# 堆分配器实验报告

# 实验主题

使用sbrk实现堆内存分配器MyMalloc

# 实验环境与工具

## 硬件环境

X64 CPU；3GHz；16G RAM；256GHD Disk 以上

## 软件环境

Windows7 64位以上；VirtualBox/Vmware 11以上；Ubuntu 16.04 LTS 64位/优麒麟 64位

## 开发工具

Vim vscode

# 动态内存分配器的基本原理

动态内存分配器维护着一个进程的虚拟内存区域，称为堆。分配器将堆视为一组不同大小的块的集合, 来维护，每个单元称之为块，每个块就是一个连续的虚拟内存片，要么是已分配的，要么是空闲的。已分配的块显式地保留为供应用程序使用。空闲块可用来分配。空闲块保持空闲，直到它显式地被应用所分配。一个已分配的块保持已分配状态，直到它被释放，这种释放要么是应用程序显式执行的，要么是内存分配器自身隐式执行的。

分配器有两种基本风格：显式分配器和隐式分配器。两种风格都要求应用显式地分配块。它们的不同之处在于由哪个实体来负责释放已分配的块。

显式分配器：要求应用显式地释放任何已分配的块。例如C程序通过调用malloc函数来分配一个块，通过调用free函数来释放一个块。其中malloc采用的总体策略是：先系统调用sbrk一次，会得到一段较大的并且是连续的空间。进程把系统内核分配给自己的这段空间留着慢慢用。之后调用malloc时就从这段空间中分配，free回收时就再还回来（而不是还给系统内核）。只有当这段空间全部被分配掉时还不够用时，才再次系统调用sbrk。当然，这一次调用sbrk后内核分配给进程的空间和刚才的那块空间一般不会是相邻的。

