**广州大学学生实验报告**

**开课学院及实验室：**计算机科学与教育软件学院计算机软件实验室 **2020年 9月 23日**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院** | **计算机科学与教育软件学院** | **年级/专业/班** | **计科182** | **姓名** | 莫广贤 | **学号** | 1806100104 |
| **实验课程名称** | Unix/linux操作系统分析实验 | | | | | **成绩** |  |
| **实验项目名称** | 内存分配与回收 | | | | | **指导老师** | 陶文正 |

**实验二 内存分配与回收**

1. **实验目的**

1.了解静态内存与动态内存的区别；

2.理解动态内存的分配和释放原理；

3.掌握如何调整动态内存的大小；

4.利用链表实现动态内存分配。

1. **实验环境**

Ubuntu20.04

1. **实验内容**

1.利用malloc和 calloc函数实现动态内存的分配；利用free函数实现动态内存的释放；

2.利用realloc函数实现调整内存空间的大小；

3.利用链表实现动态内存分配。

**四、 实验原理**

静态内存与动态内存

按分配内存空间的方式不同，一个程序所使用的内存区域可以分为静态内存与动态内存。在程序开始运行时有系统分配的内存称为静态内存，在程序运行过程中由用户自己申请分配的内存称为动态内存。

静态内存的申请是由编译器来分配的。对于用户程序中的各种变量，编译器在编译源程序时处理了为各种变量分配所需内存的工作。当程序执行时，系统就为变量分配所需的内存空间，至使用该变量的函数执行完毕返回时，自动释放所占用的内存空间。使用静态内存对用户来说是很方便的。用户并不需要了解分配内存的具体细节，也不需要时刻考虑由于程序结束前未释放所占用的内存空间而带来的可用内存泄漏。同时，静态内存也是不通过指针而使用变量的唯一方法。但是，静态内存也存在一定的缺陷：首先，由于静态内存总是预先定义了存放数据的数组大小，这就有可能因为所传入的数据量大于数组容量而引发的溢出问题。或因为定义了一个大数组，而所传入的数据量远小于数组容量，而对内存空间造成浪费。

其次，由于在某个函数中分配的静态内存将在此函数运行结束时被系统自动释放，使用指针由子函数向主函数传递数据的设想是无法被实现的。

使用动态内存时，用户可以根据需要随时申请内存，使用完毕后手动将此内存区释放。在实际应用中非常方便。但动态内存的使用也存在着巨大的隐患。任何处理过大型项目的用户都知道动态内存的使用会使内存管理变得多么复杂，以及要确切地记得在使用完毕后释放所占用的内存空间是多么困难的事情。在大型应用程序中，由于在释放某块动态内存前将指向该内存区域的指针重新赋值，从而使得此内存区域无法被释放的情况是十分常见的。通常将内存分配后没有被释放而导致可用内存减少称之为内存泄漏。避免内存泄漏耗尽系统资源正是许多服务器每隔一段时间就需要重新启动的原因。另外还要注意，由于分配动态内存时，用户得到的是一块void类型的内存，用户可以将其作为任何类型的内存空间使用，也可能引发一些无法预计的结果。

1. 动态内存的分配

分配动态内存空间所使用的函数调用如下：

#include<stdio.h>

void \*malloc(size\_t size);

void \*calloc(size\_t nmemb, size\_t size);

函数malloc和calloc都是用于分配动态内存空间的函数。

函数malloc的参数size表示申请分配的内存空间的大小，以字节记。

函数calloc的参数nmemb表示申请分配的内存空间占的数据项数目，参数size表示一个数据项的大小，以字节记。也就是说，calloc函数分配大小为nmemb\*size大小的内存空间。

函数calloc与函数malloc的最大区别就是函数calloc将初始化所分配的内存空间，把所有位置0。调用成功时，函数calloc与函数malloc的返回值都为被分配的内存空间的指针；调用失败时，返回值为NULL。

1. 动态内存的释放

释放动态内存空间所使用的函数调用如下：

#include<stdio.h>

void free(void \*ptr);

