实验8：从用户态到内核态-2023.05.26

实验概述

在本次实验中，我们首先会简单讨论保护模式下的特权级的相关内容。特权级保护是保护模式的特点之一，通过特权级保护，我们区分了内核态和用户态，从而限制用户态的代码对特权指令的使用或对资源的访问等。但是，用户态的代码有时不得不使用一些特权指令，如输入/输出等。因此，我们介绍了系统调用的概念和通过中断实现系统调用的方法。通过系统调用，我们可以实现从用户态到内核态转移，然后在内核态下执行特权指令，执行完成后返回到用户态。在实现了系统调用后，我们通过三个步骤创建进程。在这里，我们需要重点理解如何通过分页机制来实现进程之间的虚拟地址空间的隔离。最后，本次实验将介绍 `fork/wait/exit` 的一种简洁的实现思路。

Assignment 1 实现系统调用的方法

复现指导书中**“系统调用的实现”一**节，并回答以下问题。

1. 请解释为什么需要使用寄存器来传递系统调用的参数，以及我们是如何在执行 `int 0x80` 前在栈中找到参数并放入寄存器的。
2. 请使用gdb来分析在我们调用了 `int 0x80` 后，系统的栈发生了怎样的变化？ `esp` 的值和在 `setup.cpp` 中定义的变量 `tss` 有什么关系？此外还有哪些段寄存器发生了变化？变化后的内容是什么？
3. 请使用gdb来分析在进入 `asm_system_call_handler` 的那一刻，栈顶的地址是什么？栈中存放的内容是什么？为什么存放的是这些内容？
4. 请结合代码分析 `asm_system_call_handler` 是如何找到中断向量号 `index` 对应的函数的。
5. 请使用gdb来分析在 `asm_system_call_handler` 中执行 `iret` 后，哪些段寄存器发生了变

化？变化后的内容是什么？这些内容来自于什么地方？

Assignment 2 进程的创建和调度

复现**“进程的实现”**、**“进程的调度”**、**“第一个进程”**三节，并回答以下问题。

1. 请结合代码分析我们是如何在线程的基础上创建进程的PCB的（即分析进程创建的三个步骤）。
2. 在进程的PCB第一次被调度执行时，进程实际上并不是跳转到进程的第一条指令处，而是跳转到 `load_process` 。请结合代码逻辑和gdb来分析为什么 `asm_switch_thread` 在执行 `iret` 后会跳转到 `load_process` 。
3. 在跳转到 `load_process` 后，我们巧妙地设置了 `ProcessStartStack` 的内容，然后在 `asm_start_process` 中跳转到进程第一条指令处执行。请结合代码逻辑和gdb来分析我们是如何设置 `ProcessStartStack` 的内容，从而使得我们能够在 `asm_start_process` 中实现内核态到用户态的转移，即从特权级0转移到特权级3下，并使用 `iret` 指令成功启动进程的。
4. 结合代码，分析在创建进程后，我们对 `ProgramManager::schedule` 作了哪些修改？这样做的目的是什么？
5. 在进程的创建过程中，我们存在如下语句：

C++ | 复制代码

```cpp
int ProgramManager::executeProcess(const char *filename, int priority)
{
    ...

    // 找到刚刚创建的PCB
    PCB *process = ListItem2PCB(allPrograms.back(),  tagInAllList);

    ...

}
```

正如教程中所提到，“……但是，这样做是存在风险的，我们应该通过pid来找到刚刚创建的PCB。……”。现在，同学们需要编写一个 `ProgramManager` 的成员函数 `findProgramByPid` ：

C++ | 复制代码

```cpp
PCB *findProgramByPid(int pid);
```

并用上面这个函数替换指导书中提到的**"存在风险的语句"**，替换结果如下：

```cpp
int ProgramManager::executeProcess(const char *filename, int priority)
{
...

    // 找到刚刚创建的PCB
    PCB *process = findProgramByPid(pid);

...

}
```

自行测试通过后，说一说你的实现思路，并保存结果截图。

Assignment 3 fork的实现

复现“fork”一小节的内容，并回答以下问题：

1. 请根据代码逻辑概括 `fork` 的实现的基本思路，并简要分析我们是如何解决**"四个关键问题"**的。
2. 请根据gdb来分析子进程第一次被调度执行时，即在 `asm_switch_thread` 切换到子进程的栈中时， `esp` 的地址是什么？栈中保存的内容是什么？
3. 从子进程第一次被调度执行时开始，逐步跟踪子进程的执行流程一直到子进程从 `fork` 返回，根据gdb来分析子进程的跳转地址、数据寄存器和段寄存器的变化。同时，比较上述过程和父进程执行完 `ProgramManager::fork` 后的返回过程的异同。
4. 请根据代码逻辑和gdb来解释子进程的 `fork` 返回值为什么是0，而父进程的 `fork` 返回值是子进程的pid。
5. 请解释在 `ProgramManager::schedule` 中，我们是如何从一个进程的虚拟地址空间切换到另外一个进程的虚拟地址空间的。

Assignment 4 wait & exit 的实现

参考指导书中“wait”和“exit”两节的内容，实现 `wait` 函数和 `exit` 函数，回答以下问题：

1. 请结合代码逻辑和具体的实例来分析 `exit` 的执行过程。
2. 请解释进程退出后能够隐式调用 `exit` 的原因。（tips：从栈的角度分析）
3. 请结合代码逻辑和具体的实例来分析 `wait` 的执行过程。

4. 如果一个父进程先于子进程退出，那么子进程在退出之前会被称为孤儿进程。子进程在退出后，从状态被标记为 `DEAD` 开始到被回收，子进程会被称为僵尸进程。请对代码做出修改，实现回收僵尸进程的有效方法。

提交要求

1. **截止日期：**2023年6月17日 00: 00
2. **提交邮箱：**sysu_os2023@163.com
3. **邮件主题、压缩包、报告文件命名：**实验8-学号-姓名
4. **内容要求：**请大家根据上述要求，完成Assignment 1~Assignment 4，按规范撰写实验报告，并提交.asm汇编源码/C/C++源码。实验结果截图放到实验报告中，截图中尽量包含个人信息要素。
5. **实验8指导书网址：**https://gitee.com/nelsoncheung/sysu-2023-spring-operating-system/tree/main/lab8