

# 深圳大学实验报告

课程名称： 操作系统

实验项目名称： 内存分配与回收

学院： 计算机与软件学院

专业： 软件工程

指导教师： 张 滇

报告人： 郑彦薇 学号： 2020151022 班级： 软件工程 01 班

实验时间： 2023 年 05 月 11、18、25 日

实验报告提交时间： 2023/5/25

教务处制

### 一、实验目的与要求：

1. 加深对内存分配与使用操作的直观认识；
2. 掌握 Linux 操作系统的内存分配与使用的编程接口；
3. 了解 Linux 操作系统中进程的逻辑编程地址和物理地址间的映射。

### 二、方法、步骤：

#### 1. 实验相关知识内容：

##### 1.1. 内存分配、释放方法：

程序通过 C 库里的函数 `malloc` 函数进行动态分配内存。在 `malloc` 申请内存时，会有两种方式向操作系统申请堆内存：①当用户分配的内存小于 128KB 时，通过 `brk()` 系统调用从堆分配内存；②当用户分配的内存大于 128KB 时，通过 `mmap()` 系统调用在文件映射区域分配内存。

通过 `malloc()` 函数分配完动态内存并在程序使用完这些内存后，要使用 `free()` 函数释放掉该块内存空间，以免造成内存泄露等问题。`free` 函数的参数对应要被释放的内存区块。

##### 1.2. 逻辑编程地址、物理地址以及两者之间的映射：

Linux 操作系统中，逻辑地址空间是进程可见的地址空间，物理地址空间是实际的内存地址空间。进程的逻辑地址空间中的地址是虚拟地址，需要通过映射才能转换为物理地址。两者之间的映射过程：linux 操作系统中进程的逻辑编程地址和物理地址间的映射是通过虚拟内存管理模块实现的。虚拟内存管理模块将进程的虚拟地址空间划分成多个页，并将每个页映射到物理内存中的一个页框，然后维护一个页表，用于记录每个页的映射关系。当进程访问一个虚拟地址时，虚拟内存管理模块会将该地址转换为页号和页内偏移量。然后虚拟内存管理模块查找页表，找到该页对应的物理页框的地址，再将页内偏移量添加到物理页框的地址中，得到实际的物理地址。

#### 2. 程序设计思路在下一部分结合代码进行说明。

### 三、实验过程及内容：

1. 借助 google 工具查找资料，学习 Linux `proc` 文件系统中关于内存映射的部分内容（了解 `/proc/pid/` 目录下的 `maps`、`status`、`smmap` 等几个文件内部信息的解读）

#### 1.1. 进入 `proc` 目录：

通过 `cd` 命令进入 `proc` 目录，然后使用 `ps` 命令查看当前正在运行进程的 `pid`，这里我选择 2339 目录对其中的文件内部信息进行查看：

```
zhengyanwei_2020151022@ubuntu:~$ cd /proc
```

```
zhengyanwei_2020151022@ubuntu:/proc$ ps
  PID TTY          TIME CMD
 2339 pts/0        00:00:00 bash
 2347 pts/0        00:00:00 ps
zhengyanwei_2020151022@ubuntu:/proc$ cd /proc/2339
```

#### 1.2. `Maps` 信息解读：

输入 `cat maps`，可以得到如下图所示结果：

```

zhengyanwei_2020151022@ubuntu:/proc/2339$ cat maps
55cecc68000-55cecc6b3000 r--p 00000000 08:05 1048793 /usr/bin/bash
55cecc6b3000-55cecc764000 r-xp 0002d000 08:05 1048793 /usr/bin/bash
55cecc764000-55cecc79b000 r--p 000de000 08:05 1048793 /usr/bin/bash
55cecc79b000-55cecc79f000 r--p 00114000 08:05 1048793 /usr/bin/bash
55cecc79f000-55cecc7a8000 rw-p 00118000 08:05 1048793 /usr/bin/bash
55cecc7a8000-55cecc7b2000 rw-p 00000000 00:00 0
55cecd664000-55cecd7a9000 rw-p 00000000 00:00 0
7f434a508000-7f434a50b000 r--p 00000000 08:05 1059064 /usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
7f434a50b000-7f434a512000 r-xp 00003000 08:05 1059064 /usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
7f434a512000-7f434a514000 r--p 0000a000 08:05 1059064 /usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
7f434a514000-7f434a515000 r-p 0000b000 08:05 1059064 /usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
7f434a515000-7f434a516000 rw-p 0000c000 08:05 1059064 /usr/lib/x86_64-linux-gnu/libnss_files-2.31.so
7f434a516000-7f434a51c000 rw-p 00000000 00:00 0
7f434a52d000-7f434aa9d000 r--p 00000000 08:05 1051340 /usr/lib/locale/locale-archive
7f434aa9d000-7f434aaa0000 rw-p 00000000 00:00 0
7f434aaa0000-7f434aac2000 r--p 00000000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac2000-7f434aac3a000 r-xp 00022000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac3a000-7f434aac8000 r--p 0019a000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac8000-7f434aac8c000 r--p 001e7000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac8c000-7f434aac8e000 rw-p 001eb000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac8e000-7f434aac92000 rw-p 00000000 00:00 0
7f434aac92000-7f434aac93000 r--p 00000000 08:05 1059050 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac93000-7f434aac95000 r-xp 00001000 08:05 1059050 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac95000-7f434aac96000 r--p 00003000 08:05 1059050 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac96000-7f434aac97000 r--p 00003000 08:05 1059050 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac97000-7f434aac98000 rw-p 00004000 08:05 1059050 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7f434aac98000-7f434aac9e000 r--p 00000000 08:05 1056633 /usr/lib/x86_64-linux-gnu/libtinfo.so.6.2
7f434aac9e000-7f434aacb5000 r-xp 0000e000 08:05 1056633 /usr/lib/x86_64-linux-gnu/libtinfo.so.6.2
7f434aacb5000-7f434aac3000 r--p 0001d000 08:05 1056633 /usr/lib/x86_64-linux-gnu/libtinfo.so.6.2
7f434aac3000-7f434aac7000 r-p 0002a000 08:05 1056633 /usr/lib/x86_64-linux-gnu/libtinfo.so.6.2
7f434aac7000-7f434aac8000 rw-p 0002e000 08:05 1056633 /usr/lib/x86_64-linux-gnu/libtinfo.so.6.2
7f434aac8000-7f434aac9000 rw-p 00000000 00:00 0
7f434aac9000-7f434aacdb000 r--s 00000000 08:05 411676 /usr/lib/x86_64-linux-gnu/gconv/gconv-modules.cache
7f434aacdb000-7f434aacdc000 r--p 00000000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f434aacdc000-7f434aacff000 r-xp 00001000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f434aacff000-7f434ad07000 r--p 00024000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f434ad07000-7f434ad09000 r--p 0002c000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f434ad09000-7f434ad0a000 rw-p 0002d000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f434ad0a000-7f434ad0b000 rw-p 00000000 00:00 0
7fffd5adb1000-7fffd5add2000 rw-p 00000000 00:00 0
7fffd5ade3000-7fffd5ade7000 r--p 00000000 00:00 0
7fffd5ade7000-7fffd5ade9000 r-xp 00000000 00:00 0
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0 [vsyscall]

