操作系统实验报告

实验二：内存管理

姓名：胡敏臻　　袁洁

学号：2019211424　　2019211426

班级：2019211307　　2019211307

日期：2021/12/7

一、实验目的

在本次实验中，需要从不同的侧面了解 openEuler 的虚拟内存机制。在 openEuler 操作系统中，可以通过一些 API 操纵虚拟内存。主要需要了解以下几方面：

- openEuler 虚拟存储系统的组织
- 如何控制虚拟内存空间
- 如何追踪进程内存使用情况
- 详细了解与内存相关的 API 函数的使用

二、实验要求及内容

创建一个包含两个线程的进程：线程 t1 和线程 t2，进程 p1 通过执行一系列内存操作来模拟内存分配活动，线程 t2 用于跟踪线程 t1 的内存行为，两个线程通过信号量进行同步。

线程 t1 执行的内存操作类型包括：分配虚拟内存、写入虚拟内存、释放虚拟内存、锁定物理内存、解锁物理内存，可以将内存操作编号保存到输入文件中，线程 t1 从输入文件中读取内存操作并执行。

设计的实验需覆盖以下操作序列（可以通过多个输入文件做多个实验来覆盖）：

分配一块虚拟内存区域，然后打印该区域上的部分或全部内容

分配一块虚拟内存区域，然后对该区域执行写入操作

释放一块虚拟内存区域，然后打印该区域上的部分或全部内容

释放一块虚拟内存区域，然后对该区域执行写入操作

三、实验设备环境

- openEuler 20.03 (LTS), Linux version 4.19.90-2003.4.0.0036.oe1.aarch64
- gcc (GCC) 7.3.0

四、实验步骤（原理步骤、编译运行方法、实验结果说明）

本次实验采用了服务器的方式登录 openEuler 系统，找到以自己学号命名的文件夹，将输入文件 input.txt 与代码文件 memory-op.c 放在同一路径下。

编译：gcc memory_op.c -o memory_op -l pthread

运行：./memory_op

1、 运行样例输入，查看运行情况。

样例输入：

0 0 50 7

1 0 40 7

0 50 40 7

1 50 30 7

2 50 20 7

3 50 10 7

4 50 20 7

```
[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190235072 kB
MemAvailable:   191241920 kB
----------
VmSize:   150400 kB
VmRSS:      1664 kB
----------
mmap:    addr = 0x1000000000, pages = 50
VmSize:   153600 kB
VmRSS:      1664 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   153600 kB
VmRSS:      1664 kB
----------
mmap:    addr = 0x1000320000, pages = 40
VmSize:   156160 kB
VmRSS:      1664 kB
----------
write:   addr = 0x1000320000, pages = 30
VmSize:   156160 kB
VmRSS:      5824 kB
----------
mlock:   addr = 0x1000320000, pages = 20
VmSize:   156160 kB
VmRSS:      5824 kB
----------
munlock: addr = 0x1000320000, pages = 10
VmSize:   156160 kB
VmRSS:      5824 kB
----------
munmap:  addr = 0x1000320000, pages = 20
VmSize:   154880 kB
VmRSS:      4928 kB
----------
```

从样例输入中我们发现了几个问题

- 物理内存在什么情况下会增加？
- 物理内存增加与释放的大小与申请增加和释放的内存有什么关系？
- 内存释放了之后继续写会发生什么？
- 内存释放已经释放的内存会发生什么？
- 加锁之后对内存进行操作会发生什么？

