

**操作系统原理实验报告**

|  |  |
| --- | --- |
| 姓 名： | 刘东旭 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计交1901 |
| 学 号： | U201911080 |
| 指导教师： | 郑然 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2022年 1月 8日

目 录

[1 基础实验lab1 1](#_Toc92664460)

[1.1 实验目的与设计 1](#_Toc92664461)

[1.2 实验调试及心得 2](#_Toc92664462)

[2 基础实验lab2 3](#_Toc92664463)

[2.1 实验目的与设计 3](#_Toc92664464)

[2.2 实验调试及心得 4](#_Toc92664465)

[3 基础实验lab3 5](#_Toc92664466)

[3.1 实验目的与设计 5](#_Toc92664467)

[3.2 实验调试及心得 6](#_Toc92664468)

[4 实验总结 7](#_Toc92664469)

# 基础实验lab1

## 实验目的与设计

1.1.1 lab1\_1 系统调用

我们需要补充改写handle\_syscall函数，完成系统调用功能。

首先我们发现user/app\_helloworld.c中的两个函数printu和exit其内部实现调用了do\_user\_call，用过ecall中断完成一些列操作，而kernel/strap\_vector.S中完成中断向量的操作，并在之后跳转到smode\_trap\_handler函数，在该函数中若能正确执行，则执行handle\_syscall函数，所以我们找到了需要更改的handle\_syscall函数。在该函数中，我们需要完成系统调用，首先需要将当前地址加4，也就是跳转到下一个指令位置，因为进入中断，在之后会返回到原位置，而我们的需要在中断返回后执行下一个指令，因此先加4，之后可以调用已给出的do\_syscall函数，该函数实际上是根据a0的类型来选择调用的函数，a1到a7是函数的参数，观察do\_syscall函数中的返回值，在正确执行时是返回0的，再看do\_user\_call中的汇编代码，是将a0中的值提取出来进行比较判断，可以得出我们需要在handle\_syscall中将do\_syscall中返回值保存在a0中，以进行后续判断。

1.1.2 lab1\_2 异常处理

我们需要补充改写handle\_mtrap函数，完成对异常的处理。

对于异常的处理我们首先在S态中查看，kernel/machine/minit.c中我们可以看到m\_start()函数执行的delegate\_traps()，但delegate\_traps()中将没有处理该异常，而是交给了M态，之后查找kernel/machine/mtrap\_vector.S中找到了目标函数handle\_mtrap()，在kernel/machine/mtrap.c中，替换我们的目标异常CAUSE\_ILLEGAL\_INSTRUCTION，将其中的panic模拟替换成真正的进程销毁handle\_illegal\_instruction，当然当前关卡的函数也是调用panic进行的模拟，实际中需要进行改写。

1.1.3 lab1\_3 （外部）中断

我们需要补充改写handle\_mtimer\_trap函数，完成对时钟中断的处理。

同1.1.2，我们在M态mtrap.c中判断并调用了handle\_timer函数，但该函数将相关中断处理又交给了S态，在kernel/strap.c中的 smode\_trap\_handler我们找到了需要修改的目标函数handle\_mtimer\_trap，在该函数中我们需要将时钟中断的计数器g\_ticks加1，表示新增一次，同时由于之前M态中将S态的中断标记SIP\_SSIP置为了1，现在处理完中断后，我们需要结束中断，调用write\_csr函数将sip位置0即可。

## 实验调试及心得

1.2.1 lab1\_1 系统调用

将handle\_syscall中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

我觉得本关的一个难点是各种函数的相互调用，同时trapframe的结构中隐含了各类寄存器，如果我们不清楚相关结构的话，很可能会觉得没有可使用的变量而无从下手。这里我推荐的一种方法就是将代码下载到本地，在vscode上进行查看修改，即使不提供运行环境，vscode中同样可以显示函数和结构的内部关系，相比直接在educoder上直接看更直观，更快捷，也能使用git操作，运行时，在将代码复制到educoder上进行调试，同样非常方便，且节省安装环境的空间。关于问题，我们的PKE操作系统内核是如何得到应用程序中“hello world!”字符串的地址的呢？我们通关观察syscall.c/sys\_user\_print可以看到参数输入的是虚拟地址，在将其转换为物理地址，即可调用sprint，进行显示。

