目录

1. stat(2) / lstat(2) / fstat(2)
  1. 文件类型（st_mode & S_IFMT）
    - -普通文件
    - d目录文件
    - l符号链接文件
    - s套接字文件
    - c字符设备文件
    - b块设备文件
    - p管道文件
  2. 权限(st_mode)
    1. 拥有者
    2. 所属组
    3. 其他用户
    4. r w x u+s g+s o+t
  3. 硬链接(st_nlink)
    1. 同一个文件多个名字
    2. 软连接(符号链接)--->快捷方式
  4. 拥有者(st_uid)和所属组(st_gid)
    1. uid---->uname(/etc/passwd)
      - man 5 passwd
      - getpwuid(3)
    2. gid--->gname(/etc/group)
      - man 5 group
      - getgrgid(3)

5. 字节个数(st_size)!=磁盘空间大小(st_blocks)
6. 文件时间
    1. atime（st_atim）
    2. ctime(属性)(st_ctim)
    3. mtime(内容)(st_mtim)
    4. 时间类型转换
        - time(2)(time_t)
        - locatime(struct tm)
        - strftime(3)
        - mktime(3)

2. 密码校验过程
    1. /etc/shadow--->man 5 shadow
    2. crypt(3)
    3. getpass(3)
    4. getpwnam(3)
3. 读目录
    1. 目录流----》已打开目录标志
    2. opendir(3)--->readdir(3)--->closedir(3)
    3. glob(3)
    4. getopt(3) / getopt_long(3)
        - optring:
            - "-"开头，识别非选项参数
            - "c"识别选项-c
            - "h:"带参数选项 optarg指向参数
            - "w::"可选择参数
        - 返回值
            - 选项字符
            - -1 没有选项了
            - '?'未识别选项
            - 1非选项参数

并发

进程

1. 进程环境
    - c存储空间布局
        - .stack
        - .heap
        - .data
        - .bss
        - .text
    - 环境表(char **eviron)
        - key=value
        - getenv(3) / setenv(3)
    - 进程表项

- 资源限制
  - ulimit -a
  - getrlimit(2)
- 跨函数跳转
  - setjmp(3) / longjmp(3)

2. 进程控制
   - ps aux / ps axj / ps axm -L / ps axf
   - 进程状态
     - S
     - D
     - T
     - R
     - Z
   - 优先级
     - < high-priority (not nice to other users)
     - N low-priority (nice to other users)
     - L has pages locked into memory (for real-time andcustom IO)
     - s is a session leader
     - l is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
     - +is in the foreground process group
   - 进程标识pid_t
     - getpid(2)
     - getppid(2)
   - 创建
     - fork(2)--->复制父进程
     - vfork(2)
       - 阻塞调用进程
       - 共享内存空间
   - 应用
     - shell
       - fork--->父wait(2) 子exec(3)
     - 网络
   - 终止
     - main return
     - exit(3)
     - _exit(2) / _Exit(2)
     - 最后一个线程从启动例程返回
     - 最后一个线程调用pthread_exit(3)
     - abort(3)
     - 接受到一个信号
     - 最后一个线程对取消做出响应
   - 进程如何运行起来和如何终止(图)
   - 收尸
     - wait(2)
     - waitpid(2)

        - pid < -1
        - pid == -1
        - pid == 0
        - pid > 0
    - 进程替换
        - exec(3)
        - execl(3) / execlp(3) / execv(3) / execvp(3)
    - system(3)

3. 进程关系

- 会话（session）承载进程组(process group)
- 进程组承载进程
- 进程承载线程
- getsid(2) / setsid(2)
- getpgid(2) / setpgid(2)

4. 守护进程
    - PIG == PGID == SID 脱离控制终端
    - 编程规则
        - umask(2)--->0
        - fork(2)--->父exit(0) 子setsid(2)
        - chdir("/")
        - 0, 1, 2 > "/dev/null"
    - 日志
        - syslog
        - openlog(3)
        - syslog(3)
        - closelog(3)
        - "/var/log/syslog"
    - 单实例守护进程
        - 同时只能运行一次
        - "/var/run/xxx.pid"--->flock(2) / lockf(3)
    - daemon(3)
5. 进程间通信

信号

1. 信号的类型
    - 标准信号(1~31)
    - 实时信号(34~64)
2. 查看所有信号
    - kill -l
    - man 7 signal
3. 信号的默认行为
    - Term
    - Core
    - Ign
    - Stop
    - Cont
4. 信号的行为注册