2、 我们修改输入，在释放完内存之后做写操作

此时输入：

0 0 50 7

1 0 40 7

0 50 40 7

4 50 30 7

1 50 20 7

我们发现此时发生了段错误。因此得知，**在未分配的内存上做写操作是不被允许的**。

3、 接着，我们尝试在已经释放的内存上做释放操作，探究会发生什么？

输入修改如下：

0 0 50 7

1 0 40 7

0 50 40 7

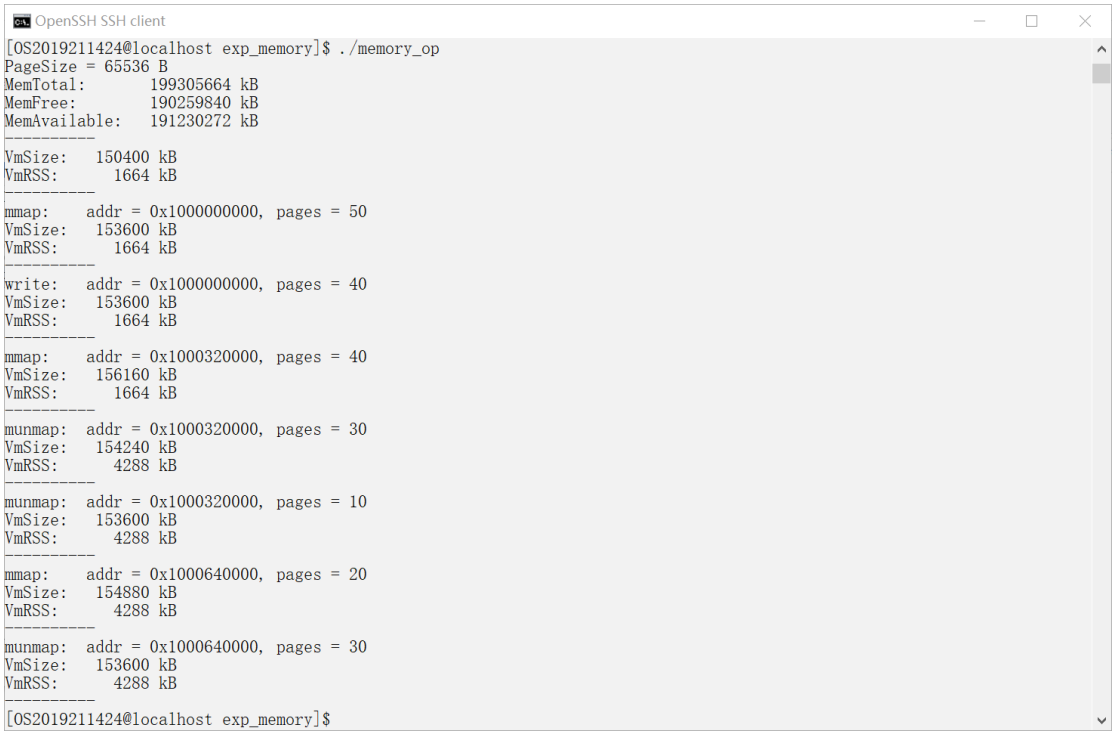
4 50 30 7

4 80 30 7

0 100 20 7

4 100 30 7

本次输入修改，有两个目的，一个是同一片内存分两次释放，另一个是释放比分配内存多的部分会发生什么？



我们发现，不论是从哪里开始释放，地址的起始地址都是同一个，这与我们的常理不同，并且释放函数的含义也是从起始开始释放。我们于是查看了源代码，发现此时的打印为

```
        printf("munmap:  addr = %p, pages =
            %d\n", region, block);
}
```

打印的为 region，而 rigion 是指最近分配的内存地址，不符合释放的首地址，我们将其修改如下：

```
printf("munmap:  addr = %p, pages = %d\n", pagesize * (base + start), block);
```

这样打印的就是此时释放的首地址了。

此外，当我们多释放地址时，此时也没有发生错误。通过查阅资料我们得知，如果释放失败，则会返回-1，于是我们修改如下：

```
int flag=munmap((void*) (pagesize * (base + start)), block * pagesize);
if(flag==-1){
    printf("Wrong munmap!");
}
else{
    printf("munmap:  addr = %p, pages = %d\n", pagesize * (base + start), block);
}
```