1.2.2 lab1\_2 异常处理

将handle\_mtrap中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

我觉得本关非常清楚，和前一关一样都是了解pke内部的函数使用情况，明白异常的处理次序及相关函数的使用，实际需要完成的工作量很小。

1.2.3 lab1\_3（外部）中断

将handle\_mtimer\_trap中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

我觉得本关和前两个的大体思路相同，都是理解pke的运行逻辑和思路，代码量不大，另一个需要注意的点是使用write\_csr将sip置0，提示中都是使用大写，但实际是小写，可以参考mtrap.c中的handle\_timer里的写法。关于思考，请读者做完lab1\_3的实验后思考为什么死循环并不会导致死机，因为一旦超时后进行时钟中断，由于开始时该中断并不完善，在第一次中断后就会崩溃，继而直接结束。

# 基础实验lab2

## 实验目的与设计

2.1.1 lab2\_1 虚实地址转换

我们需要补充改写user\_va\_to\_pa函数，完成虚实地址转换。

找到函数所在位置vmm.c，首先我们需要将已知的逻辑地址找到其对应的页表项，这里我们使用page\_walk函数来寻找，找到后进行相关合法性的判断，如果为0，说明不存在，我们返回0，否则进行后续操作。之后为了转换为物理地址，我们将页号转换成对应的物理块号，调用PTE2PA，而偏移量实际上与虚拟地址的偏移量相当，即只取逻辑地址的低12位作为偏移量，与高位的块号相加，即为所求的实地址。

2.1.2 lab2\_2 简单内存分配和回收

我们需要补充改写user\_vm\_unmap函数，完成简单内存分配和回收。

通过观察内存分配情况，在syscall.c中找到sys\_user\_allocate\_page函数，该函数中先为物理地址分配内存，找到一个可用的逻辑地址，通过调用user\_vm\_map建立其映射关系，在user\_vm\_map函数实际调用map\_pages，map\_pages中我们可以看到建立映射时是需要或上PTE\_V有效位表示地址有效性的。

回收中，找到函数所在位置vmm.c，我们回收前首先要确定具体的物理地址，由于逻辑地址已给出，用如2.1.1的方法找到对应的物理地址，之后通过调用free\_page来释放对应的物理地址的内存，同时我们也要取消之前虚地址到实地址的映射，所以我们将该pte异或PTE\_V来取消映射。

2.1.3 lab2\_3 缺页异常

我们需要补充改写handle\_user\_page\_fault函数，完成缺页异常处理。

首先我们在S态处理中断的strap.c中找到目标函数，之后针对缺页异常，我们的处理方式是建立当前虚地址与新建物理地址的映射，这里我们使用map\_pages来建立新的映射，相关的访问权限写法可以仿照内存分配的格式来，关于size的大小，我们的目的只是对当前的缺页进行映射，也就是说只用映射1页，查看map\_pages的内部，若只用映射一页，即满足first==last，填1即可，循环只进行1次。

