華中科技大學

操作系统原理课程实验报告

姓　　名：　　徐瑞达

学　　院：　　计算机科学与技术学院

专　　业：　　计算机科学与技术

班　　级：　　CS2008

学　　号：　　U202015533

指导教师：　　胡贯荣

分数	
教师签名	

2022 年 12 月 26 日

目　录

实验一 复杂缺页异常

1.1 实验目的

该实验通过处理更为复杂的复杂缺页异常，以加深对缺页异常的理解，更好地掌握如何处理缺页异常。

1.2 实验内容

1.2.1 问题分析

问题分为以下两部分：

1. 当发生用户栈缺页异常，重新分配一个物理页，并将其映射到对应的虚拟地址上。
2. 当处理其他类型缺页异常，如对动态申请的数组进行越界访问时，应当警告用户非法访问地址，报错退出。

可以通过判断缺页地址是否在栈内来判断缺页类型。

1.2.2 解决过程

1. 解决方案

从文档的提示入手——监控进程的虚拟地址空间。在进程的数据结构中定义变量 alloc_page_count，表示已为栈分配的物理页数。假设栈最大为 50 个 PGSIZE，那么当 stval>USER_STACK_TOP-50*PGSIZE 时说明为用户栈缺页异常，否则判定为数组越界访问引起的缺页异常。

2. 其他思路

观察代码发现，数组 ans 通过 naive_malloc 函数向堆申请地址空间，而递归时产生的是递归调用栈。堆从低地址(0x00000000 + PGSIZE * 1024)处开始往高地址增长，而栈则从高地址(栈顶 0x7ffff000)处开始往低处增长。当发生数据越界访问时，发生缺页异常的地址 stval 应小于当前的栈帧寄存器 sp，因此当满足 stval<sp 时即可报错退出。使用这种方法最终也得到正确答案。

1.3 实验调试及心得

在开始写该挑战实验时，因为没有彻底搞懂缺页异常，写起来云里雾里。查

阅资料后知道，缺页异常一般包括两种情况，一是程序设计不当导致访问了非法的地址（如本挑战中的数组越界），二是访问的地址是合法的，但还未为该地址分配物理页（如用户栈缺页）。在处理复杂缺页异常时，首先判断是哪种缺页异常然后再进行相应的处理，因此只需在 lab2_3 的基础上添加判断即可。

向同学请教时，我还解决了另一个问题。在 lab2_3 中，我的解决代码如下：

```
//MY lab2_3
user_vm_map((pagetable_t)current->pagetable, stval, 1,
(uint64)alloc_page(), prot_to_type(PROT_WRITE | PROT_READ, 1));
```

开始时，我赋给函数的第三个参数为 PGSIZE，也即一个页的大小，却总是发生映射错误，后来误打误撞改为 1 竟得到正确结果，令我百思不得其解。在写挑战实验请教同学时，才知道其中缘由。其另一种解决代码是函数第 2、3 个参数为 RUNDOWN(stval,PAGESIZE)和 PAGESIZE，即按照 PGSIZE 进行了对齐操作。追溯至函数 map_pages 时发现，该函数在进行映射时，也会将地址按照 PGSIZE 对齐。当把参数写成 stval 和 1 时，在实验中并没有影响对齐后的结果；而当把参数写成 stval 和 PGSIZE 时，将会使对齐结果错误，从而导致映射错误。

从这个实验我知道，在进行实践之前要先把理论基础扎牢，同时在完成实验时，要敢于自己思考，不局限于他人的指导和提示。在做实验时，要耐下心来阅读代码，推敲代码中的逻辑，独立解决读代码或者解决问题时遇到的不懂之处。

实验二 进程等待和数据段复制

1.1 实验目的

1. 通过完成数据段复制，更好地掌握 fork 的工作原理和过程。
2. 通过完成进程等待，更好地掌握进程调度。

1.2 实验内容

1.2.1 问题分析

1. 由于存在全局变量 flag，如果不复制数据段，父子进程将共用该变量，因此需要在 fork 操作中实现数据段的复制以使进程间数据段相互独立，避免某个进程修改变量对其他进程造成影响。
2. 增加系统调用 SYS_user_wait，实现进程等待的相关功能。

1.2.2 解决过程

1. 数据段复制

查阅代码得知，进程的相关数据以 mapped_region 的数据结构进行存储。每个 mapped_region 包含 npages 个页，类型为 seg_type，对应虚拟地址为 va。复制数据段时，需要逐页复制数据并进行地址映射。

在 process.c 文件中的 do_fork 函数内，添加 DATA_SEGMENT 的 case 子句。在该子句内，为子进程分配新的物理页，将父进程页的数据复制到该新页内，并使用 map_pages 函数进行地址映射，以此循环直至 npages 个页均复制完成。

2. 进程等待

首先在 syscall 文件中定义宏 SYS_user_wait 和函数 sys_user_wait，并在函数 do_syscall 中添加对 SYS_user_wait 的支持。函数 sys_user_wait 通过调用 do_wait 函数实现。

在函数 do_wait 中，如果 pid 为-1，说明等待任意一个子进程结束后返回其 pid，此时遍历所有进程，如果其为当前进程的子进程，若进程已结束则返回其 pid，否则在当前进程的 blockmap 中记录下该子进程。如果当前进程有子进程且均未结束，则返回-2 表示应阻塞当前进程等待子进程之一结束。

如果 pid 大于 0，说明等待进程 id 为 pid 的子进程结束后返回 pid，此时与上述情况中的单次循环类似。其余情况均返回-1。

而在 sys_user_wait 函数中，当 wait 的返回值为-2，则将当前进程标记为 BLOCKED 并转进程调度。同时，当某个进程结束调用 free_process 时，需要根据其父进程的 blockmap 判断父进程是否因其阻塞，如果是则将父进程标记为 FREE 并插入就绪队列中。

1.3 实验调试及心得

在写该挑战实验时，不像挑战二一样一头雾水，却也遇到了许多挫折。在进行数据段复制时，我先尝试的是简单地进行创建新页、复制页、映射，却没有考虑到数据段可能有多个页，导致程序运行时抛出映射错误。最后需要通过循环复制每个页，直到 npages 个数据段页复制完成。

在写进程等待功能时，由于需要在进程的数据结构中记录引起阻塞的子进程列表，我采用了整型数组的方式实现，这种方式满足了功能的实现。而后查阅了助教的解题指导，发现可以使用一个整型变量 blockmap 实现，其第 n 位为 1 表示需要等待进程 id 为 n 的子进程结束，可以使用位运算达成需求，大大减小了进程数据所需的空间大小。

从这次实验我知道，硬件资源是十分宝贵的，操作系统在完成调度工作的同时也要注意自身的资源消耗，以进一步提升系统性能。