1.

Ctrl-a 回到第一个光标

Ctrl-e 到最后

Ctrl-u 删除所有输入

上 Ctrl-p 上一条指令

下 Ctrl-n 下一条指令

Ctrl-c 终止进程

连续按两次gg回到开头

按G回到末尾

2.

sudo apt-get update 更新软件资源到本地

sudo apt-get install “软件” 安装命令

sudo apt-get remove “软件” 卸载命令

Ubuntu安装包以.deb为结尾 （使用软件包安装 sudo dpkg -i 安装包名）

tar压缩：

tar zcvf 要生成的压缩包名 压缩材料

（tar zcvf test.tar.gz file1 dir2使用gzip方式压缩

tar jcvf test.tar.gz file1 dir2使用bzip2方式压缩

）

tar解压缩：

tar zxvf test.tar.gz使用gzip方式解压缩

tar jxvf test.tar.gz使用bzip2方式解压缩）

rar压缩：

rar a -r 要生成的压缩包名 压缩材料

(

rar a -r test.rar file1 dir2使用rar方式压缩

)

rar 解压：

unrar x rartest.rar

zip压缩：

zip -r testzip.zip file1 dir2

zip解压缩：

unzip testzip.zip

./当前目录

cd ~回到用户目录

history 查看历史命令

pwd 查看当前目录

exit 退出用户

mkdir “文件夹名” 创建文件夹

rmdir 删除空文件夹

rm –rf 强制删除文件

cp 文件/目录 文件/目录 将文件拷贝到目录里

touch file\_name(s) 创建新的空文件

which 查看命令所在的目录

cat 查看文件内容

tac 倒着查看文件内容

tree 以树状显示当前目录

mv 文件/目录 文件/目录 移动文件

whoami 查看当前用户

sudo chmod u+x a.c修改文件权限 (文字设定法)

sudo chmod 471 a.c (数字设定法)

{

u 表示用户

g 表示同组

o 表示其他用户

a 表示所有用户

r 对应4

w 对应2

x 对应1

}

sudo chown 新用户名 待修改文件 修改文件所属用户

sudo chgrp 新用户组名 待修改文件 修改文件所属用户组

sudo chown nobody:nogroup hello.cpp 一起改

find ./ -type ‘l’ 在当前目录下按文件类型找 （找文件）

find ./ -name ‘\*.jpg’ 在当前目录下按文件名字找

find ./ -maxdepth 1 -name ‘\*.jpg’ （限制层级深度为一层，应写为第一个参数）

（还有-size按大小）

grep –r ‘copy’ ./ -n（按文件内容搜索对象）

ps aux 查看当前执行的进程 (ps aux | grep ‘root’ ,两个命令结合使用)

ps ajx 参看pid/ppid/gid/sid

3

bin：存放二进制可执行文件

boot：存放开机启动程序

dev：存放设备文件

home：存放普通用户

etc：用户信息和系统配置文件

lib：库文件

root：管理员宿主目录

usr：用户资源管理目录

4.

-l 详细信息

-a 显示隐藏文件

-d 显示目录

-a 全部

-r 递归

例子：ls –l hello.cpp

5.

Linux系统文件类型：8种（真正占用磁盘空间的是普通文件/目录文件/软连接，其他文件类型称为伪文件，只占用内存空间）

普通文件：-

目录文件：d

字符设备文件：c

块设备文件：b

软连接：l

管道文件：p

套接字：s

未知文件

6.

软连接：为保证软连接可以任意搬移，创建是务必对源文件使用绝对路径

ln –s “文件目录” “软连接名字” 创建软连接也就相当于Windows快捷方式

硬连接：操作系统给每一个文件赋予唯一的inode，当有相同inode的文件存在时，彼此同步。删除时只讲硬连接数减一，减为0时inode被释放

ln “文件目录” “软连接名字” 创建硬连接 文件之间同步

6.

makefile:

命名：makefile Makefile

1个规则，

目标：依赖条件

（一个他把缩进）命令

1. 目标的时间必须晚于依赖条件的时间，否则，更新目录
2. 依赖条件如果不存在，找寻新的规则去产生依赖

ALL:指定makefile的终极目标

2个函数： ($()相当于取值)

src=$(wildcard \*.c):匹配当前工作目录下的所有.c文件。将文件名组成列表，赋值给变量src。

obj=$(patsubst %.c,%.o,$(src))：将参数3中，包含参数1的部分，替换为参数2.