## 项目整体结构



Makefile 自动化编译脚本

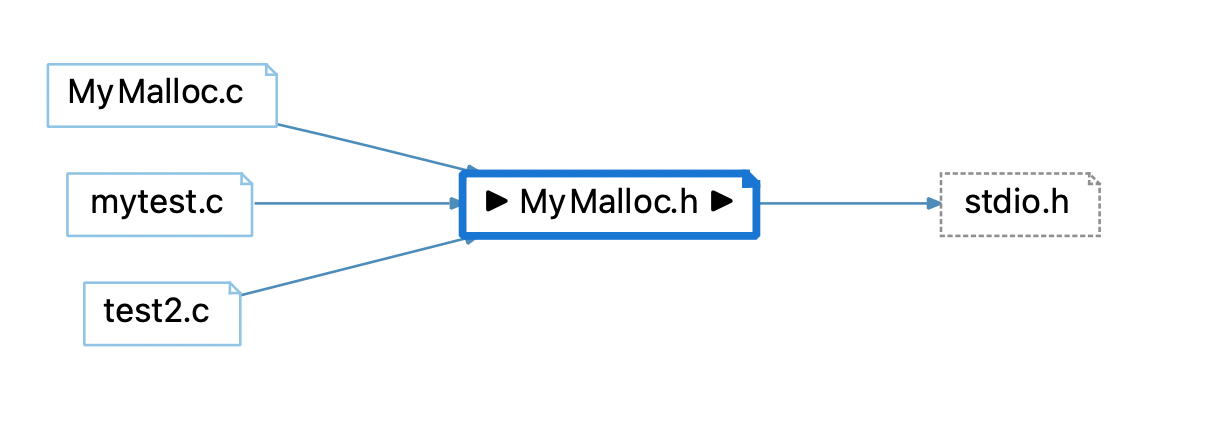
MyMalloc.c 函数源代码

MyMalloc.h 函数头文件

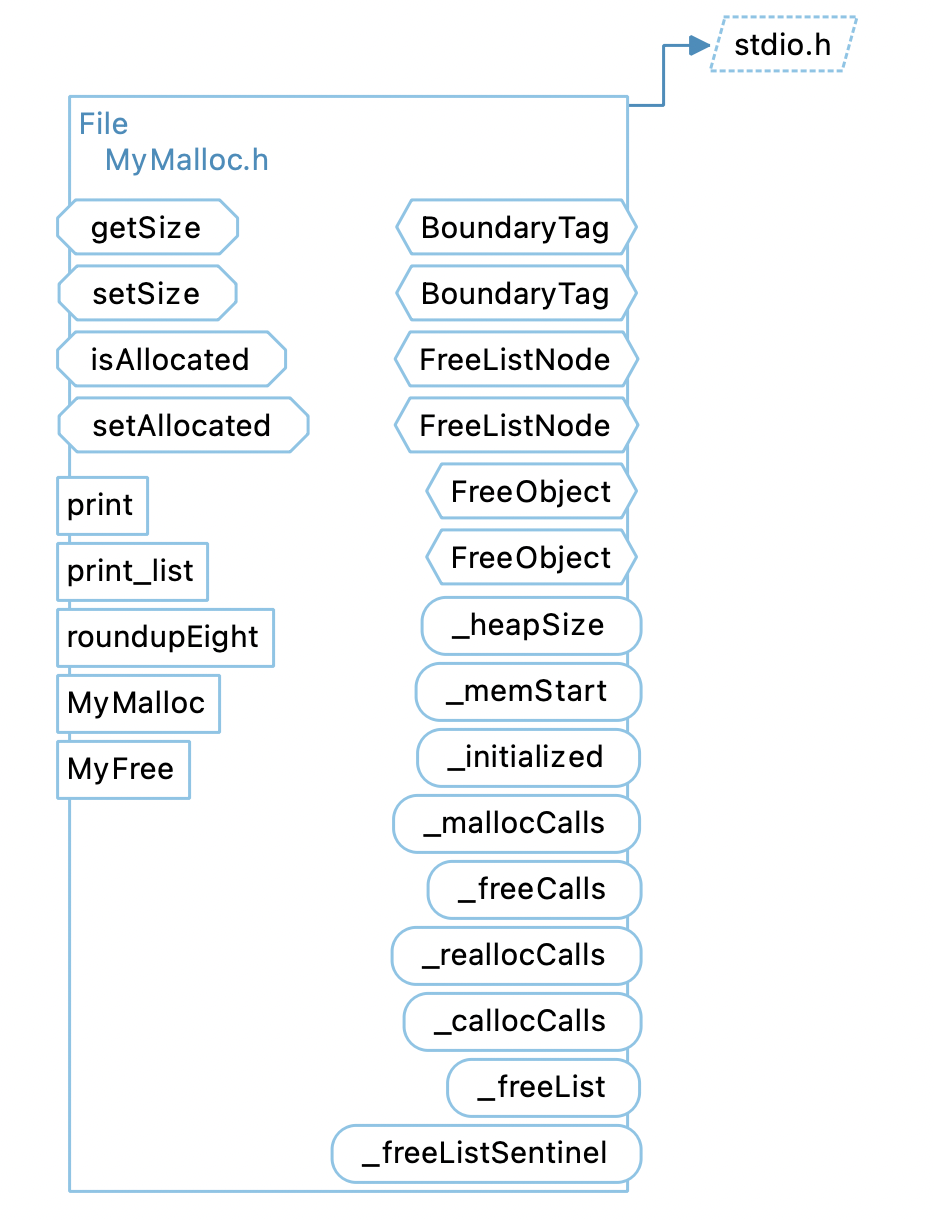