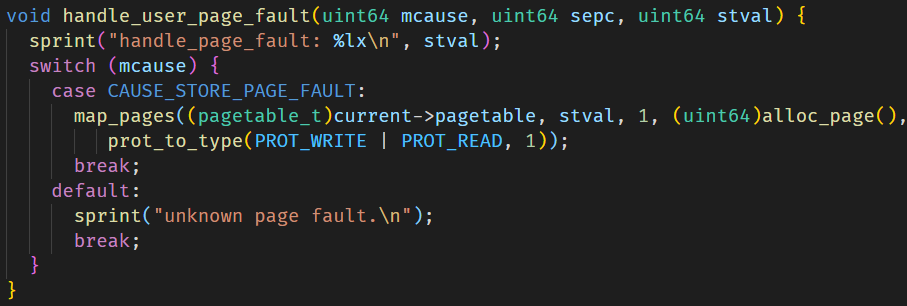


图2.1 handle\_user\_page\_fault具体实现方法

## 实验调试及心得

2.2.1 lab2\_1 虚实地址转换

将user\_va\_to\_pa中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

本关的重点在于对于逻辑地址、物理地址各位的理解，明白页号和块号的转换关系，明白两种地址的偏移量实际上是相同的，之后的转换就是调用相对应的函数，比较易理解。

2.2.2 lab2\_2 简单内存分配和回收

将user\_vm\_unmap中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

本关虽然只用完成回收的代码操作，但是我们需要先阅读内存分配操作，学习了解处理步骤和细节，如PTE\_V有效位的使用就是从分配中学来的，之后才能顺利完成回收内存函数的修改，代码的另一个重点是找到物理地址，这个与上一关实际上是一致的。

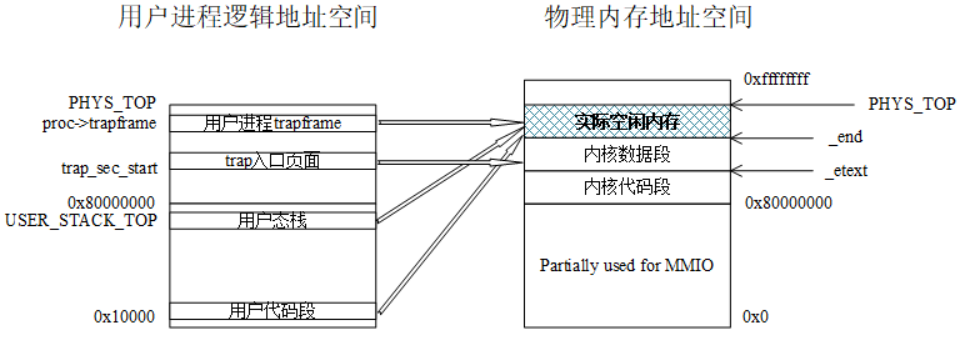


图2.2 用户进程的逻辑地址空间和到物理地址空间的映射

问题：这里的USER\_FREE\_ADDRESS\_START对应图2.2中的用户进程的逻辑地址空间的哪个部分呢？这一点请读者自行判断，并分析为什么是4MB，以及能不能用其他的逻辑地址？对应的是用户代码段结束后往上的一段地址，介于用户态栈和用户代码段之间。因为用户态栈是在高地址往下分配，所以我们的USER\_FREE\_ADDRESS应该尽量与用户态栈错开，从用户代码段结束后开始，所能分配的地址更多，分配的4MB其实可以再大一点，但刚好从4MB开始应当是最好的。

2.2.3 lab2\_3 缺页异常

将handle\_user\_page\_fault中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

本关如果理解了对S态异常处理的流程和内存分配后，再加上对与页表操作相关的重要函数的理解后，分配新的物理地址和建立新的地址映射这些操作其实都非常容易理解。

# 基础实验lab3

## 实验目的与设计

3.1.1 lab3\_1 进程创建（fork）

我们需要补充改写do\_fork函数，完成进程的创建。

首先从应用程序开始跟踪进程创建的函数调用，user/app\_naive\_fork.c --> user/user\_lib.c --> kernel/strap\_vector.S --> kernel/strap.c --> kernel/syscall.c找到了kernel/process.c文件中的do\_fork函数，由于在子进程中不存在由页表到块号的映射，而子进程和父进程的虚拟地址是一致的，同时应当映射到同一个物理地址，所以在该函数中我们首先要使用map\_pages新建立一个在子进程中的映射关系，而这个映射关系中的虚拟地址和物理地址都是父进程的相应地址，关于size大小，由于只涉及单页的映射，同2.1.3中的思想，我们直接令size为1即可完成1次映射，同时注意该段为代码段，所以我们建立的映射权限是可读和可执行的，通过上述思想，我们可以成功在子进程中创建。

3.1.2 lab3\_2 进程yield

我们需要完善sys\_user\_yield系统调用，实现进程执行过程中的主动释放CPU的动作。

user/app\_yield.c-->kernel/syscall.c找到sys\_user\_yield()函数，进程释放CPU的动作应该首先将当前进程状态置为READY，通过调用insert\_to\_ready\_queue函数将当前进程放入就绪队列队尾，之后通过schedule()转进程调度。

