操作系统课程设计

实验报告

姓名：　张云非　学号：　201600301144

一、实验目标

内存管理模块运行可视化：以一个最简单的进程的运行全过程为例，展示内存管理模块中函数的执行情况。

二、实验环境

代码阅读与测试环境：Visual Studio 2013 + bochs 虚拟机 + Linux-0.11 项目（来源于网络）

数据提取环境：Vmware 虚拟机运行 Debian 系统 + bochs 虚拟机 + gdb 调试（由陈宇翔同学小组提供的环境）

可视化编程环境：Visual Studio 2015 + OpenSceneGraph （3.4.1 版本，是一个使用 C++的基于 OpenGL 的面向对象的 3D 图形引擎）

三、实验过程

第一周至第九周：阅读并调试 Linux0.11 源码

经过了一系列的搜索和实验之后，决定网络上提供的一个 VS2013 的项目来调试和分析源码。当然，为了能在 VS2013 中运行和调试，这个项目中的汇编代码都用了 masm 的风格重写了，简化了一部分阅读汇编源码的难度。同时，由于在 IDE 中调试内核，使得修改和重新编译变得十分迅速。

由于本次实验的目标设定为内存管理模块的可视化，主要的关注点就在完成内存管理功能的 memory.c 中。除此之外，由于要关注进程的运行过程，还涉及了一部分进程管理和磁盘管理的内容，例如 fork.c，exec.c，exit.c 等等。

第十周至第十二周：调试内核并提取系统运行日志

由于使用了基于 VS2013 的项目调试源码，所以修改源码以输出信息似乎是

一个不错且唯一的选择（无法使用 GDB 这样的调试工具）。但是在 Linux 内核编程的过程中，修改源码极其容易导致段错误的发生，而本次实验的目的又是内存管理模块，这就使得提取数据的难度进一步增加。所以最后只得放弃，并改用由陈宇翔同学小组提供的环境提取数据。

第十三周至第十五周：编写程序，将系统运行日志可视化

图形编程最为典型 API 当然是 OpenGL，但是使用 OpenGL 编程却又过于复杂，所以本次实验中使用了基于 OpenGL 的面向对象封装的图形库 OpenSceneGraph 来可视化产生的系统运行日志。

四、实验结果

1. Linux0.11 系统源代码分析

本次实验分析的主要代码集中在以下处于 memory.c 中的函数内：

get_free_page *---得到内存中空闲的一页*
free_page *---释放一页*
free_page_tables---释放一整块内存
put_page *---将一页映射到线性地址空间*
un_wp_page *---取消一页的写保护*
do_wp_page *---响应写保护异常*
write_verify *---确认一页是否只读*
share_page *---共享内存一页*
do_no_page *---响应缺页异常*

这些函数虽然功能各异，但是却都体现除了 Linux0.11 内核代码的特点。总结下来大致有三点：

其一是面向过程，也就是体现了 C 语言的命令式编程语言的特点。

虽然这一点看上去平平无奇，但是这却是 memory.c 中的代码的一个显著特点。与磁盘管理模块和进程管理模块都不同的是，内存管理模块中几乎不存在数据结构，仅有的一个就是 mem_map 这个管理内存分配情况的全局映射数组。如果不简单地使用二分法来区分面向对象和面向过程，而是将它们之间的区别更加地细分，那么可以认为，维护数据结构完整性的程序，更加偏向于面向对象，而实现特定函数功能的代码，更加偏向于面向过程。如此一来，与内核之中的其他模块相比，内存管理模块无疑是最能体现内核的面向过程属性。

其二是大量的位操作与类型转换。

Linux0.11 使用了二级页表的内存管理机制（分段管理的细节则被屏蔽在 CPU 给出的线性地址中，故无需讨论），本质上的原理可以说并不是很难，但是具体实现的时候就包括了大量的位操作，以及指针类型与普通类型的转换等等。这样编程的一个原因很可能是为了最大限度的重用变量，也就是减少内核对内存的使用，不过这样也不可避免地导致了可读性的下降。

其三是和汇编语言以及硬件紧密结合。

这一点无需多言，内存管理必须有硬件的支持。而为了速度的需求，函数中也会不可避免地使用汇编语言（例如 *get_free_page* 函数）。同时，线性地址和物理地址的区分又需要时刻注意。

2. 系统运行过程的形式化描述方法

由于是典型的面向过程风格的代码，提取出的数据当然也很难组织成结构化信息，所以系统的运行过程，也就表示为了一组函数，以及对应的函数中的参数或者中间变量的信息（提取的系统运行日志的具体内容见附录）。

而为了展示内存管理模块的功能，选择使用了从一个简单进程的一生，看它的运行过程：

简单进程：

一个 Hello World 程序的运行过程。

由 gcc1.4 版本直接编译

```
#include<stdio.h>
int main(){
printf(“Hello World!”);
return 0;
}
```

为.out 执行文件，即可获得。

无论是使用系统镜像中自带的 a.out（软盘镜像自带），或者是写上面的代码然后使用 gcc1.4（硬盘镜像自带）编译，效果都是一样的，而且十分方便。

3. 内核运行数据输出方法

正如之前所述，本次提取系统运行数据的方法是使用了由陈宇翔同学小组提供的环境，所以在此不作更多说明。但是值得一提的是过滤数据的方式，用 GDB 脚本在函数之中增加断点，那么可能会输出大量的无关信息，将有效的信息淹没，尤其是在内存管理这样的频繁被调用的模块之中。那么为了准确地提取所需要的简单进程运行时内存管理模块的运行情况，那么过滤的方法就十分重要了。通过阅读和调试源码，可以得知：如果使用命令行，直接执行的第一个进程，它的进程号是六；而描述当前进程的一个全局变量的指针是 current，那么使用一个 current->pid == 6 的条件断点，则可以极其有效地过滤数据。

4. 可视化方法的描述

在 memory.c 中，称得上是数据结构的只有 mem_map 这一个数组，但是为了可视化的效果，本次实验中还是将 CPU，mem_map 以及内存抽象为三个不同的实体（虽然它们的地位显然不是对等的），并且使用它们之间的信息传递来模拟系统运行的过程。

