**《操作系统》实验报告**

徽标, 公司名称

描述已自动生成

**题 目 内存管理**

**学 院 计算机学院**

**班 级 2021211304**

**学 号 2021212484**

**姓 名 张梓良**

**2024年 4 月 14**

## 目 录

第一章 实验概述 1

1.1 实验目的 1

1.2 实验内容 1

1.2.1 XXX 1

1.2.2 XXXX 1

1.3 XXXX 1

第二章 实验步骤 2

2.1 XXXX 2

2.1.1 XXXX 2

2.1.2 XXX 2

2.2 XXXX 2

2.2.1 XXX 2

2.2.2 XXX 2

2.3 XXXXX 10

2.3.1 XXXX 10

2.4 XXXXX 12

2.4.1 XXXX 12

2.4.2 XXXX 16

第三章 实验结果分析 22

3.1 XXX 22

3.1.1 XXXX 22

3.1.2 XXXX 22

3.2 XXXXX 22

3.2.1 XXX 22

3.2.2 XXX 24

3.3 XXXX 24

3.3.1 XXXX 24

3.3.2 XXX 26

3.4 XXX 26

3.4.1 XXXXX 26

3.4.2 XXXX 29

3.4.3 XXXX 29

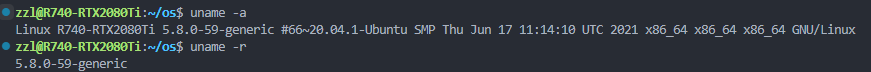
第四章 总结 36

4.1XXXXX 36

# 实验概述

# 实验环境

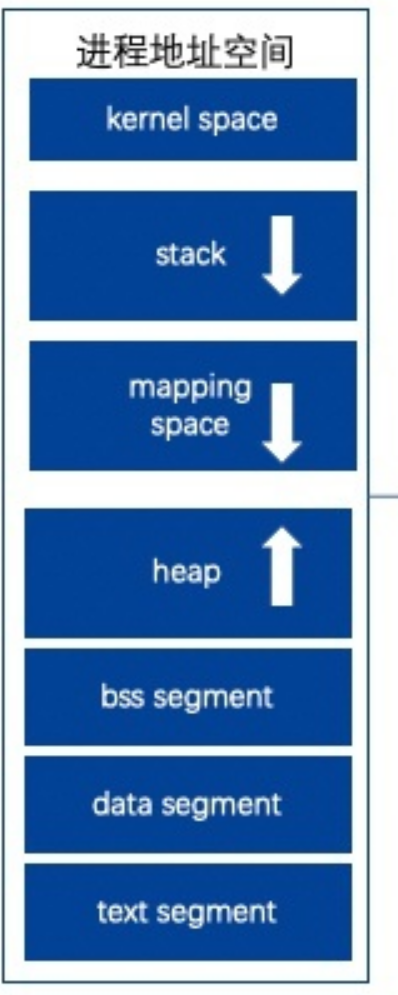
* Linux R740-RTX2080Ti 5.8.0-59-generic #66~20.04.1-Ubuntu SMP Thu Jun 17 11:14:10 UTC 2021 x86\_64 x86\_64 x86\_64 GNU/Linux
* 5.8.0-59-generic



# 实验内容及目的

实验4-1-1 观察进程虚拟地址空间内的变量和数据分布。了解Linux进程虚拟地址空间的组成；编写C语言程序，分配位于堆和栈上的变量，观察变量的地址分布和变化

* 1. Linux进程的虚拟地址空间通常可以分为以下几个部分：
     1. 内核空间：这是操作系统内核和设备驱动程序所使用的空间。在64位架构中，内核空间通常是从0xFFFF\_8000\_0000\_0000到0xFFFF\_FFFF\_FFFF\_FFFF。
     2. 用户空间：这是用户程序所使用的空间。在64位架构中，用户空间通常是从0x0000\_0000\_0000\_0000到0x0000\_7FFF\_FFFF\_FFFF。
     3. 共享库空间：这是共享库文件所使用的空间。在64位架构中，共享库空间通常是从0x0000\_7FFF\_FFFF\_FFFF到0xFFFF\_7FFF\_FFFF\_FFFF。
     4. 堆空间：这是程序运行时使用的动态内存空间。堆空间通常从高地址向低地址扩展，可以使用malloc()和free()等函数来动态分配和释放堆内存。
     5. 栈空间：这是程序运行时使用的函数调用栈空间。栈空间通常从低地址向高地址扩展，可以使用局部变量和函数参数等方式来分配栈内存。
  2. 每一个进程的地址空间如下图所示：