clean:（没有依赖）

-rm -rf $(obj) a.out “-“：作用是，删除不存在文件时，不报错

3个自动变量：

$@:在规则的命令中，表示规则的目标

$^:在规则的命令中，表示所有依赖条件

$<:在规则的命令中，表示第一个依赖条件,如果将该变量应用在模式规则中，它可将依赖条件列表中的依赖依次取出，套用模式规则。

模式规则：

%.o:%.c

gcc –c $< –o $@

静态模式规则：

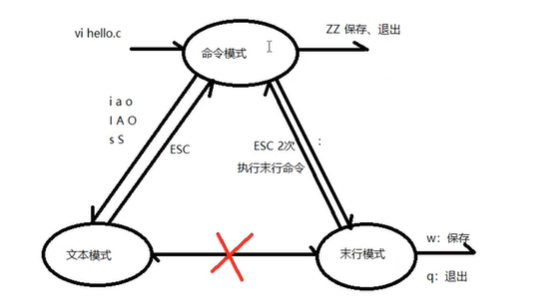
$(obj):%.o:%.c

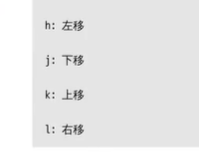
Gcc –c $< -o %@

伪目标：

.PHONY:clean ALL

7.





跳转到指定行：

1）88G(命令模式)

2）:88（末行模式）

3）连续按两次gg回到开头

4）按G回到末尾

5）自动格式化 gg=G（命令模式）

6）删除单个字符 x（命令模式）

7）删除光标至行尾D

8) 删除光标至行尾d0

9）复制一行 yy

10）粘贴 p：向后 P：向前

11）查找 1找设想内容，按“/“输入欲搜索关键字，回车，使用n检索下一个

2找看到的内容，将光标置于单词任意一个字符上，按“\*“

12）撤销 u（命令模式）

13）竖分屏 vsp（末行模式） Crtl+ww切换

8.gcc编译可以执行程序4步骤：预处理、编译。汇编。链接



-I 指定头文件目录位置

-c 只做预处理、编译、汇编 得到二进制文件

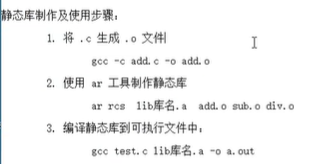
-g 编译是增加调试语句。主要支持gdb调试

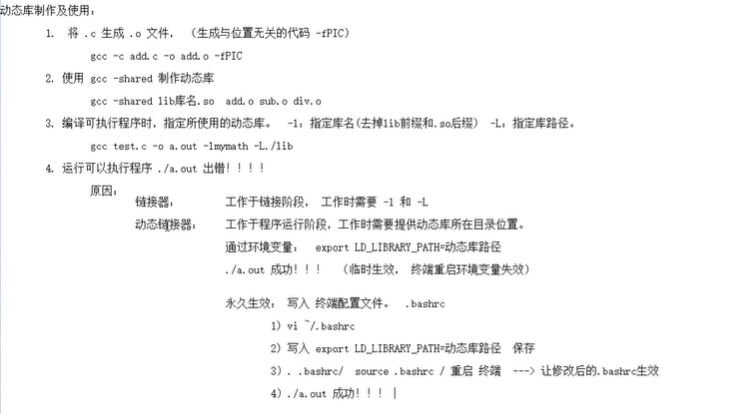
-Wall 显示所有警告信息

-D 向程序动态注册宏定义

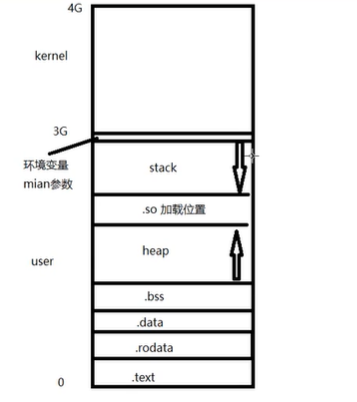
9.静态库：对空间要求较低。而对时间要求较高。动态库：对时间要求较低，对空间要求较高。