Mytest.c 测试文件1

Test2.c 测试文件2

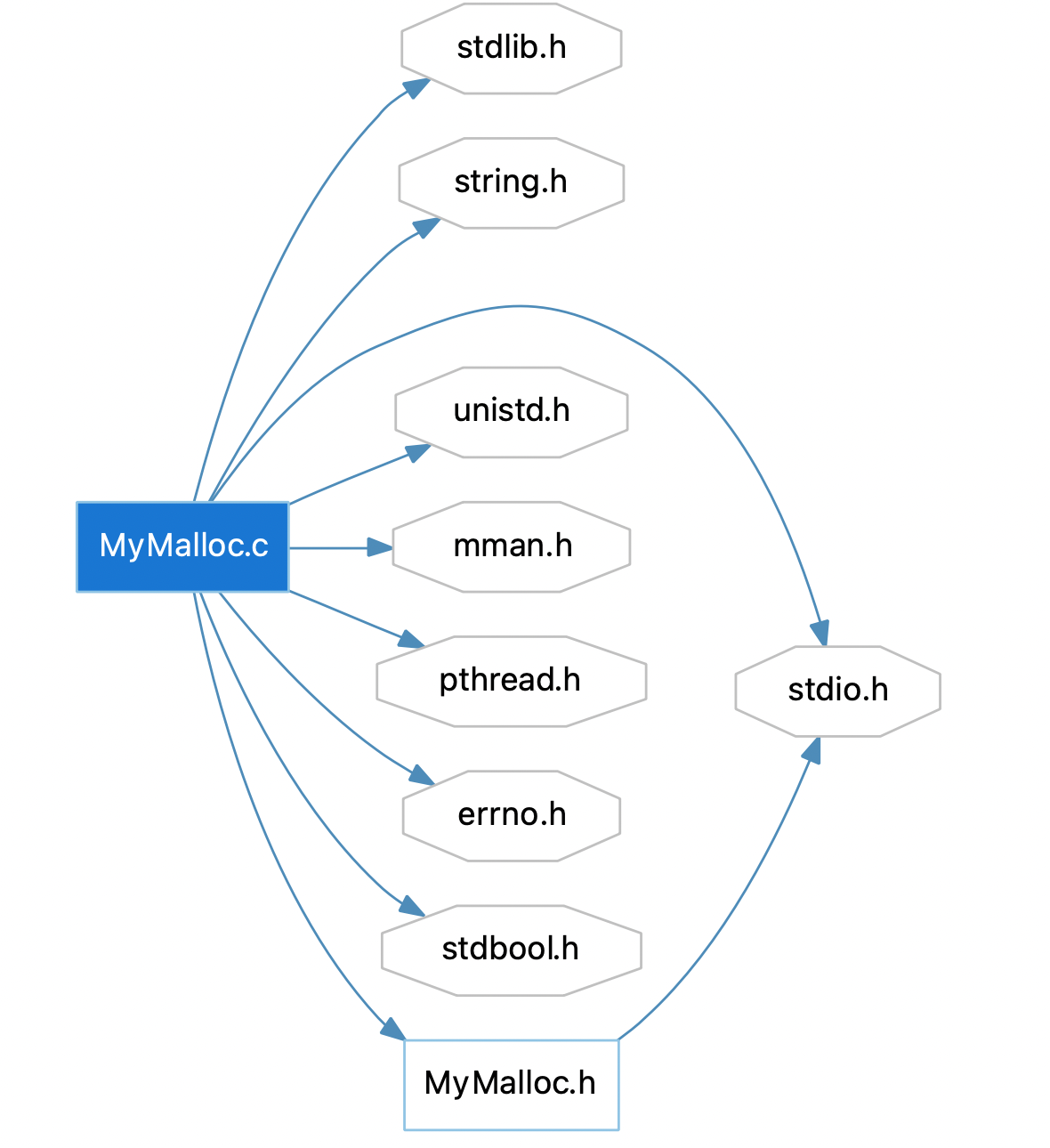
## 头文件依赖



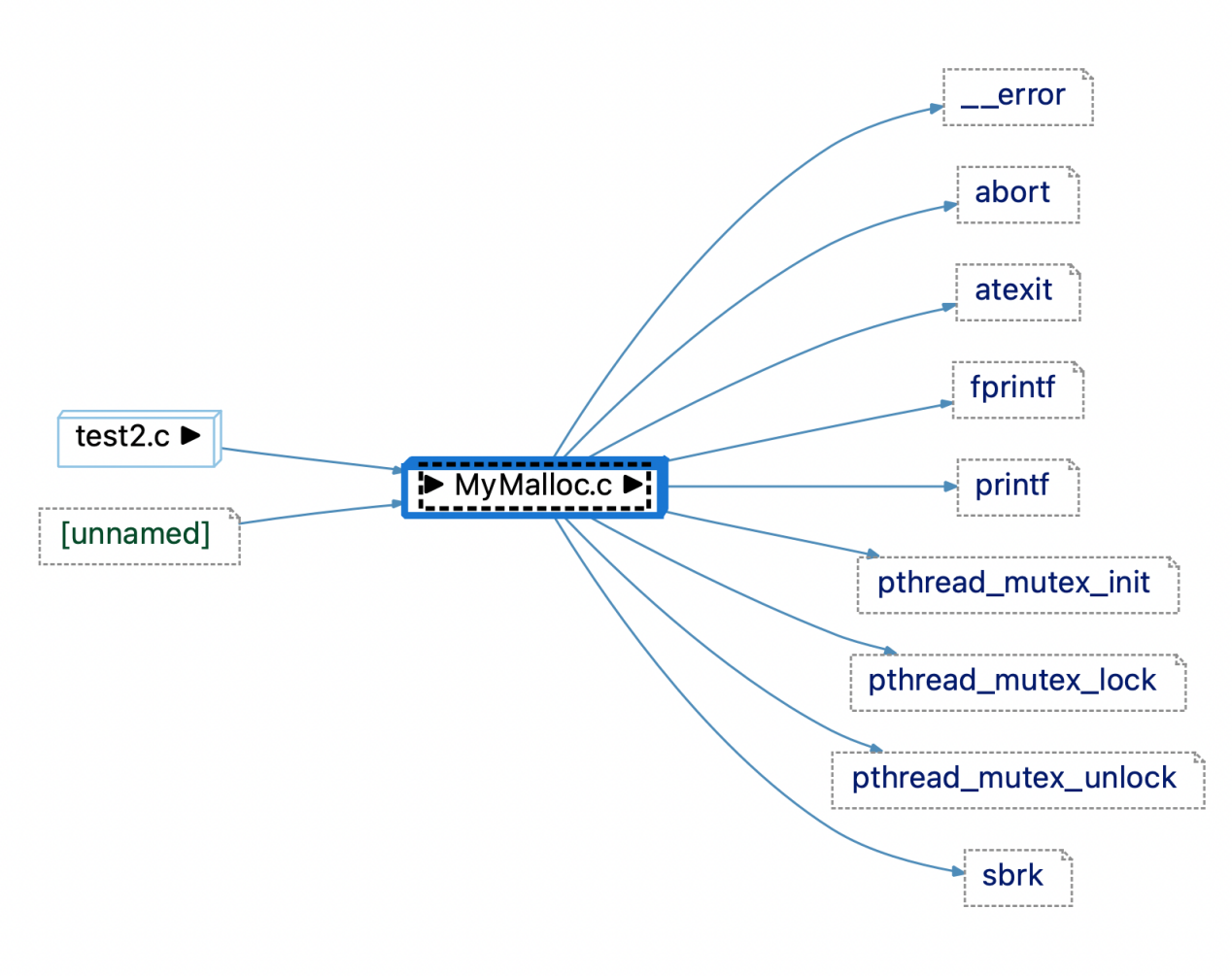
## 头文件属性

