操作系统——内存管理报告

151250176 薛恺丰

实现方式：

段页式内存管理方式

建立的抽象：

进程：管理内存读写是否合法问题，其主要文件是process.c, process.h
内存：仅关注页的分配和缺页处理，其主要文件是manager.c, manager.h
常量及类型：指定了一些常量及类型，以便提高代码的可读性和可复用性，其主要文件是const.h

建立的工具函数：

主要是实现一次存取内存的连续单位，实现可以在内存中存取short, unsigned int等数据，其主要文件是my_lib.c, mylib.h

一些实现细节：

1.数据结构细节：每个进程配备一个段表，整个内存共用一个页表，读写、申请释放内存时先查询段表，再访问页表

2.内存分配：
首先分配进程段表(process_alloc)，再分配虚拟内存页(alloc_pages)

3.内存存取：
首先查看进程段表决定是否合法(process_write, process_read, access_range_check)，不合法直接拒绝请求；若合法，则经过页表取到合适的页(find_memory)，决定是否缺页替换(page_fault_handler)，最终读写指定的内存区域

4.内存释放
首先查看进程段表决定是否合法(process_free, free_range_check)，不合法直接拒绝请求；若合法，则释放虚拟内存中所占的页(memory_free)

遇到的问题：

1.为每个进程建立一张页表所占储存空间太大，无法实现
解决方案：仅保存一张页表，使用双向链表将同一次申请的页连接起来，使用一个标志位确定其是否被占用，仅用一张页表就可以实现内存的管理。

2.对问题一的解决导致了新的一个问题：链表查找速度太慢
解决方案：利用程序的局部性，建立一个共用页表cache，使用FIFO调度方式保存近期存取过的10份页表，则不用每次都沿着链表查到所需要的页，加快了存取速度，减少了查找次数。

3.处理缺页时，必须将需要换入的页写在一个temp内存页，再把需要换出的页写出，最后把temp内存页的内容移动到换入的位置，这样效率较低（即交换两个块需要类似如下的代码：temp = out; out = in; in = temp）

解决方案：（用了一个交换指针的小技巧~）因为页表是用双向链表连接起来的，只需要把指针指到这个temp内存页，将下一次的temp内存页位置指定为需要换出的那一页，就减少了4KB的内存移动。(即仅需要：temp = out; out = in)