最后运行时仍能正常运行没有报错。因此我们可以判断，当**释放已释放的内存时，为合法释放，只是释放内存量为 0。**

4、 脏页提交（物理内存增加）规律

最开始观察老师提供的输入，发现有的时候虽然逻辑内存里的页被修改了，但是并不会被提交即扩大物理内存。通过查阅资料，我们最初认为当写不是同一片连续区域，或者

内存被释放时会被提交。但是当测试样例扩大，我们发现如果脏页过少，也不会被提交，于是我们猜测只有脏页到达一定值后才会被提交。

```
[OS20192114424@localhost exp_memory]$ cat input.txt
0 0 80 7
0 100 80 7
0 200 100 7
1 0 40 7
1 40 24 7
1 64 1 7
4 0 10 7
[OS20192114424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190283776 kB
MemAvailable:   191306304 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 24
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000400000, pages = 1
VmSize:   167040 kB
VmRSS:      5760 kB
----------
munmap:  addr = 0x1000000000, pages = 10
VmSize:   166400 kB
VmRSS:      5184 kB
----------
[OS20192114424@localhost exp_memory]$
```

```
[OS2019211424@localhost exp_memory]$ cat input.txt
0 0 80 7
0 100 80 7
0 200 100 7
1 0 40 7
1 40 25 7
1 100 20 7
1 120 46 7
1 200 66 7
0 300 50 7
1 300 1 7
[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190259008 kB
MemAvailable:   191272256 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 25
VmSize:   167040 kB
VmRSS:      5760 kB
----------
write:   addr = 0x1000640000, pages = 20
VmSize:   167040 kB
VmRSS:      5760 kB
----------
write:   addr = 0x1000780000, pages = 46
VmSize:   167040 kB
VmRSS:      9984 kB
----------
write:   addr = 0x1000c80000, pages = 66
VmSize:   167040 kB
VmRSS:     14208 kB
----------
mmap:    addr = 0x10012c0000, pages = 50
VmSize:   170240 kB
VmRSS:     14208 kB
----------
write:   addr = 0x10012c0000, pages = 1
VmSize:   170240 kB
VmRSS:     14464 kB
----------
[OS2019211424@localhost exp_memory]$
```

```
[OS2019211424@localhost exp_memory]$ cat input.txt
0 0 80 7
0 100 80 7
0 200 100 7
1 0 40 7
1 40 25 7
1 100 20 7
1 120 46 7
1 200 65 7
0 300 50 7
1 300 1 7
[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190259200 kB
MemAvailable:   191271488 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 25
VmSize:   167040 kB
VmRSS:      5760 kB
----------
write:   addr = 0x1000640000, pages = 20
VmSize:   167040 kB
VmRSS:      5760 kB
----------
write:   addr = 0x1000780000, pages = 46
VmSize:   167040 kB
VmRSS:      9984 kB
----------
write:   addr = 0x1000c80000, pages = 65
VmSize:   167040 kB
VmRSS:      9984 kB
----------
mmap:    addr = 0x10012c0000, pages = 50
VmSize:   170240 kB
VmRSS:      9984 kB
----------
write:   addr = 0x10012c0000, pages = 1
VmSize:   170240 kB
VmRSS:     14400 kB
----------
[OS2019211424@localhost exp_memory]$ vim input.txt
[OS2019211424@localhost exp_memory]$ [OS2019211424@localhost exp_memory]$
[OS2019211424@localhost exp_memory]$ cat input.txt
```

通过控制变量可发现，第一个输入的写没有写到 65 页，没有被提交，但是第二个第三个第一次写到 65 页即被提交。**所以我们认为我们得出结论第一次脏页达到 65 页，就可以被提交。**第三个输入第三次写 65 页没有被提交，而第二个输入第三次写 66 页就可以被提交。所以接着得出结论第一次提交后，**后续要被提交脏页的数量一定要达到 66 页才会被提交。**但是测试过程中我们发现，程序如果最后一次为写即使不达到 66 页依然被提交。

```
[OS2019211424@localhost exp_memory]$ cat input.txt
0 0 80 7
0 100 80 7
0 200 100 7
1 0 40 7
1 40 25 7
[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190260160 kB
MemAvailable:   191273408 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 25
VmSize:   167040 kB
VmRSS:      5952 kB
----------
```

所以我们认为，如果程序最后的操作为写，无论大小为多少，都会被提交。

```
0 200 100 7
1 0 40 7
1 40 15 7
4 0 40 7

[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190257216 kB
MemAvailable:   191270784 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 15
VmSize:   167040 kB
VmRSS:      1600 kB
----------
munmap:  addr = 0x1000000000, pages = 40
VmSize:   164480 kB
VmRSS:      2752 kB
----------
VmSize:   164480 kB
VmRSS:      2752 kB
----------
```