编译阶段出错有行号，链接阶段出错没有行号，且有collect2（链接器）





10.进程的内存分布



11.gdb调试工具：

基础指令：

-g：使用该参数编译可执行文件，得到调试表

gdb a.out

list: list/l 1 列出源码。根据源码指定行号设置断点

b：break/b 20 在20行处设置断点

d：delete/d 20 删除某个断点

run/r 运行程序

next/n 下一条指令（会越过函数）

step/s 下一条指令（会进入函数）

print/p p i 查看变量i的值

continue：继续执行断点后续程序

quit 退出gdb当前调试

其他指令：

run：进入gdb直接run就可以查看段错误

set args 设置main 函数命令行参数

run 参数1 参数2 设置main 函数命令行参数

start:直接从第一行开始进行调试

finsh:结束当前函数的调用

info b 查看设置的断点位置

b 20 if i = 5 ：设置条件断点

ptype arr:查看arr变量的变量类型

backtrace/bt:列出当前程序正存活的栈帧

frame/f 切换函数的栈帧

display:设置跟踪变量

undisplay：取消设置跟踪变量，使用跟踪变量的编号

使用gdb调试进程

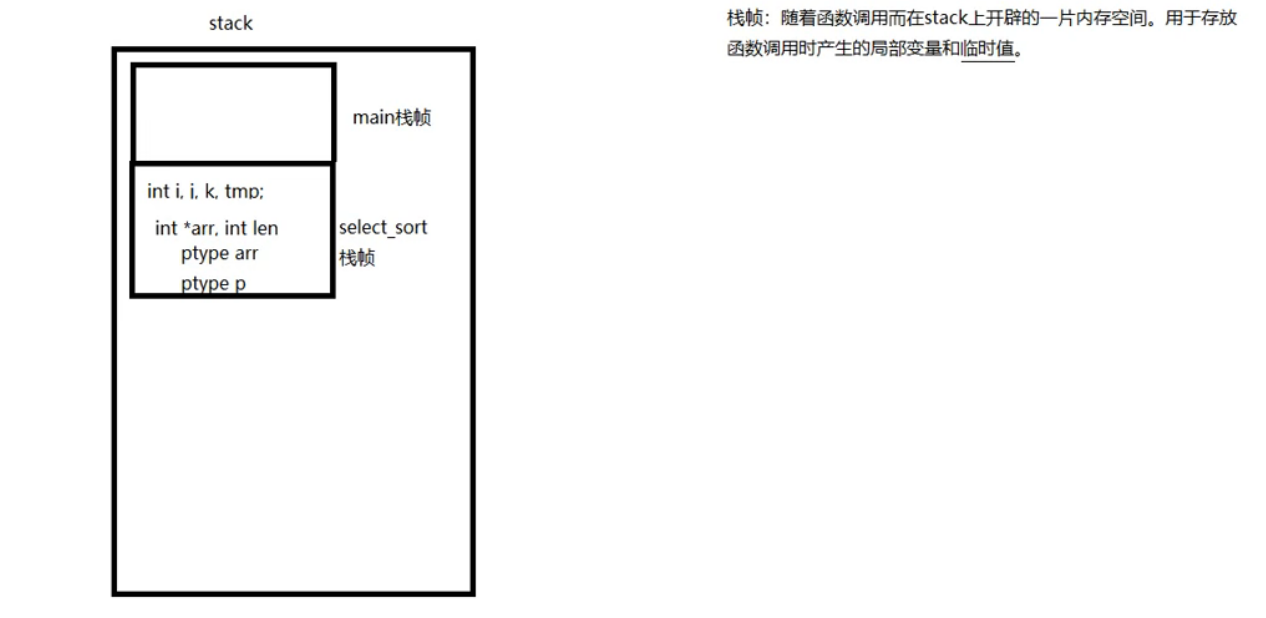
set follow-fork-mode child 命令设置gdb在fork之后跟踪子进程

