课程设计

**设 计 题 目：**

**专 业 班 级：**

**学 生 姓 名：**

**指 导 教 师：**

**设 计 时 间：**

XXX学院课程设计任务书

任务下达日期：

设计题目：

设计主要内容和要求：

利用linux系统调用实现内存分配器。

1.程序运行环境为Linux下。

2 .程序是线程安全的。

摘 要

我国操作系统技术随着国家经济的发展正处在一个飞速发展的时代。给互联网带来了很多新的问题和任务。经过几十年的发展， 我国计算机技术似乎已经达到了世界领先的技术， 直到中兴事件的发生才让我们意识到了底层和基础知识的重要性。 中兴作为国内最大的制造商之一，因缺乏制造芯片的核心技术被美国制裁，股票顿时，甚至威胁到了公司的安全。操作系统作为软件界最重要的成员也是我们国家的软肋。

该项目试图设计操作系统中一个重要的部分，堆内存分配器。该程序运用基于c语言，通过调用linux系统调用进行内存管理。以vim, gcc, make等为主要开发工具，设计出基于linux下简单内存分配器。

**关键字**：c语言，堆内存，内存管理

目录

[1 课程设计单位概况 1](#_Toc1947201955)

[1.1课程设计时间 1](#_Toc1119961052)

[1.2课程设计地点 1](#_Toc491235009)

[2 绪 论 2](#_Toc1259657195)

[2.1设计背景 2](#_Toc1164684239)

[3 相关技术 3](#_Toc534562468)

[3.1 线程概述 3](#_Toc1467304275)

[3.2 linux 3](#_Toc1428231424)

[3.3 brk and sbkr 4](#_Toc1860820649)

[4 需求分析 5](#_Toc1008296482)

[4.1功能性需求 5](#_Toc645514497)

[4.2 非功能性需求 5](#_Toc74766435)

[4.2.1软硬件环境需求 5](#_Toc321539550)

[5 系统设计 6](#_Toc1046360998)

[5.1设计思路 6](#_Toc445708103)

[5.2结构图 7](#_Toc593126385)

[5.2.1 项目结构图 7](#_Toc56063321)

[5.2.3 头文件属性 9](#_Toc1658398661)

[5.2.4 函数依赖情况 10](#_Toc516041014)

[5.2.5 函数调用情况 11](#_Toc1562355712)

[5.2.6 common函数具体分析 13](#_Toc1229899715)

[5.3主函数流程图 18](#_Toc1394407630)

[5.3.1 MyMalloc模块 18](#_Toc319997699)

[6 效果分析（测试） 20](#_Toc902275005)

[6.1 普通测试 20](#_Toc1153977568)

[6.2 多线程测试 21](#_Toc976169319)

[6.3 自动化Makefile 22](#_Toc1850165000)

[参 考 文 献 23](#_Toc159946440)

# 1 课程设计单位概况

## 1.1课程设计时间

2021年5月24日至2021年6月18日。

## 1.2课程设计地点

天津大学XXX。

# 2 绪 论

## 2.1设计背景

在计算机科学中, 动态内存分配（Dynamic memory allocation）又称为堆内存分配，是指计算机程序在运行期中分配使用内存。它可以当成是一种分配有限内存资源所有权的方法。

# 3 相关技术

## 3.1 线程概述

线程（英语：thread）是操作系统能够进行运算调度的最小单位。大部分情况下，它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。在Unix System V及SunOS中也被称为轻量进程（lightweight processes），但轻量进程更多指内核线程（kernel thread），而把用户线程（user thread）称为线程。

线程是独立调度和分派的基本单位。线程可以为操作系统内核调度的内核线程，如Win32线程；由用户进程自行调度的用户线程，如Linux平台的POSIX Thread；或者由内核与用户进程，如Windows 7的线程，进行混合调度。

同一进程中的多条线程将共享该进程中的全部系统资源，如虚拟地址空间，文件描述符和信号处理等等。但同一进程中的多个线程有各自的调用栈（call stack），自己的寄存器环境（register context），自己的线程本地存储（thread-local storage）。

一个进程可以有很多线程来处理，每条线程并行执行不同的任务。如果进程要完成的任务很多，这样需很多线程，也要调用很多核心，在多核或多CPU，或支持Hyper-threading的CPU上使用多线程程序设计的好处是显而易见的，即提高了程序的执行吞吐率。以人工作的样子想像，核心相当于人，人越多则能同时处理的事情越多，而线程相当于手，手越多则工作效率越高。在单CPU单核的计算机上，使用多线程技术，也可以把进程中负责I/O处理、人机交互而常被阻塞的部分与密集计算的部分分开来执行，编写专门的workhorse线程执行密集计算，虽然多任务比不上多核，但因为具备多线程的能力，从而提高了程序的执行效率。

## 3.2 linux

Linux是一种自由和开放源码的类UNIX操作系统。该操作系统的内核由林纳斯·托瓦兹在1991年10月5日首次发布[5][6]，在加上用户空间的应用程序之后，成为Linux操作系统。Linux也是自由软件和开放源代码软件发展中最著名的例子。只要遵循GNU 通用公共许可证（GPL），任何个人和机构都可以自由地使用Linux的所有底层源代码，也可以自由地修改和再发布。大多数Linux系统还包括像提供GUI的X Window之类的程序。除了一部分专家之外，大多数人都是直接使用Linux 发行版，而不是自己选择每一样组件或自行设置。

Linux严格来说是单指操作系统的内核，因操作系统中包含了许多用户图形接口和其他实用工具。如今Linux常用来指基于Linux的完整操作系统，内核则改以Linux内核称之。由于这些支持用户空间的系统工具和库主要由理查德·斯托曼于1983年发起的GNU计划提供，自由软件基金会提议将其组合系统命名为GNU/Linux[7][8]，但Linux不属于GNU计划，这个名称并没有得到社群的一致认同。

Linux最初是作为支持英特尔x86架构的个人电脑的一个自由操作系统。目前Linux已经被移植到更多的计算机硬件平台，远远超出其他任何操作系统。Linux可以运行在服务器和其他大型平台之上，如大型计算机和超级计算机。世界上500个最快的超级计算机已100％运行Linux发行版或变种[9]。Linux也广泛应用在嵌入式系统上，如手机（Mobile Phone）、平板电脑（Tablet）、路由器（Router）、电视（TV）和电子游戏机等。在移动设备上广泛使用的Android操作系统就是创建在Linux内核之上。

## 3.3 brk and sbkr

brk 和 sbrk 是 Unix 和类 Unix 操作系统中使用的基本内存管理系统调用，用于控制分配给进程数据段的内存量。 [1]这些函数通常是从更高级别的内存管理库函数（例如 malloc）调用的。在最初的 Unix 系统中，brk 和 sbrk 是应用程序获取额外数据空间的唯一方式；更高版本也允许使用 mmap 调用来完成此操作。

# 4 需求分析

## 4.1功能性需求

实现一个最简单的堆内存分配器，解决内存碎片和多线程并发的问题。能够正确分配与测试。

## 4.2 非功能性需求

### 4.2.1软硬件环境需求

系统应运行于linux平台。

# 5 系统设计

## 5.1设计思路

虽然可以使用低级的 mmap 和 munmap 函数来创建和删除虚拟内存的区域，但是运行时需要额外虚拟内存时，用动态内存分配器更方便，也有更好的可移植性。

动态内存分配器维护着一个进程的虚拟内存区域，称为堆。系统之间细节不同，但是不失通用性，假设堆是一个请求二进制零的区域，它紧接在未初始化的数据区域后开始，并向上生长。分配器将堆视为一组不同大小的块的集合来维护。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。通过分配一段连续内存，然后将它分成固定大小的内存块，每次分配时从空余的内存块中取出一块使用。虽然池分配器也会产生内存碎片，但碎片的大小是固定的，我们可以继续使用它们，池分配器以链表的形式存在。