```

对上述结果中相关信息进行解读：

/proc/pid/maps 文件可以用来查看进程的内存映射信息。该文件包含了进程的虚拟内存空间的映射关系，包括了每个虚拟内存区域的起始地址、结束地址、权限、偏移量、映射设备等信息。

在上图中，每一行都有 6 列信息。第一列为内存段的虚拟地址，包括起始地址和结束地址，以十六进制表示。第二列为虚拟内存区域的执行权限信息，其中 r 表示读，w 表示写，x 表示可执行；p 表示私有，s 表示共享。第三列表示虚拟内存区域在映射文件中的偏移量，如果不是文件映射则为 0。第四列映射文件中的主设备号和次设备号。第五列表示如果该虚拟内存区域是文件映射，则为其所在文件的节点号。第六列表示如果该虚拟内存区域是文件映射，则为其所在文件的路径名。

### 1.3. Status 信息解读：

输入 cat status，可以得到如下图所示结果：

[illegible]

对上述结果中相关信息进行解读:

`/proc/pid/status` 文件可以用来查看进程的状态信息。该文件包含了进程的各种状态信息，包括进程的名称、状态、优先级、进程组、线程数和内存使用情况等。下面列出上图中常见信息的含义：

**Name:** 当前正在执行的进程的名称。

**State:** 进程的状态。R 表示运行，S 表示睡眠，D 表示不可中断睡眠，Z 表示僵尸进程。

Tgid: 线程组 ID, 即进程 ID。

Pid: 进程 ID。

PPid: 父进程 ID。

TracerPid: 跟踪进程 ID。

**Uid:** 进程的用户 ID。

Gid: 进程的组 ID。

**FDSize:** 进程打开的文件描述符数量。

**Threads:** 进程中线程的数量。

**VmSize:** 进程虚拟内存大小（在下面的实验中会用到该数据对程序执行过程进行分析）。

VmRSS: 进程驻留内存大小。

**VmData:** 进程数据段大小。

VmStk: 进程栈大小。

**VmExe:** 进程代码段大小。

#### 1.4. Smap 信息解读：

输入 `cat smaps`，可以得到如下图所示结果：

```
zhengyanwei_2020151022@ubuntu:/proc/2333$ cat smaps
55cecc686000-55cecc6b3000 r--p 00000000 08:05 1048793 /usr/bin/bash
Size: 180 kB
KernelPageSize: 4 kB
MMUPageSize: 4 kB
Rss: 180 kB
Pss: 180 kB
Shared_Clean: 0 kB
Shared_Dirty: 0 kB
Private_Clean: 180 kB
Private_Dirty: 0 kB
Referenced: 180 kB
Anonymous: 0 kB
LazyFree: 0 kB
AnonHugePages: 0 kB
ShmemPmdMapped: 0 kB
FilePmdMapped: 0 kB
Shared_Hugetlb: 0 kB
Private_Hugetlb: 0 kB
Swap: 0 kB
SwapPss: 0 kB
Locked: 0 kB
THPeligible: 0
VmFlags: rd mr mw me sd
```

对上述结果中相关信息进行解读：

`/proc/pid/smaps` 文件可以用来查看进程的详细内存映射信息。该文件包含了进程的每个虚拟内存区域的详细信息，包括该内存区域的大小、起始地址、权限、映射文件、所在的物理内存页等信息。下面列出文件中主要信息的含义：

第一行列出虚拟内存区域的起始地址和结束地址、该虚拟内存区域的权限信息、偏移量、映射文件的设备号和节点号。

**Size:** 内存区域的大小。

**Rss:** 该内存区域占用的物理内存大小，包括共享和私有的部分。

**Pss:** 该内存区域占用的物理内存大小，按比例分配共享部分。

**Shared\_Clean:** 该内存区域被共享的干净页的大小。

**Shared\_Dirty:** 该内存区域被共享的脏页的大小。

**Private\_Clean:** 该内存区域私有的干净页的大小。

**Private\_Dirty:** 该内存区域私有的脏页的大小。

**Referenced:** 该内存区域被引用的页的大小。

**AnonHuePages:** 该内存区域使用的巨大页面的大小。

**Swap:** 该内存区域被交换出的大小。

- 编写程序，连续申请分配六个 128MB 空间（记为 1~6 号），然后释放第 2、3、5 号的 128MB 空间。然后再分配 1024MB，记录该进程的虚存空间变化（`/proc/pid/maps`），每次操作前后检查 `/proc/pid/status` 文件中关于内存的情况，简要说明虚拟内存变化情况。推测此时再分配 64M 内存将出现在什么位置，实测后是否和你的预测一致？解释说明用户进程空间分配属于课本中的离散还是连续分配算法？首次适应还是最佳适应算法？用户空间存在碎片问题吗？