此函数的作用是释放由函数calloc或函数malloc分配的动态内存。参数ptr是指向要释放的动态内存的指针。

注意：当动态内存被释放后，原来指向它的指针就会变为悬空指针。此时使用该指针将会产生错误。

下面的例子就是使用动态内存实现将子函数中分配的内存空间指针返回主函数，实现数据的传递。

#include <stdio.h>

#include <string.h>

char \*upcase(char \*inputstring);

int main(void)

{

char \*str1, \*str2;

str1=upcase("Hello" );

str2=capitao("Goodbye");

printf("str1=%s, str2=%s\n", str1, str2);

free(str1);

free(str2);

return 0;

}

char \*upcase(char \*inputstring)

{

char \*newstring;

int counter;

if(!(newstring=malloc(strlen(inputstring)+1)))

{

printf("ERROR ALLOCATING MEMORY! \n");

exit(255);

}

strcpy(newstring, inputstring);

for(counter=0; counter<strlen(newstring); counter++)

{

if(newstring[counter]>=97&&newstring[counter]<=122)

newstring[counter]-=32;

}

return newstring;

}

在这个程序中由于使用了动态内存，使得子函数可以灵活地分配所需的内存空间。

1. 调整动态内存的大小

对于用函数calloc与函数malloc分配好的动态内存，可以使用realloc函数来调整它的大小。该函数的说明如下：

#include<stdio.h>

void\*realloc(void \*ptr, size\_t size);

realloc函数的作用是重新调整一块动态内存区域的大小，参数ptr是指向要调整的动态内存的指针，应是函数calloc与函数malloc的返回值。参数size是新定义的动态内存的大小。Size可以大于或小于动态内存的原大小，调用realloc函数时，通常是在原来的内存空间调整动态内存的大小，原有数据不被改动。当size大于原大小，而原位置中无法完成调整时，将重新开辟内存空间并将原数据拷贝到新的内存空间中。

注意：如果参数ptr为NULL，则函数realloc的作用相当于函数malloc。如果参数size为0，则函数realloc的作用相当于函数free。

下面的例子说明了该函数的应用：在这个程序中，针对函数upcase的参数newstring是否为NULL，采取了不同的处理方式。如果newstring不为NULL，则直接分配内存空间。否则调整原空间的大小以适应新的需要。

#include <stdio.h>

#include <string.h>

void upcase(char \*inputstring, char \*newstring);

int main(void)

{

char \*string;

upcase("Hello",string);

printf("str1=%s \n", string);

capitao("Goodbye", string);

printf("str2=%s\n", string);

free(string);

return 0;

}

void upcase(char \*inputstring, char \*newstring)

{

int counter;

if(!newstring)

{

if(!(newstring=realloc(NULL, strlen(inputstring)+1)))

{

printf("ERROR ALLOCATING MEMORY! \n");

exit(255);

}

}

else

{

if(!(newstring=realloc(newstring, sizeof(inputstring)+1)))

{

printf("ERROR REALLOCATING MEMORY! \n");

exit(255);

}

}

strcpy(newstring, inputstring);

for(counter=0; counter<strlen(newstring); counter++)

{

if(newstring[counter]>=97&&newstring[counter]<=122)

newstring[counter]-=32;

}

return ;

}

1. 使用链表进行动态内存的分配

虽然使用动态内存可以方便地使用内存，但动态内存也有局限性，就是在数据输入到程序之前必须知道数据的大小，以便申请相应的动态内存。然而，在很多情况下，用户都无法事先知道这个值，因而也就无法申请相应的内存空间。对于这种事先未知大小的数据输入，可以使用链表将其分块保存。

链表是一种动态地进行存储分配的结构。链表中的各个元素是一个结构，每个元素称为链表的一个结点。此结构中包含有一个指向此结构的指针，用于指向链表中的下一个结点。链表的最后一个结点的指针NULL，表示链表结束。

下面的例子说明了使用链表获得动态内存的方法：这个程序将终端输入的一系列字符串用链表的形式保存下来。然后再将这些数据组装起来，回显到输出终端。链表的结点为stringdata结构。stringdata结构中的整型量iscontinuing用于表示当前结点是否为链表的末尾。如果iscontinuing有值，则表示此结点不是链表的末尾。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define DATASIZE 10

typedef struct stringdata

{

char \*string;

int iscontinuing;

struct stringdata \*next;

} mydata;

mydata \*append(mydata \*start, char \*input);

void displaydata(mydata \*start);

void freedata(mydata \*start);

int main(void)

