操作系统 Lab 3

151242041, 王昊庭, hatsuyukiw@gmail.com

2017 年 5 月 2 日

1 实验内容

本次实验中主要完成了进程的调度. 具体地说, 在内核的层面, 这个作者实现了队列式的时间片轮转的调度方法. 对于用户程序, 操作系统提供 `fork()`, `exit()`, `sleep()`, 和 `getpid()` 这四个系统调用的封装.

函数 `fork()` 的行为是从父进程克隆出一个新的进程, 在父进程中返回值为新进程的 PID, 在新进程中返回值为 0. 其它函数的行为是平凡的.

这个作者实现了这样的用户态程序: 用户态的第一个进程为 `empty`. 进程 `empty` 通过 `fork()` 运行游戏, `empty` 主进程随后进入闲置状态. 由于 `empty` 永远不会被销毁, 当没有进程运行时, 系统将调度至运行 `empty` 的状态.

这个作者实现的是最简单的调度方法 (时间片轮转), 再加上这个作者故意将调度频率调整地很低 (10Hz), 实际上游戏的卡顿极度严重, 原因是 `empty` 进程也是一个和游戏进程地位平等的进程, 其会被分配同等的运行时间. 这样这个作者就能证明已经正确实现进程调度了.

关于 `fork`, 这个作者提供了一个小的单元测试方便助教先生检验其实现正确性. 在 `game/common.h` 中注释掉 Macro `GAME`, 打开 Macro `TEST_FORK`. 即可运行实验讲义中提到的关于 `fork` 的趣味题. 该题的答案是 6. 这个测试程序的结果输出至串口.

2 实验中遇到的问题

本次实验中出现过的 Bug 是这个作者忘记了在调度时, 需要更改 `esp` 至新的当前进程的内核栈栈底指针. 从一个进程调度至另一个进程时, 需要更改的有如下四个信息: 内核中维护的进程数据结构, 内核栈栈底指针, Task Gate 中的 `esp` 的值, `CR3` 寄存器的值. 缺一不可.

函数 `fork` 的实现有一个比较麻烦的地方. 在对内存内容进行深拷贝时, 当前的页表页目录 (`CR3` 寄存器) 是属于父进程的, 源地址是父进程的虚拟地址, 目标地址是子进程的虚拟地址, 所以无法直接引用目标地址, 这个作者使用了一个很丑陋的解决方案.