（1）思路：

- ①根据题目要求，编写的程序按顺序执行每一个步骤。
- ②在进行空间的连续分配时，可以使用 `malloc` 方法指定分配空间大小，并对所分配的内存地址进行展示。
- ③编写一个 `Pause` 方法，要求用户输入任意字符串，每次操作完成后调用一次该方法，只有在用户输入字符串后才会执行下一操作，从而实现“每次操作前后都能对信息进行查看”的要求。



(2) 编写程序并给出每一部分解释如下:

Malloc 方法, 分配指定大小的内存空间并输出所分配到的内存地址。

```
char * Malloc(int size){
    char * buffer = malloc(size);
    //for循环将分配的空间中每个字节的值设置为该字节在内存块中的位置除以128的余数
    //这个操作可以确保分配的内存空间是可读可写的, 并且可以检测是否存在内存空间的越界问题。
    //选择128:2的幂, 可以保证每个字节的值都是0~127之间的数且循环出现
    for(int i=0;i<size;i++){
        buffer[i] = i % 128;
    }
    printf("alloc %d MB, %p-%p\n", size/MB, buffer, buffer + size);
    return buffer;
}
```

Pause 方法, 要求用户输入任意字符串。主函数执行每一步操作前调用该方法, 实现程序的“中断”以对内存信息变化进行查看。

```
void Pause(){
    char s[128];
    printf("input anything to continue(input quit to exit)\n");
    scanf("%s", s);
    if(s == "quit")
        return;
}
```

主函数中, 首先分配 6 个 128MB 的空间。

```
Pause();
//分配6个128MB的空间
char * buffer1 = Malloc(128 * MB);
char * buffer2 = Malloc(128 * MB);
char * buffer3 = Malloc(128 * MB);
char * buffer4 = Malloc(128 * MB);
char * buffer5 = Malloc(128 * MB);
char * buffer6 = Malloc(128 * MB);
printf("assign 6 buffers finish\n");
```

释放第 2、3、5 号的 128MB 空间。

```
Pause();
//释放第2、3、5号的128MB空间
free(buffer2);
free(buffer3);
free(buffer5);
printf("free buffer 2,3,5 finish\n");
```

再分配 1024MB 空间和 64MB 空间。

```
Pause();
//再分配1024MB空间
char * buffer7 = Malloc(1024 * MB);

Pause();
//再分配64MB内存
char * buffer8 = Malloc(64 * MB);
```

最后在程序结束前释放未释放的, 防止内存泄露。

```
Pause();

//程序结束前释放未释放的, 防止内存泄露
free(buffer1);
free(buffer4);
free(buffer6);
free(buffer7);
free(buffer8);
printf("buffer 1,4,6,7,8 have been free,process finish!\n");
return 0;
```

(3) 编译并运行程序, 打开一个新的 terminal 窗口, 使用 ps 命令查看当前正在运行的程序 assign 的 pid 值, 进入相应目录。

```

zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test3$ ./assign
input anything to continue

zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test3$ ps aux | grep assign
zhengya+  4367  0.0  0.0  2496  580 pts/0    S+   23:52   0:00  ./assign
zhengya+  4405  0.0  0.0  9040  724 pts/1    S+   23:53   0:00  grep --color=auto  assign
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test3$ cd /proc/4367
zhengyanwei_2020151022@ubuntu: /proc/4367$

```

(4) 在分配 6 个 128MB 空间之前，查看 maps，status 的情况：

```

zhengyanwei_2020151022@ubuntu: /proc/4367$ cat maps
5599e18d6000-5599e18d7000 r--p 00000000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Test3/assign
5599e18d7000-5599e18d8000 r-xp 00001000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Test3/assign
5599e18d8000-5599e18d9000 r--p 00002000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Test3/assign
5599e18d9000-5599e18da000 r--p 00002000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Test3/assign
5599e18da000-5599e18db000 rw-p 00003000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Test3/assign
5599e251b000-5599e253c000 rw-p 00000000 00:00 0 [heap]
7faace07c000-7faace07c000 r--p 00000000 00:00 0 [vdso]
7faace09e000-7faace216000 r-xp 00022000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace216000-7faace264000 r--p 0019a000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace264000-7faace268000 r--p 001e7000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace268000-7faace26a000 rw-p 001eb000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace26a000-7faace270000 rw-p 00000000 00:00 0
7faace281000-7faace282000 r--p 00000000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace282000-7faace2a5000 r-xp 00001000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2a5000-7faace2ad000 r--p 00024000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2ae000-7faace2af000 r--p 0002c000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2af000-7faace2b0000 rw-p 0002d000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2b0000-7faace2b1000 rw-p 00000000 00:00 0
7ffe70068000-7ffe70089000 rw-p 00000000 00:00 0 [stack]
7ffe700aa000-7ffe700aa000 r--p 00000000 00:00 0 [vvar]
7ffe700aa000-7ffe700ac000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0 [vsyscall]

```

Maps 信息显示，heap 区从 5599e251b000-5599e253c000。

```

zhengyanwei_2020151022@ubuntu: /proc/4367$ cat status
Name: assign
Umask: 0002
State: S (sleeping)
Tgid: 4367
Ngid: 0
Pid: 4367
PPid: 4309
TracerPid: 0
Uid: 1002 1002 1002 1002
Gid: 1002 1002 1002 1002
FDSize: 256
Groups: 1002
NSTgid: 4367
NSpid: 4367
NSpgid: 4367
NSsid: 4309
VmCmk: 2496 kB
VmSize: 2496 kB
VmLck: 0 kB
VmPin: 0 kB
VmHWM: 580 kB
VmRSS: 580 kB
RssAnon: 68 kB
RssFile: 512 kB
RssShmem: 0 kB
VmData: 176 kB
VmStk: 132 kB
VmExe: 4 kB