更加具体的可视化编程过程的描述如下：（代码见附录）

首先建立了一个表示各种对象的基类。

然后建立了一个比特，一个 32 位字，一块 4k 的页面等对象类型。

然后将信息传递的对象（也就是线段和文字）封装。

写一个能够按时间调用回调函数的动画执行类。

将基本的动作（例如从一个位置画到另一个位置的线）封装成函数。

读入生成的日志文件。

根据函数以及其对应的信息，为每个函数写一个可视化函数。

5. 系统运行过程实例

生成的系统运行视频总共长度为 17 分 35 秒（见附录）。

可视化视频的构图如下：

使用左边的一个大方块，用来表示 CPU。

使用右边的一个大矩形，用来表示内存。

而在内存上面的一个小矩形，用来表示数据结构 MEM_MAP 这个内存映射数组。

使用线段和文字的移动，用来表示它们三者之间的信息传递过程，以及 CPU 对应过程的处理（认为是 CPU 的思考过程）。

使用内存上面的更小的单位，来表示内存中的一页（4K 大小）。

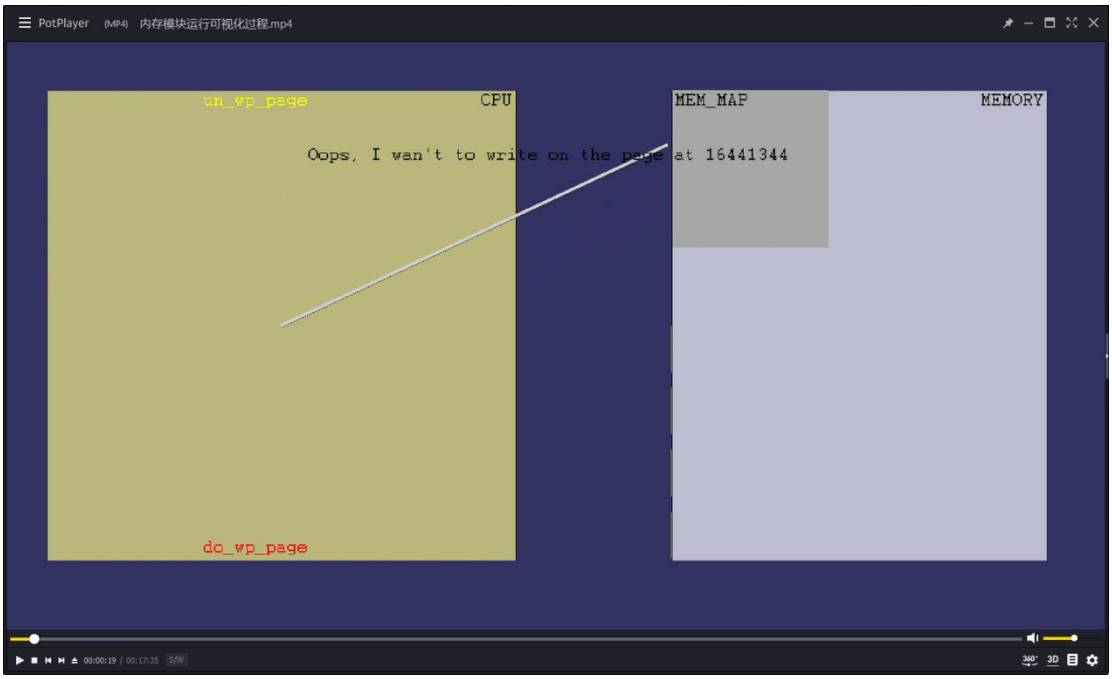
一个最简单的进程的运行的情况如下：

孕育期-进程创建过程，操作系统在为进程的执行做准备

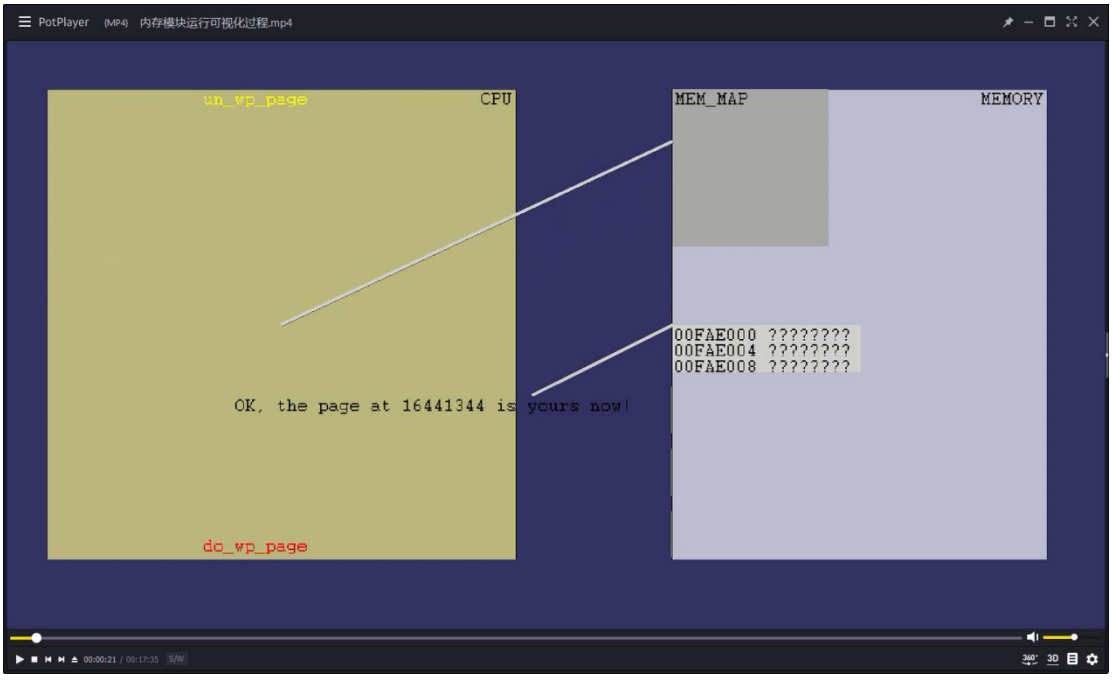
对应的时间为 0:05~13:41

在一开始首先是解决页面写保护错误处理函数 do_wp_page 的过程：

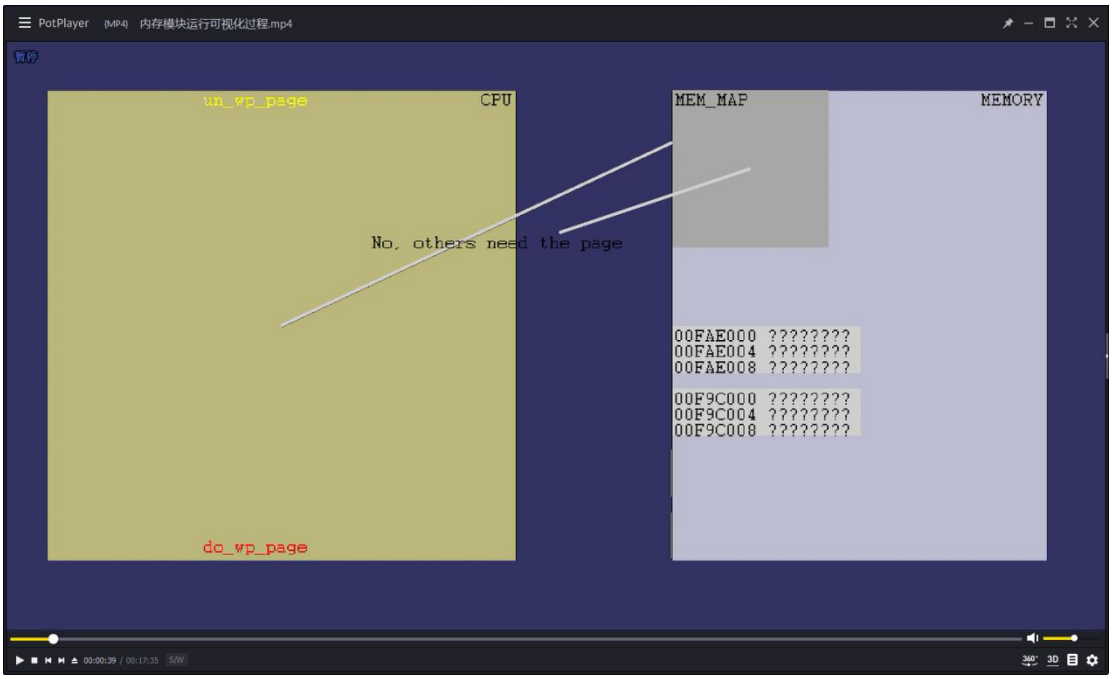
do_wp_page 转换线性地址为物理地址之后，就调用 un_wp_page 函数取消页面的写保护：

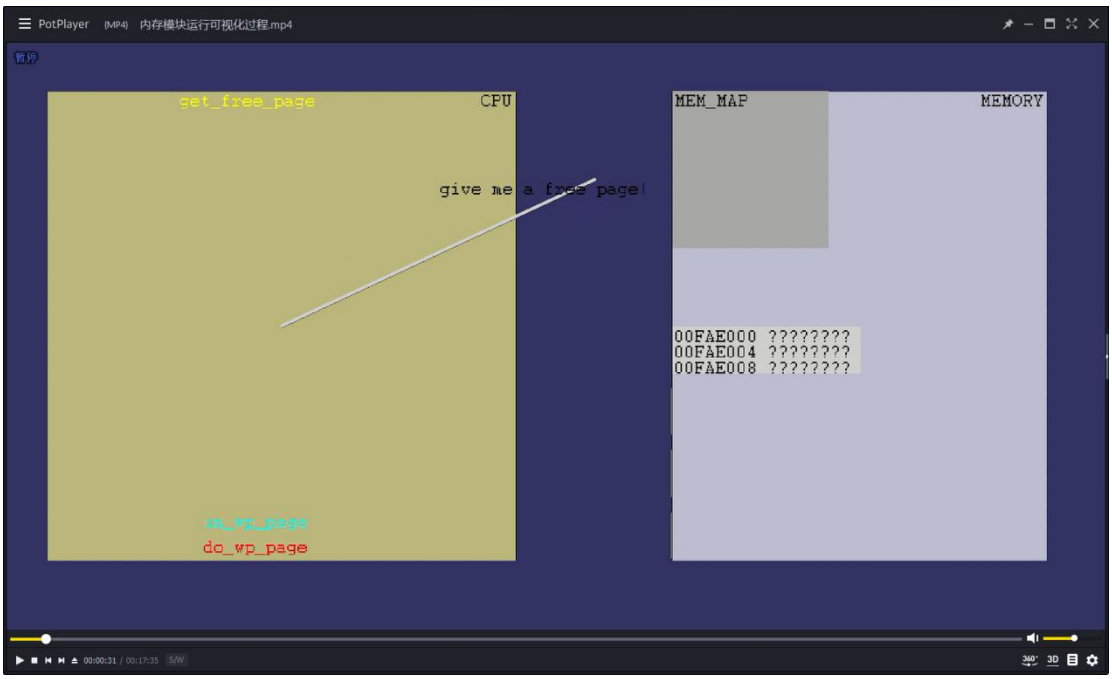


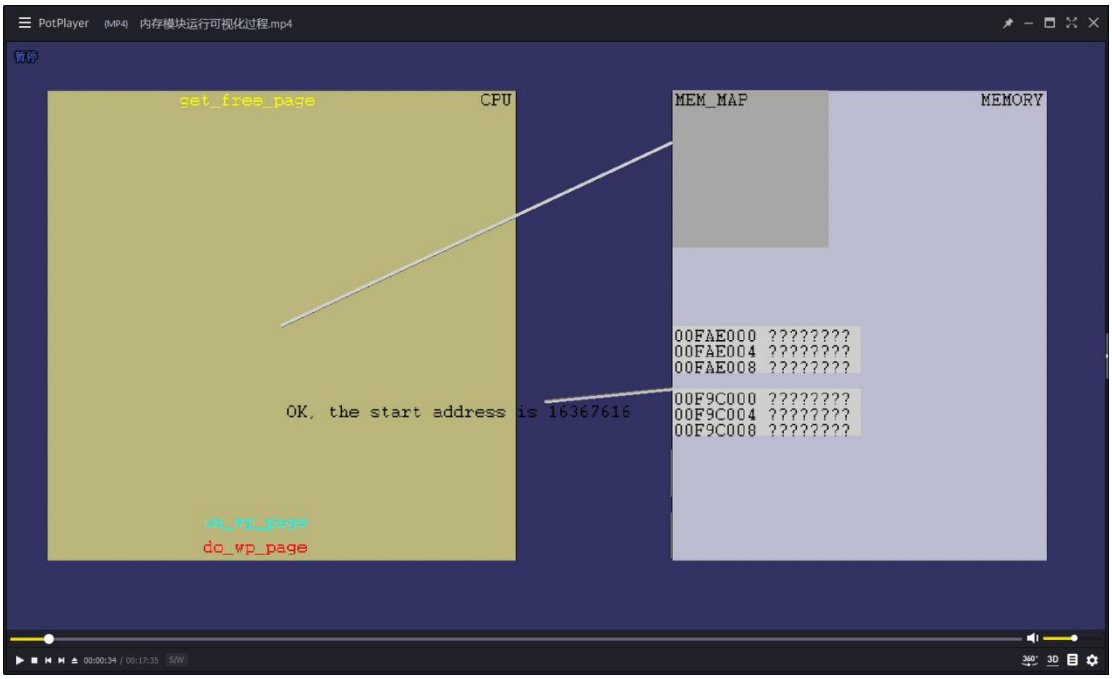
un_wp_page 函数中，如果发现页面只有当前进程在使用，而没有其他进程共享，则直接取消页面的写保护，将页面分配给当前进程使用即可：



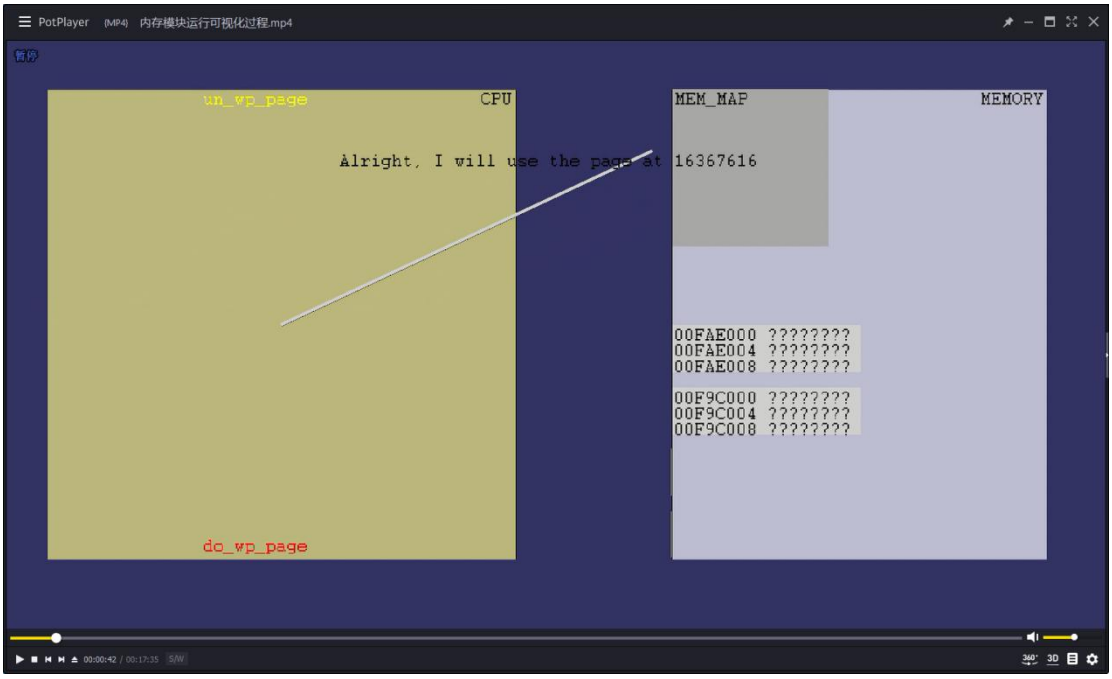
而如果其他进程在共享这个页面，那么就需要复制这页（写时复制），调用了 get_free_page 函数：