通过posix提供的线程相关和互斥相关函数，防止程序在多线程环境下进程竞争互斥资源造成的内存破坏。

## 5.2结构图

### 5.2.1 项目结构图



图5.1 项目结构图

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

5.2.2 头文件依赖

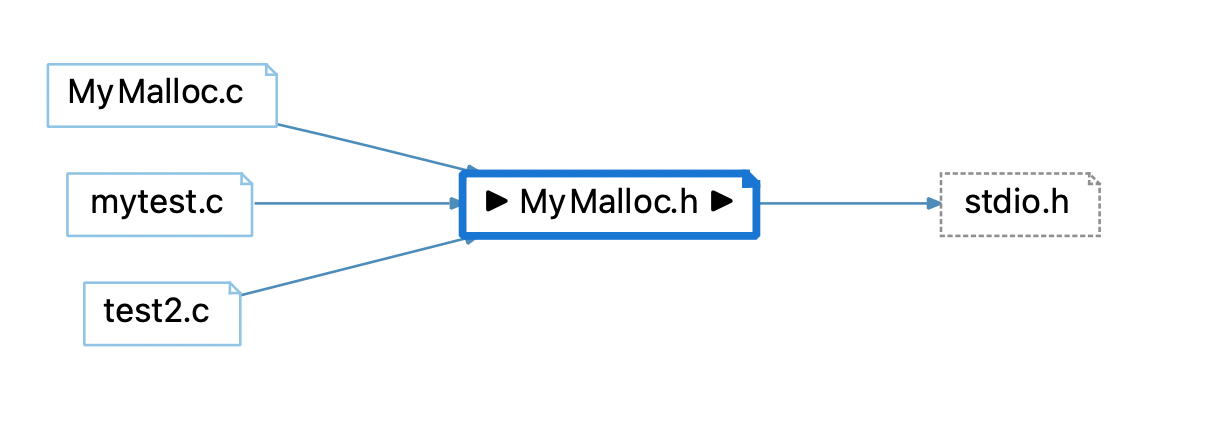
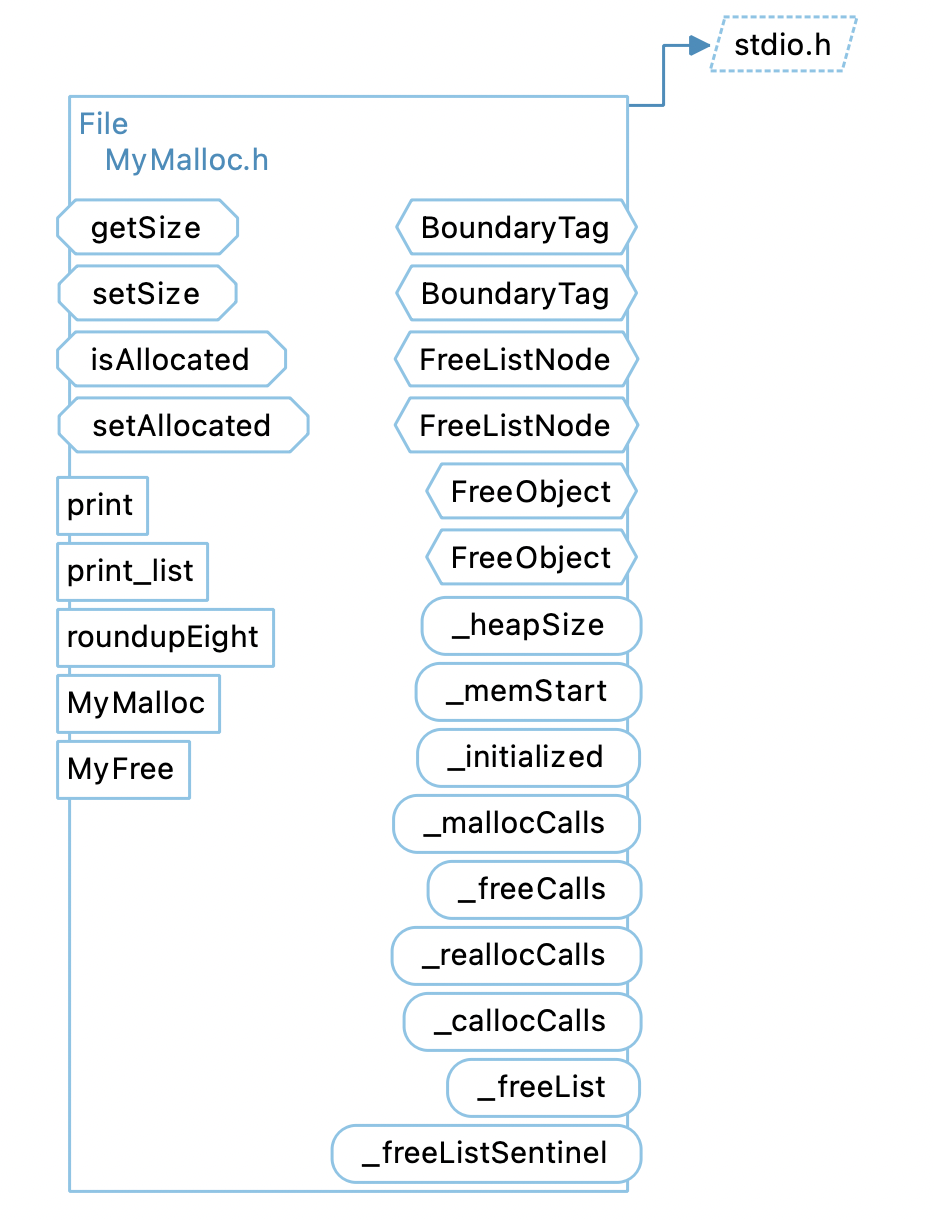
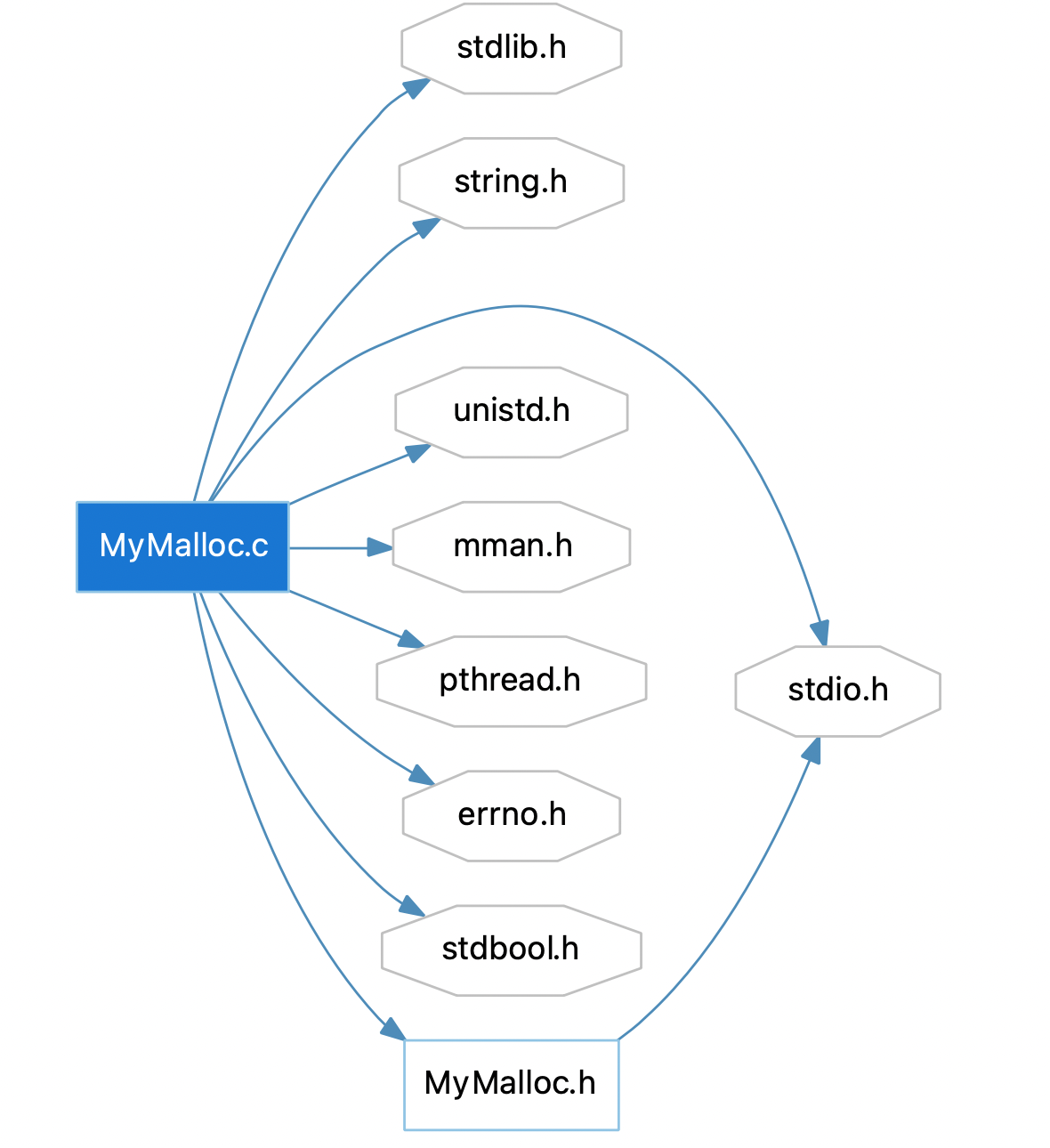