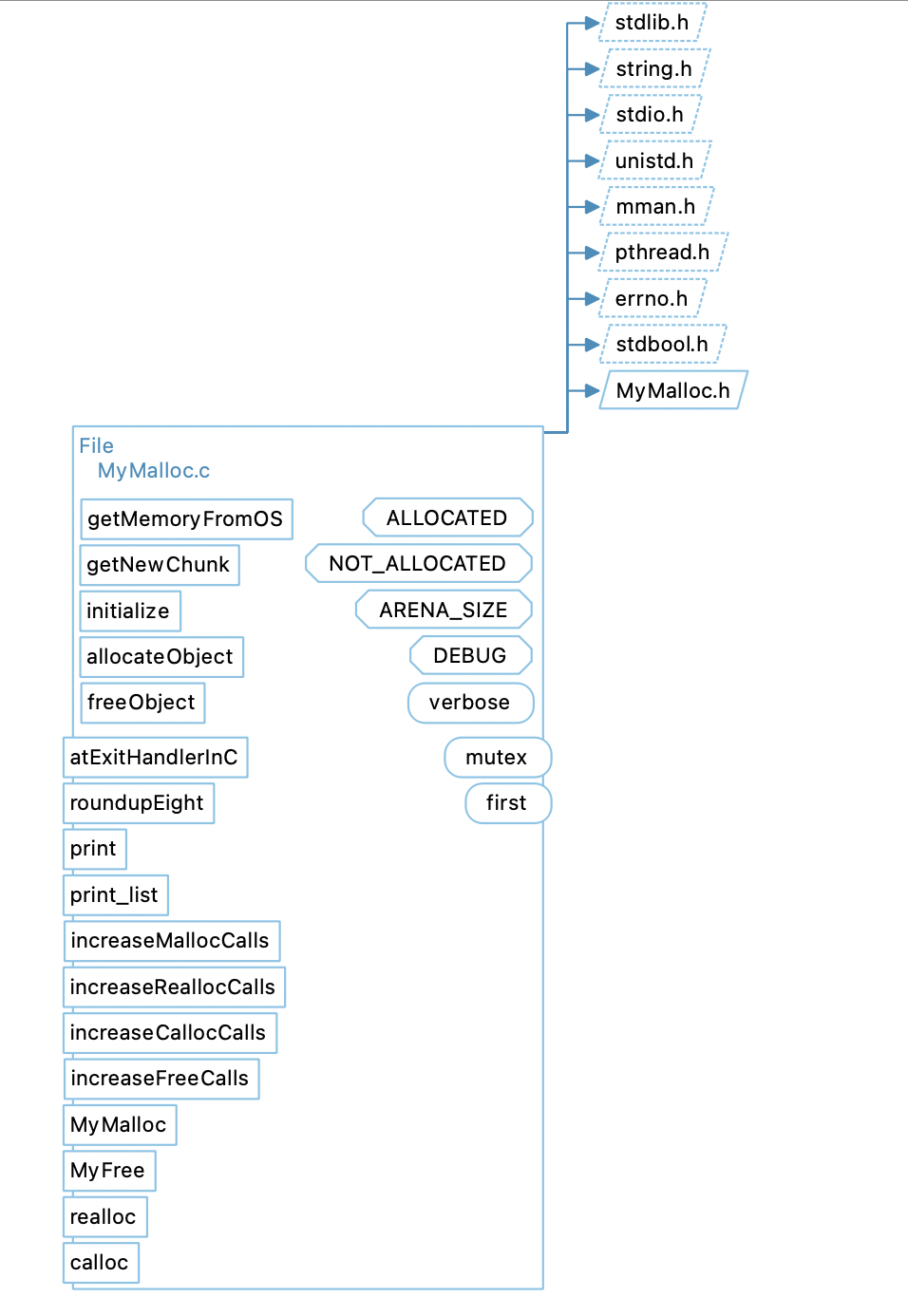


## 函数依赖



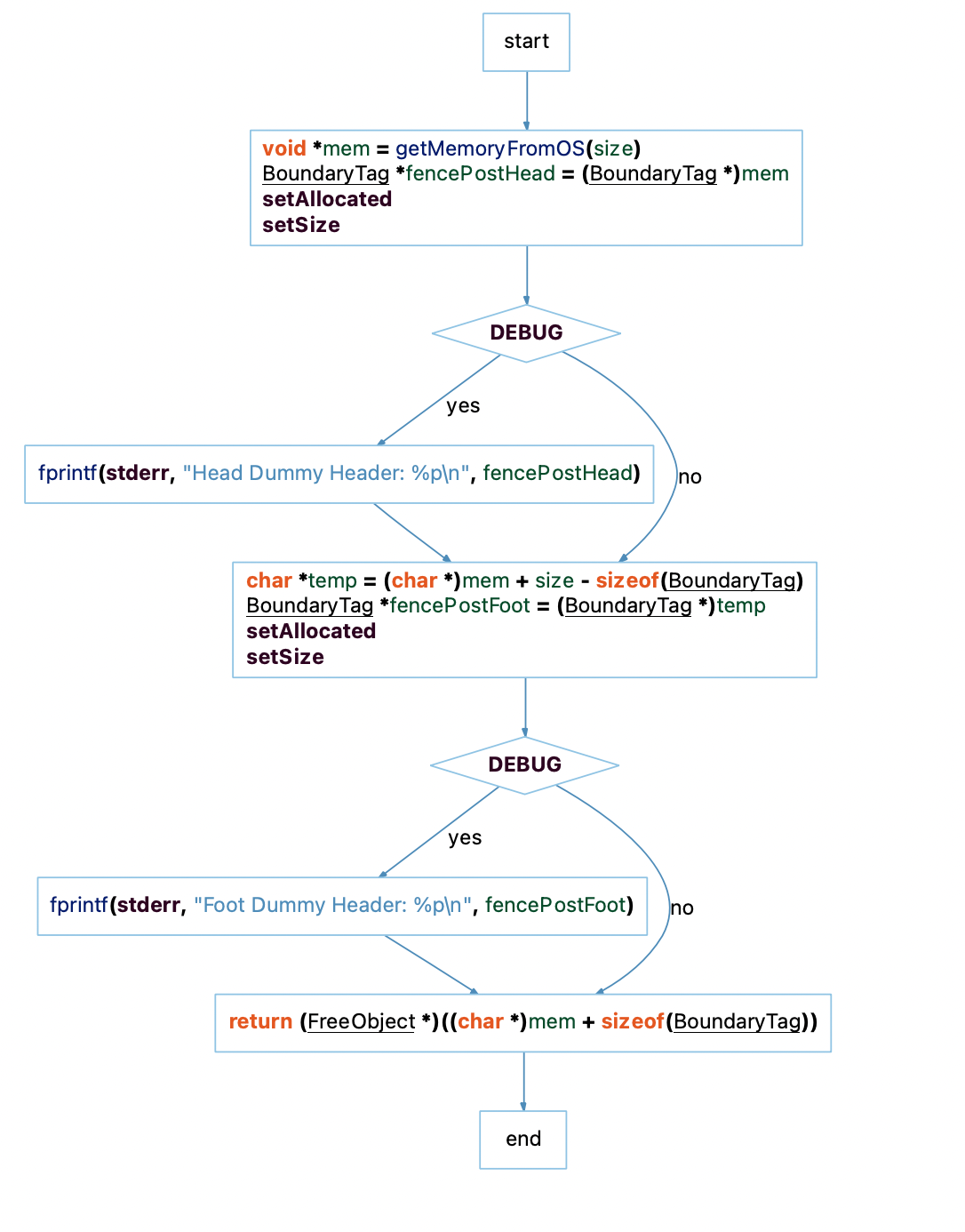
## 函数调用情况



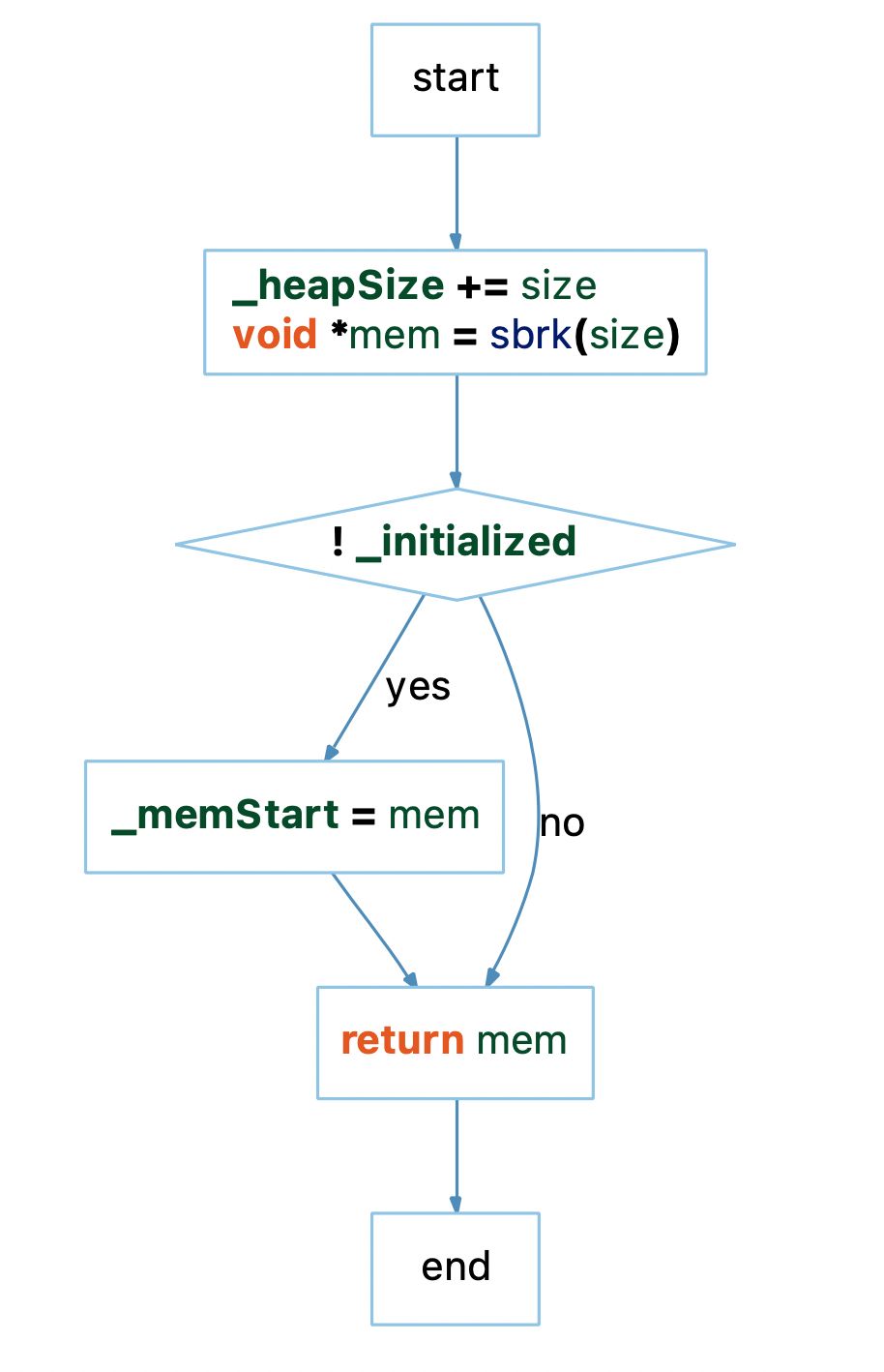