- signal(2)
    - SIGKILL和SIGTOP不能改变

5. 产生信号
    - Ctrl+c --->SIGINT Ctrl+\ --->SIGQUIT Ctrl+z --->SIGSTOP
    - kill(2) / alarm(2)
    - 程序出错---》段错误
6. 信号特点
    - 丢失---》显示信号结构是位图
    - 打断阻塞的系统调用
    - 响应过程是嵌套的
    - 忽略信号的本质mask=1
    - 信号处理函数中不能用longjmp跳转---》信号处理函数调用结束将mask置0
    - 信号处理函数使用可重入函数
7. 信号集
    - sigset_t
    - sigemptyset(2)
    - sigaddset(2)
    - sigdelset(2)
    - sigfillset(2)
    - sigismember(2)
8. 设置信号屏蔽字(mask位图)
    - sigprocmask(2)
        - SIG_BLOCK
        - SIG_UNBLOCK
        - SIG_SETMASK
    - sigsuspend(2)--->信号驱动程序运行
        - sigprocmask(SIG_MASK, &set, &oldset)
        - pause(2)
        - sigprocmask(SIG_MASK, &oldset, NULL)
9. sigaction(2)!!!!!!
    - struct sigaction
        - sa_handler
            - SIG_DFL
            - SIG_IGN
            - 函数
        - sa_flags
            - SA_SIGINFO--->sa_sigaction信号处理函数
            - SA_NOCHLDWAIT 子进程终止后不是僵尸进程
        - sa_mask
10. setitimer(2)
    - TIMER_REAL--->SIGALRM
    - 指定时间间隔产生信号
    - 最小时间单位微秒
11. 流量控制
    - 漏桶
    - 令牌桶

线程

1. 线程标识
    - pthread_t
    - pthread_self(3) 获取调用线程线程标识
    - pthread_equal(3) 比较线程标识是否一致
2. 创建
    - pthread_create(3)
3. 终止
    - return
    - pthread_exit(3)
4. 收尸
    - pthread_join(3)
    - 默认属性 joined
    - pthread_detach(3)--->分离则不能pthread_join(3)
5. 取消
    - pthread_cancel(3)
        - whether--->state enabled
            - pthread_setcancelstate(3)
        - when--->type deferred
            - pthread_setcanceltype(3)
            - 取消点 man 7 pthreads
6. 同步
    - 临界区:多线程竞争的代码
    - 互斥量
        - pthread_mutex_t
        - pthread_mutex_init(3)
        - pthread_mutex_lock(3)
        - pthread_mutex_unlock(3)
        - pthread_mutex_destroy(3)
    - 多线程竞争的条件有变化的情况下----》条件变量
        - pthread_cond_t
        - pthread_cond_init(3)
        - pthread_cond_wait(3)
        - pthread_cond_signal(3)
        - pthread_cond_broadcast(3)
        - pthread_cond_destroy(3)
7. 线程和信号
    - sigwait(3) // 等待指定信号集中信号的到来
    - pthread_sigmask(3) // 线程有自己的mask位图
    - 多线程mask位图独立的，信号的行为共享的，顾减少多线程+信号
8. 线程和fork
    - pthread_atfork(3)
9. 线程和io
    - pread(3) / pwrite(3)

网络编程

1. ipv4地址的划分
    - A 0~127 内网（10.0.0.0~10.255.255.255）
    - B 128~191 内网（172.16~172.31）
    - C 192~223 内网（192.168）
    - D 224~239 组播地址
    - E 保留
    - 0.0.0.0---->本地所有网卡地址
    - 127.0.0.1 ---->环回测试地址
    - 255.255.255.255 广播地址
    - 点分十进制"192.168.1.10"---->uint32_t inet_aton(3) / inet_ntoa(3)
2. 端口(port)
    - 运行服务的标识
    - 0~65535（0~1024周知端口）
3. 字节序
    - 本地字节序(htons(3))
    - 网络字节序(ntohs(3))
4. 网络模型
    - OSI 7层网络模型
        - 应用层
        - 表示层
        - 会话层
        - 传输层
        - 网络层
        - 数据链路层
        - 物理层
    - TCP/IP 四层网络模型
        - 应用
        - 传输
        - 网络
        - 链路
5. 套接字
    - man 7 socket
    - man 7 ip
    - TCP
        - socket(AF_INET, SOCK_STREAM, 0);
        - 基于连接的，可靠传输，有序的
        - server
            - socket(2)
            - bind(2)
            - listen(2)
            - accept(2)
            - read(2) / write(2) recv(2) / send(2)
            - close(2)
        - client
            - socket(2)
            - connect(2)
            - read(2) / write(2) recv(2) / send(2)