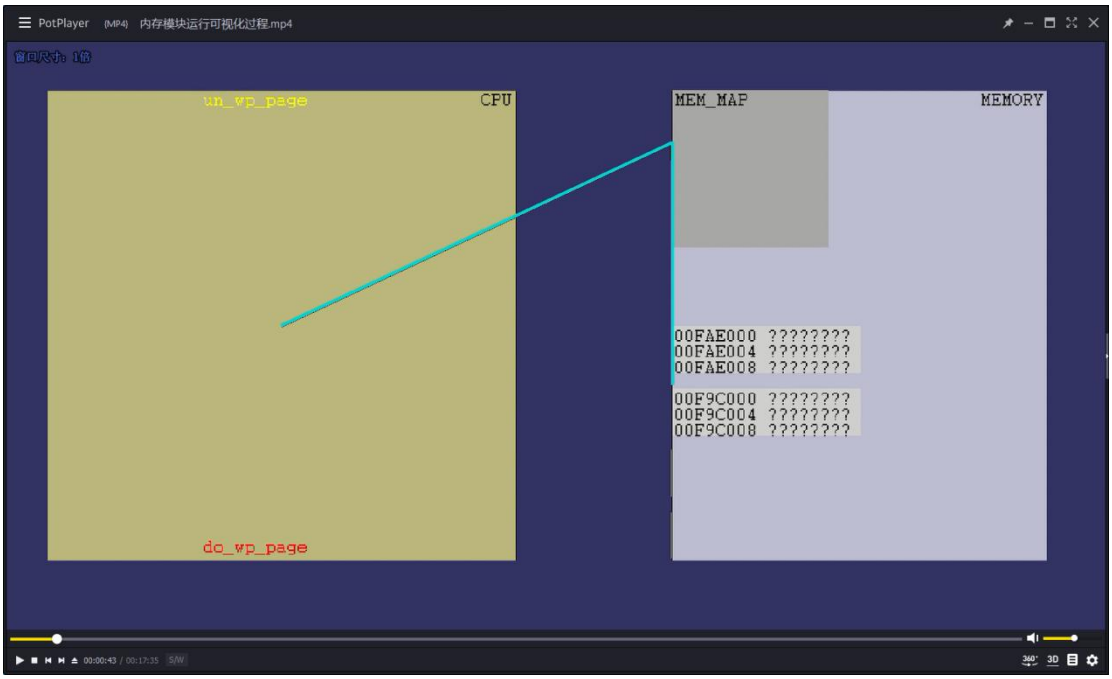

PotPlayer (MP4) 内存模块运行可视化过程.mp4
暂停
un_wp_page
CPU
MEM_MAP
MEMORY
No, others need the page
00FAE000 ????????
00FAE004 ????????
00FAE008 ????????
00F9C000 ????????
00F9C004 ????????
00F9C008 ????????
do_wp_page

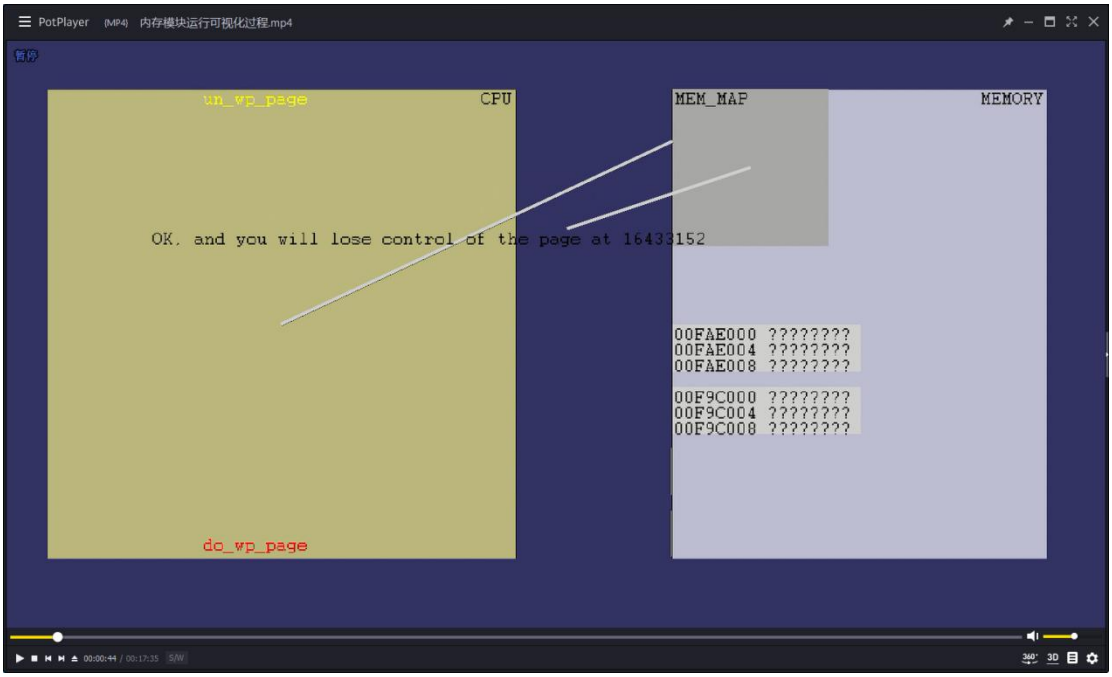

PotPlayer (MP4) 内存模块运行可视化过程.mp4
暂停
get_free_page
CPU
MEM_MAP
MEMORY
give me a free page!
00FAE000 ????????
00FAE004 ????????
00FAE008 ????????
un_wp_page
do_wp_page



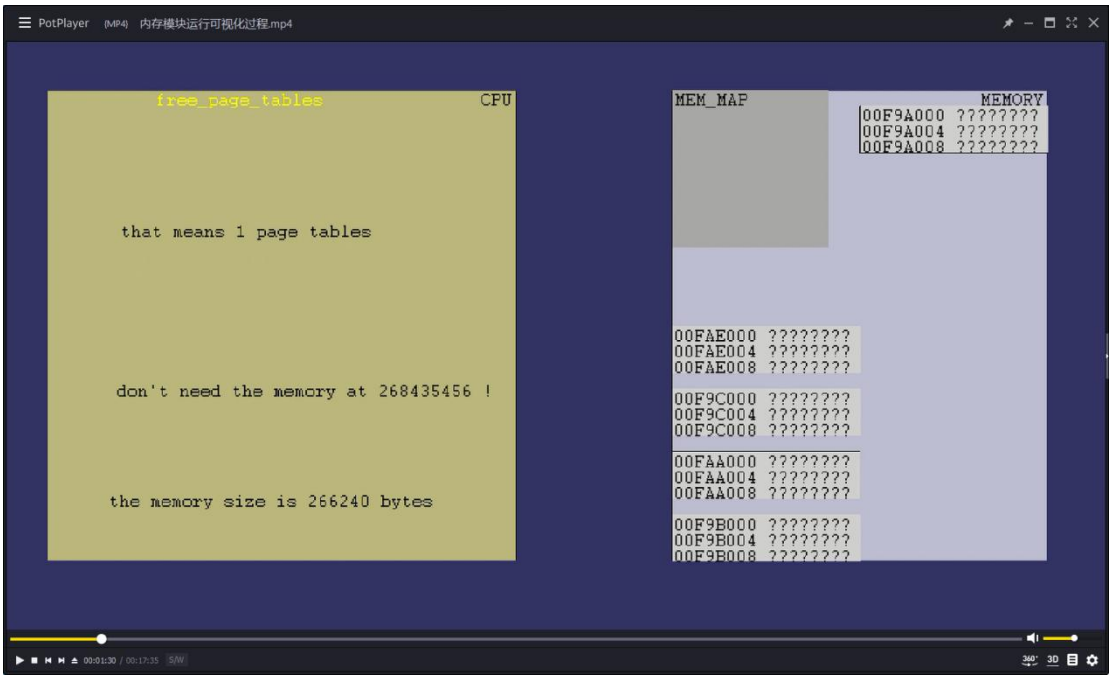
然后进程则使用新的一页，同时取消了进程对原来页的使用权：

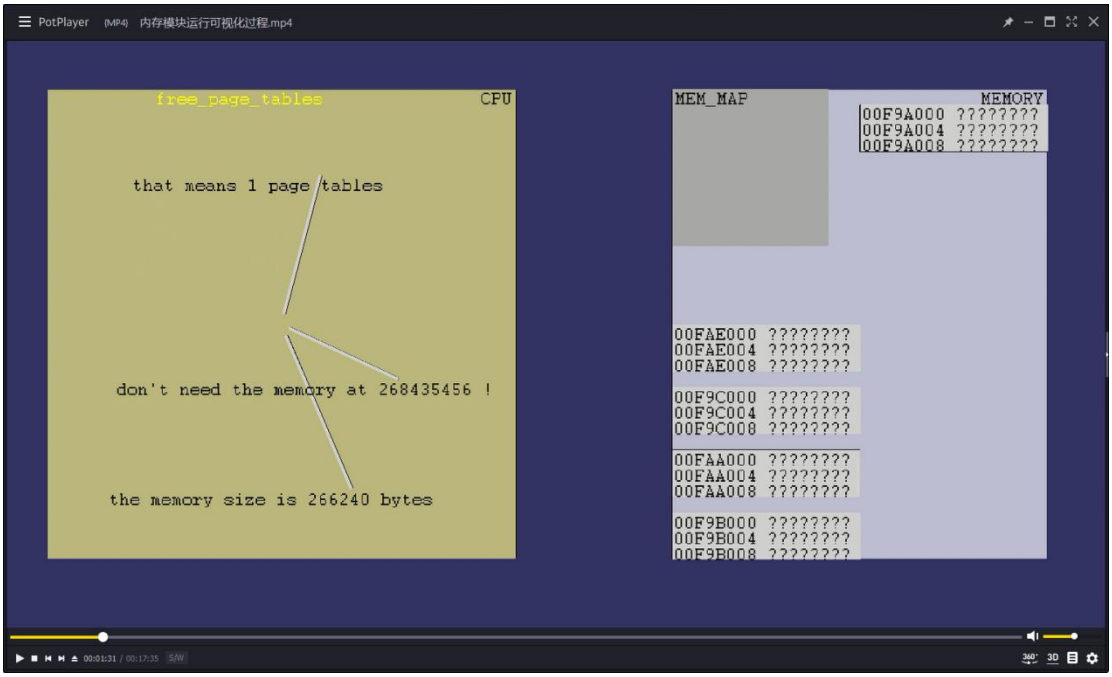




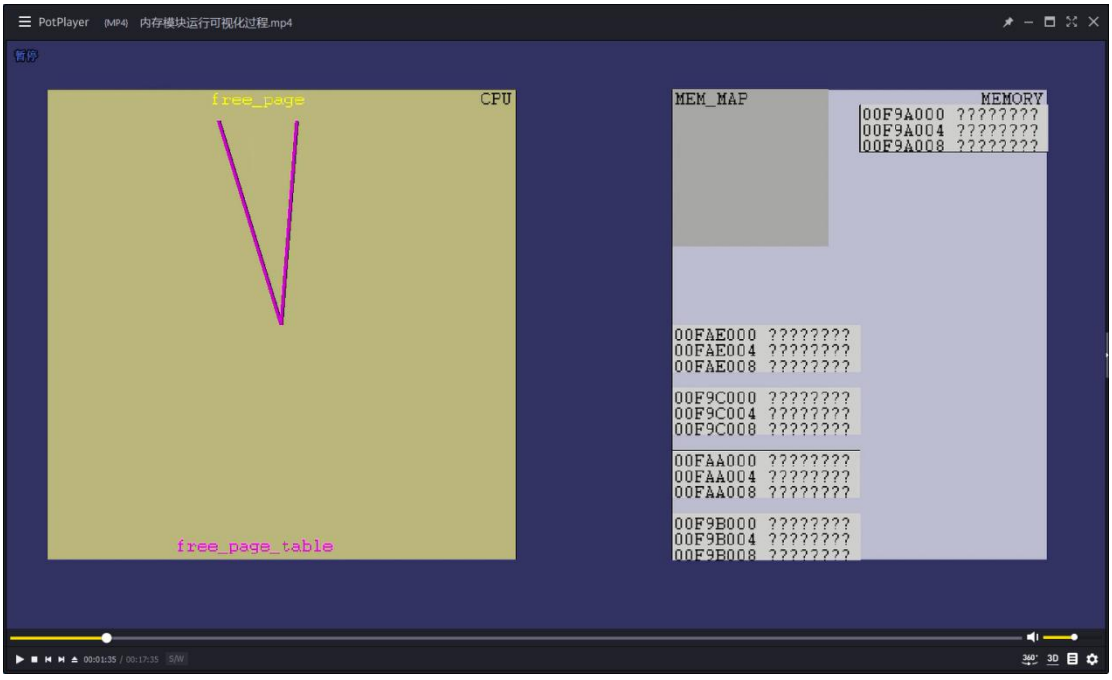


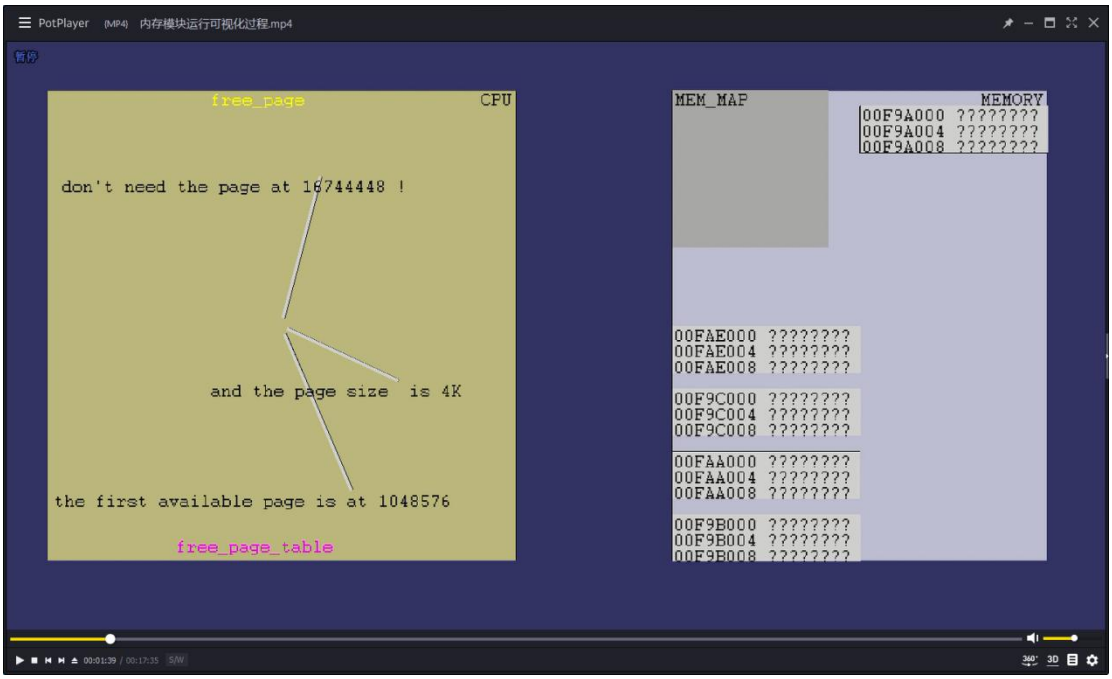
在重复的几次写保护错误之后，就开始了释放父进程所占用页面的过程（由于进程使用 fork 函数创建，所以共享着父进程的页面），这一过程调用 free_page_tables 函数实现：

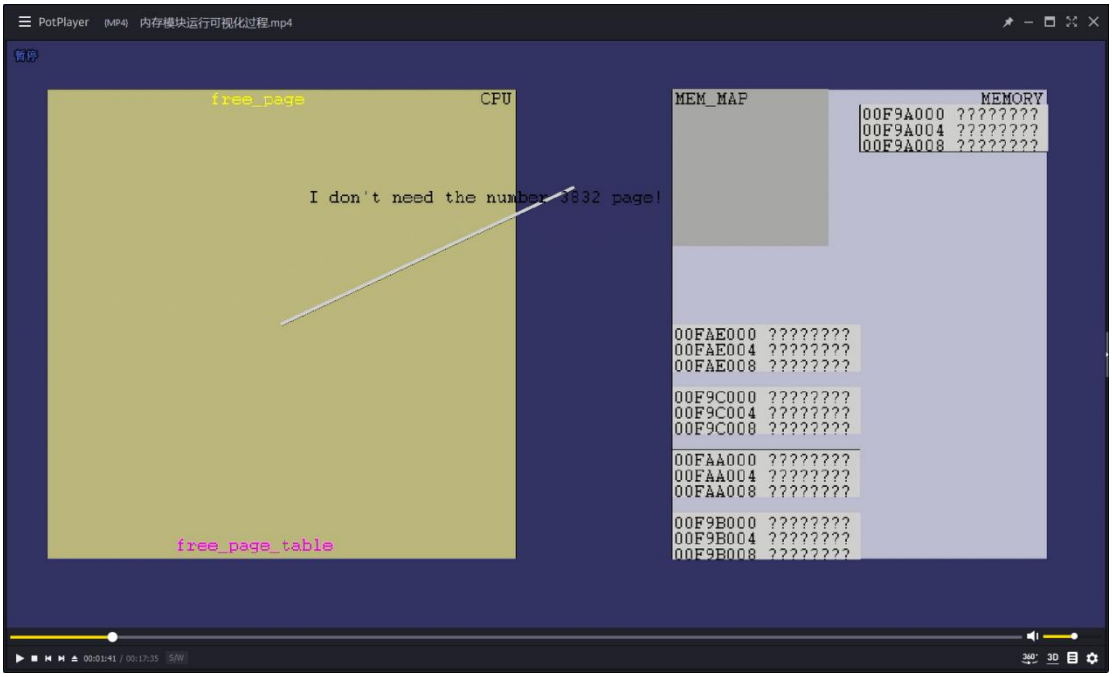

PotPlayer (MP4) 内存模块运行可视化过程.mp4
free_page_tables
CPU
that means 1 page tables
don't need the memory at 268435456 !
the memory size is 266240 bytes
MEM_MAP
MEMORY
00F9A000 ????????
00F9A004 ????????
00F9A008 ????????
00FAE000 ????????
00FAE004 ????????
00FAE008 ????????
00F9C000 ????????
00F9C004 ????????
00F9C008 ????????
00FAA000 ????????
00FAA004 ????????
00FAA008 ????????
00F9B000 ????????
00F9B004 ????????
00F9B008 ????????


PotPlayer (MP4) 内存模块运行可视化过程.mp4
free_page_tables
CPU
that means 1 page tables
don't need the memory at 268435456 !
the memory size is 266240 bytes
MEM_MAP
MEMORY
00F9A000 ????????
00F9A004 ????????
00F9A008 ????????
00FAE000 ????????
00FAE004 ????????
00FAE008 ????????
00F9C000 ????????
00F9C004 ????????
00F9C008 ????????
00FAA000 ????????
00FAA004 ????????
00FAA008 ????????
00F9B000 ????????
00F9B004 ????????
00F9B008 ????????

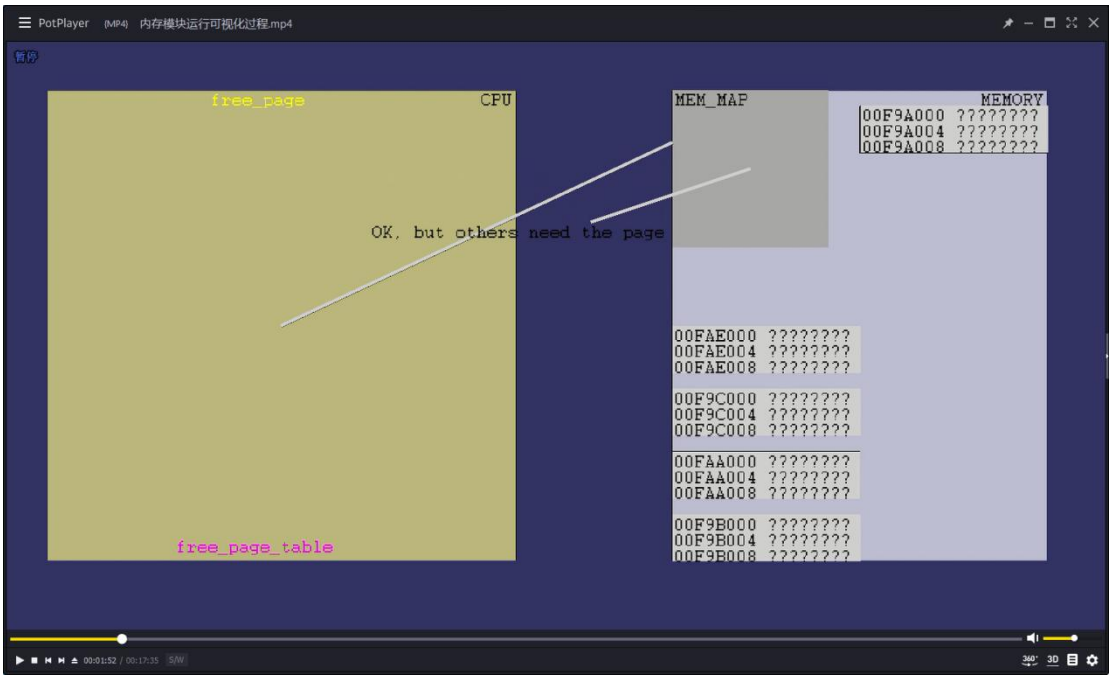
而 free_page_tables 函数则调用 free_page 函数，每次释放一个页面，来释放整块的连续内存：



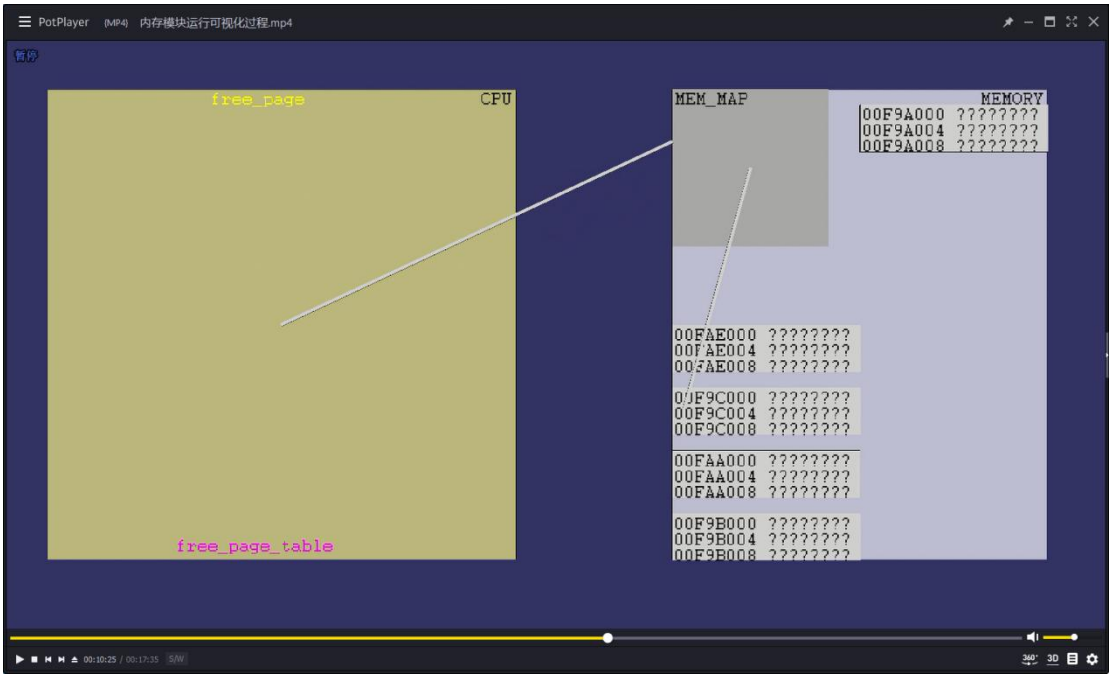


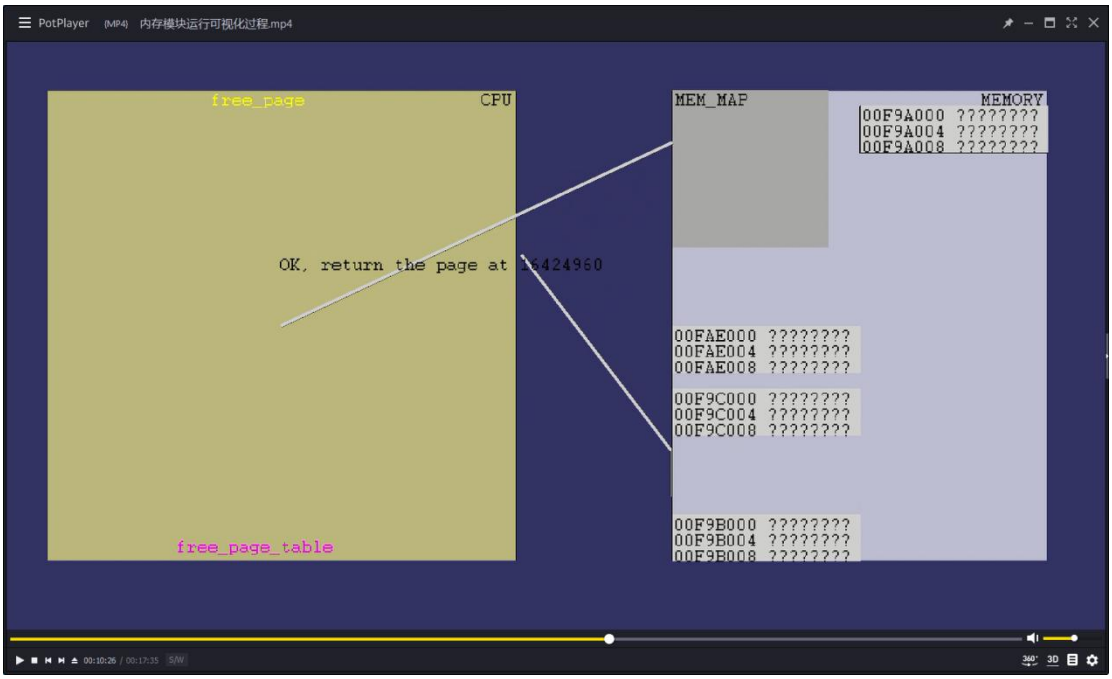


由于大部分是共享的页面，所以并没有真正的释放内存页面，而仅仅是取消了进程对这个页面的使用权：

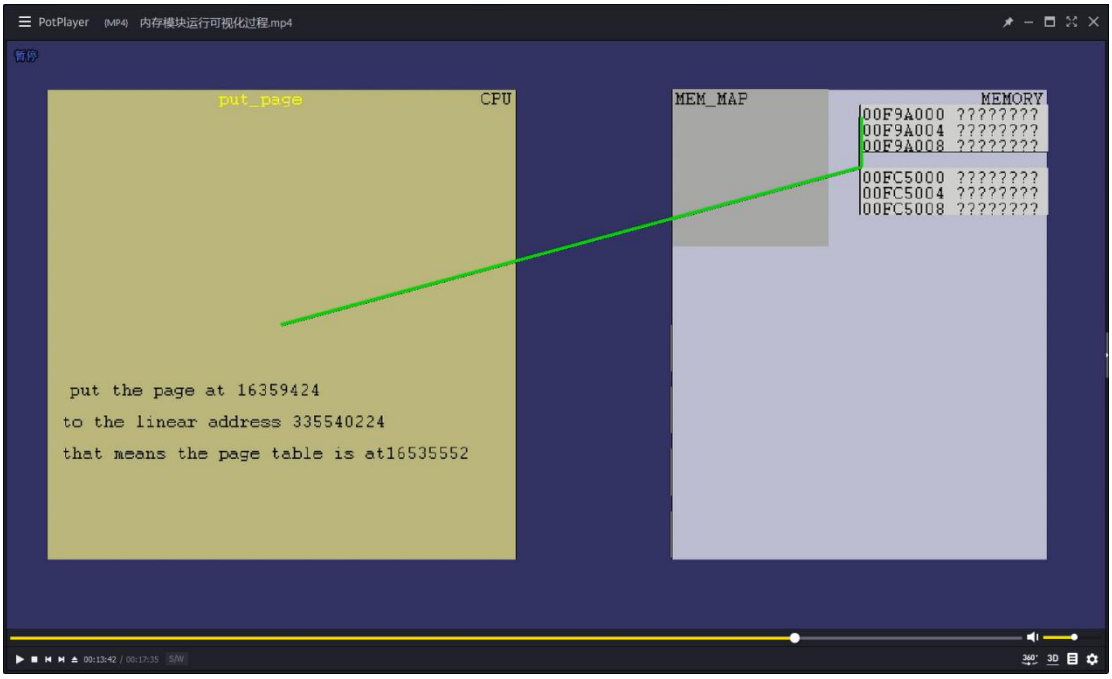


当然，也有当前进程独占的页面，那么则会被释放：





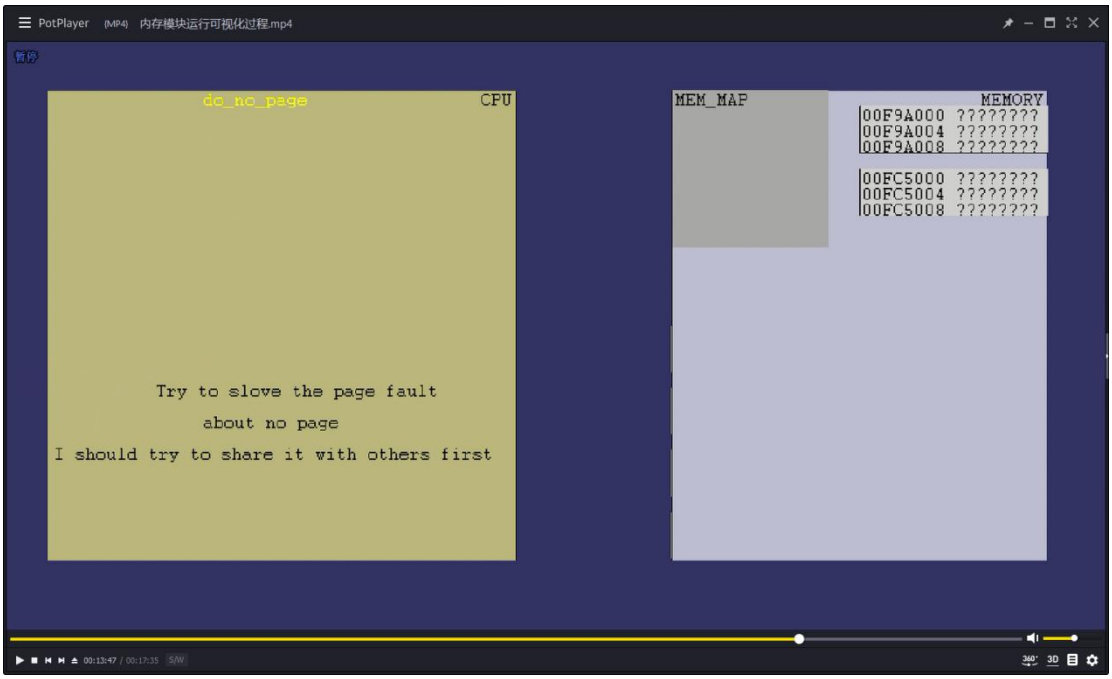
然后设置进程的局部描述符，并把进程执行的传入参数的页面使进程能够使用：



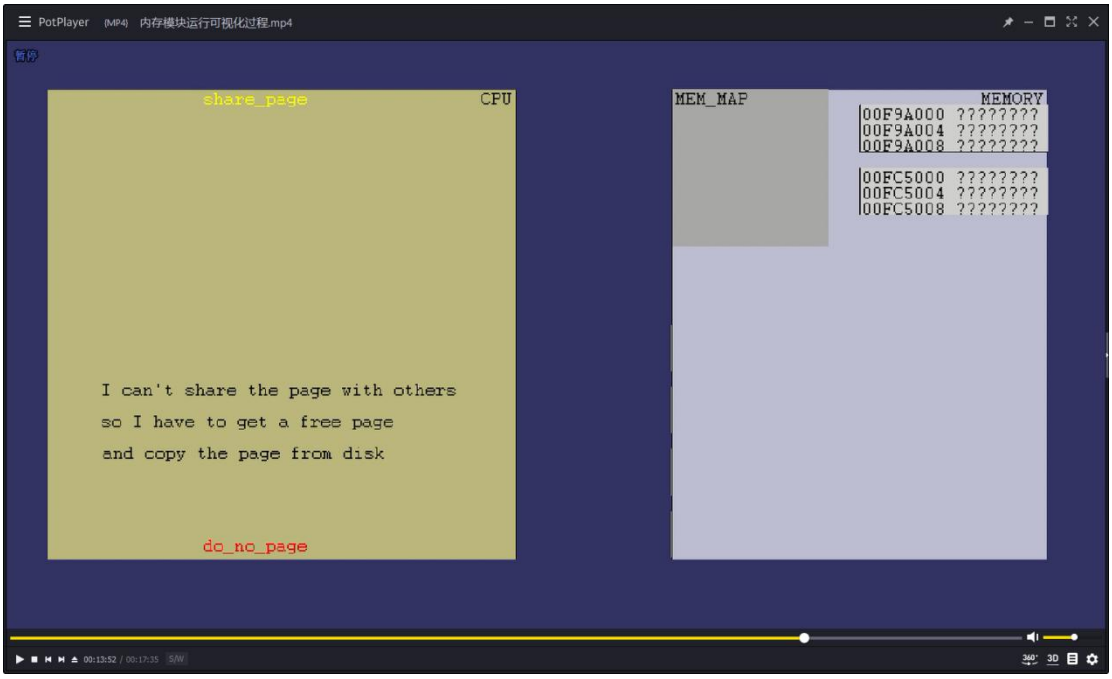
成长期-进程执行过程

这一时期对应的时间为 13:42~16:06

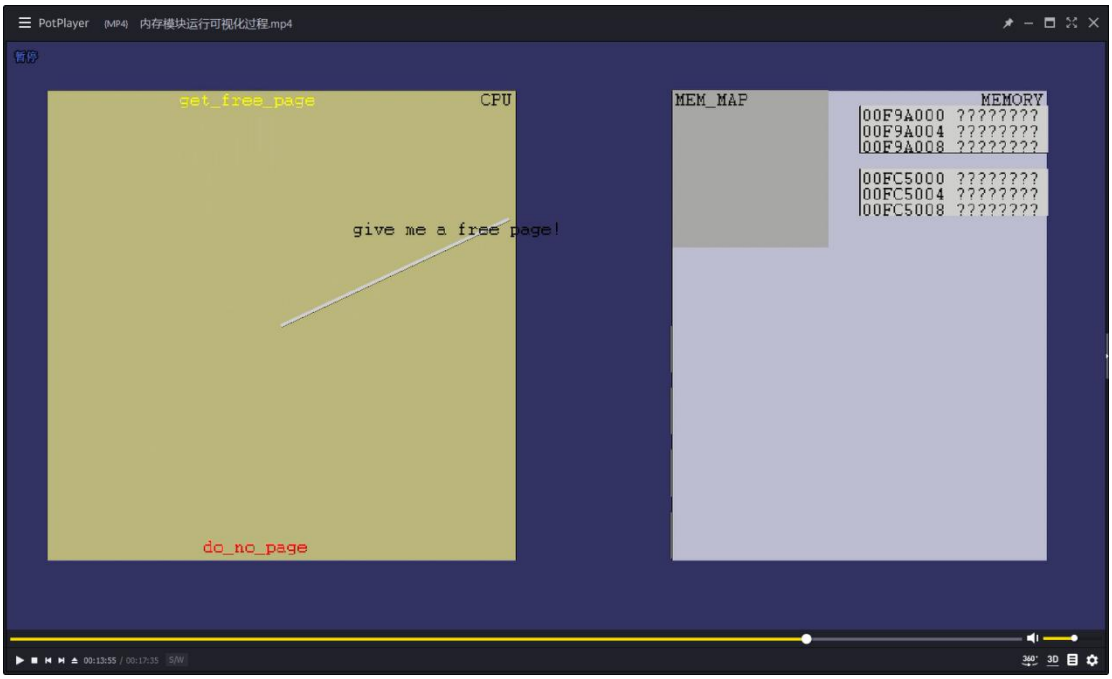
一开始首先是处理缺页异常的 do_no_page 函数：



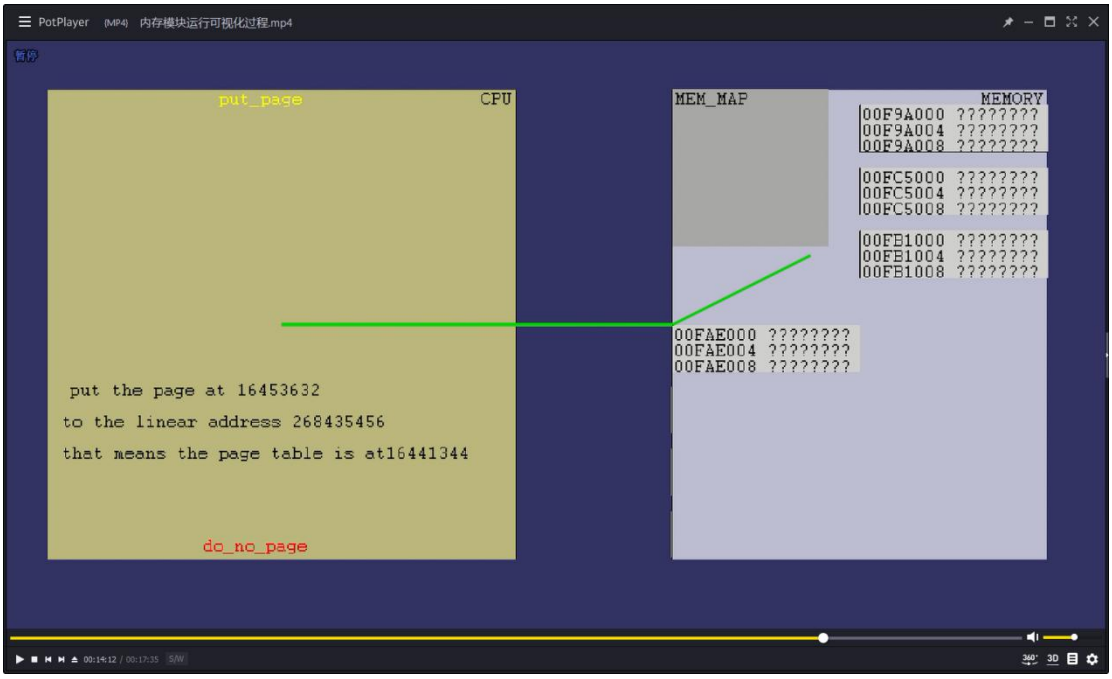
首先会尝试去和其他进程共享一个相同的页面，即调用 share_page 函数：



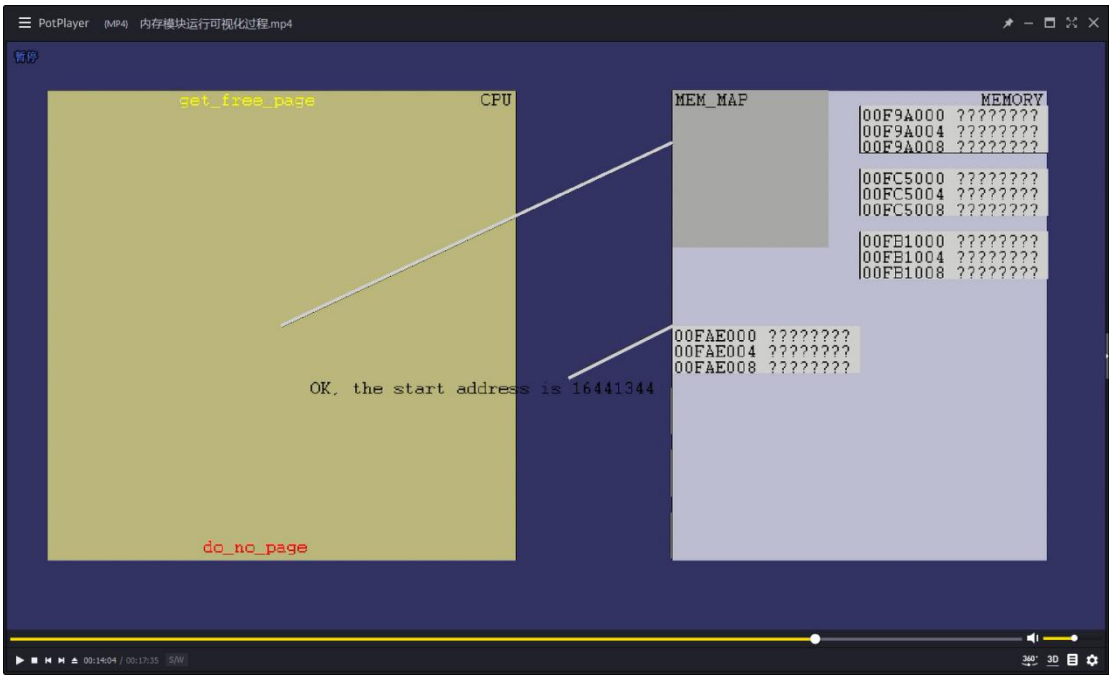
但是无法共享页面时，就要申请一个新的页面：



然后调用 put_page 函数将申请到的页面，映射到线性地址上去：

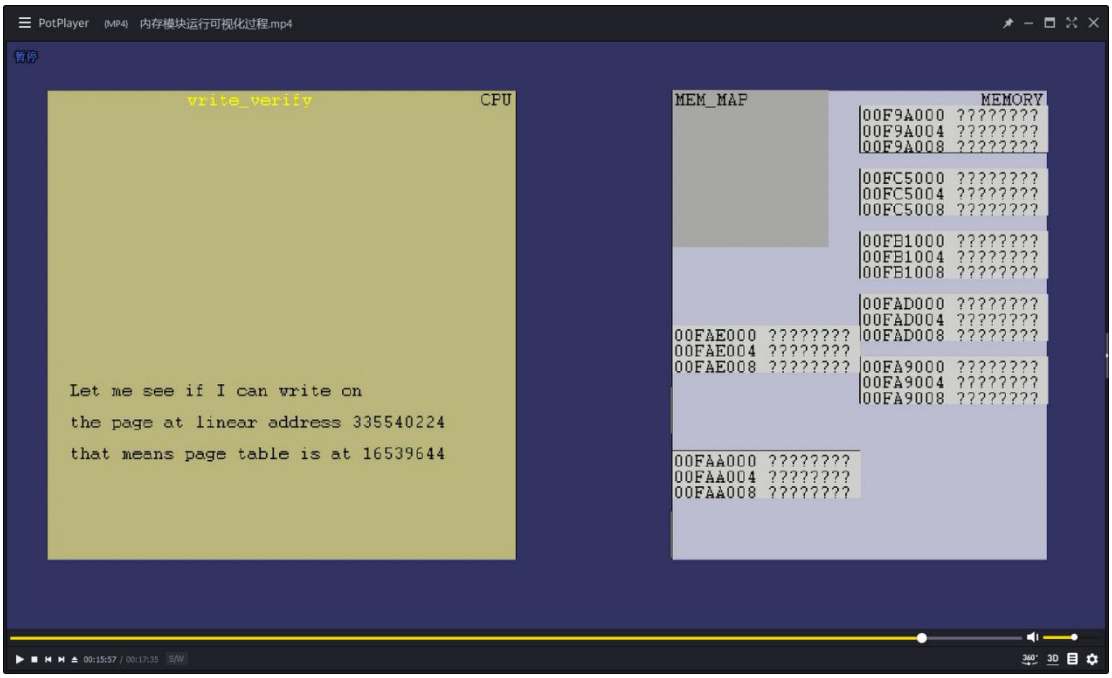


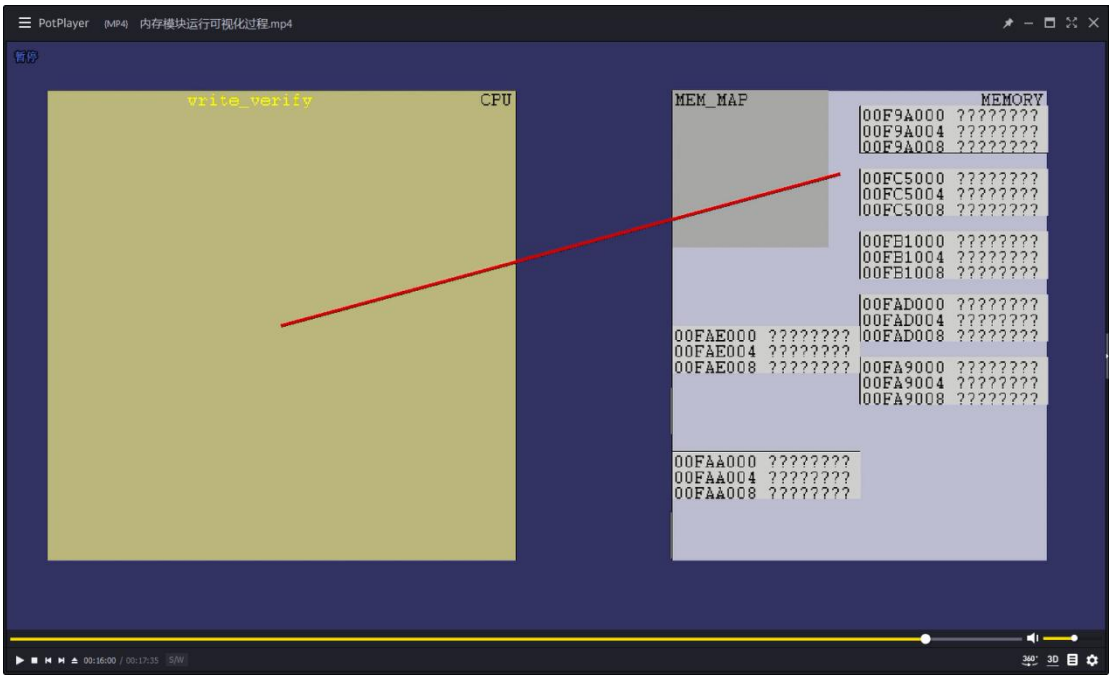
如果这时页表也无效，那么 put_page 函数会申请一个页面给页表使用：

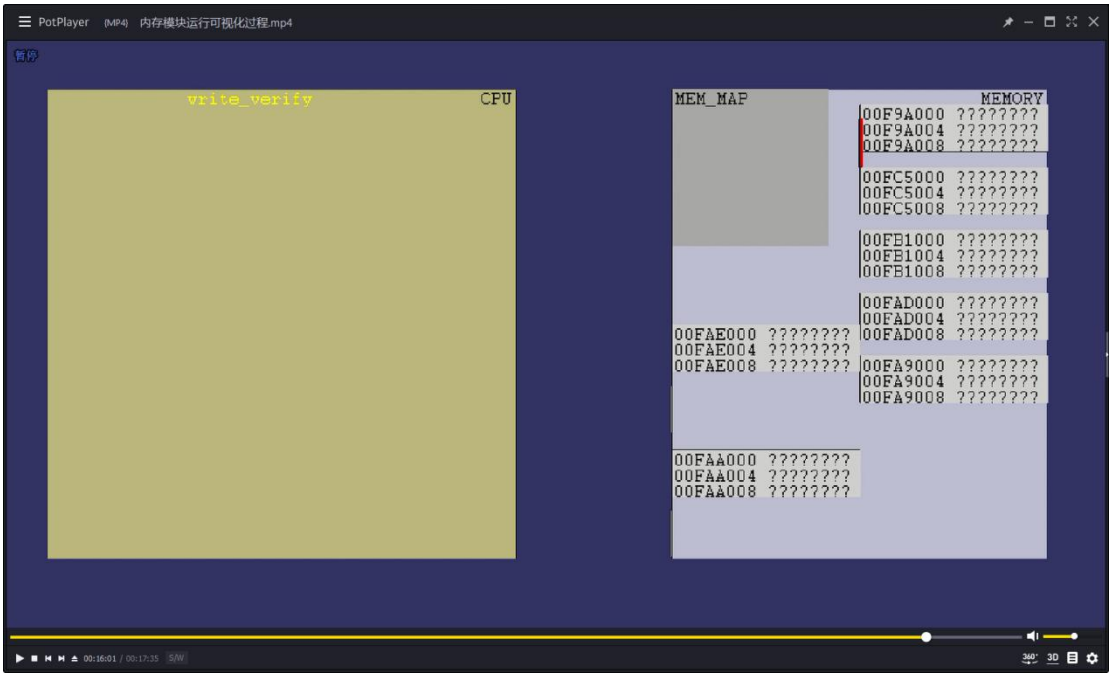


如此重复了多次，由于进程简单，所以只会引起缺页异常。

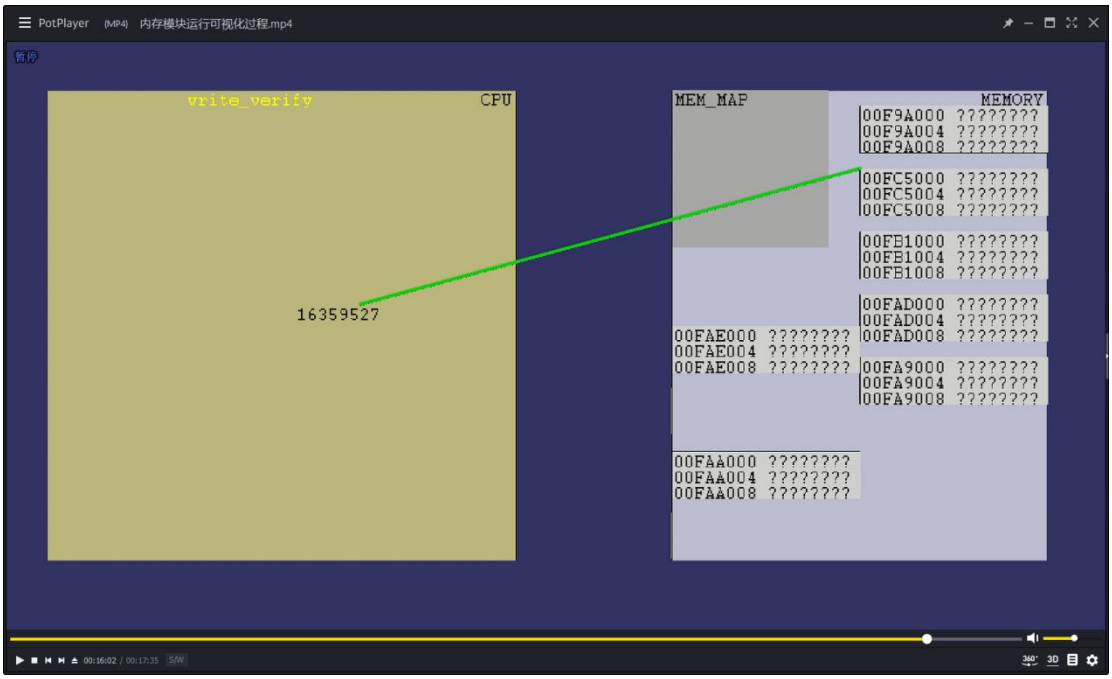
最后会调用一次 write_verify 函数，确定内核是否能向一个页面写入，这时由于进程在控制台输出了 Hello World!，而产生的系统调用中对内存管理模块的使用：

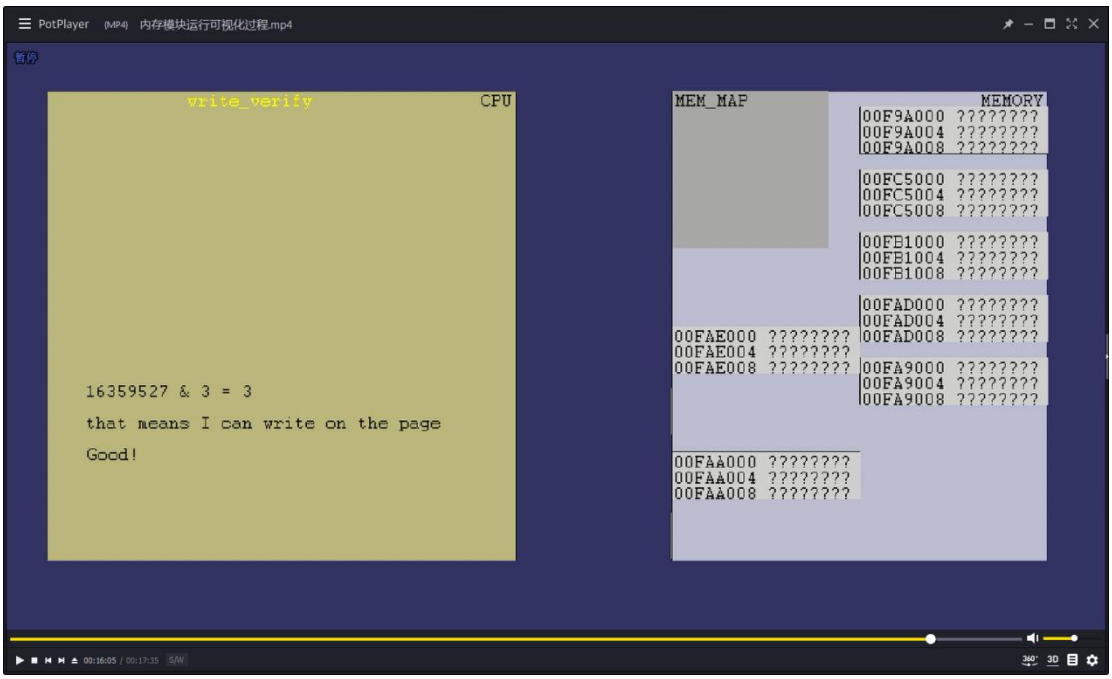






通过查页表的对应项的 W/R 位就可得知：



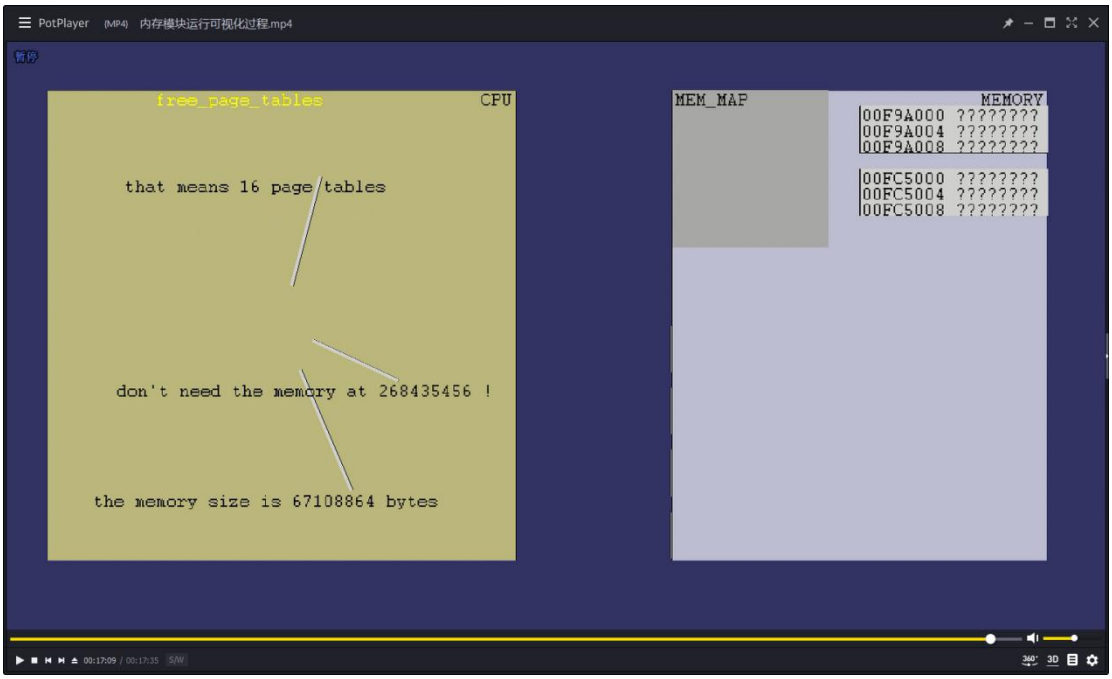


至此，进程主体运行完毕

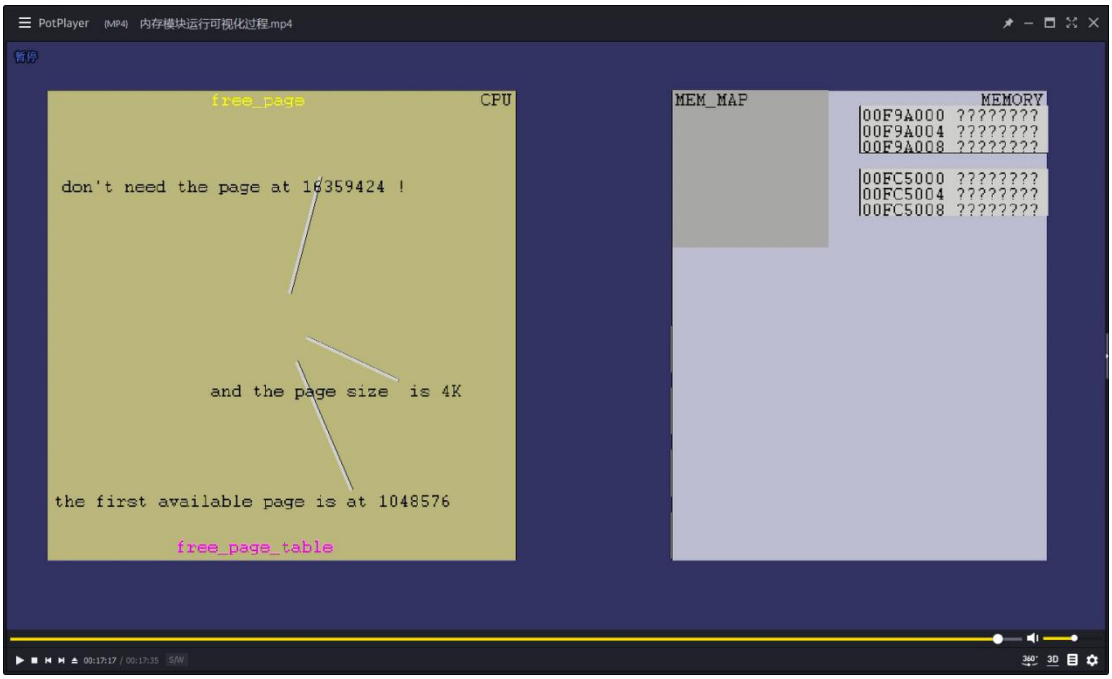
衰亡期-进程销毁过程，操作系统回收进程占用的资源

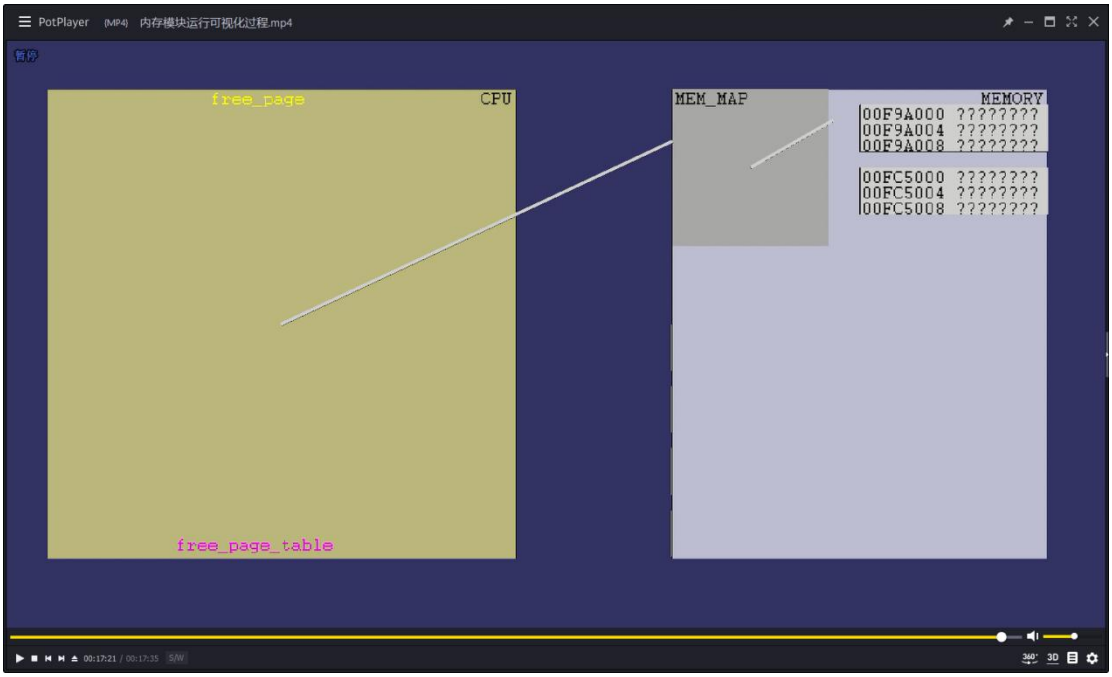
这一时期对应的时间为 16:07~17:32

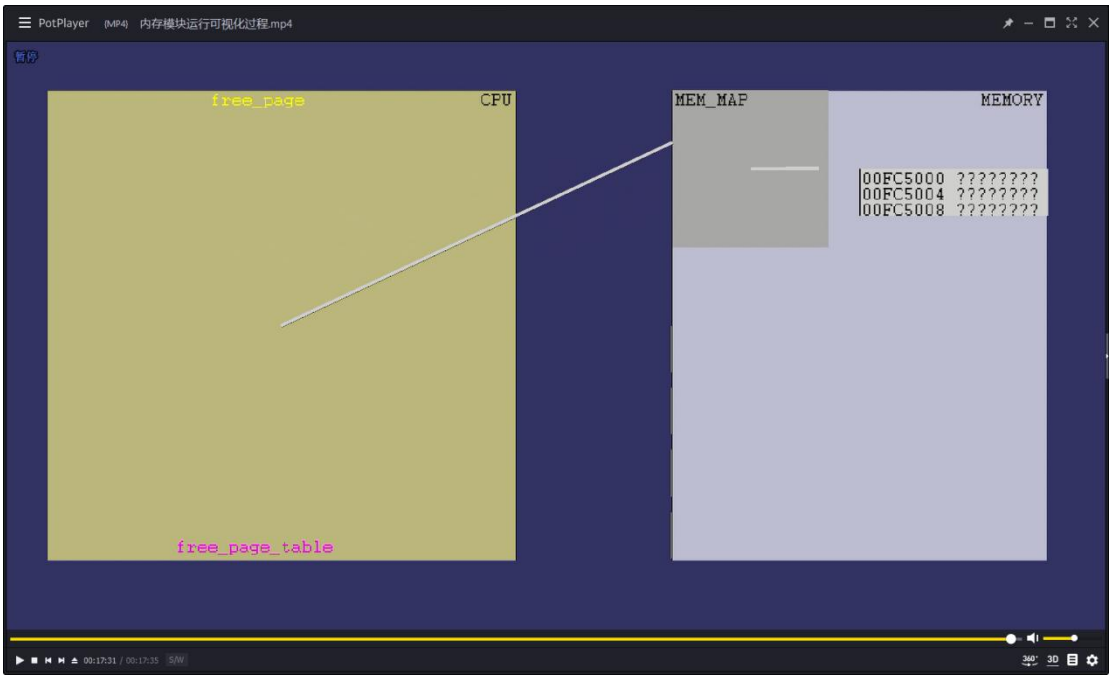
进程结束后会释放占用的所有内存，即调用 free_page_tables 函数：



然后就调用了 free_page 函数释放每一页：


PotPlayer (MP4) 内存模块运行可视化过程.mp4
暂停
free_page
CPU
don't need the page at 16359424 !
and the page size is 4K
the first available page is at 1048576
free_page_table
MEM_MAP
MEMORY
00F9A000 ????????
00F9A004 ????????
00F9A008 ????????
00FC5000 ????????
00FC5004 ????????
00FC5008 ????????


PotPlayer (MP4) 内存模块运行可视化过程.mp4
暂停
free_page
CPU
free_page_table
MEM_MAP
MEMORY
00F9A000 ????????
00F9A004 ????????
00F9A008 ????????
00FC5000 ????????
00FC5004 ????????
00FC5008 ????????



当然，总是先释放页面，最后释放页表，至此，进程彻底被销毁。

五、实验总结

一个简单进程的一生，可以如下概括：

孕育期

在母体中天翻地覆 --do_wp_page
而母体千辛万苦 --free_page_tables

终于顺利出生　--put_page

成长期

成长过程中屡屡碰壁　--do_no_page
最后终于如愿以偿　--write_verify

衰亡期

暮年岁月静好，驾鹤西去　--free_page_tables

六、实验体会

本次实验中，最大的收获有二：

其一是关于内核知识的大量实现级别的了解。比如说是80386CPU的一些硬件构造与约定，还有其汇编语言的写法等等。

其二是关于面向过程编程的了解。虽然面向对象显然是一种更加高级的编程范式，而且能够完全兼容面向过程，但是这也导致对于编写纯面向过程代码的忽视。自学习编程以来，面向对象似乎就是主旋律，这就导致了只会C++而不会纯C语言的情况发生。例如：对于大量的全局函数，全局变量的使用，甚至是头文件和实现文件分离的写法都感觉十分陌生。

而本次实验也有不足：

首先是比重分配的问题。真正说得上是操作系统核心内容，占实验的部分略显不足，而基础的硬件知识和汇编语言等内容的占比过高。不过这一点利弊参半，也使得整个实验的综合性更强。

其次是可视化的工作略显繁琐。由于现代图形引擎，基于面向对象的占绝大多数，而用来可视化面向过程的系统运行过程，显得格格不入，而且搭建可视化的框架，耗费的代码量也是巨大的，而且最后的效果也并不理想。并且，生成的视频虽然长度足够，但是其中的大

部分都是重复性的动作。

如果使用编写可视化程序说是面向过程的可视化会造成大量的重复和效果不佳，那么不妨使用动画软件手工制作的可视化，这样可以更加精确地可视化，也会使视频更加短小精悍，效果更好。所以，在操作系统课程设计中，我认为没必要拘泥于用编程的方法进行可视化。

七、附录

可视化工程的代码，可以 gitee.com/zyfbis/LinuxVisualization 中找到，编译运行则需要 VS2015 和 OSG 3.4.1 的依赖库。

而为可视化提取的系统运行日志则在上面 git 仓库中的 log 文件夹下找到，为 functions.txt，而同目录下还额外包括了提取日志的 gdb 脚本和提取的系统运行的其他信息。

可视化视频可以在 pan.baidu.com/s/1cID_yabdLi942aGJYVMVJQ 找到。