# 具体函数分析

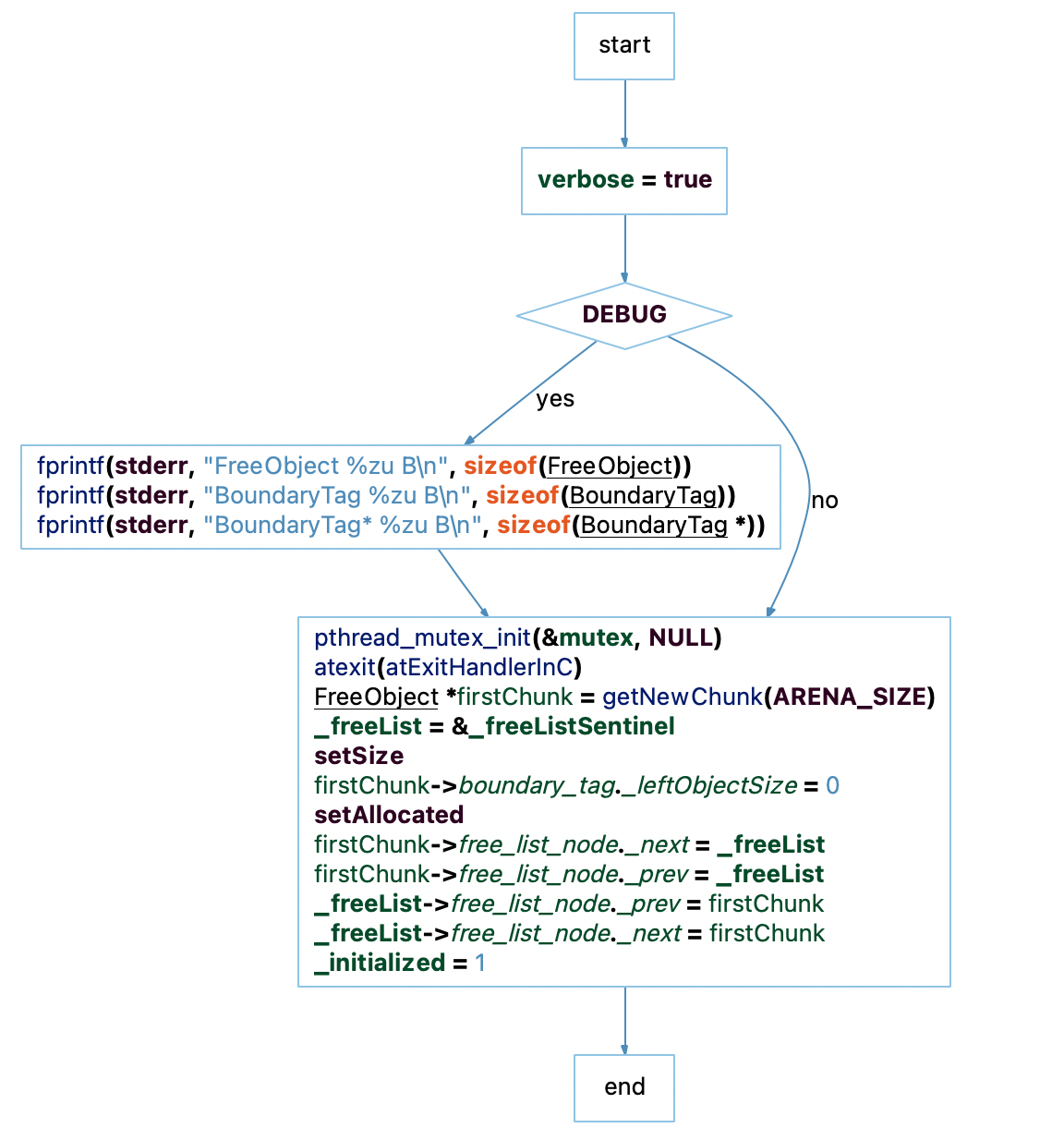
getNewTrunk获取新的内存块



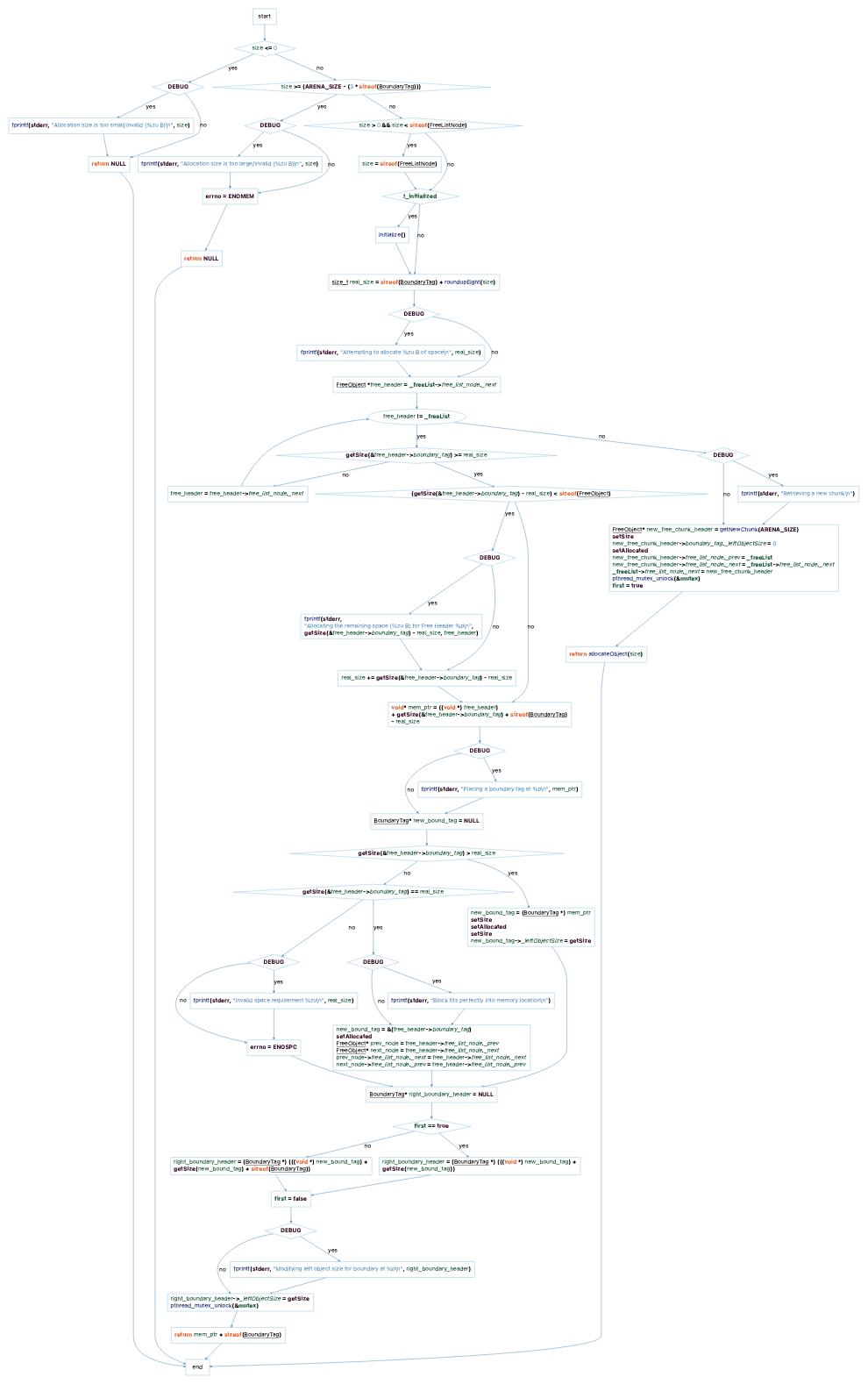
getMemoryFromOS通过sbrk系统调用获取内存



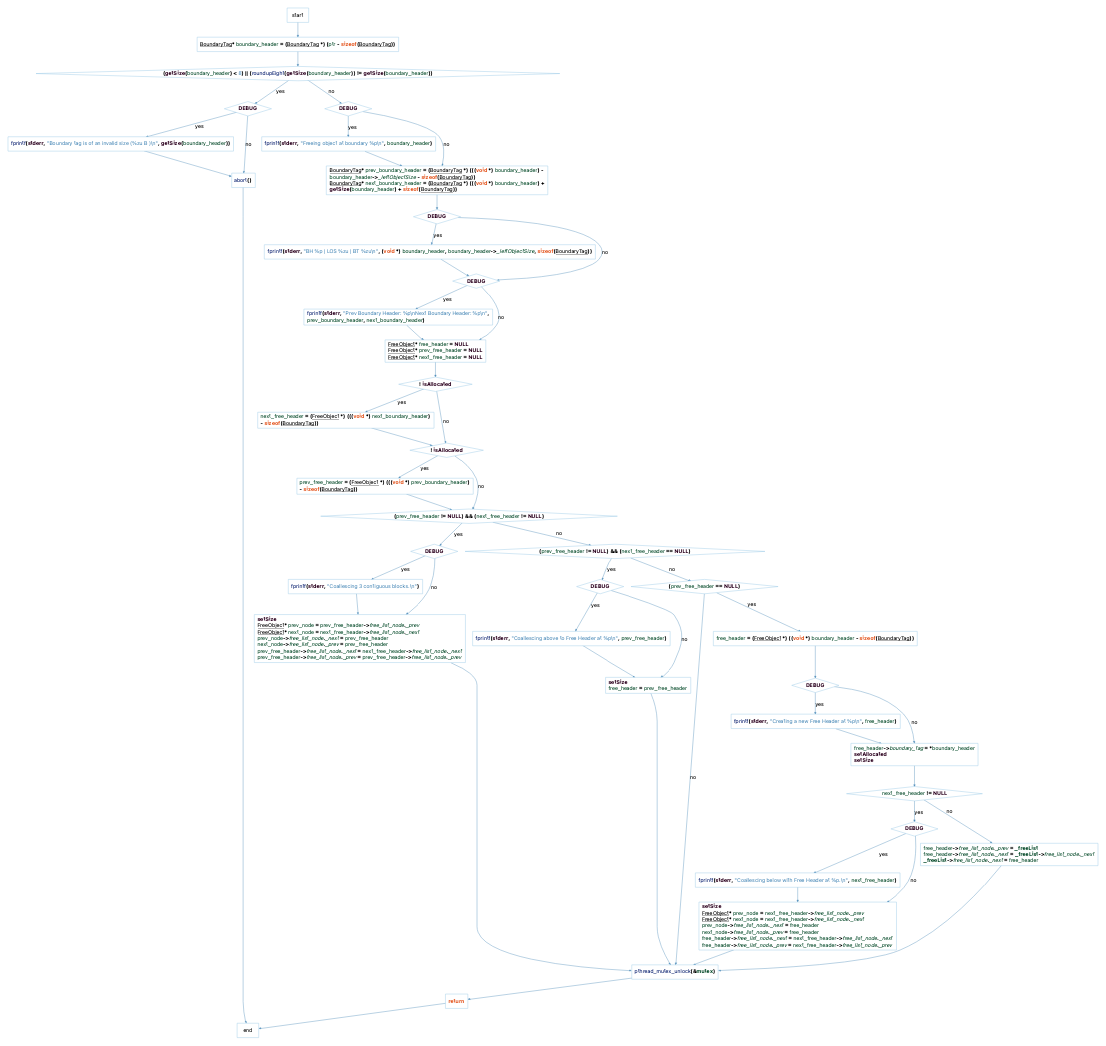