3.1.3 lab3\_3 循环轮转调度

我们需要补充改写rrsched函数，完成进程的循环轮转调度。

每次增加时间计数，当超过时间片的长度时表示当前进程时间片到，将时间片置0，同时同3.1.2的操作，需要放回就绪队列的队尾，之后在就绪队列中转进程调度，即可完成循环轮转调度。

## 实验调试及心得

3.2.1 lab3\_1 进程创建（fork）

将do\_fork中的代码按设计思路完成后，输入到educoder上进行调试，发现运行失败，原因是显示缺少了1行代码。

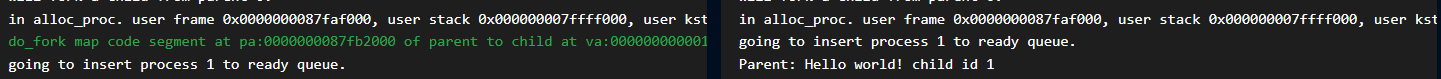


图3.1 运行错误

在浏览了所有代码后，发现没有任何地方有输出该信息的位置，合理猜测可能是老师之前误删了相关代码，我们可以依据正确输出，手动添加这段提示信息即可，之后再运行代码，发现成功通过。

本体中比较难理解的一个点就是在调用map\_pages函数的时候，有父子两个进程，而不像之前都是单一的当前进程，这样十分容易混淆两个进程的使用，所以我们更应该清楚相关逻辑，才能建立起正确的映射，另一个问题就是缺少一句提示信息，我认为这个地方是今后课程组应该提前修改好的。

3.2.2 lab3\_2 进程yield

将sys\_user\_yield中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

本关非常简单，重点是理解状态的转换关系，释放进程，相当于将该进程放回就绪队列，理解了这其中的操作步骤就非常简单。

3.2.3 lab3\_3 循环轮转调度

将rrsched中的代码按设计思路完成后，输入到educoder上进行调试，发现成功通过。

这关其实考察的是对分时系统的轮转调度的理解与掌握，知道时间片到后将主动释放CPU的操作后，本关也非常容易解决。

# 实验总结

我觉得新课程实验是一个非常好的尝试，不用学生装配环境，同时能在头哥上一键测评，对于学生和老师来说都是一件便利高效的方案。而在本次实验中，我们围绕系统调用、异常和外部中断，内存管理和进程管理这三个在操作系统中非常重要的知识点展开实验，这些实验不仅仅是学生自己写埋头代码完成，更多的知识点是整个框架的实际，都是模板已经提供好的，学生在阅读这些模板代码的时候同样可以学习到许多知识要点，我觉得这相比于其它实验缺少提示纯靠学生自学完成要友好的多。提示文档相当详细丰富，教会了同学详细的阅读代码，理解架构的方法，我觉得也是本实验非常优秀的地方。

当然，在这里我也看到了一些在自己做实验时，认为实验存在的问题，比如git操作，虽然我在vscode上可以很方便的使用git操作进行记录、恢复，但是在educoder平台上由于其每关独立，需要人为的进行复制粘贴，非常麻烦，而且越到后来，随着完成的任务量增多，越发麻烦，希望改进。同时educoder上写代码也不是很方便，不能提供相关提示和跳转，如果只在educoder平台上完成代码，还是比较费时的，这一点是希望educoder平台能提供更好的提示功能。

再谈及实验本身，我觉得一个是知识点的涵盖问题，像文件管理这种占比很大的知识点在实验中没有涉及还是有些缺憾。同时这些基础题基本上都是属于代码填空题，可发挥的空间很少，我建议是留出更多位置交给同学来完成代码功能的实现同时增加相对应的更为详细的提示予以告知。有些题目非常简单如lab3\_2和 lab3\_3，我觉得进程这块难度可以稍加提升。

总的来说，本次实验带给我的是一次全新的体验，一个实验重在对已有框架代码的阅读与理解也是一种非常有趣的学习过程，而对riskv框架的学习，也是我们接触掌握前沿知识的一种有效手段，我们的课程能够与时俱进的进行改革，我们学生也才能更快的与时代接轨。