        - close(2)
    - tcp三次握手(连接)
        - client向server发送syn序列号，假定为y，client处于SYN_SENT
        - server接收到client的syn之后，向client端发送ack=y+1,syn=x，server处于SYN_RECV
        - client接收到后，发送server应答数据包ack=x+1，处于ESTABLISHED
    - tcp四次挥手(连接断开)
        - client向server发送fin断开序列号，假定为z
        - server收到后立即向client发送ack=z+1
        - server向client发送fin=m
        - client收到后向server发送ack=m+1
- UDP
    - socket(AF_INET, SOCK_DGRAM, 0);
    - 无连接，不可靠
    - 优势，效率高，多应用于视频，音频
    - server
        - socket(2)
        - bind(2)
        - recvfrom(2) / sendto(2)
        - close(2)
    - client
        - socket(2)
        - sendto(2) / recvfrom(2)
        - close(2)
    - 组播/ 多播(man 7 ip)
        - setsockopt(2)
        - server
            - setsockopt(sd, IPPROTO_IP，IP_MULTICAST_IF, struct ip_mreqn,)
        - client
            - setsockopt(sd, IPPROTO_IP，IP_ADD_MEMBERSHIP, struct ip_mreqn,)
    - 广播(man 7 socket)
        - setsockopt(sd, SOL_SOCKET, SO_BROADCAST, val=1,);
- UNIX
    - AF_UNIX / AF_LOCAL
    - struc sockaddr_un
    - socketpair(2)

作业

2020.10.26

1. 读入10个成绩，判断由多少个不及格的
2. 读入输入的月份，判断属于哪一个季节
3. 读入用户输入的一个整型数，判断是否为质数
4. 读入一个用户输入的整型数，将这个整型数的第3位置1，将这个数的第5位清0，其他位不变

5. 打印乘法口诀
6. 求得Fibonacci数列的前10项(前两项位1，从第三项开始每一项是前两项的和)
7. 打印一下图形

```
    *
  * * *
* * * * *

* * * *
  * * * *
    * * * *

A
A B
A B C
A B C D
```

#

2020.10.27

1. 定义以下接口函数：
    1. 计算一个整型数的阶乘
    2. 判断是否是质数
    3. 判断是否是一个水仙花数（每一位的立方和等于数值本身）
    4. 比较两个整型变量的大小关系
    5. 求给定整型数的逆序数 例如：123--》321
2. 将给定整型数中的偶数位累加，将结果值返回
3. 求出两个整型的最大公约数，和最小公倍数。

温馨提示，不要过多纠结数学问题，概念上网查！！！！！

2020.10.28

1. 输入10个数，存储到数组中，分别统计其中正数、负数、零的个数。
2. 定义一个整型数组int arr[10],读入用户输入的10个整型数，赋值给数组，
   求这个数组的最大值和最小值以及平均值。
3. 初始化一个整型变量int arr[10], 值随机。给这个数组从小到大排序。

2020.10.28

1. 读入20个学生的c成绩，显示成绩分布图
   例如:
   100: *
   90~99:* * * *
   80~89:* * *
   70~79:* * * * * *
   .....
   0~9: *

一个星表示一个学生

2. 定义一个字符数组，有100个成员，从终端读入一个字符串(%s),将字符串转置（倒叙）并输出
3. 定义一个能容纳100个元素的字符数组，从终端读入若干字符，统计大写字母的个数，小写字母的个数以及数字字符的个数，并将大写转小写，小写转大写后输出该字符串
4. 用多维数组显示一下图形(结合vt控制码)

   []

   [] [] []

   [] [] []

   []

   [] []

   [] []

   []

   []

   []

   []

2020.11.23

1. 实现将argv[1]文件中按单词输出到标准输出
2. 统计argv[1]文件中有多少argv[2]字符串
3. 将argv[1]文件倒序写入argv[2]

2020.11.24

1. 用文件io完成11.23作业3
2. 将argv[1]文件的第argv[2]行行首写入argv[3]
3. ./mycat filepath 如果argv[2]不为空，则将argv[1]文件写入argv[2]，为空写标准输出

2020.12.3

1. 用信号实现流量控制功能，将argv[1]文件按照指定的流量写到终端，1s中写10个字节