Initialize初始化freelist



allocateObject分配内存对象



freeObject释放内存对象

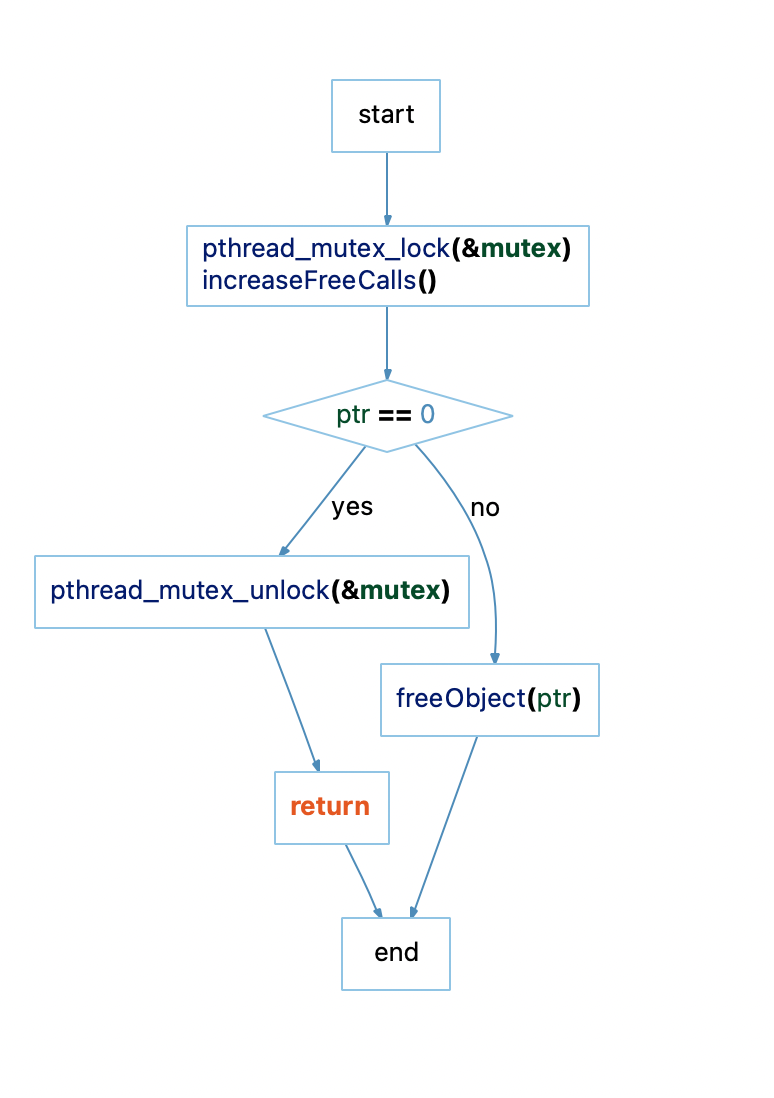


MyMalloc函数分配堆内存