最后我们印证了，**当取消映射时，物理内存会变大，即脏页提交。**

通过实验我们粗略得到以下规律：

1、正常情况下脏页被提交，不是以一页为单位，而是为了减少了 io 次数，提高效率，积攒足够的脏页后才被提交通常该值为 65 或 66

2、进程在映射空间的对共享内容的改变并不直接写回到磁盘文件中，往往在调用 munmap()后才执行该操作，物理内存此时会增加。

3、如果最后运行写操作，无论脏页积攒是否达到 66 或 65 都会被提交

5、物理地址映射规律

通过多次测试实验样例，修改测试样例我们发现，当虚拟内存开始被提交，线程物理内存变大时，并不总是按照我们预想的，弄脏多少页，物理内存就增加相应的页数，而是总会比该值多出一或三，而且该值会随着运行次数而改变，目前小组成员猜测，逻辑内到**虚拟内存的映射也不是以页为单位进行映射，而是多页作为整体来映射，每次映射会比需要的多出几页。**但由于每次运行的结果都不同，小组成员，也未能得出准确结论。

```
0 200 100 7
1 0 40 7
1 40 15 7
4 0 40 7

[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190257216 kB
MemAvailable:   191270784 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 15
VmSize:   167040 kB
VmRSS:      1600 kB
----------
munmap:  addr = 0x1000000000, pages = 40
VmSize:   164480 kB
VmRSS:      2752 kB
----------
VmSize:   164480 kB
VmRSS:      2752 kB
----------
```

```
[OS2019211424@localhost exp_memory]$ cat input.txt
0 0 80 7
0 100 80 7
0 200 100 7
1 0 40 7
2 40 10 7
1 40 15 7
4 40 10 7
3 40 10 7
[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190256064 kB
MemAvailable:   191270208 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
mlock:   addr = 0x1000280000, pages = 10
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 15
VmSize:   167040 kB
VmRSS:      1600 kB
----------
munmap:  addr = 0x1000280000, pages = 10
VmSize:   166400 kB
VmRSS:      4544 kB
----------
munlock: addr = 0x1000280000, pages = 10
VmSize:   166400 kB
VmRSS:      4672 kB
----------
[OS2019211424@localhost exp_memory]$
```

6、锁对写操作的影响

小组成员最初，对锁机制的锁没有太明白锁住的是什么，于是对测试样例进行修改得出结论。

```
[OS2019211424@localhost exp_memory]$ cat input.txt
0 0 80 7
0 100 80 7
0 200 100 7
1 0 40 7
2 40 10 7
1 40 15 7
4 40 10 7
3 40 10 7
[OS2019211424@localhost exp_memory]$ ./memory_op
PageSize = 65536 B
MemTotal:       199305664 kB
MemFree:        190256064 kB
MemAvailable:   191270208 kB
----------
VmSize:   150400 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000000000, pages = 80
VmSize:   155520 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000640000, pages = 80
VmSize:   160640 kB
VmRSS:      1600 kB
----------
mmap:    addr = 0x1000c80000, pages = 100
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000000000, pages = 40
VmSize:   167040 kB
VmRSS:      1600 kB
----------
mlock:   addr = 0x1000280000, pages = 10
VmSize:   167040 kB
VmRSS:      1600 kB
----------
write:   addr = 0x1000280000, pages = 15
VmSize:   167040 kB
VmRSS:      1600 kB
----------
munmap:  addr = 0x1000280000, pages = 10
VmSize:   166400 kB
VmRSS:      4544 kB
----------
munlock: addr = 0x1000280000, pages = 10
VmSize:   166400 kB
VmRSS:      4672 kB
----------
[OS2019211424@localhost exp_memory]$
```

通过该输入我们可以发现关于程序的锁机制，实现的是防止这部分内存页被调度到交换空间，实际上对于该程序的操作而言，没有体现锁机制，**也就是虽然加了锁，但是写操作等没有用到该特征。**

五、实验心得体会

本次上机总时长约为 8 小时，花费时间较长。

此次实验小组成员花费时间较长，主要时间花在测试样例的构想，以及对测试结果的分析上。在实验的最开始，小组成员就平台就遇到了问题，因为本次实验输出较多，会造成页的滚动。然而，在 vmware 的虚拟机上，我们无法做到页的自由滚动。经过数次尝试换平台安装虚拟机无果之后，我们最终需选择了远程登录 openEuler 系统，完成本次实验。

开始实验后，由于对 api 的不熟悉，对代码无法理解，不能明白每个参数的含义。通过认真阅读指导书，了解每个 api 的作用，再通过尝试运行测试样例，理解了每个参数的含义以及输出结果的意义，使得实验可以继续进行。

最初成员理论知识掌握的不充分，对于实验中出现的各种非正常现象无法进行解释，

为了解决这个问题，我们构建了大量的输入测试样例，通过控制变量，观察每个不同的变量，依次递增取不同的值会对结果造成怎样的影响，过程中成员通过网络，书籍积极查找资料，对理论知识有了更深层的掌握之后，可以解释每种现象出现背后的原理。

最后，本次实验并不像第一次实验，代码修改的空间很小，任务更多的落在了对测试结果的分析上，然而这就需要扎实的理论基础作为支撑，通过本次试验，小组成员对虚拟内存的理论知识都有了更深层的理解，并且通过实验，可以做到将理论知识运用到实际的应用上，再通过实际运行的结果不断完善自己的理论知识。总之，本次实验虽然花费了成员大量的精力，但是也给成员们带来了不小的收获，以及实验完成的满足感。