set follow-fork-mode parent 设置跟踪父进程

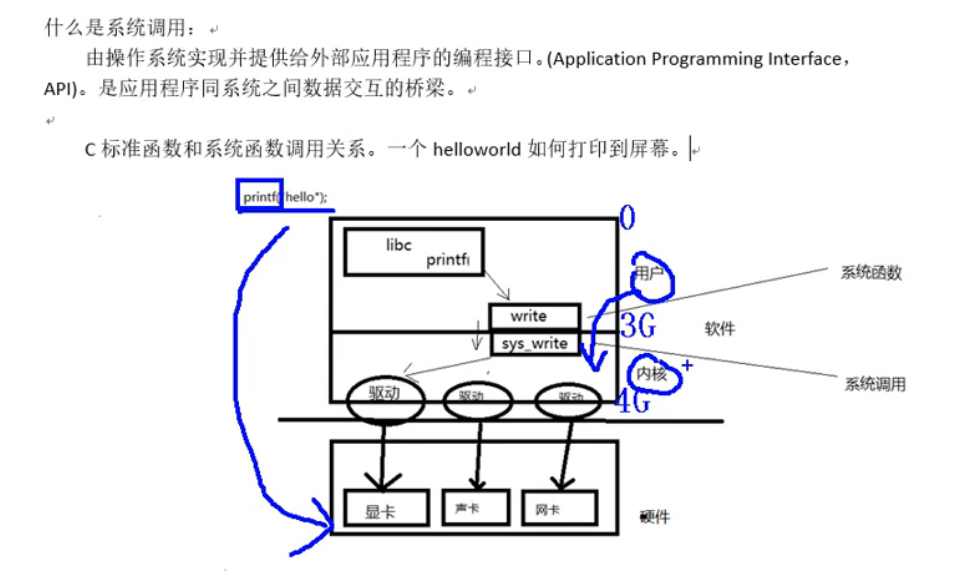
不进行设置默认跟踪父进程

注意：一定要在fork函数调用前设置才有效

12.



13.



14.文件描述符就是一个整数

open函数：

int open(char \*pathname, int flags) #include<unistd.h>

参数：

pathname:欲打开的文件路径名

flags：文件打开方式，O\_RDONLY|O\_WRONLY|O\_RDWD #include<fcntl.h>

返回值：

成功：返回打开文件所得到对应的文件描述符（整数）

失败：-1，设置errno

int open（char \*pathname, int flags, mode\_t mode）

参数：

pathname:欲打开的文件路径名

flags：文件打开方式，O\_RDONLY|O\_WRONLY|O\_RDWD

mode:参数三使用的前提是，参二指定了O\_CREAT。取值8进制数，用来描述文件的访问权限。rwx 0664

创建文件最终得到的权限= mode & ~umask

返回值：

成功：返回打开文件所得到对应的文件描述符（整数）

失败：-1，设置errno

close函数

int close(int fd)

参数：

是前面open函数返回的文件描述符

错误处理函数： 与errno相关

printf（“xxx error： %d\n”,errno）

char \*strerror(int errnum);

printf(“%s\n”,strerror(errnum))

void perror(const char \*s);

perror(“open error”);等于实现了printf(“%s\n”,strerror(errnum))

read函数：

ssize\_t read(int fd, void \*buf, size\_t count);

参数：

fd：文件描述符

buf：存数据的缓冲区

count：缓冲区大小

放回值：

0表示读到文件末尾

成功：读到的字节数

失败：-1，设置errno

-1：并且errno = EAGIN 或EWOULDBLOCK,说明不是read失败，而是read在以非阻塞方式去读一个设备文件（网络文件）并且文件无数据

write函数：

ssize\_t write(int fd, const void \*buf, size\_t count);