```

Status 信息表明虚拟内存的大小为 2496kb。

(5) 输入任意字符，进行 6 个空间的创建，可以看到 6 个空间的内存地址连续，头尾相差 0x1000 即 4096 字节。

```

input anything to continue(input quit to exit)
start
alloc 128 MB, 0x7faac607b010-0x7faac607b010
alloc 128 MB, 0x7faac607a010-0x7faac607a010
alloc 128 MB, 0x7faab6079010-0x7faab6079010
alloc 128 MB, 0x7faaa6078010-0x7faaa6078010
alloc 128 MB, 0x7faaa6077010-0x7faaa6077010
alloc 128 MB, 0x7faa9e076010-0x7faa9e076010
assign 6 buffers finish
input anything to continue(input quit to exit)

```

然后查看 maps 信息，可以看到此时多了一块 7faa9e076000-7faace07c000 的内存，大小为 805330944B=786456KB=768MB=6\*128MB，刚好是 6 个 128MB 的连续内存。

```

zhengyanwei_2020151022@ubuntu:/proc/4367$ cat maps
5599e18d6000-5599e18d7000 r--p 00000000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18d7000-5599e18d8000 r-xp 00001000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18d8000-5599e18d9000 r--p 00002000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18d9000-5599e18da000 r--p 00002000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18da000-5599e18db000 rw-p 00003000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e251b000-5599e253c000 rw-p 00000000 00:00 0 [heap]
7faa9e076000-7faa9e07c000 rw-p 00000000 00:00 0
7faa9e07c000-7faa9e09e000 r--p 00000000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faa9e09e000-7faa9e216000 r-xp 00022000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faa9e216000-7faa9e264000 r--p 0019a000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faa9e264000-7faa9e268000 r--p 001e7000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faa9e268000-7faa9e26a000 rw-p 001eb000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faa9e26a000-7faa9e270000 rw-p 00000000 00:00 0
7faa9e281000-7faa9e282000 r--p 00000000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faa9e282000-7faa9e2a5000 r-xp 00001000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faa9e2a5000-7faa9e2ad000 r--p 00024000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faa9e2ae000-7faa9e2af000 r--p 0002c000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faa9e2af000-7faa9e2b0000 rw-p 0002d000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faa9e2b0000-7faa9e2b1000 rw-p 00000000 00:00 0
7ffe70068000-7ffe70089000 rw-p 00000000 00:00 0 [stack]
7ffe700aa000-7ffe700aa000 r--p 00000000 00:00 0 [vvar]
7ffe700aa000-7ffe700ac000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0 [vsyscall]

```

查看 status 信息，可以看到虚拟内存大小变为 788952KB（即为初始值 2496KB 加上分配的内存大小 786456KB）：

```

zhengyanwei_2020151022@ubuntu:/proc/4367$ cat status
Name: assign
Umask: 0002
State: S (sleeping)
Tgid: 4367
Ngid: 0
Pid: 4367
PPid: 4309
TracerPid: 0
Uid: 1002 1002 1002 1002
Gid: 1002 1002 1002 1002
FDSize: 256
Groups: 1002
NSTgid: 4367
NSpid: 4367
NSpgid: 4367
NSSid: 4309
VmPeak: 788952 kB
VmSize: 788952 kB
VmLck: 0 kB
VmPin: 0 kB
VmHWM: 787836 kB
VmRSS: 787836 kB
RssAnon: 786452 kB
RssFile: 1384 kB
RssShmem: 0 kB
VmData: 786632 kB
VmStk: 132 kB
VmExe: 4 kB
VmLib: 1652 kB
VmPTE: 1580 kB
VmSwap: 0 kB
HugetlbPages: 0 kB

```

（6）输入任意字符，释放 2、3、5 号缓冲区

```

input anything to continue
release
free buffer 2,3,5 finish
input anything to continue

```

重新查看 status 信息，可以看到此时虚拟内存大小为 395724KB，减少了 393228KB=384MB=3\*128MB，刚好是释放的 3 块内存的大小。



```
zhengyanwei_2020151022@ubuntu:/proc/4367$ cat status
Name: assign
Umask: 0002
State: S (sleeping)
Tgid: 4367
Ngid: 0
Pid: 4367
PPid: 4309
TracerPid: 0
Uid: 1002 1002 1002 1002
Gid: 1002 1002 1002 1002
FDSize: 256
Groups: 1002
NSTgid: 4367
NSpid: 4367
NSpgid: 4367
NSSid: 4309
VmPeak: 788952 kB
VmSize: 395724 kB
VmLck: 0 kB
VmPin: 0 kB
VmHWM: 787836 kB
VmRSS: 394696 kB
RssAnon: 393312 kB
RssFile: 1384 kB
RssShmem: 0 kB
VmData: 393404 kB
VmStk: 132 kB
VmExe: 4 kB
VmLib: 1652 kB
VmPTE: 820 kB
VmSwap: 0 kB
HugetlbPages: 0 kB
```

然后查看 maps 信息，可以看到 heap 已经拆分为 3 块不连续的地址。

```
zhengyanwei_2020151022@ubuntu:/proc/4367$ cat maps
5599e18d6000-5599e18d7000 r--p 00000000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18d7000-5599e18d8000 r-xp 00001000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18d8000-5599e18d9000 r--p 00002000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18d9000-5599e18da000 r--p 00002000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e18da000-5599e18db000 rw-p 00003000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
t3/assign
5599e251b000-5599e253c000 rw-p 00000000 00:00 0 [heap]
7faa9e076000-7faaa6077000 rw-p 00000000 00:00 0
7faaae078000-7faab6079000 rw-p 00000000 00:00 0
7faac607b000-7faace07c000 rw-p 00000000 00:00 0
7faace07c000-7faace09e000 r--p 00000000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace09e000-7faace216000 r-xp 00022000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace216000-7faace264000 r--p 0019a000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace264000-7faace268000 r--p 001e7000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace268000-7faace26a000 rw-p 001eb000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace26a000-7faace270000 rw-p 00000000 00:00 0
7faace281000-7faace282000 r--p 00000000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace282000-7faace2a5000 r-xp 00001000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2a5000-7faace2ad000 r--p 00024000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2ae000-7faace2af000 r--p 0002c000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2af000-7faace2b0000 rw-p 0002d000 08:05 1059040 /usr/lib/x86_64-linux-gnu/ld-2.31.so
7faace2b0000-7faace2b1000 rw-p 00000000 00:00 0
7ffe70068000-7ffe70089000 rw-p 00000000 00:00 0 [stack]
7ffe700a6000-7ffe700aa000 r--p 00000000 00:00 0 [vvar]
7ffe700aa000-7ffe700ac000 r-xp 00000000 00:00 0 [vdso]
fffffffff60000-fffffffff601000 --xp 00000000 00:00 0 [vsyscall]
```