使用posix的pthread互斥加锁来保障多线程安全

MyFree函数释放已经分配了的内存

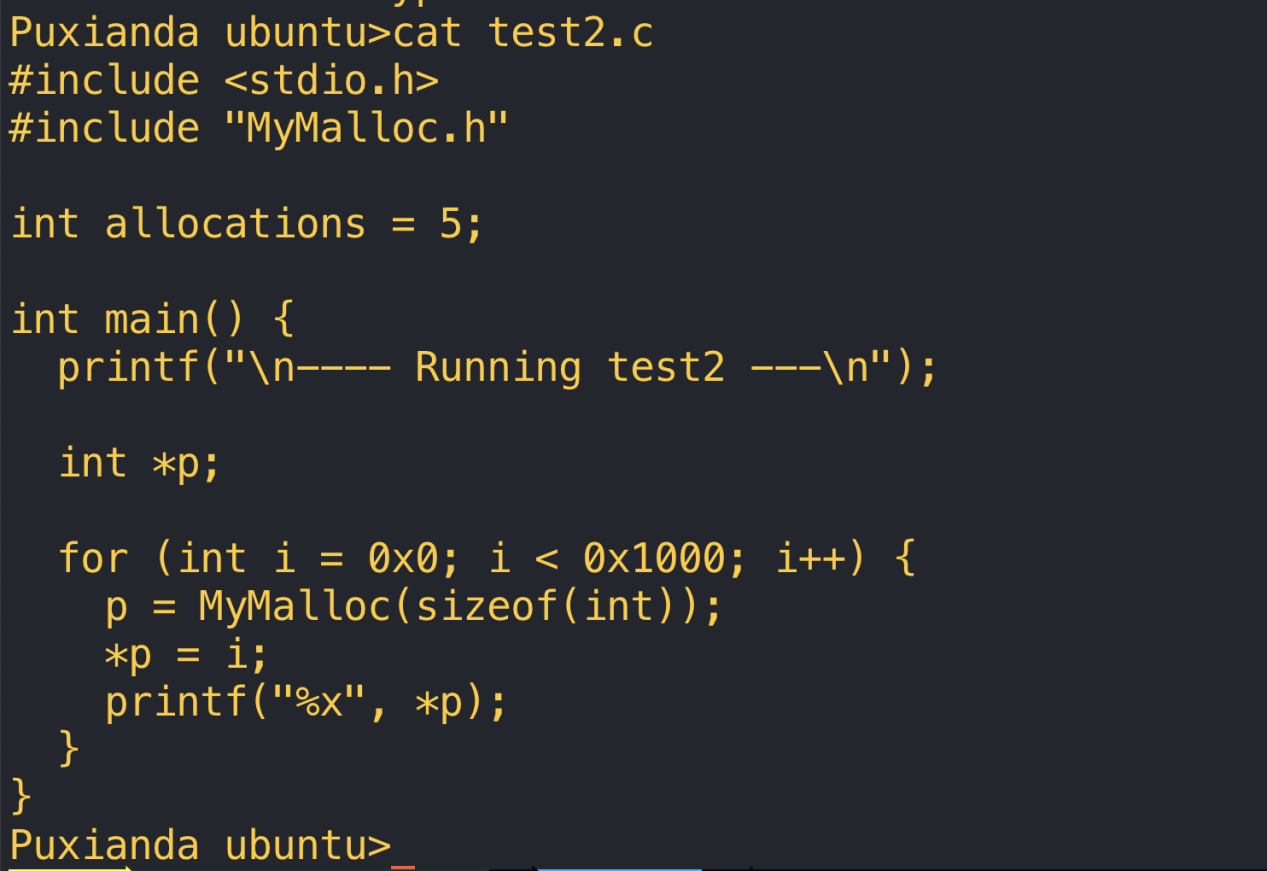
\

# 测试

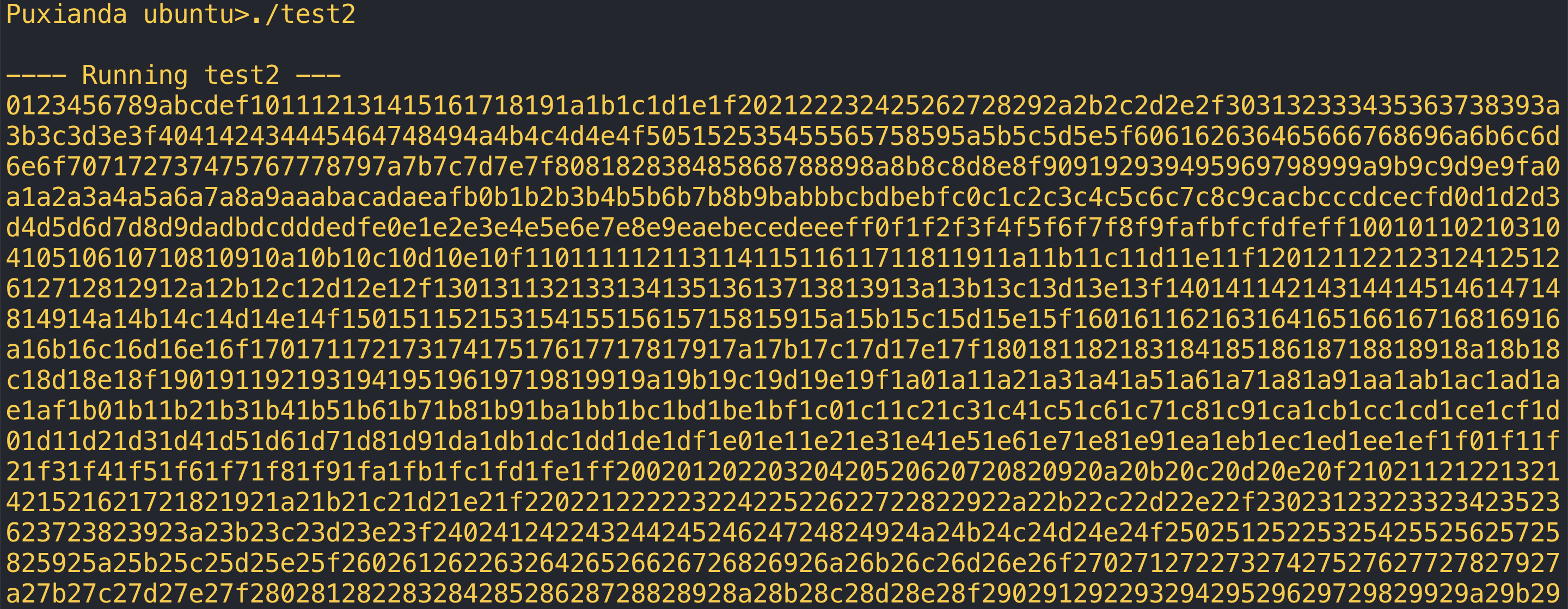
## 普通测试

源代码:

使用for循环0x1000次进行压力测试



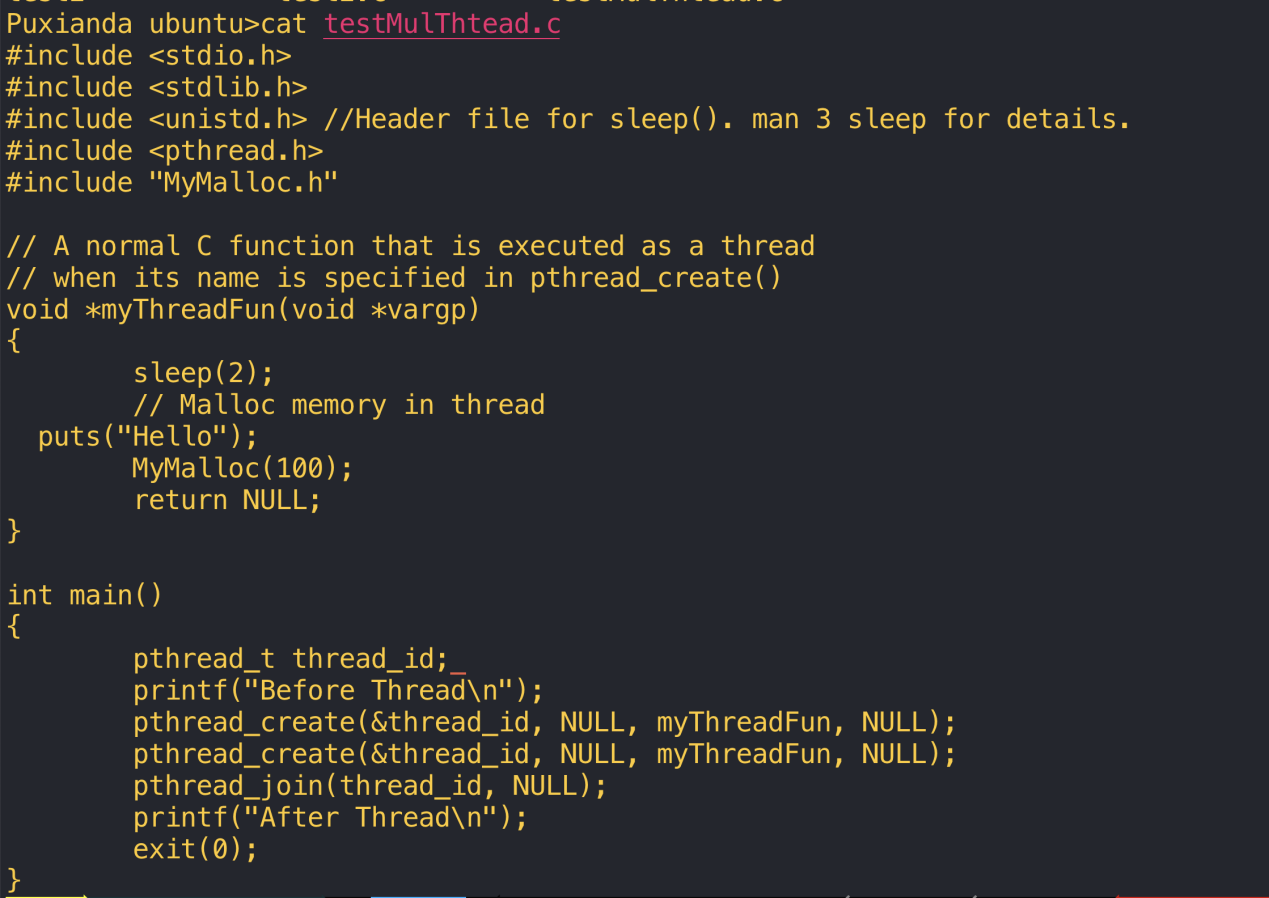
结果



输出0x1-0x1000与预期结果相符，测试通过。

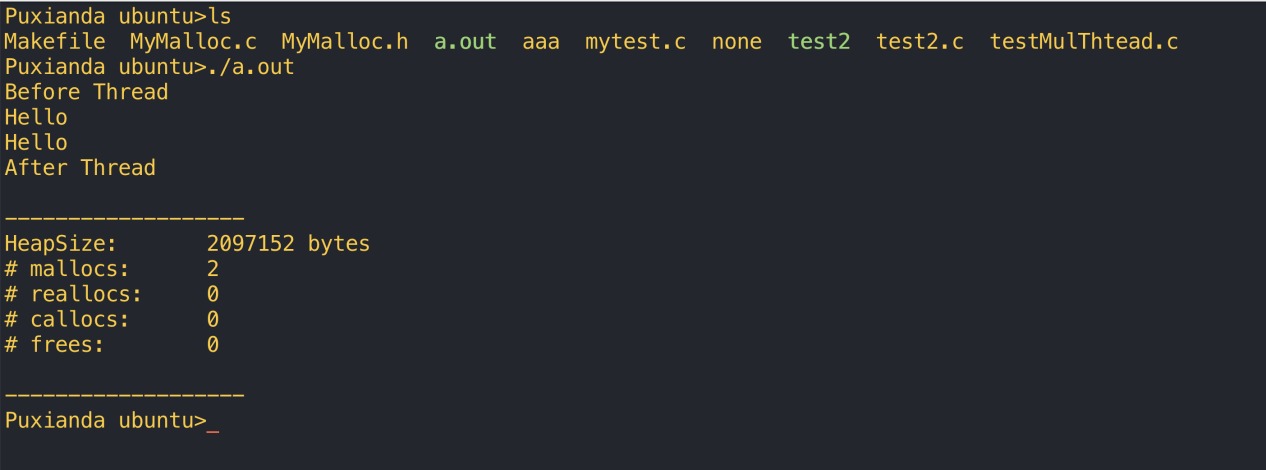
## 多线程测试

源代码:



这里使用两个pthread\_create函数创建两个线程， 每个线程分别分配一次内存。

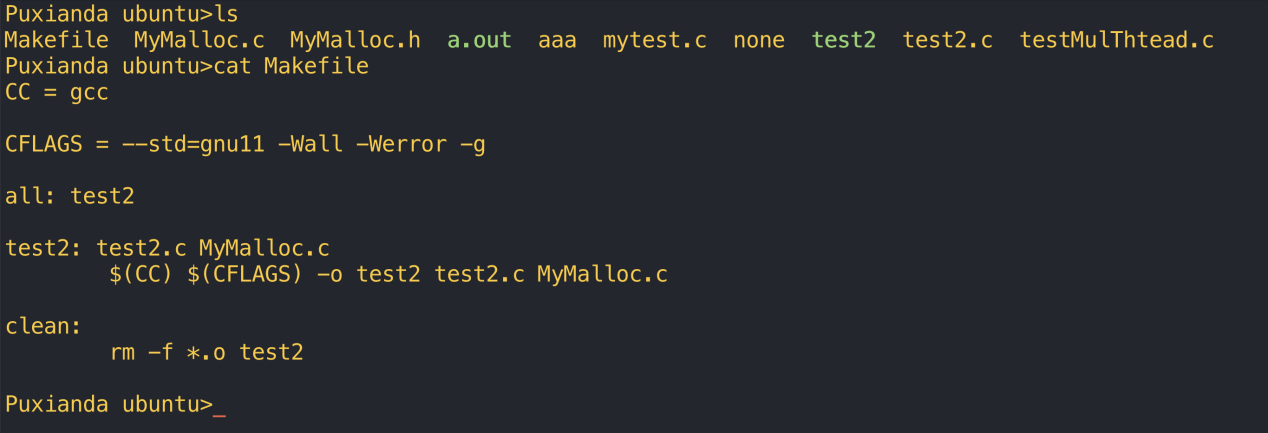
结果:



可以看到程序在多线程情况下也能正常工作, 测试通过。

## 自动化makefile

源代码:



Make:

