会议记录

参加人员：所有人

讨论内容：

1. 对于需要修改的地方进行了讨论，主要为如下几项功能：
   1. 更改为FIFO方式读入请求；
   2. 更改为多级页表多进程；
   3. 更换页面替换算法；
   4. 增加输入输出功能。
2. 针对以上几项功能，讨论了具体实现方式：
   1. 针对更改为FIFO，另建立一进程，使用FIFO向vmm发送访存请求，vmm通过FIFO向请求产生器回复结果（仅针对read操作，回复读到的数据）；
   2. 针对更改为多级页表多进程，采用简单的加大数组的改法，一个维度表示进程，另外两个维度为两级页表，相应的辅助存储初始化、程序初始化以及部分功能作相应修改，对于多进程，只支持固定进程数，不支持进程的创建与撤销；
   3. 针对增加的多进程功能，新建一请求类型，即切换，用于整体更换页表，原先针对单进程的请求不变，都当做对当前页表的操作，此外增加用于输入输出的请求类型；
   4. 增加输入输出功能，通过遍历实际内存、辅助内存和页表来实现；
   5. 更改替换算法，一开始我们还不知道老化算法，所以考虑更改为LRU，LFU与LRU都是淘汰某一参数（使用次数和最后使用时间）最小的页面，所以算法本身是没有区别的，只需要更改统计信息即可。
3. 针对以上功能进行分工，并约定相关接口：
   1. 上述a与c功能属于通信部分，统一分给一个人实现，退出程序功能也在此实现；
   2. b功能除初始化以外，涉及针对大量代码细节的更改，统一分给一个人实现；
   3. b中的初始化以及d功能，为代码的整体添加，可以分给1-2人实现；
   4. 由于通信部分需要与响应部分进行沟通，所以在此约定了请求的格式：
      1. 请求由4个字段组成，请求类型、虚拟地址、操作值和返回值，请求产生时，这一结构携带请求的信息到响应部分，响应完成后，这一结构携带处理结果回复给请求程序；
      2. 其中操作值是一复用字段，在读操作时，操作值由response填写为结果（以供回复给gen\_request），在写操作时，操作值由gen\_request填写为将写入的值，在切换进程操作时，由gen\_request填写为目标进程号；
      3. 返回值为操作结果，例如操作成功、地址越界、权限不足等，实际上没有使用。
4. 会议还讨论了其他问题：
   1. 会议再次讨论了OO课程的意义。在OO作业中，我们训练规格的方法是这样的：在个人完成的500行规模，周期为1星期的程序里，自己写规格给自己看，也就是说，写一些文档，自己告诉自己，自己昨天写的函数是干什么用的。这显然是没有任何意义的。而OS作业，需要组员间的合作，各自书写不同的代码对相同的数据进行操作，这个时候规格的作用就自然而然地体现出来了。OO课程的这种“没有困难，创造困难也要折腾人”的逗比思想由此可略见一斑。
   2. 会议讨论了关于内存分配的问题。在实际中，不可能每运行一个进程，就在交换分区中占用3GB空间，所以页面的分配肯定是在程序运行过程中，（至少有一部分是）根据程序需求而动态分配的。我们本来的设想是访问时分配，也就是至少对所指虚地址进行一次读、写或执行时，才为所在页真实地分配辅存。在此基础上，会议还讨论了添加、删除进程，以及fork的做法。但是后来一看，给的代码辅存就是程序的地址空间大小，所以就还是按照原来的写了，这样动态建立进程和fork也做不了了。
   3. 会议讨论了关于将页表放入主存的想法。后来发现，这样要涉及到加大页面大小，而且要有实际价值，还要把页表换出，这太难了，完全不会，感觉写这个有点zuo，就没写。
   4. 后来我们讨论了一下关于页面老化算法的问题。我们感觉，这个页面老化算法的实现还是需要商榷的。以x86为例，实际的x86系统通常有100万个页面，如果仅靠全部扫描一遍来更新信息，效率好像不是很高，特别是这还是一个中断响应程序，经常性地使系统长期阻塞可能不是很好。但是我们想了半天，感觉即使能把整体移位做成O(1)复杂度，其代价却是将每次页面访问更新信息的复杂度提升为O(logn)（更新统计信息），而且常数巨大，而页面访问比整体更新要发生的频繁的多。但是后来我们又想，由局部性原理，每一段时间内访问到的页面是很有限的，也就是说，这个O(logn)仅发生在每个页面在一段时间内第一次访问时，而这只会在比较小的一部分访存的时候发生。我们估计以我们的做法，大概在一段时间内，小于1%的页面被访问的时候，后者要好一些。真实情况下的这一数据为多少，以及现行linux中的实际页面替换算法，还有待进一步研究。我们推测，事实上的页面管理肯定是有预淘汰功能的，一个直观感受是我们很少能看到物理内存真正被占满，另一个方面，一旦物理内存被占满，此时进行的页面替换将可能面临两次磁盘访问，响应时间将大大延长。另外，对于不同用途的页面可能有不同的淘汰策略，例如对于只读页面或代码段，这些页面通常不会被更改，因而换出时通常不需要写回，系统可能更倾向于选择这些页面进行替换。