参数：

fd：文件描述符

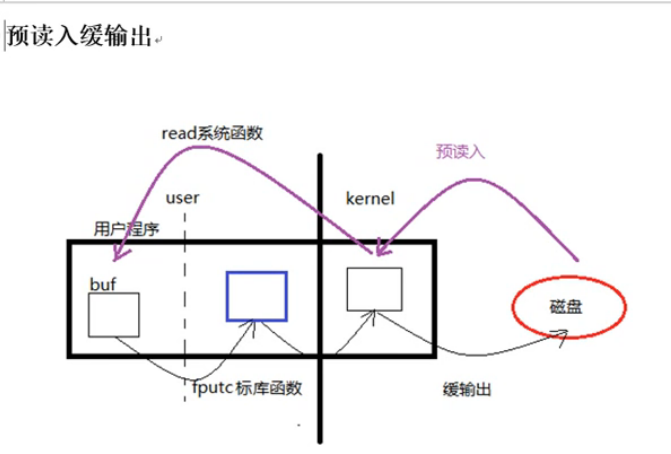
buf：待写出数据的缓冲区

count：数据大小

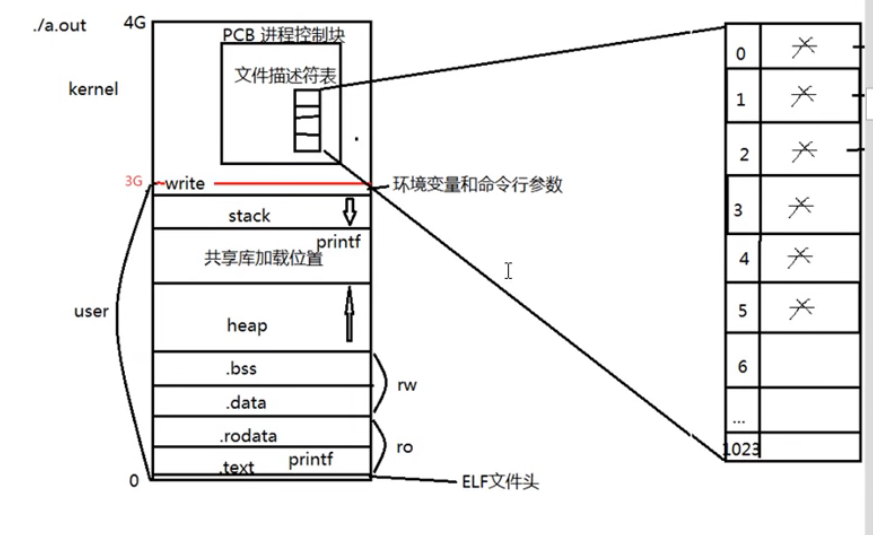
放回值：

成功：写入的字节数

失败：-1，设置errno



文件描述符：



PCB进程控制块，本质结构体

成员，文件描述符表

文件描述符：0/1/2/3/…./1024 表中可用的最小的

0 – STDIN\_FILENO

1 – STDOUT\_FILENO

2 – STDERR\_FILENO

15.

阻塞和非阻塞： 是设备文件，网络文件的属性

产生阻塞的场景。读设备文件/和读网络文件。（读常规文件无阻塞概念）

/dev/tty –-终端文件。（默认终端文件都是阻塞文件）

open(“/dev/tty”, O\_RDWR|O\_NONBLOCK) --设置/dev/tty非阻塞状态。（默认为阻塞状态）

fcntl函数:在不打开文件的状况下改变文件状态

int fcntl(int fd, int cmd, ... /\* arg \*/ );变参函数

int flags = fcntl(fd, F\_GETFL)

flags |= O\_NOBLOCK

fcntl(fd, F\_SETGL, flags)

获取文件状态：F\_GETFL

设置文件状态：F\_SETFL

lseek函数：修改文件偏移量（读写位置）

off\_t lseek(int fd, off\_t offset, int whence)

参数：

fd：文件描述符

offset：偏移量

whence：起始偏移位置： SEEK\_SET/SEEK\_CUR/SEEK\_END

返回值：

成功：较起始位置偏移量

失败：-1，设置errno

应用场景：

1.文件的读和写使用的是同一偏移位置

2.使用lseek获取。扩展文件的大小/也可使用truncate函数直接拓展

3.使用lseek函数拓展文件大小，要想是文件真正拓展，要进行IO操作（比如说使用write函数）

od -tcx filename 查看文件的16进制表示形式

od -tcd filename 查看文件的10进制表示形式

16.