对未被释放的内存区和上图中的地址对应如下：

```
input anything to continue(input quit to exit) 5599e18da000-5599e18db000 rw-p 00003000 08:05 658051 /home/zhengyanwei_2020151022/Desktop/OS_Tes
start t3/assign
alloc 128 MB, 0x7faac607b010-0x7faace07b010 5599e251b000-5599e253c000 rw-p 00000000 00:00 0 [heap]
alloc 128 MB, 0x7faabe07a010-0x7faac607a010 7faa9e076000-7faaa6077000 rw-p 00000000 00:00 0
alloc 128 MB, 0x7faab6079010-0x7faabe079010 7faaae078000-7faab6079000 rw-p 00000000 00:00 0
alloc 128 MB, 0x7faaa6078010-0x7faab6078010 7faac607b000-7faace07c000 rw-p 00000000 00:00 0
alloc 128 MB, 0x7faaa6077010-0x7faaa6077010 7faace07c000-7faace09e000 r--p 00000000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
alloc 128 MB, 0x7faa9e076010-0x7faa9e076010 7faace09e000-7faace216000 r-xp 00022000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
assign 6 buffers finish 7faace216000-7faace264000 r--p 0019a000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
7faace264000-7faace268000 r--p 001e7000 08:05 1059048 /usr/lib/x86_64-linux-gnu/libc-2.31.so
```

(7) 输入任意字符，再分配 1024MB 的空间，这段空间的地址是 0x7faa5e075010-0x7faa8e075010。

```
input anything to continue(input quit to exit)
assign
alloc 1024 MB, 0x7faa5e075010-0x7faa8e075010
input anything to continue(input quit to exit)
```

查看 maps 信息，可以看到内存地址在原来 6 号内存空间的基础上向后进行拓展

```
5599e251b000-5599e253c000 rw-p 00000000 00:00 0 [heap]
7faa5e075000-7faaa6077000 rw-p 00000000 00:00 0
7faaae078000-7faab6079000 rw-p 00000000 00:00 0
7faac607b000-7faace07c000 rw-p 00000000 00:00 0
```

(8) 输入任意字符再分配 64MB 的内存，通过输出可以知道这段空间的地址为

0x7faac207a010-0x7faac607a010。

```
assign
alloc 64 MB, 0x7faac207a010-0x7faac607a010
input anything to continue(input quit to exit)
```

查看 maps 信息，可以发现新内存跟在第一块 128MB 的内存后面，相当于占用之前被释放的第二块 128MB 的内存

```
5599e251b000-5599e253c000 rw-p 00000000 00:00 0 [heap]
7faa5e075000-7faa60770000 rw-p 00000000 00:00 0
7faaae078000-7faab6079000 rw-p 00000000 00:00 0
7faac207a000-7faace07c000 rw-p 00000000 00:00 0
```

观察程序的输出，也可以发现新分配的 64MB 的内存的地址，就是一开始分配的第二块内存的尾地址。

```
start
alloc 128 MB, 0x7faac607b010-0x7faace07b010
alloc 128 MB, 0x7faabe07a010-0x7faac607a010
alloc 128 MB, 0x7faab6079010-0x7faab079010
alloc 128 MB, 0x7faaae078010-0x7faab6078010
alloc 128 MB, 0x7faaa6077010-0x7faaae077010
alloc 128 MB, 0x7faa9e076010-0x7faaa6076010
assign 6 buffers finish
input anything to continue(input quit to exit)
release
free buffer 2,3,5 finish
input anything to continue(input quit to exit)
assign
alloc 1024 MB, 0x7faa5e075010-0x7faa9e075010
input anything to continue(input quit to exit)
assign
alloc 64 MB, 0x7faac207a010-0x7faac607a010
input anything to continue(input quit to exit)
```

(9) 修改程序验证算法是首次适应还是最佳适应：

首次适应：每次都从低地址开始查找，找到第一个能满足大小的空闲分区。实现方法是空闲分区以地址递增的次序排列。每次分配内存时顺序查找空闲分区链（或空闲分区表），找到大小能满足要求的第一个空闲分区。

最佳适应：由于动态分区分配是一种连续分配方式，为各进程分配的空间必须是连续的一整片区域，因此为了保证当“大进程”到来时能有连续的大片空间，可以尽可能多地留下大片的空闲区，即优先使用更小的空闲区。实现方法是空闲分区按容量递增次序链接。每次分配内存时顺序查找空闲分区链（或空闲分区表），找到大小能满足要求的第一个空闲分区。

根据上述第 7 步中，对 64MB 空间分配结果的特点，可以修改程序代码，比如将第二个分区大小修改为 200MB，第四个分区大小修改为 100MB，然后释放这两块空间，再次执行 64MB 内存空间的分配，通过 maps 信息观察分配的内存地址占用的是被释放的哪一块空间。

