Contents

1 内存管理实验指南

1.1 Overview

1.1.1 设计目的:

屏蔽底层繁琐的细节，让同学们了解并熟悉操作系统的内存管理的实现思路

1.1.2 实验方式:

1. 助教们用 C 风格的函数模拟对于底层的操作，从而屏蔽现实操作系统的繁琐复杂的部分（详见下面 “底层模拟接口” 部分）
2. 同学们实现内存的管理，满足计算机其他模块对于内存使用的需要。具体需要实现的接口见下面 “内存管理接口” 部分
3. 助教们将同学们实现的代码编译，运行测试（详见下面 “测试说明” 部分），并进行打分

1.1.3 评分方式:

1. 代码不能通过编译，0 分
2. 代码被查出和其他同学雷同，0 分
3. 代码通过正确性测试，得到加分 4 分，如有少量测试无法通过，酌情扣分
4. 代码能支持申请超过 128 MB 的内存，并且性能达到标准线，得到加分 7 分，如有少量测试无法通过，酌情扣分
5. 代码在完成上述所有要求的前提下，在三项性能测试中任意项获得第一的同学，得到加分 10 分

1.2 底层模拟接口

1.2.1 模拟设备细节

- 虚拟空间大小（磁盘可用交换区大小）：512MB
- 物理空间大小（内存可用空间大小）：128MB
- 字节可寻址

1.2.2 设计说明

- 底层接口只负责：
- 数据的读和写
- 对模拟设备的访问的记录
- 需要你实现的功能：
- 存储保护
- 虚拟地址到物理地址的转换
- 什么时候，将哪些数据，从内存写进磁盘 or 从磁盘读进内存

1.2.3 底层模拟接口

- 读取内存某个物理地址上的字节，若 address 越界，会退出程序，开销小

  ```
  data_unit mem_read(p_address address);
  ```

- 往内存某个物理地址写入字节，若 address 越界，会退出程序，开销小

  ```
  void mem_write(data_unit data, p_address address);
  ```

- 将硬盘某个地址开始的一段连续的数据加载到内存，若 x_offset + size 越界，会退出程序；开销与 size 大小有关，比内存读写开销大很多，且每次调用都有始动开销

  ```
  void disk_load(p_address mem_offset, p_address disk_offset, m_size_t size);
  ```

- 将内存某个地址开始的一段连续的数据保存到磁盘，若 x_offset + size 越界，会退出程序；开销与 size 大小有关，比内存读写开销大很多，且每次调用都有始动开销

  ```
  void disk_save(p_address mem_offset, p_address disk_offset, m_size_t size);
  ```

- 获取内存和硬盘的读写开销情况，并往控制台输出

  ```
  void evaluate(count_t m_read, count_t m_write, count_t d_read, count_t d_write);
  ```

1.3 内存调用接口

1.3.1 调用细节

- 进程号为 1-999，不会出现超过范围的 pid
- 每个进程可申请的内存空间无限制
- 调用 allocate 的次数会有一个比较小的上限，因此不必太过担心一些 corner case
- 大部分情况下，read 和 write 调用会根据之前返回的 address 来调；但是，我们有时候会故意调用错误的地址，这时候你的程序应该拒绝我们的调用
- 在内存空间不足的时候（不管你有没有实现磁盘的虚拟内存），都可以返回申请内存失败
- 测试用例会杂乱地访问各个进程的各个地址，但是对各个地址的访问会遵循一定的 locality 原则，因此理想情况下你的程序不应该频繁地对磁盘数据换入换出

1.3.2 设计说明

- 实现的时候，不允许申请任何额外的内存（即不允许声明全局变量，不允许调用 malloc）
- 需要使用数据记录内存使用状况的时候（如采用动态划分的话要用链表，用分页的话存储页表等），也需要调用底层模拟接口（即 mem_read, mem_write ）
- 这样设计是保证为了不同实现方法之间的公平性，当然这样使得你们实现起来需要额外花点功夫。建议根据自己的实现方式，抽象出一些工具函数并放到不同的 c 文件里，这样会使得你们的代码的可读性和可维护性大大提高。这也是考验你们的地方之一。

1.3.3 需要实现的接口

- 初始化函数，会在每个测试用例开始调用一次，你可以在这个函数里面做一些初始化操作

  ```
  void init();
  ```

- 进程号为 pid 的进程希望访问 address 处的数据。如果访问合法，往 data 指针里写入数据（通过 *data = xxx），并返回 0；如果访问不合法，返回 -1

  ```
  int read(data_unit *data, v_address address, m_pid_t pid);
  ```

- 进程号为 pid 的进程希望往 address 处写入数据。如果访问合法，往 address 处写入 data，并返回 0；如果访问不合法，返回 -1

  ```
  int write(data_unit data, v_address address, m_pid_t pid);
  ```

- 进程号为 pid 的进程希望申请大小为 size 的空间。如果空间有剩余，往 address 指针里写入申请到的地址（通过 *address = xxx），并返回 0；如果空间不足，返回 -1

  ```
  int allocate(v_address *address, m_size_t size, m_pid_t pid);
  ```

- 进程号为 pid 的进程希望归还 address 处开始的空间。如果访问合法，回收空间，返回 0；如果访问不合法（address 处的空间不属于 pid 进程），返回 -1

  ```
  int free(v_address address, m_pid_t pid);
  ```

1.4 测试说明

1.4.1 测试步骤

1. 将 test.c 文件，你的代码，我们准备的头文件和 bottom.c 一起编译
2. 在 test.c 里的 main 方法包含一系列的测试用例，每个测试用例会先调用你的 init 进行初始化
3. 每个测试用例内部会按各个测试用例的要求调用你的 read, write, allocate, free 方法，并检查返回的结果

1.4.2 测试种类

1. 正确性测试
    1. 进程 A 试图访问不属于它的地址的时候，应拒绝访问
    2. 往一个地址 a 里写入数据 d，如果后面的访问没有修改 a 上的值，那么读取 a 的时候应返回 d
2. 时间测试
    1. 对同一个进程的相近几个位置的访问，除了第一次可能需要在磁盘上换入换出，后续的访问应该都要能在较短的时间内完成
    2. 多个进程的多个距离较远的地址轮流访问，如果调度做的足够好，那么不应该在磁盘上频繁地换入换出
3. 容量测试
    1. 多次申请比较小的空间，理想情况下应该都要能够申请成功
    2. 多次申请比较小的空间，然后退回部分，再次申请几个较小的空间，理想情况下应该都要能够申请成功，而且不需要在磁盘上换入换出
    3. 多次申请比较小的空间，然后退回部分，再次申请一个较大的空间（总用量不超过 128 MB），理想情况下应该都要能够申请成功，而且不需要在磁盘上换入换出，考察的是对内存碎片的处理
    4. 一次申请超过 128 MB 的空间，若支持磁盘换入换出，应该能够申请成功
4. 综合测试

- 即综合上面的几种进行测试

1.4.3 注意事项

- 由于测试用例是一个接着一个跑的，因此请注意你在 `init` 里面把上次用过的变量重新初始化，避免上次的测试对本次测试造成干扰

1.5 提交事项

1.5.1 提交截止时间

- 6 月 11 日 23:59:59

1.5.2 目录结构

- report.pdf
- src
  - xxx.h
  - yyy.c
  - …

1.5.3 注意事项

- 提交到 cms 的时候请将上面的 report.pdf 和 src 压缩到“学号-姓名.zip”，report.pdf 和 src 直接放到压缩文件的根目录
- 请确认 src 目录内不含 call.h, bottom.h, type.h, bottom.o 等我们已提供的文件，即仅需要包含你的代码。
- src 目录里请不要嵌套任何目录结构。
- 请不要在代码里定义 main 函数，否则我们编译的时候无法通过。
- report.pdf 里面请写清楚你的内存管理的实现方式，中途遇到的困难，以及解决方案

1.6 其他

如有任何疑问或建议，请与负责实验加分项的助教取得联系：

- 吴嘉荣：141250148@smail.nju.edu.cn
- 李振昊：141250066@smail.nju.edu.cn