传入参数：

1. 指针作为函数参数
2. 通常用const关键字修饰
3. 指针指向有效区域，在函数内部做读操作

传出参数：

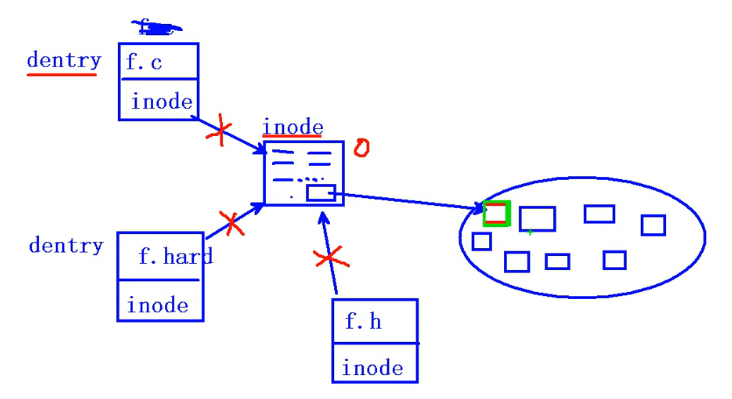
1. 指针作为函数参数
2. 在函数调用之间，指针指向的空间可以无意义，但必须有效
3. 在函数内部，做写操作
4. 函数调用结束后，充当函数返回值

传入传出参数：

1. 指针作为函数参数
2. 在函数调用之前，指针指向的空间有实际意义
3. 在函数内部，先做读操作，后写操作
4. 函数调用结束后，充当函数返回值

17.

inode其本质也是一个结构体，存储文件的属性信息



dentry:目录项

18.

stat函数：获取文件属性

int stat(const char \*pathname, struct stat \*statbuf);

参数：

pathname：文件路径

stabuf：（传出参数）存放文件属性

返回值：

成功：0

失败：-1 设置errno

获取文件大小：buf.st\_size

获取文件类型：buf.st\_mode

获取文件权限：buf.st\_mode

lstat函数：获取文件属性

int lstat(const char \*pathname, struct stat \*statbuf);

参数：

pathname：文件路径

stabuf：（传出参数）存放文件属性

返回值：

成功：0

失败：-1 设置errno

lstat 不会穿透符号连接，stat会穿透符号链接

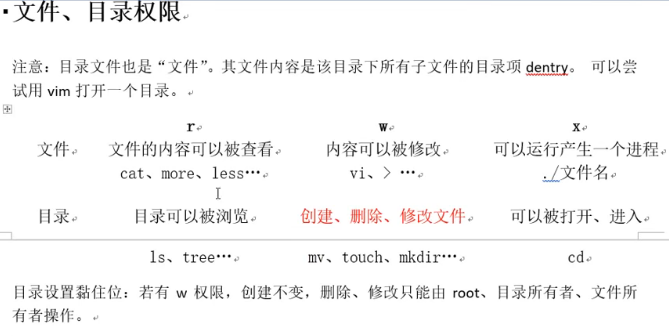
link函数：创建硬连接

unlink函数：删除硬连接（只有当打开文件的进程都关闭了，系统才会择机释放）

19.

隐式回收：当进程结束运行时，所有该进程打开的文件会被关闭，申请的内存空间会被释放。

20.



21.目录操作函数：

DIR \*opendir(char \*name)

int closedir(DIR \*dp)

struct dirent \*readdir(DIR \*dp)

struct dirent{

inode

char dname[256]

…

}

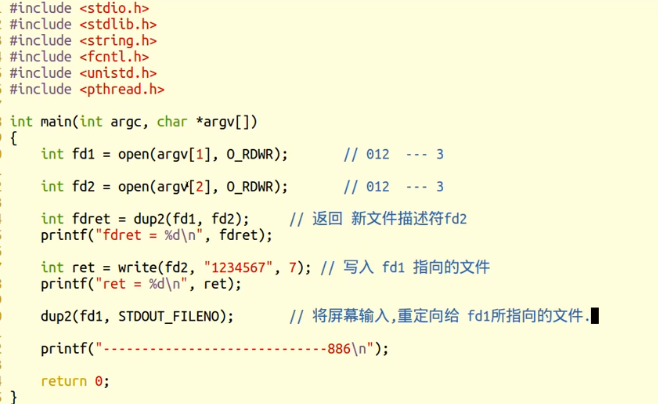
22.重定向

dup和dup2

int dup2(int oldfd, int newfd);