修改程序如下：

```

Pause();
//分配6个128MB的空间
char * buffer1 = Malloc(128 * MB);
//char * buffer2 = Malloc(128 * MB);
char * buffer2 = Malloc(200 * MB);
char * buffer3 = Malloc(128 * MB);
//char * buffer4 = Malloc(128 * MB);
char * buffer4 = Malloc(100 * MB);
char * buffer5 = Malloc(128 * MB);
char * buffer6 = Malloc(128 * MB);
printf("assign 6 buffers finish\n");

Pause();
//释放第2、4号的空间
free(buffer2);
free(buffer4);
printf("free buffer 2,4 finish\n");
//释放第2、3、5号的128MB空间
//free(buffer2);
//free(buffer3);
//free(buffer5);
//printf("free buffer 2,3,5 finish\n");

//Pause();
//再分配1024MB空间
//char * buffer7 = Malloc(1024 * MB);

Pause();
//再分配64MB内存
char * buffer8 = Malloc(64 * MB);

Pause();|
return 0;

```

编译并执行代码，然后对第 2、4 块分区进行释放，查看 maps 信息查看内存对应关系

```

zhengyanwei_2020151022@ubuntu:~/Desktop/05_Test3$ ./assign
input anything to continue(input quit to exit)
start
alloc 128 MB, 0x7f1712666010-0x7f171a666010
alloc 200 MB, 0x7f1705e65010-0x7f1712665010
alloc 128 MB, 0x7f16fde64010-0x7f1705e64010
alloc 100 MB, 0x7f16f7a63010-0x7f16fde63010
alloc 128 MB, 0x7f16efa62010-0x7f16f7a62010
alloc 128 MB, 0x7f16e7a61010-0x7f16efa61010
assign 6 buffers finish
input anything to continue(input quit to exit)
free
free buffer 2,4 finish
input anything to continue(input quit to exit)

```

```

start
alloc 128 MB, 0x7f1712666010-0x7f171a666010 555c878ae000-555c878af000 rw-p 00003000 08:05 656761 /home
alloc 200 MB, 0x7f1705e65010-0x7f1712665010 st3/assign 555c889e4000-555c88a05000 rw-p 00000000 00:00 0 [heap]
alloc 128 MB, 0x7f16fde64010-0x7f1705e64010 7f16e7a61000-7f16f7a63000 rw-p 00000000 00:00 0
alloc 100 MB, 0x7f16f7a63010-0x7f16fde63010 7f16fde64000-7f1705e65000 rw-p 00000000 00:00 0
alloc 128 MB, 0x7f16efa62010-0x7f16f7a62010 7f1712666000-7f171a667000 rw-p 00000000 00:00 0
alloc 128 MB, 0x7f16e7a61010-0x7f16efa61010 7f171a667000-7f171a689000 r--p 00000000 08:05 1059048 /usr
assign 6 buffers finish 7f171a689000-7f171a801000 r-xp 00022000 08:05 1059048 /usr
input anything to continue(input quit to exit) 7f171a801000-7f171a84f000 r--p 0019a000 08:05 1059048 /usr

```

分配 64MB 内存区，观察内存地址

```

zhengyanwei_2020151022@ubuntu:~/Desktop/05_Test3$ ./assign
input anything to continue(input quit to exit)
start
alloc 128 MB, 0x7f1712666010-0x7f171a666010
alloc 200 MB, 0x7f1705e65010-0x7f1712665010
alloc 128 MB, 0x7f16fde64010-0x7f1705e64010
alloc 100 MB, 0x7f16f7a63010-0x7f16fde63010
alloc 128 MB, 0x7f16efa62010-0x7f16f7a62010
alloc 128 MB, 0x7f16e7a61010-0x7f16efa61010
assign 6 buffers finish
input anything to continue(input quit to exit)
free
free buffer 2,4 finish
input anything to continue(input quit to exit)
assign
alloc 64 MB, 0x7f170e665010-0x7f1712665010
input anything to continue(input quit to exit)

```

通过上述结果可以发现 64MB 内存占用的是 200MB 的空间，因此可以说明使用的是首次适应算法。

(10) 解释说明用户进程空间分配属于课本中的离散还是连续分配算法

连续分配：每个进程分配一段地址空间连续的内存空间。

离散分配：允许将一个进程分散的分配到许多不相邻的分区中，程序全部装入内存。

在代码中，通过调用 malloc 函数为每个分配的内存块分配一段连续的内存空间，并返回分配的内存块首地址。在释放内存块时，free 函数将内存块的首地址作为参数传递给 free



函数，释放该内存块所占用的连续内存空间，因此该程序使用的是连续分配算法。

(11) 程序用户空间存在碎片问题吗？

存在。当程序多次分配和释放内存块后，可能会出现内存空间的碎片化问题，即存在一些大小较小的、不连续的空间。例如程序中对 2、3、5 块内存块进行释放，这些空间中，第二块内存空间被后来分配的 64MB 占用，而其他空间未被使用，此时便出现了内存空间的碎片化问题。要避免这一问题，可以采用内存池技术或者动态内存管理算法。

