# 进程环境

## main函数

int main(int argc ,char \*\*argv)

int main(int argc ,char \*argv[]){

return 0;//返回这个值是给父进程看的（./程序名 运行时父进程为shell）

}

运行一个ELF可执行文件时，系统shell进程会创建一个子进程来运行main函数。

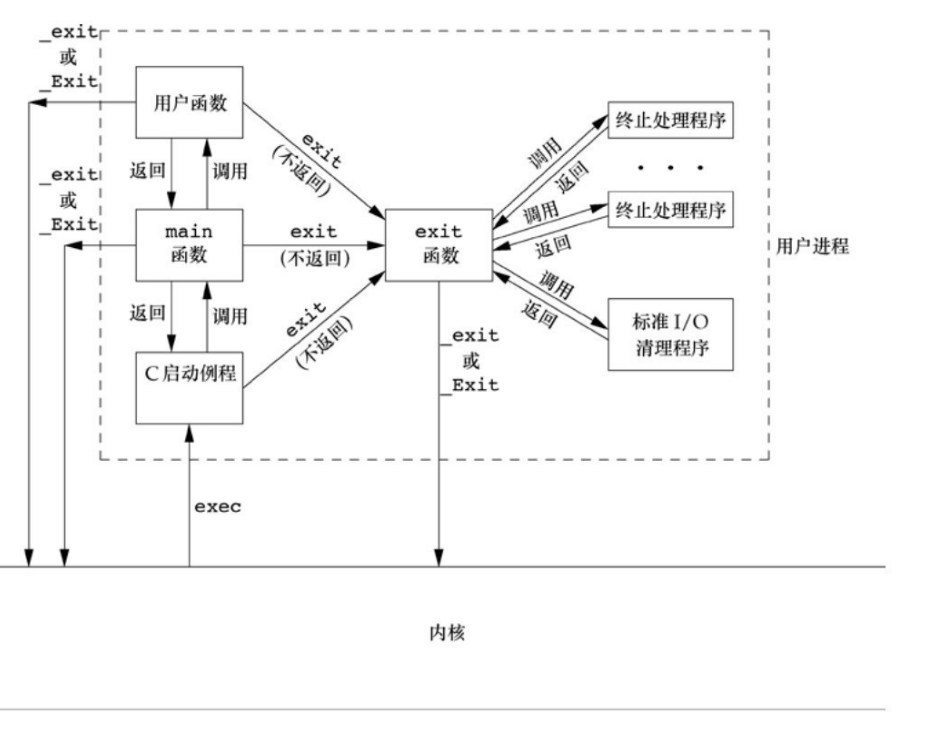
note:linux只有线程的概念（父线程/子线程），当一个程序只有一个线程时，称此线程为进程。

main函数必须是进程的入口和出口

## 进程的终止

（1）正常终止：

* 从main返回 return
* 调用exit （标准库函数 会进行钩子函数调用 IO刷新等清理工作）
* 调用\_exit或者\_Exit (系统调用 不会进行钩子函数调用 IO刷新等清理工作)



* 最后一个线程，从启动例程返回
* 最后一个线程调用pthread\_exit (最后一个线程终止)

1. 异常终止：

* 调用abort
* 接到一个信号并终止
* 接到一个线程对其取消请求并作出响应

void exit(int status);//status & 0377 is returned to the parent 低8位有符号

int atexit(void (\*function)(void));//钩子函数 注册一个在进程正常终止时要调用的函数 在exit（）执行前执行 先注册后执行

## 命令行参数的分析

分析命令行传参的标准函数：

int getopt(int argc, char \* const argv[],

const char \*optstring);

用到的全局变量：

extern char \*optarg;

extern int optind, opterr, optopt;//optind 当前解析到argv的位置

int getopt\_long(int argc, char \* const argv[],

const char \*optstring,

const struct option \*longopts, int \*longindex);

//一次只能解析出一个选项，需要多次运行该函数，解析出所有选项

ffplay -i /dev/video0 -s 800x400

-i 成为一个选项 后面的称为选项参数

ls -a /dev

其中的/dev并不是选项参数，是一个非选项的传参。

const char \*optstring 规定了所有有效的选项

char \*optarg；指向了该选项参数起始的地址

如果optstring第一个符号为 '-', 解析到所有的非选项参数将返回1；

## 环境变量

export 命令显示所有环境变量

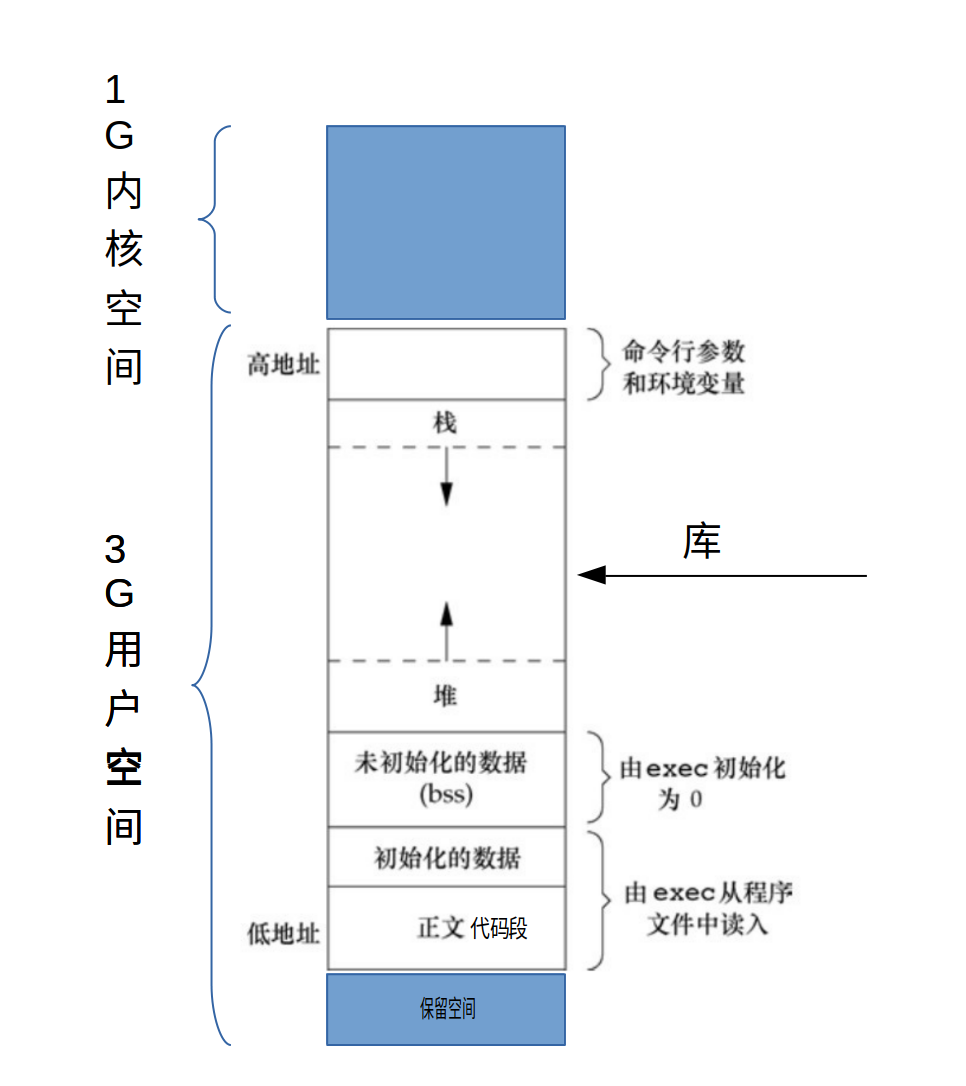
extern char \*\*environ; //该地址是存储环境变量数组的地址

char \*getenv(const char \*name);

int setenv(const char \*name, const char \*value, int overwrite);

int putenv(char \*string);

## C程序的存储空间布局



ps auf 命令查看进程树

pmap 命令查看进程内存

## 库

动态库

静态库

手工装载库:

void \*dlopen(const char \*filename, int flags);//装载一个库

void \*dlsym(void \*handle, const char \*symbol);//查找一个函数

Link with -ldl.

int dlclose(void \*handle);//关闭一个库

note:

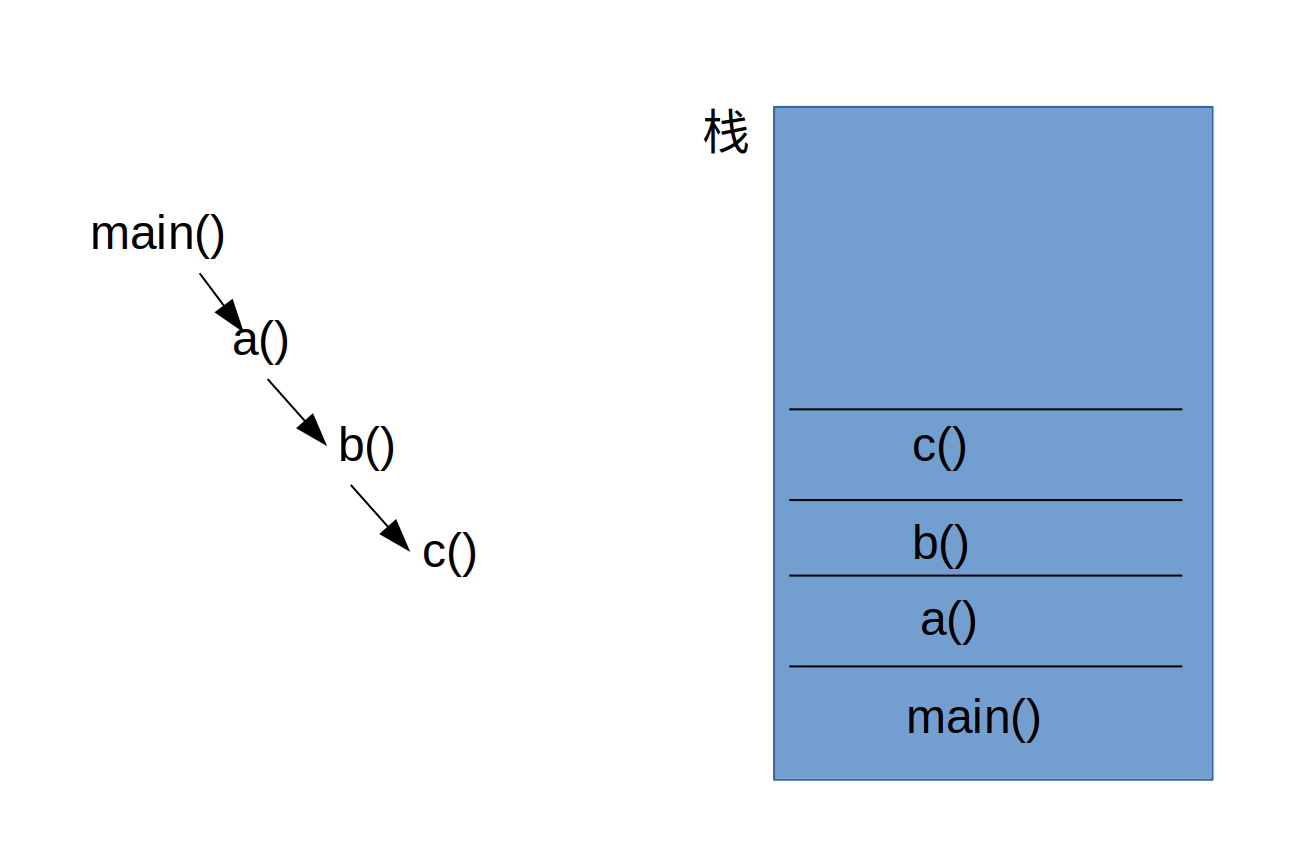
根据C标准，函数指针和void\* 之间的转换是未定义的。POSIX.1-2003 and POSIX.1-2008 accepted this state of affairs and proposed the following workaround:

\*(void \*\*) (&cosine) = dlsym(handle, "cos");

## 函数跳转(跨函数 安全跳转)

int setjmp(jmp\_buf env); //设置跳转点 设置跳转点成功返回0 从跳转处返回，返回非0值。

void longjmp(jmp\_buf env, int val);//跳转



goto 跳转时，不会对函数环境进行恢复，是不安全的跳转，所以goto一般只用在函数内部。

当一系列函数嵌套，不会改变函数环境时，比如单纯的进行搜索（ag:递归搜索），找到目标值后，在出栈一步一步跳转回去是没有必要的，此时可以直接跳转回最上层。比如: 调用顺序为main()——>a()——>b()——>c() ,在c函数找到目标值,直接从c函数跳转到mian() 函数。

## 资源的获取与控制

int getrlimit(int resource, struct rlimit \*rlim);

int setrlimit(int resource, const struct rlimit \*rlim);

struct rlimit {

rlim\_t rlim\_cur; /\* Soft limit 软限制最高不能超过硬限制\*/

rlim\_t rlim\_max; /\* Hard limit (ceiling for rlim\_cur) \*/

};

普通用户只能降低硬限制，可以提高或降低软限制。

root用户可以升高或降低硬限制。