图5.2 头文件依赖

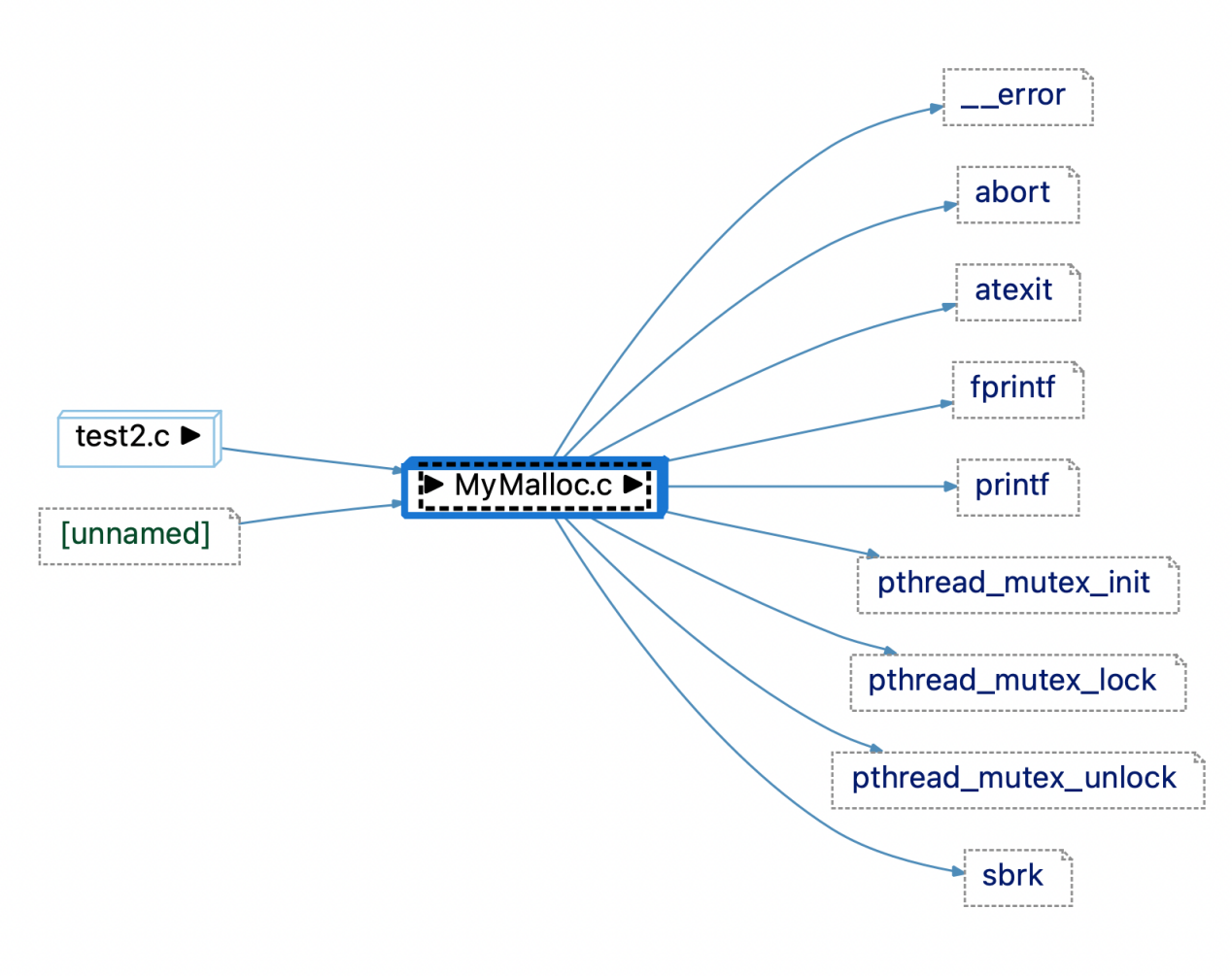
### 5.2.3 头文件属性

图5.3 头文件属性

### 5.2.4 函数依赖情况

图5.4 函数依赖情况