* + 1. 代码段(text)：程序可执行代码，字符串字面值，只读变量，一般为只读区域。代码段指令由操作码和操作对象(或对象地址引用)组成。如果操作对象是立即数(具体数值)，将直接包含在代码中；若是局部数据，将在栈区分配空间，然后引用该数据地址；若位于BSS段和数据段，同样引用该数据地址
    2. 数据段(Data)：程序中已初始化且初值不为0的全局变量和静态局部变量。属于静态内存分配(静态存储区)，可读可写
    3. BSS(Block Started by Symbol)段：未初始化的全局变量和静态局部变量。初始值为0的全局变量和静态局部变量(依赖于编译器实现)。未定义且初值不为0的符号(该初值即common block的大小)。程序运行时，数据段和BSS段的整个区段通常称为数据区，数据段与BSS段的区别如下：
       1. BSS段不占用物理文件尺寸，但占用内存空间；数据段占用物理文件，也占用内存空间。E.g. 对于大型数组如int ar0[10000] = {1, 2, 3, ...}和int ar1[10000]，ar1放在BSS段，只记录共有10000\*4个字节需要初始化为0，而不是像ar0那样记录每个数据1、2、3...，此时BSS为目标文件所节省的磁盘空间相当可观
       2. 当程序读取不在内存的数据段数据时，系统产生page fault，为数据段数据分配相应的物理内存；当程序读取BSS段的数据时，内核会将其转到一个全零页面，不会发生page fault ，也不会为其分配相应的物理内存。
    4. 栈stack：存储局部变量、函数参数、返回地址等，用于为函数内部声明的非静态局部变量(C语言中称“自动变量”)提供存储空间。以及记录函数调用过程相关的维护性信息，称为栈帧(Stack Frame)或过程活动记录(Procedure Activation Record)，包括：函数返回地址，不适合装入寄存器的函数参数及一些寄存器值的保存。除递归调用外，堆栈并非必需。因为编译时可获知局部变量，参数和返回地址 所需空间，并将其分配于BSS段。临时存储区，用于暂存长算术表达式部分计算结果或alloca()函数分配的栈内内存。每个进程/线程都有属于自己的栈，栈的内容随着线程执行动态变化（增长，收缩）。栈有最大值限制RLIMIT\_STACK(通常是8M)，使用Linux命令ulimit -s可查看和设置堆栈最大值
    5. 堆heap：存放进程运行时动态分配的内存段，由程序自己控制，可动态扩张或缩减
       1. 调用malloc(C)/new(C++)等分配内存时，新分配的内存动态添加到堆上(扩张)
       2. 调用free(C)/delete(C++)等释放内存时，被释放的内存从堆中剔除(缩减)

堆管理器通过链表管理程序申请的内存，由于堆申请和释放是无序的，最终会产生内存碎片。堆内存一般由应用程序分配释放，回收的内存可供重新使用。若程序不释放，程序结束时操作系统可能会自动回收。堆中内容是匿名的，无法按名字直接访问，只能通过指针间接访问。堆的末端由break指针标识，当堆管理器需要更多内存时，可通过系统调用brk()和sbrk()来移动break指针以扩张堆，一般由系统自动调用。堆使用时经常出现两种问题：1. 内存破坏：释放或改写仍在使用的内存。2. 内存泄漏：未释放不再使用的内存。释放次数少于申请次数，可能造成内存泄漏。

* + 1. 内存映射段(mmap)：将硬盘文件的内容直接映射到内存—内存映射文件。应用程序使用Windows CreateFileMapping()/MapViewOfFile()或Linux的mmap()系统调用建立这种映射。内存映射是一种方便高效的文件I/O方式， 被用于装载动态共享库。内存映射段用于映射可执行文件用到的动态链接库。用户也可创建匿名内存映射，该映射没有对应的文件, 用于存放程序数据。在 Linux中，通过malloc()请求一大块内存，C运行库将创建一个匿名内存映射，而不使用堆内存。“大块” 意味着比阈值 MMAP\_THRESHOLD还大，缺省为128KB，可通过mallopt()调整。