放回值：新文件描述符newfd

例子：dup2（2， 4）就是把后面的指向前面的



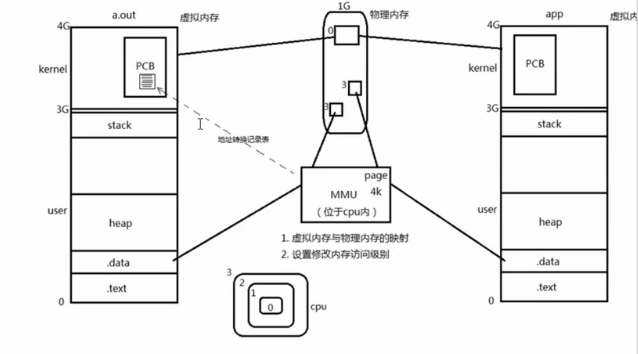
23.

进程：

程序：死的。只占用磁盘空间。 --剧本

进程：活的。运行起来的程序，占用内存、cpu等系统资源。 --戏

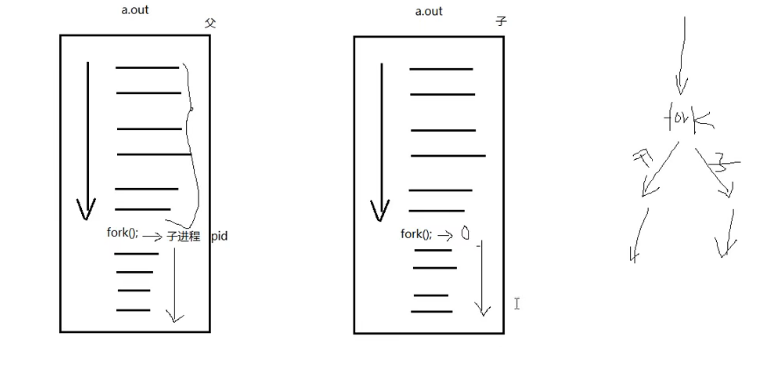
MMU：虚拟内存映射单元



进程控制块：PCB，进程控制块是一个结构体

结构体中包含很多，其中重点掌握：进程id/进程状态（初始态/就绪态/运行态/终止态/挂起态）/进程切换时需要保存和恢复的一些CPU寄存器/描述虚拟地址空间的信息/描述控制终端的信息/当前工作目录位置/umask掩码/文件描述符表包含很多指向file结构体的指针/和信号相关的信息/用户id和组id/会话和进程组/进程可以使用的资源上限

24.



25.

fork函数：

pid\_t fork(void)

创建子进程，父子进程各自返回，父进程返回紫禁城pid。子进程返回0

getpid();获取当前进程的pid

getppid()；获取子进程的pid

特别的，fork之后父进程先执行还是子进程先执行不确定，取决于内核所使用的调度算法

父子进程相同：

刚fork后。data段/test段/堆/栈/环境变量/全局变量/宿主目录位置/进程工作目录位置/信号处理方式

父子进程不同：

进程id/返回值/各自的父进程/进程创建时间/闹钟/未决信号集

父子进程共享：

读时共享，写时复制。（父子进程不共享全局变量）

1. 共享文件描述符2.mmamp建立的映射区

26.

exec函数族(使进程执行某一程序，调用成功不会返回，只有失败才会返回-1)

先只学习两个函数

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

/\* (char \*) NULL \*/);执行自己的进程

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

/\* (char \*) NULL \*/);执行系统的进程

参1：程序名

参2：argv0

参3：argv1

…：argvN

哨兵:NULL

因为是变参函数，所以通过在传入的最后一个参数后再加NULL，可以告诉函数表示传入参数结束，这个操作就叫做加哨兵

例子：

execlp(“ls”, “ls”, “-l”, NULL);

execl(“/bin/ls”, “ls”, “-l”, NULL);

27.