### 5.2.5 函数调用情况



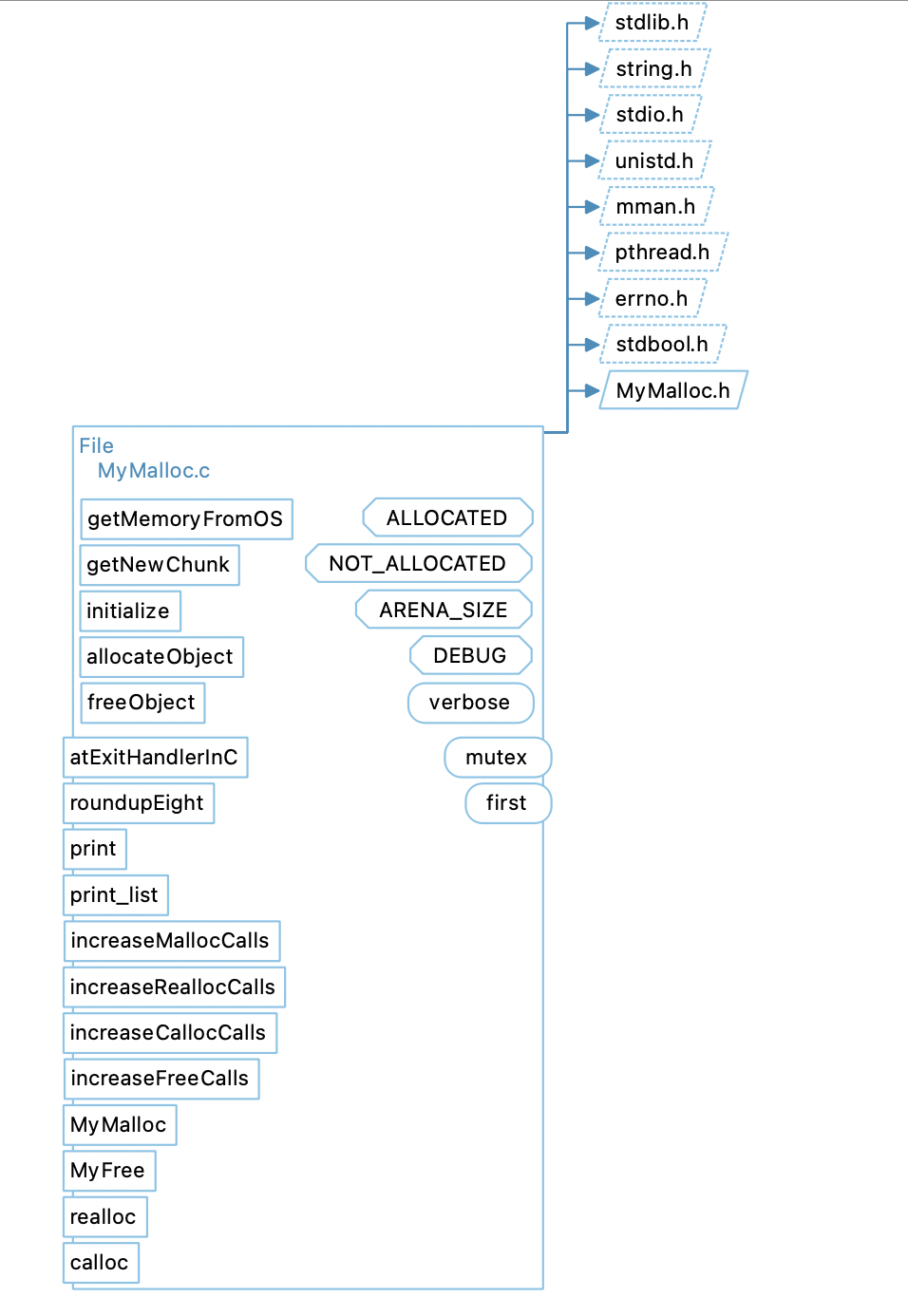
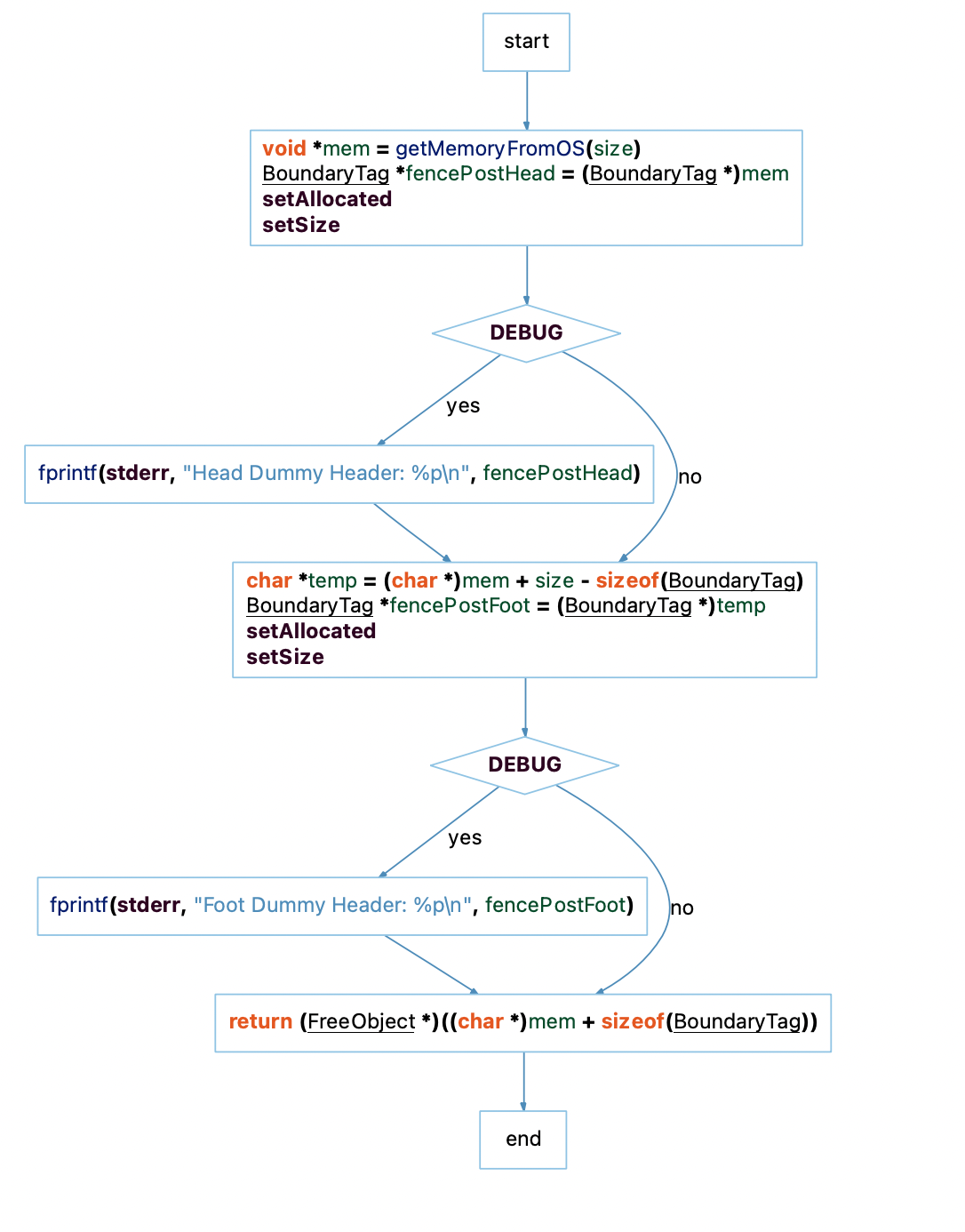