3. 设计一个程序测试出你的系统单个进程所能分配到的最大虚拟内存空间为多大？

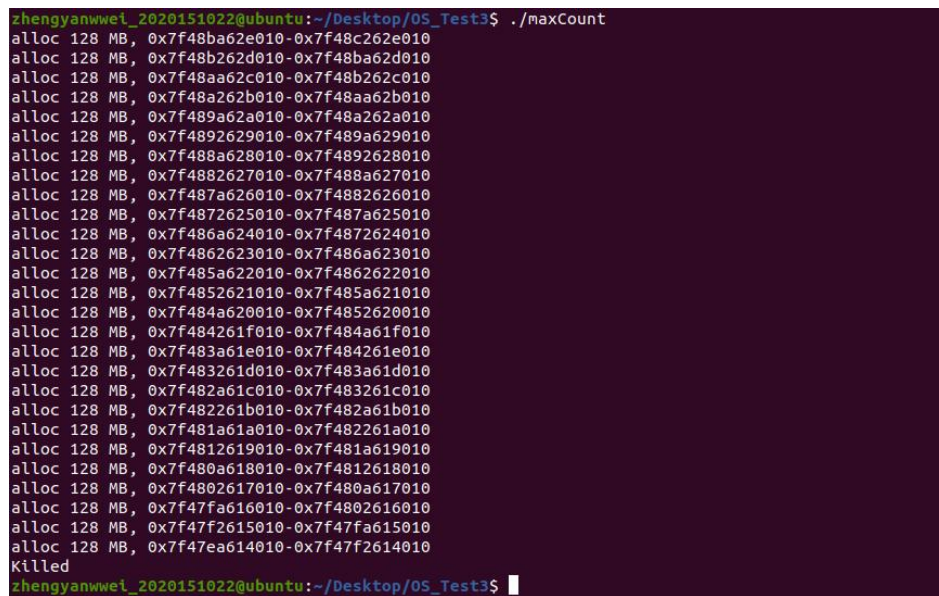
(1) 思路：要看单个进程能分配到的最大虚拟内存空间，可以在程序中编写一个循环，不断申请内存，每次申请大小为 1MB，直到申请失败为止。在申请过程中，输出当前所申请的内存大小和内存地址，申请失败则退出。最后通过申请的内存块数量和大小对系统单个进程所能分配的最大虚拟内存空间进行统计。

(2) 编写程序及解释如下：

同样使用上述 Malloc 方法对指定大小的内存空间进行分配并输出所分配的内存地址。在主函数中，设置每次申请的空间大小为 128MB，然后设置循环不断进行内存的分配，直到发生故障。

```
int size = 128 * MB;
while(1){
    //不断申请大小为128MB的内存空间
    char * buf = Malloc(size);
    //申请失败，退出
    if(buf == NULL){
        printf("Failed to allocate the new one\n");
        break;
    }
}
return 0;
```

(3) 编译并执行程序，程序不断申请空间。当申请空间不足时，程序出错。



```
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test3$ ./maxCount
alloc 128 MB, 0x7f48ba62e010-0x7f48c262e010
alloc 128 MB, 0x7f48b262d010-0x7f48ba62d010
alloc 128 MB, 0x7f48aa62c010-0x7f48b262c010
alloc 128 MB, 0x7f48a262b010-0x7f48aa62b010
alloc 128 MB, 0x7f489a62a010-0x7f48a262a010
alloc 128 MB, 0x7f4892629010-0x7f489a629010
alloc 128 MB, 0x7f488a628010-0x7f4892628010
alloc 128 MB, 0x7f4882627010-0x7f488a627010
alloc 128 MB, 0x7f487a626010-0x7f4882626010
alloc 128 MB, 0x7f4872625010-0x7f487a625010
alloc 128 MB, 0x7f486a624010-0x7f4872624010
alloc 128 MB, 0x7f4862623010-0x7f486a623010
alloc 128 MB, 0x7f485a622010-0x7f4862622010
alloc 128 MB, 0x7f4852621010-0x7f485a621010
alloc 128 MB, 0x7f484a620010-0x7f4852620010
alloc 128 MB, 0x7f484261f010-0x7f484a61f010
alloc 128 MB, 0x7f483a61e010-0x7f484261e010
alloc 128 MB, 0x7f483261d010-0x7f483a61d010
alloc 128 MB, 0x7f482a61c010-0x7f483261c010
alloc 128 MB, 0x7f482261b010-0x7f482a61b010
alloc 128 MB, 0x7f481a61a010-0x7f482261a010
alloc 128 MB, 0x7f4812619010-0x7f481a619010
alloc 128 MB, 0x7f480a618010-0x7f4812618010
alloc 128 MB, 0x7f4802617010-0x7f480a617010
alloc 128 MB, 0x7f47fa616010-0x7f4802616010
alloc 128 MB, 0x7f47f2615010-0x7f47fa615010
alloc 128 MB, 0x7f47ea614010-0x7f47f2614010
Killed
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test3$
```

(4) 统计虚拟机出错前程序申请的内存空间大小

通过上图左侧的输出可以看到，在内存空间满之前，程序一共申请了 27 个大小为 128MB 的内存区，即系统单个进程所能分配到的最大虚拟内存空间为  $27 \times 128\text{MB} = 3456\text{MB} \approx 3.4\text{GB}$ 。

(5) 通过资料查询，可以直到 linux 单个进程能分配的最大虚拟内存受限于系统架构、



进程标志、内存资源限制和物理内存等因素。

4. 编写一个程序，分配 256MB 内存空间（或其他足够大的空间），检查分配前后 /proc/pid/status 文件中关于虚拟内存和物理内存的使用情况，然后每隔 4KB 间隔将相应地址进行读操作，再次检查 /proc/pid/status 文件中关于内存的情况，对比前后两次内存情况，说明所分配物理内存（物理内存块）的变化。然后重复上面操作，不过此时为写操作，再观察其变化

（1）思路：

①在程序中分配 256MB 的内存空间。然后以这个大小为条件，设置循环从 0 开始，每次增加 4KB 的大小，在循环体中设置语句对所申请的内存空间进行读操作。然后再设置一个同样的循环，在循环体中执行写操作。

②为在观察读操作的变化后再执行写操作，可以使用上述 Pause 方法，在程序执行写操作的循环体之前调用该方法，实现程序的“中断”。

（2）编写程序和解释如下：

分配 256MB 的空间

```
Pause();  
  
//assign  
int size = 256*MB;  
//分配256MB的空间  
char * buffer = malloc(size);  
printf("assign finish!\n");
```

设置循环体执行读操作：

```
Pause();  
//read  
unsigned long long sum;  
int block;  
for(int i=0;i<size;i+=4*KB){  
    block = buffer[i];  
    sum += block;  
}  
printf("read finish!\n");
```