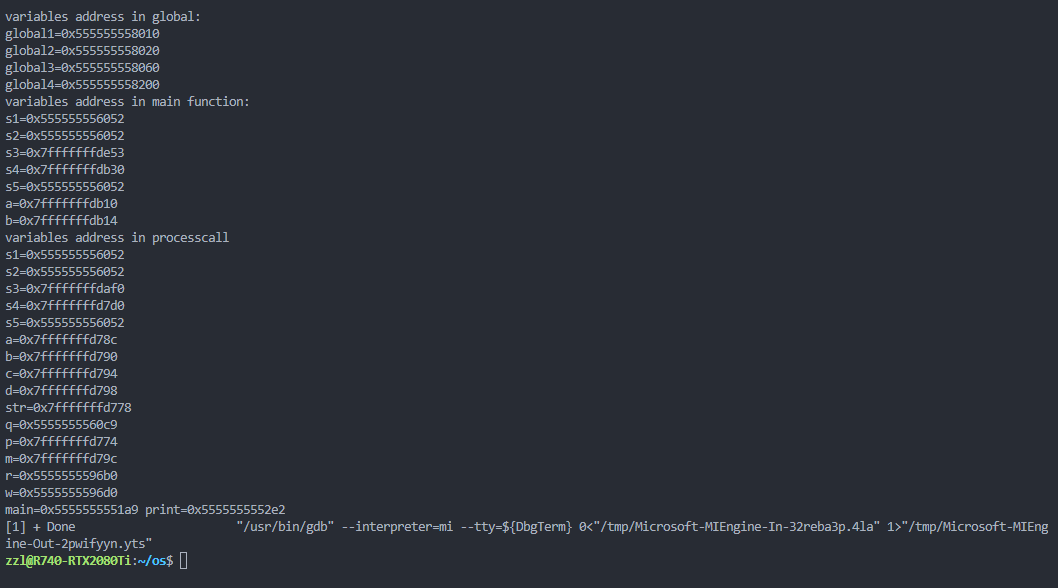
# 实验步骤

# 实验代码

1. #include <stdio.h>
2. #include <malloc.h>
3. void print(char \*, int);
4. int global1[3] = {1, 2, 3};
5. int global2[3] = {4, 5, 6};
6. int global3[100];
7. int global4[100];
8. int main()
9. {
10. printf("variables address in global:\nglobal1=%p\nglobal2=%p\nglobal3=%p\nglobal4=%p\n",
11. global1, global2, global3, global4);
12. *//"abcde"作为字符串常量存储在常量区 s1、s2、s5拥有相同的地址*
13. char \*s1 = "abcde";
14. char \*s2 = "abcde";
15. char s3[] = "abcd";
16. long int \*s4[100];
17. char \*s5 = "abcde";
18. int a = 5;
19. int b = 6; *// a,b在栈上，&a>&b地址反向增长*
20. printf("variables address in main function:\ns1=%p\ns2=%p\ns3=%p\ns4=%p\ns5=%p\na=%p\nb=%p\n",
21. s1, s2, s3, s4, s5, &a, &b);
22. printf("variables address in processcall\n");
23. print("ddddddddd", 5);
24. *// 参数入栈从右至左进行,p先进栈,str后进 &p>&str*
25. printf("main=%p print=%p \n", main, print);
26. *// 打印代码段中主函数和子函数的地址，编译时先编译的地址低，后编译的地址高main < print*
27. }
28. void print(char \*str, int p)
29. {
30. char \*s1 = "abcde"; *// abcde在常量区，s1在栈上*
31. char \*s2 = "abcde"; *// abcde在常量区，s2在栈上，s2 - s1可能等于0，编译器优化了相同的常量，只在内存保存一份*
32. *// 而&s1>&s2*
33. char s3[] = "abcdeee"; *// abcdeee在常量区，s3在栈上，数组保存的内容为abcdeee的一份拷贝*
34. long int \*s4[100];
35. char \*s5 = "abcde";
36. int a = 5;
37. int b = 6;
38. int c;
39. int d; *// a,b,c,d均在栈上，&a>&b>&c>&d地址反向增长*
40. char \*q = str;
41. int m = p;
42. char \*r = (char \*)malloc(1);
43. char \*w = (char \*)malloc(1); *// r<w 堆正向增长*
44. printf("s1=%p\ns2=%p\ns3=%p\ns4=%p\ns5=%p\na=%p\nb=%p\nc=%p\nd=%p\nstr=%p\nq=%p\np=%p\nm=%p\nr=%p\nw=%p\n",
45. s1, s2, s3, s4, s5, &a, &b, &c, &d, &str, q, &p, &m, r, w);
46. */\* 栈和堆是在程序运行时候动态分配的，局部变量均在栈上分配。*
47. 栈是反向增长的，地址递减；malloc等分配的内存空间在堆空间。堆是正向增长的，地址递增。
48. r,w变量在栈上(则&r>&w)，r,w所指内容在堆中(即r<w)。\*/
49. }