孤儿进程：父进程先于子进程结束，则子进程成为孤儿进程，子进程的父进程成为init进程，称为init进程领养孤儿进程。

僵尸进程：进程终止，父进程尚未回收，子进程残留资源（PCB）存放于内核中，变成僵尸进程（kill命令对其无效）

28.

wait函数： 回收子进程退出资源，阻塞回收任意一个

pid\_t wait(int \*wstatus);

参数：（传出）回收进程的状态

返回值：成功：回收进程的pid

失败：-1，errno

函数作用：

1. 阻塞等待子进程退出
2. 清理子进程残留在内核中的pcb资源
3. 通过传出参数，得到子进程的结束状态

获取子进程正常终止值：

WIFEXITED(wstatus)—》为真—》调用WEXITSTATUS(wstatus)—》得到子进程的退出值

WIFSIGNALED(wstatus)—》为真—》调用WTERMSIG(wstatus)—》得到导致子进程一场终止的信号编号

waitpid函数：指定对某一个进程进行回收，可以设置非阻塞或阻塞。

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

返回值：

>0 ：表示成功回收的子进程pid

0 ：函数调用时，参3指定WNOHANG，并且，没有子进程结束。

-1：失败。errno

参数：

pid：指定回收的进程pid

>0 :指定的一个进程

-1 回收任意子进程

0回收和当前调用waitpid一个组的所有子进程

< -1回收指定进程组内的任意子进程

wstatus：（传出）回收进程的状态

options：WNOHANG，指定回收方式为非阻塞

总结：

一次wait/waitpid函数调用，只能回收一个子进程

想要回收多个子进程，使用循环

waitpid(-1, &status, 0) == wait(&status)

29.

进程间通信的常用方式，特征

管道：简单

信号：开销小

mmp映射：用在非血缘关系进程间

socket（本地套接字）：稳定

管道：默认管道大小4096

实现原理：

内核借助环形队列机制，使用内核缓冲区实现

特征：

1. 伪文件
2. 管道中的数据只能一次读取
3. 数据在管道中只能单向流动（确保管道只有一个读端和一个写端，虽然其实允许多读端，多写端）

局限性：

1.自己写，不能自己读

2.数据不可以反复读

3.采用的通信方式是半双工通信

4.管道必须作用在有血缘关系的进程之间

30.

pipe函数：创建并打开管道

int pipe(int fd[2])

参数：

fd[0]：读端

fd[1]：写端

返回值：

成功：0

失败：-1 .errno

管道的读写行为：

读管道：

1. 管道有数据，read返回实际读到的字节数
2. 管道无数据：1）无写端，read返回0（类似读到文件结尾）

2）有写端，read阻塞等待

写管道：

1. 无读端，异常终止，由SIGPIPE导致
2. 有读端，1）管道已满，阻塞等待

2）管道未满，返回写入的字节数

管道的优劣：

优点：

简单，相比信号，套接字实现进程间通信，简单的多

缺点：

1. 只能单向通信，双向通信需要建立两个管道
2. 只能用于父子，兄弟进程（有共同祖先）间通信，该问题后来使用fifo有名管道解决

pipe管道：用于有血缘关系的进程通信

父子进程间通信

兄弟进程间通信

fifo管道：可以用于无血缘关系的进程通信

命名管道：mkfifo

无血缘关系进程间通信

读端：open（fifo， O\_RDONLY）

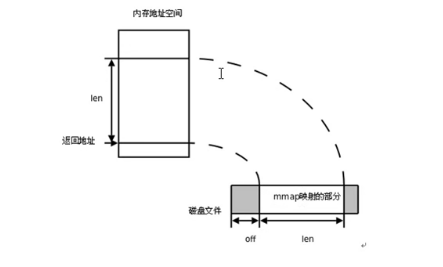
写端：open（fifo， O\_WRONLY）

文件实现进程间通信：

依赖打开的文件是内核中的一块缓冲区，多个无血缘关系的进程，可以同时访问该文件

31.

共享内存映射：将磁盘上的东西映射到内存



void \*mmap(void \*addr, size\_t length, int prot, int flags,

int fd, off\_t offset);

作用创建共享内存映射。

参数：

addr，指定映射区的首地址，通常传NULL，表示让系统自动分配

length：共享内存映射区的大小（<= 文件的实际大小）

prot：共享内存映射区的读写属性，PROT\_READ/PROT\_WRITE

flags:标注共享内存的共享属性。MAP\_SHARED/MAP\_PRIVATE（共享表示对内存的修改会反应到磁盘上）

fd：用于创建共享内存映射区的那个文件

offset：默认0，表示映射文件全部。偏移位置。必须是4k的整数倍

返回值：

成功：返回映射区的首地址

失败：MAP\_FAILED（void \*(-1)），errno