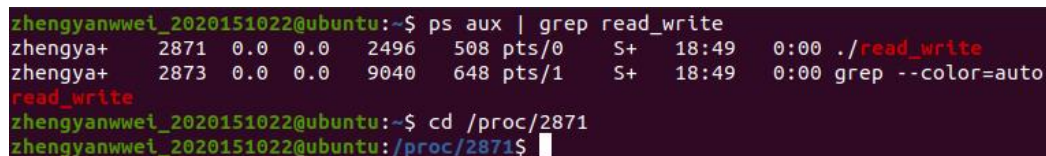
设置循环体执行写操作：

```
Pause();  
//write  
for(int i=0;i<size;i+=4*KB)  
    buffer[i] = i;  
printf("write finish!\n");
```

释放所申请的空间，防止内存泄露。

```
Pause();  
//释放一下buffer，防止内存泄露  
free(buffer);  
printf("buffer has been freed!\n");  
return 0;
```

（3）编译并执行程序，在另一个终端中查看进程号，进入相应目录



```
zhengyanwei_2020151022@ubuntu:~$ ps aux | grep read_write  
zhengya+ 2871 0.0 0.0 2496 508 pts/0 S+ 18:49 0:00 ./read_write  
zhengya+ 2873 0.0 0.0 9040 648 pts/1 S+ 18:49 0:00 grep --color=auto  
read_write  
zhengyanwei_2020151022@ubuntu:~$ cd /proc/2871  
zhengyanwei_2020151022@ubuntu:/proc/2871$
```

（4）分配 256MB 空间前，查看进程内存信息。通过下图可以看到当前虚拟内存大小为 2496KB，物理内存大小为 508KB。

```
zhengyanwei_2020151022@ubuntu:/proc/2871$ cat status
Name: read_write
Umask: 0002
State: S (sleeping)
Tgid: 2871
Ngid: 0
Pid: 2871
PPid: 2126
TracerPid: 0
Uid: 1002 1002 1002 1002
Gid: 1002 1002 1002 1002
FDSize: 256
Groups: 1002
NStgid: 2871
NSpid: 2871
NSpgid: 2871
NSsid: 2126
VmPeak: 2496 kB
VmSize: 2496 kB
VmLck: 0 kB
VmPin: 0 kB
VmHWM: 508 kB
VmRSS: 508 kB
RssAnon: 64 kB
RssFile: 444 kB
RssShmem: 0 kB
```

(5) 输入任意语句，开始分配空间

```
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test3$ ./read_write
input anything to continue(input quit to exit)
assign
assign finish!
input anything to continue(input quit to exit)
```

查看分配空间后的进程内存信息，可以看到虚拟内存大小：

$264644\text{KB} = 2496\text{KB} + 262148\text{KB} = 2496\text{KB} + 256\text{MB}$ ，刚好增加了 256MB。

```
zhengyanwei_2020151022@ubuntu:/proc/2871$ cat status
Name: read_write
Umask: 0002
State: S (sleeping)
Tgid: 2871
Ngid: 0
Pid: 2871
PPid: 2126
TracerPid: 0
Uid: 1002 1002 1002 1002
Gid: 1002 1002 1002 1002
FDSize: 256
Groups: 1002
NStgid: 2871
NSpid: 2871
NSpgid: 2871
NSsid: 2126
VmPeak: 264644 kB
VmSize: 264644 kB
VmLck: 0 kB
VmPin: 0 kB
VmHWM: 508 kB
VmRSS: 508 kB
RssAnon: 64 kB
RssFile: 444 kB
```

(6) 继续输入任意语句，开始执行读操作的循环体：

```
read
read finish!
input anything to continue(input quit to exit)
```

查看内存信息，可以看到虚拟内存的大小不发生改变，物理内存增加大约 1MB。

```

zhengyanwei_2020151022@ubuntu:/proc/2871$ cat status
Name:   read_write
Umask:  0002
State:  S (sleeping)
Tgid:   2871
Ngid:   0
Pid:    2871
PPid:   2126
TracerPid: 0
Uid:    1002   1002   1002   1002
Gid:    1002   1002   1002   1002
FDSize: 256
Groups: 1002
NSTgid: 2871
NSpid:  2871
NSpgid: 2871
NSSid:  2126
VmPeak: 264644 kB
VmSize: 264644 kB
VmLck:  0 kB
VmPin:  0 kB
VmHWM:  1324 kB
VmRSS:  1324 kB
RssAnon:      84 kB
RssFile:     1240 kB
RssShmem:      0 kB
VmData: 262324 kB
VmStk:   132 kB

```

(7) 输入语句执行写操作的循环体

```

write
write finish!
input anything to continue(input quit to exit)

```

查看内存信息，可以看到物理内存的大小明显增加，相比于分配完成时的值，增加了256MB。

```

zhengyanwei_2020151022@ubuntu:/proc/2871$ cat status
Name:   read_write
Umask:  0002
State:  S (sleeping)
Tgid:   2871
Ngid:   0
Pid:    2871
PPid:   2126
TracerPid: 0
Uid:    1002   1002   1002   1002
Gid:    1002   1002   1002   1002
FDSize: 256
Groups: 1002
NSTgid: 2871
NSpid:  2871
NSpgid: 2871
NSSid:  2126
VmPeak: 264644 kB
VmSize: 264644 kB
VmLck:  0 kB
VmPin:  0 kB
VmHWM:  263400 kB
VmRSS:  263400 kB
RssAnon: 262160 kB
RssFile:  1240 kB
RssShmem:  0 kB
VmData: 262324 kB
VmStk:   132 kB

```

(8) 最后输入任意语句退出程序，对所申请空间进行释放。

```

input anything to continue(input quit to exit)
quit
buffer has been freed!
zhengyanwei_2020151022@ubuntu:~/Desktop/OS_Test3$

```

#### 四、实验体会：

1. 通过本次实验，对如何在 linux 中编写程序实现指定大小的内存块进行分配和使用、进程结束前需要对所分配空间进行释放等有了一定的掌握。
2. 对于 proc 文件下的 status、maps 等文件有了一定的认识，了解了文件中几个重要的参数所表示的意义，并能够使用其所展示的信息对进程执行过程中出现的信息变化进行分析。



指导教师批阅意见：

成绩评定：

指导教师签字：

年 月 日

备注：