图5.5 函数调用情况

### 5.2.6 common函数具体分析

图5.6 getNewTrunk函数流程

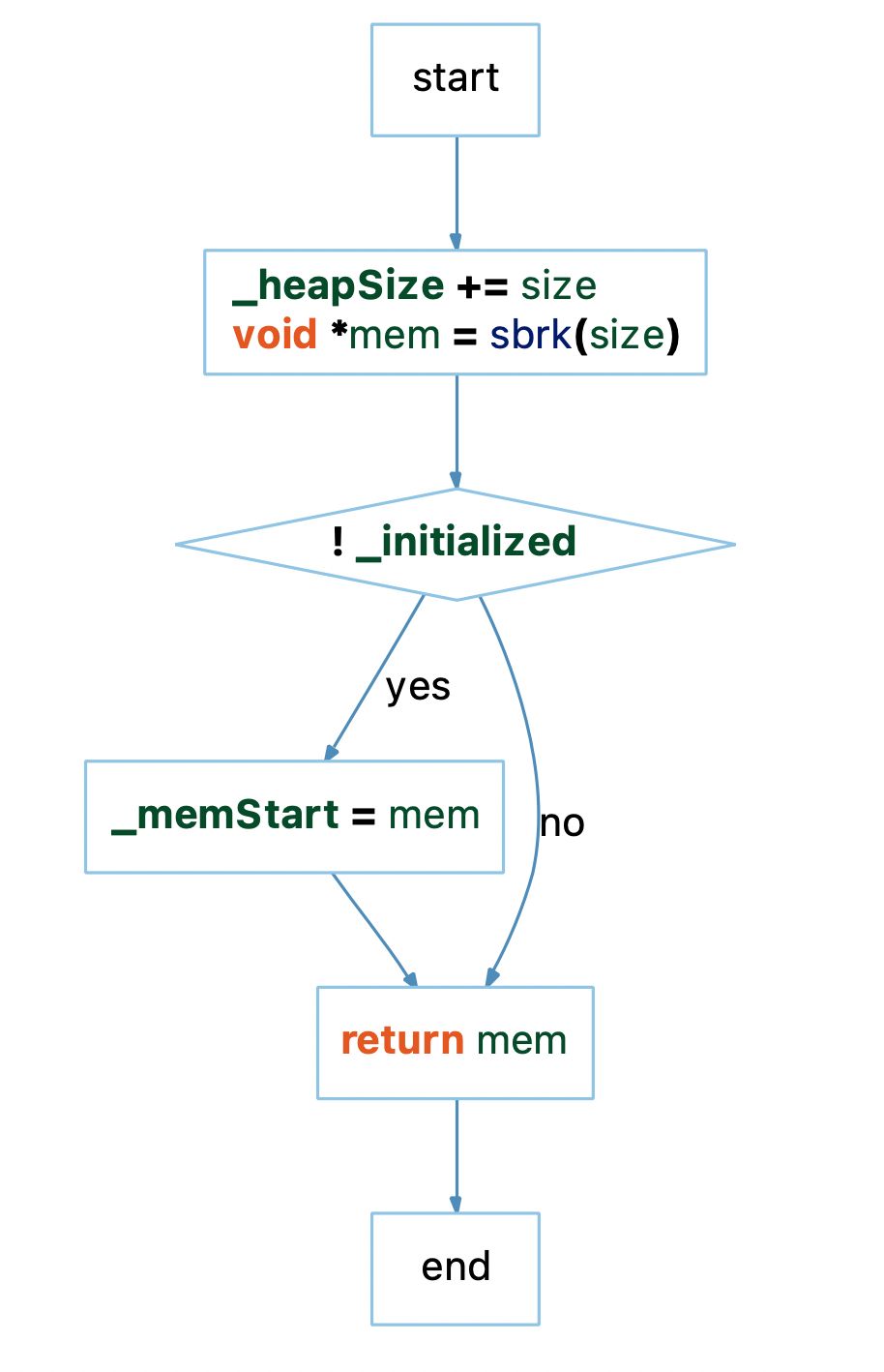


图5.6.1 getMemoryFromOS函数流程

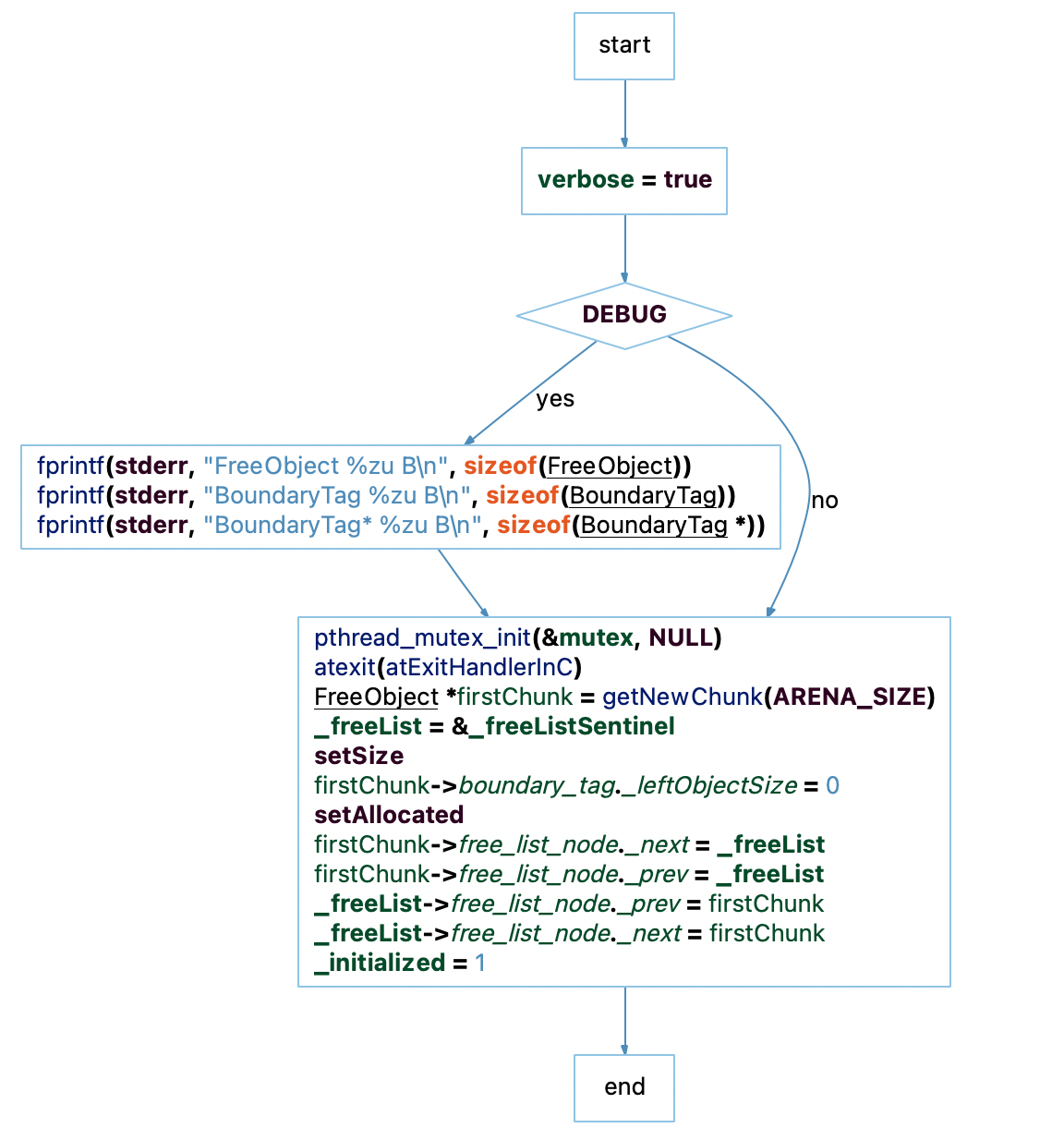


图5.6.2 Initialize函数流程

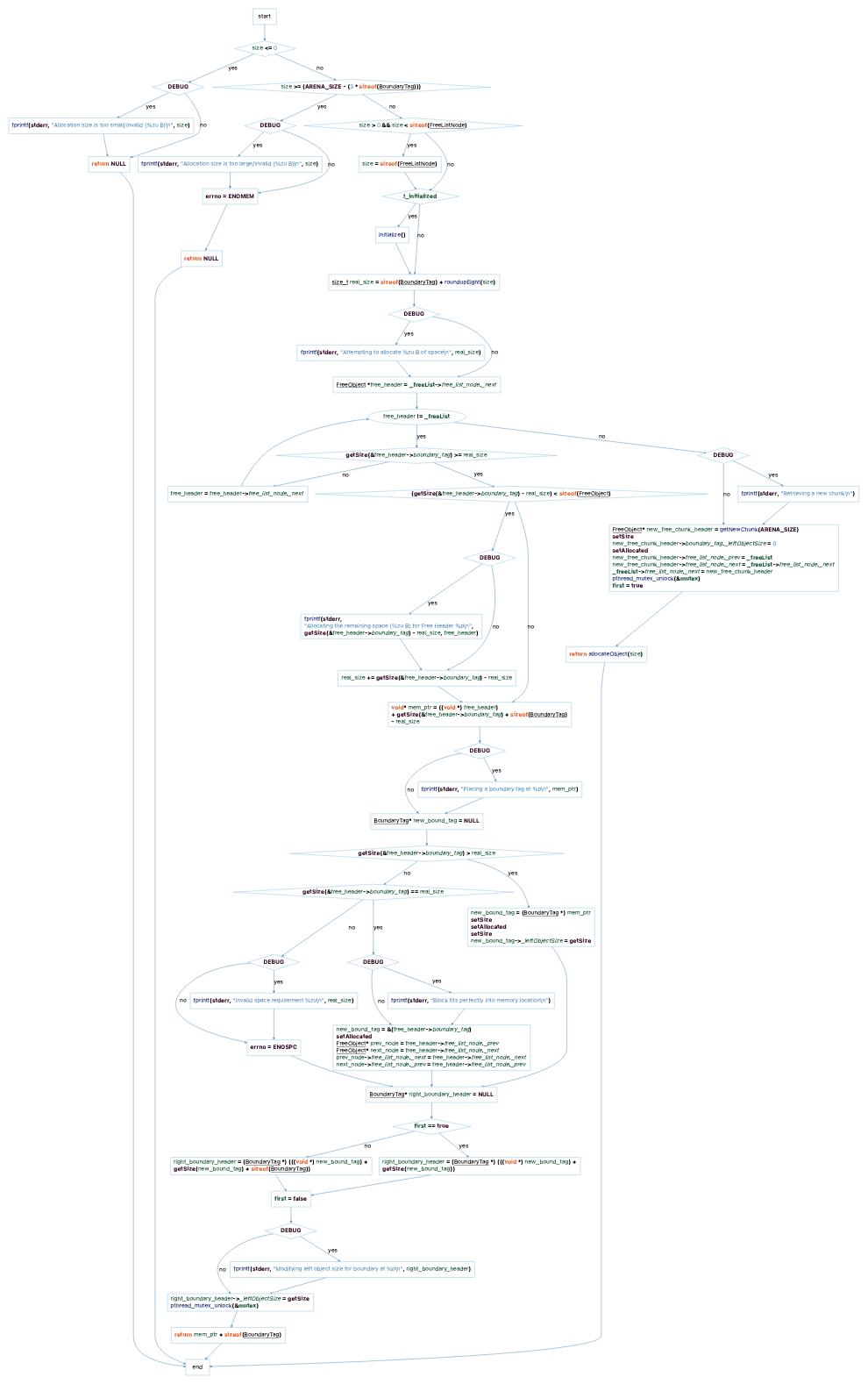


图5.6.3 allocateObject函数流程

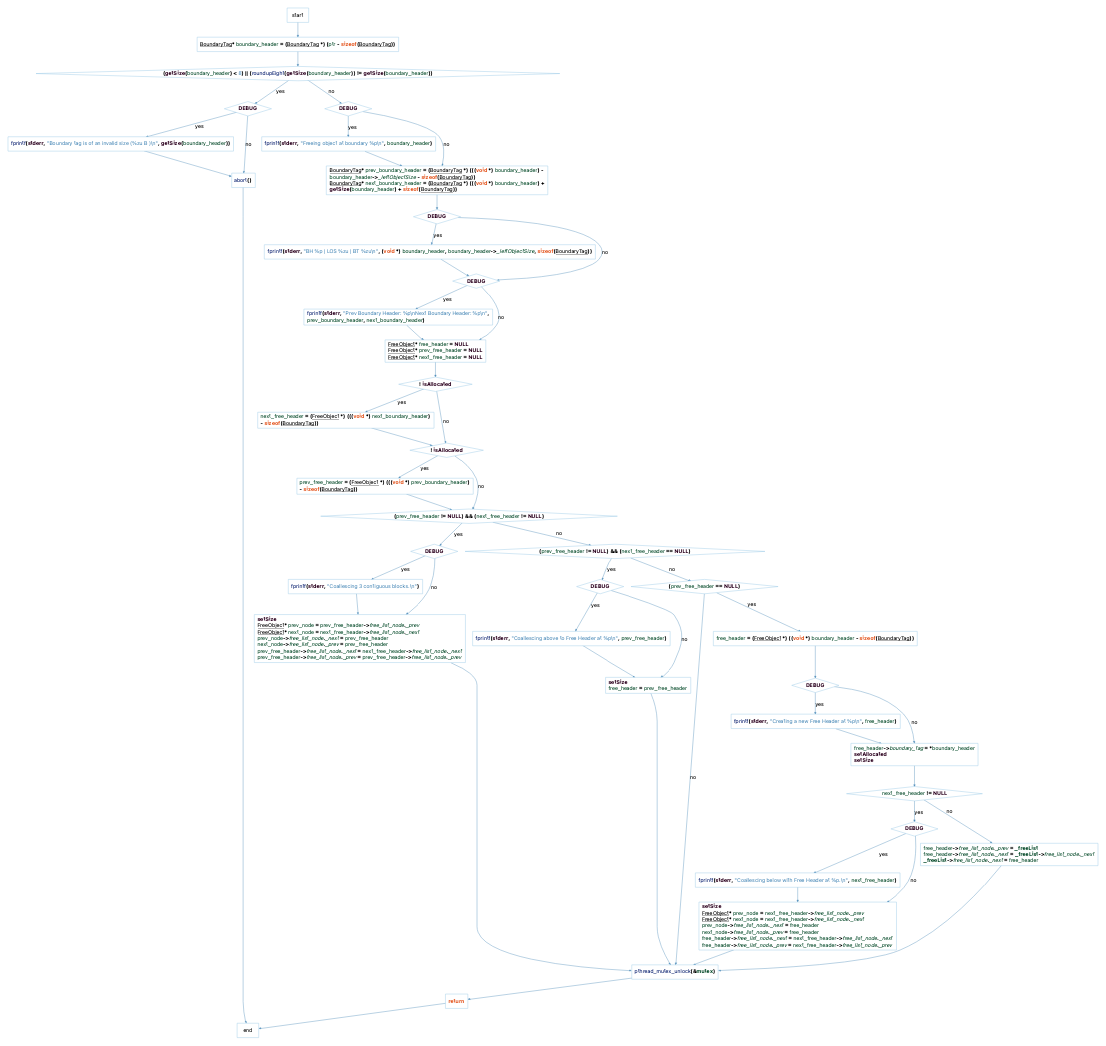


图5.6.3 freeObject函数流程

## 5.3主函数流程图

### 5.3.1 MyMalloc模块



图5.7 MyMalloc函数流程图