{

char input[DATASIZE];

mydata \*start=NULL;

printf("ENTER SOME DATA,AND PRESS Ctrl+D WHEN DONE. \n");

while(fgets(input, sizeof(input), stdin))

{

start=append(start, input);

}

displaydata(start);

freedata(start);

return 0;

}

mydata \*append(mydata \*start, char \*input)

{

mydata \*cur=start, \*prev=NULL, \*new;

while(cur)

{

prev=cur;

cur=cur->next;

}

cur=prev;

new=malloc(sizeof(mydata));

if(!new)

{

printf("COULDN’T ALLOCATE MEMORY! \n");

exit(255);

}

if(cur)

cur->next=new;

else

start=new;

cur=new;

if(!(cur->string=malloc(sizeof(input)+1)))

{

printf("ERROR ALLOCATING MEMORY! \n");

exit(255);

}

strcpy(cur->string, input);

cur->iscontinuing=!(input[strlen(input)-1]=='\n'||input[strlen(input)-1]=='\r');

cur->next=NULL;

return start;

}

void displaydata(mydata \*start)

{

mydata \*cur;

int linecounter=0, structcounter=0;

int newline=1;

cur=start;

while(cur)

{

if(newline)

printf("LINE %d:",++linecounter);

structcounter++;

printf("%s",cur->string);

newline=!cur->iscontinuing;

cur=cur->next;

}

printf("THIS DATA CONTAINED %d LINES AND WAS STORED IN %d STRUCTS. \n",

linecounter,structcounter);

}

void freedata(mydata \*start)

{

mydata \*cur, \*next=NULL;

cur=start;

while(cur)

{

next=cur->next;

free(cur->string);

free(cur);

cur=next;

}

}

**五、实验中用到的系统调用函数（包括实验原理中介绍的和自己采用的）**

exit()

**六、实验步骤**

1. 利用malloc函数和 free函数来实现动态内存的申请和释放；
2. 利用realloc函数来修改上述malloc函数申请的动态内存的大小；
3. 利用链表结构，malloc函数和 free函数实现将终端输入的一系列字符串用链表的形式保存下来。然后再将这些数据组装起来，回显到输出终端。

**七、实验结果分析（截屏的实验结果，与实验结果对应的实验分析）**

1.



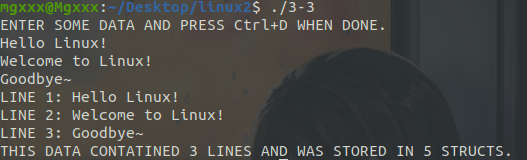
该程序主要的函数时upcase()函数。该函数在内部创建了一个指针newstring，用malloc()函数向其分配内存，然后把传进来的参数全部转化为大写放入newstring中，并返回newstring。主函数打印输出了经过upcase()处理的结果，在最后通过free()函数释放了申请的内存。

2.



该C程序中的upcase()函数第二个参数newstring是二级指针，是因为主函数中要传入的是一级指针的地址。\*newstring为传入二级指针指向的一级指针的地址。当传入的\*newstring为空值时，则向\*newstring分配内存。当\*newstring不为空值时，则根据第一个参数inputstring来调整\*newstring的大小。主程序在处理完字符产后，释放新开辟的空间。

3.



C程序中结构stringdata为链表的结点。string用来存储字符串数据，宏定义中DATASIZE定义了一次从输入流中接收多少个字符。每次调用append都会新建一个链表结点并开辟一个空间。实验结果中有3行数据并使用了5个结构体来存储数据。第一行字符串长度大于10小于20，使用了两个结构体，第二行数据也是使用了两个结构体，第三行字符串长度小于等于10，所以只占用了一个结构体，总共占用了五个结构体。

**八、实验总结**

本次实验了解了静态内存和动态内存的区别，学会了使用malloc分配内存和使用remalloc调整动态内存大小，使用free来释放分配的内存。还学会了使用链表实现动态内存的分配。最后还加深了对指针的理解。

**九、实验数据及源代码（学生必须提交自己设计的程序源代码，并有注释，源代码电子版也一并提交），包括思考题的程序。**

源码数据见同目录下的文件夹” 1806100104-莫广贤-实验1-计科182”。