# 实验结果及分析

## 实验结果



## 结果分析

**variables address in global:**

* global1,global2是指向已经初始化的全局数组的指针，全局数组位于**Data segment**。
* global3,global4是指向未初始化的全局数组的指针，全局数组位于**Bss segment**。

**variables address in main function:**

* s1、s2、s5是指向常量“abcde”的指针（所以它们的值相同），s1、s2、s5位于**Stack**，常量“abcde”位于**Text segment。**
* s3是指向常量“abcdeee”副本的指针，s3位于**Stack，**s3指向的常量副本也在**Stack。**
* s4是指向一个指针数组，s4位于**Stack，**s4指向的指针数组也在Stack**。**
* a,b为变量，它们都位于**Stack，**同时&a>&b地址反向增长。

**variables address in processcall：**

* s1、s2、s5是指向常量“abcde”的指针（所以它们的值相同），s1、s2、s5位于**Stack**，常量"abcde"位于**Text segment。**
* s3是指向常量“abcdeee”副本的指针，s3位于**Stack，**s3指向的常量副本也在**Stack。**
* s4是指向一个指针数组，s4位于**Stack，**s4指向的指针数组也在Stack**。**
* a,b,c,d为变量，它们都位于**Stack，**同时&a>&b>&c>&d地址反向增长。
* str和p为函数的参数，位于**Stack，**参数是从右往左入栈，p先入栈，str后入栈，&str<&p。
* q是指向常量“ddddddddd”的指针，q位于**Stack，**常量“ddddddddd”位于**Text segment。**
* m为变量，m位于**Stack。**
* r和w是指向动态分配的内存的指针，r和w位于**Stack，**它们指向的内容位于**Heap，**同时堆是正向增长的，所以r<w**。**

# 总结

在这次期末大作业论文撰写中，曾经无数次的困难和技术难关困扰着我，曾经有过放弃本系统的念头，曾经有过敷衍了事的想法，但在XXX学院XXX专业同学们共同的帮助下和相互间的鼓舞下，所面临的技术问题都一步一步地解决了。每一次技术突破的喜悦，都让我坚定的胜利的信念。让我深深地体会到进行学术研究开发不是一件简单的事情，它需要设计者具有全面的专业知识、慎密的思维、严谨的工作态度以及较高的分析问题、解决问题的能力，而我在很多方面还有欠缺。虽然如此，我一直都相信，when there is a will,there is a way.（有志者事竟成）。

##### 参考文献

[1] 李建彬.一键排版人工智能引擎研究:[硕士学位论文].上海:华西理工大学，2015.

[2] 程元军.一键排版PaperYes的理论基础.北京:机械出版社，2007.11.

[3] 陈炜煜，李建彬.毕业论文格式要求.特区经济，2013，（2）:221-222.

[4] 樊伟.论项目时间管理.现代商贸工业，2010，（5）:46-47.

[5] 郝利华.JC公司基于PDM系统的研发项目进度管理研究: [硕士学位论文]. 济南:山东大学，2015.

[6] 何国伟.管理质量的三要素:效果、效率及可靠性.质量与可靠性，2007，（4）:4-6.