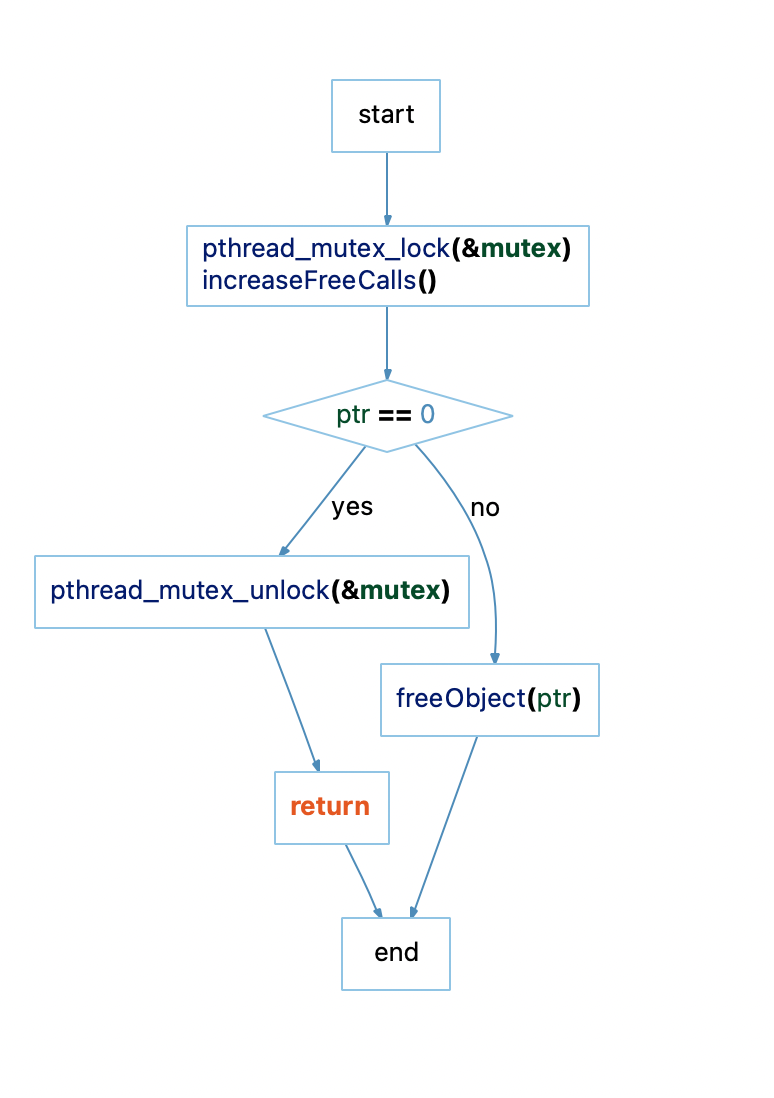


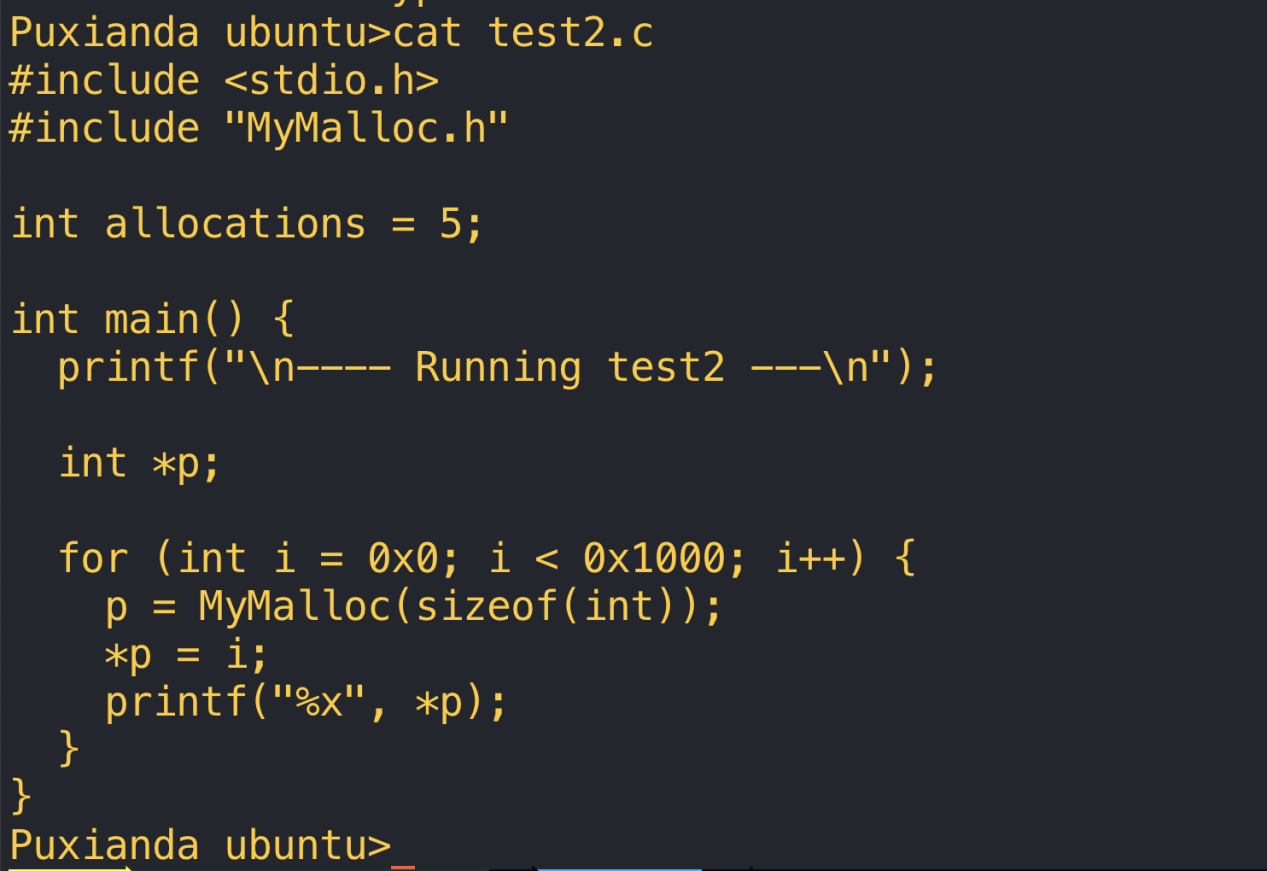
图5.7.1 MyFree函数流程图

# 6 效果分析（测试）

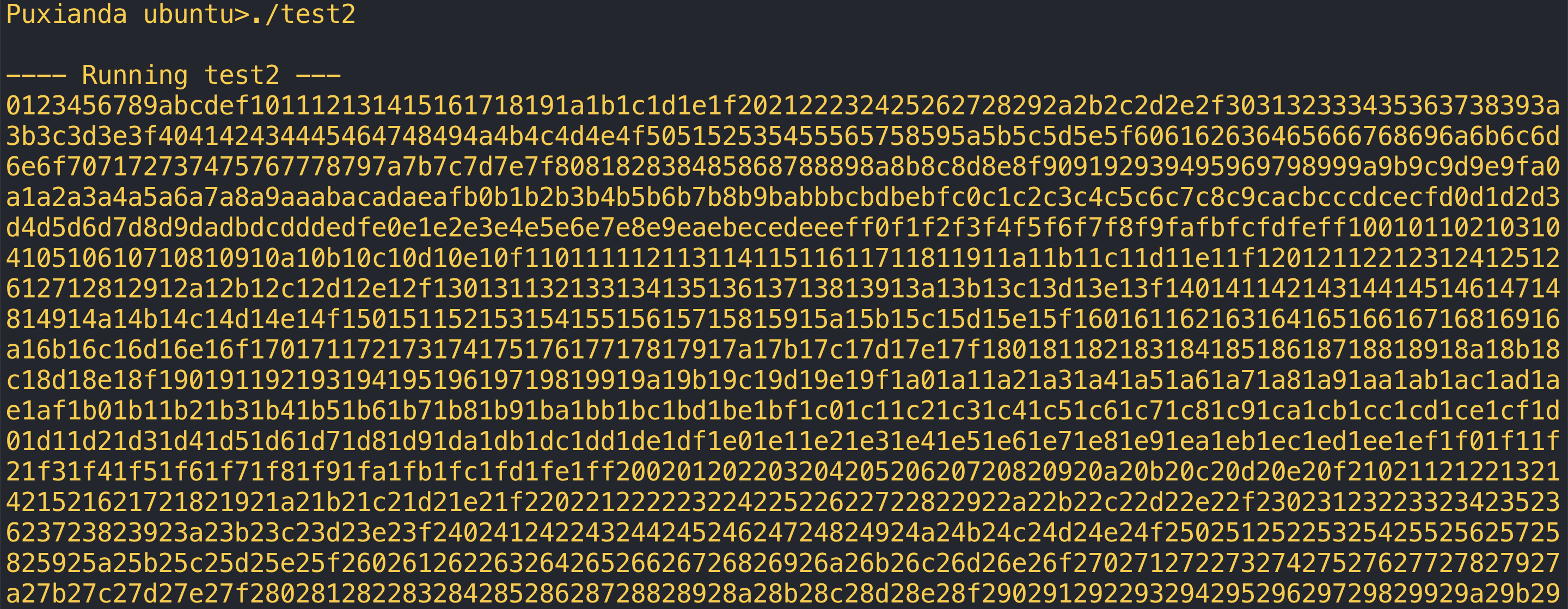
## 6.1 普通测试

源代码:

使用for循环0x1000次进行压力测试



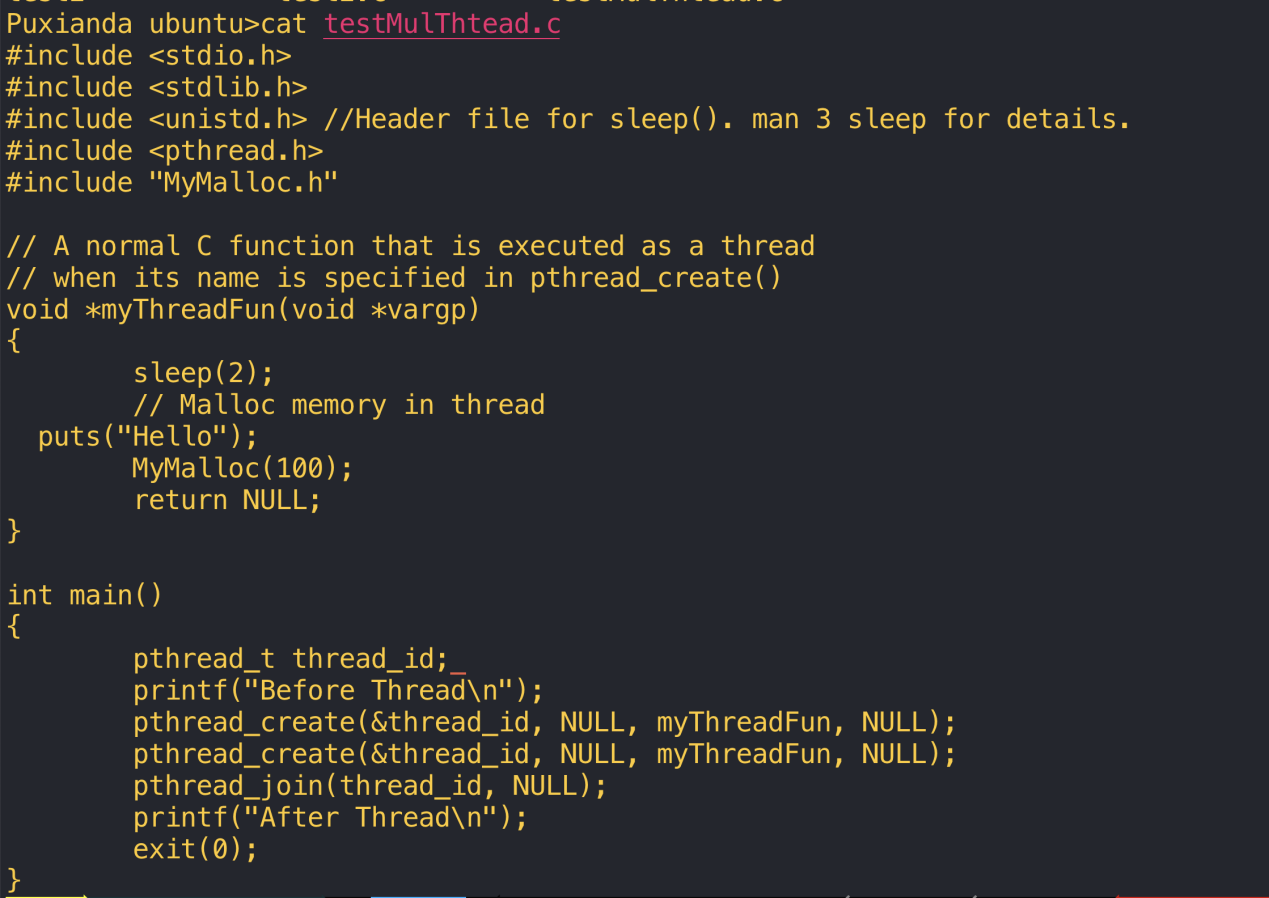
结果



输出0x1-0x1000与预期结果相符，测试通过。

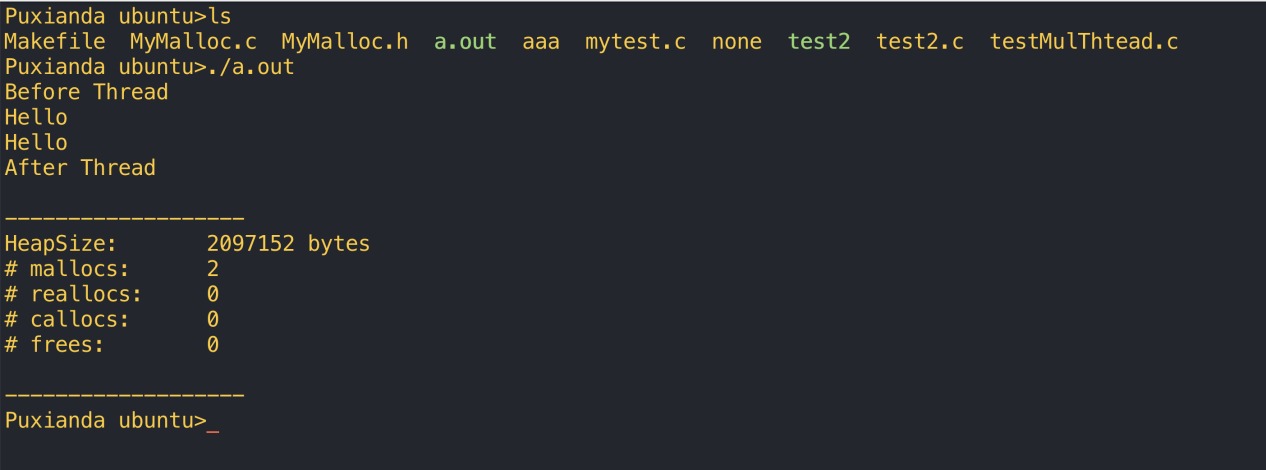
## 6.2 多线程测试

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。

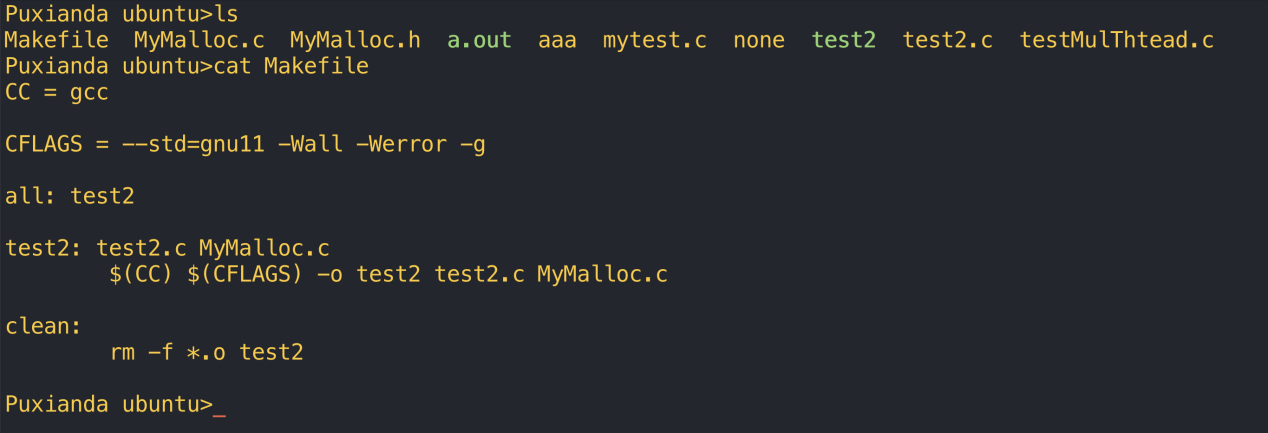
结果:



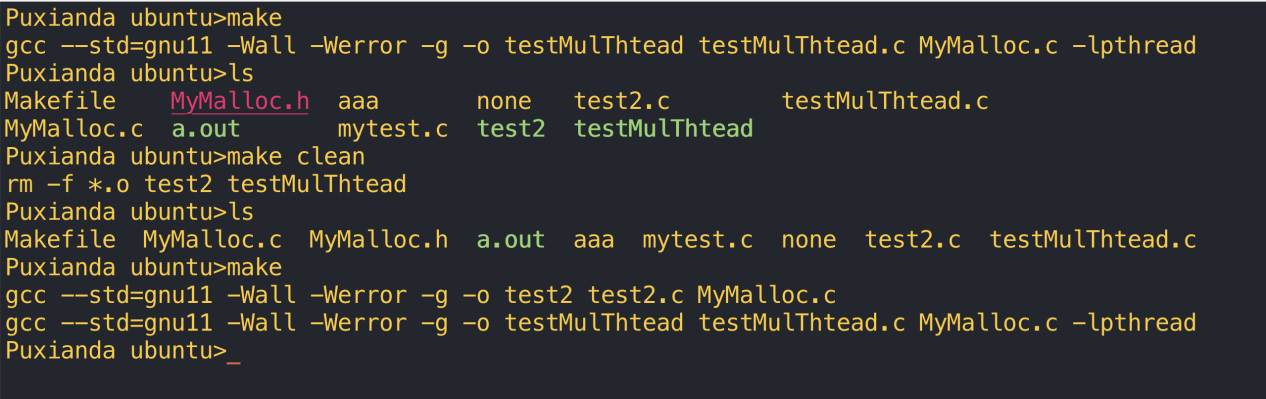
可以看到程序在多线程情况下也能正常工作, 测试通过。

## 6.3 自动化Makefile

源代码:



Make:



# 参 考 文 献

[1]潘爱民著. 《程序员的自我修养:链接,装载与库》2017,(10):22-29