Linux 开发常用工具

1. Strings 输出ELF文件中的所有字符串
2. Strip 删除elf 文件中一些无用的信息，减少空间
3. Nm 列出目标文件符号，地址，符号类型
4. Size 显示目标文件段和文件大小
5. Readelf 显示elf格式文件的内容
6. Objdump 显示目标文件信息，可以用来反汇编
7. Ar 创建静态库，添加或删除文件内容
8. Addr2line 将地址转换成文件行号

堆内存优化：

1. 堆内存最小单位是16B，尽量减少小块内存的申请，避免内存浪费
2. 用mallopt 函数设置M\_MMAP\_THRESHOLD 参数，降低mmap的门槛，会降低内存空洞的危险，但是增加了系统调用，降低了性能
3. 调整用mallopt 函数设置M\_TRIM\_THRESHOLD参数，减少堆顶连续内存门槛，释放更多堆顶的内存

栈内存优化：

栈的内存申请后不会释放，出栈时只是空间可以复用，物理内存还是没有释放

1. 尽量避免在栈空间中申请大量内存
2. 尽量避免使用递归函数，因为递归会不断的加大栈的空间

环境变量内存的优化

1. 尽量在程序启动前设置好环境变量，使得环境变量精密的排列在栈空间，在程序内增加或修改环境变量会导致在堆中申请内存

数据段优化

1. 尽可能减少全局变量和静态变量，可以使用nm –format=sysv elf\_file | grep -w .data或者 nm –format=sysv elf\_file | grep -w .bss
2. 对于库，可以把数据段改到代码段，因为数据段独立，代码段共享

Int i =1; 改成const int i = 1

1. 全局变量定义尽量放在.c文件中定义，在.h文件中extern，避免逻辑和编译时内存占用
2. 不要在共享库中通过extern 的方式引用共享库的全局变量，可以包一个接口函数操作全局变量
3. 不要把一些无用的动态库链接到程序中，否则程序加载重定位的时候会占用物理页面
4. C库的小so可以考虑合并动态库，注意防止合并导致loader 加载一些无关的内容
5. C++ 大库，可以考虑拆分
6. 如果一个a.so仅仅只有一个b.so 或可执行程序依赖它，且同时出现，可以把a.so 编译成静态库，然后链接到b.so 或可执行程序中
7. 链接静态库时，会把静态库的所有内容都copy到so 或可执行文件中，所以在生成静态库的时候，没有必要的东西，可以从静态库中删除

代码段优化

1. 编译时不要加上-export-dynamic, 加上“-Wl, -export-dynamic” 编译选项会增加代码段的大小，从而加大内存
2. 删除程序永远也不到的代码，加上编译选项“-Wunused” 和“—Wunreachable-code” 检测冗余代码
3. 对于共享库的代码，减少不必要的导出符号，可以使用static，推荐使用编译参数“—version-script”, 使用方式“-Wl，--version-script=scriptName”

File:scriptName

{

Global:

funcA;

funcB;

…

local:\*;

}

线程内存优化

1. 进程的默认栈8MB，线程默认栈2MB，可以根据情况设置大小，线程退出后，调用pthread\_join 释放栈内存
2. 减少线程数量，考虑异步通信

系统内存的优化

1. /tmp 目录占用内存的

内存泄漏

1. 怎样通过出错的地址确定调用的是哪个函数，nm -n a.out 会列出程序的使用的函数对应的地址
2. 编译时加上编译选项“-g”, 使用addr2line -e a.out 0xabcde 可以定位文件的具体行号
3. Malloc 和free Hook几种
4. Coredump 无法定位的时候，在编译的时候加上“-mapcs-frame”
5. 减少守护进程的代码量，尽量简单，因为守护进程长期运行，一点点泄露都非常验证，普通进程有泄露问题不大，因为运行完就结束了，内存系统会释放普通进程的内存
6. root@mylinkit:/proc/1491# cat stat | awk '{print $30}' 得到进程号1491的EIP的当前值

性能分析

1. gprof 工具 产生程序主要耗时在上面地方：
   1. gcc -pg -g test.c -o test
   2. ./test
   3. Gprof test gmon.out > anlysis.txt
   4. Vim analysis.txt
2. Oprofile 工具查看程序主要运行耗时的代码段

opcontrol –setup–no-vmlinux

opcontrol –init

opcontrol –start  //启动

opcontrol –dump  //采集

./a.out

opcontrol –stop   //停止

opreport     //查看

opreport –l   //查看，包括函数

嵌入式设备需要通过strip -s a.out 把程序瘦身

1. Splint /flawfinder检测代码