

**操作系统原理课程实验报告**

|  |  |
| --- | --- |
| 姓 名： | 汪宇飞 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS2003班 |
| 学 号： | U202015375 |
| 指导教师： | 谢美意 |

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

2022年12月28日

目 录

[实验一 lab2\_challenge1：复杂缺页异常 1](#_Toc123244734)

[1.1 实验目的 1](#_Toc123244735)

[1.2 实验内容 1](#_Toc123244736)

[1.3 实验调试及心得 2](#_Toc123244737)

[实验二 lab3\_challenge1：进程等待和数据段复制 3](#_Toc123244738)

[2.1 实验目的 3](#_Toc123244741)

[2.2 实验内容 3](#_Toc123244742)

[2.3 实验调试及心得 6](#_Toc123244743)

# 实验一 lab2\_challenge1：复杂缺页异常

## 实验目的

给定应用app\_sum\_sequence.c，对给定n递归计算从0到n的和，并将每一次递归的结果保存至数组ans中。

由于采用的是递归算法，函数调用会被保存至栈中，每一次函数调用都会将上一次函数调用现场压栈，从而导致栈越来越深也即使用的内存空间越来越大。同时由于实验将n设置为1024，导致栈一定会被压爆，所分配的内存空间不存从而出错，因此需要对这一情况进行处理。在判断属于缺页异常且是用户栈缺页时需要分配一个新的物理页从而进行扩充。

另一方面，由于函数设置的原因，最后一次计算会访问数组越界地址，这里的越界地址的虚拟地址无对应映射的物理地址，因此是非法访问，需要输出提示并退出程序的运行。

## 实验内容

由上述实验目的的讲述与分析可得出实验内容分为两个部分：一是对用户栈的扩充从而使递归算法顺利运行；二是判断数组的访问是否越界，若越界则提示并退出。

对于第一个内容，要求和实现均和基础实验lab2\_3相同，根据基础实验lab2\_3的实验指导可以找到对于缺页异常处理的函数，即kernel/strap.c文件中的handle\_user\_page\_fault函数，对于mcause是CAUSE\_STORE\_PAGE\_FAULT时的情况的代码进行补充。

首先需要判断引发缺页异常的虚拟地址stavl是否位于预设的用户栈中，即小于栈顶USER\_STACK\_TOP并大于栈底USER\_STACK\_TOP – PGSIZE\*20，符合则说明stavl合法，可以分配新的物理页并进行映射。

使用实验所提供的alloc\_page函数进行新的物理页的分配，并使用user\_vm\_map函数进行物理地址到虚拟地址的映射。

其次对于第二个内容，首先需要判断数组的访问是否越界，这一判断也是通过stavl进行的。由于引起这一缺页异常的虚拟地址并不位于预设的用户栈中，因此只需在第一个内容的判断之后加上响应的处理即可，根据实验给出的与其输出填入对于的panic输出。

由此即可完成这一挑战实验的内容，所得出的代码如代码1.1所示。

**代码1.1**

case CAUSE\_STORE\_PAGE\_FAULT:

if(stval<=0x7ffff000&& (stval >= 0x7ffff000 - PGSIZE \* 20)){

user\_vm\_map((pagetable\_t)current->pagetable, ROUNDDOWN(stval,PGSIZE),PGSIZE,(uint64)alloc\_page(), prot\_to\_type(PROT\_WRITE | PROT\_READ, 1));

}else{

panic("this address is not available!");

}

break;

## 实验调试及心得

这一挑战实验是六个挑战实验中所标注的最简单的一个，因此我选择这一实验作为第一个做的实验。虽然这一实验原理上并不复杂，所需要实现的代码量仅为不到十行并不多，但它很好地带我了解体会到了挑战实验的面貌，即没有基础实验那样详尽明确的实验指导，需要按照要求自行修改完善代码。通过这一次简单的实验，我对于挑战实验有了初步的认识和准备，也有了相应完成挑战实验的经验。

由于我是使用educoder进行的实验，且由于基础实验的实验指导十分详细明确，我做基础实验时只需按照指导填入代码后测试通关即可，尚未使用过命令行输出。而在这一挑战实验中第一次使用了educoder所提供的命令行进行测试观察，对实验的完成很有帮助。educoder提供了很方便的实验平台和环境，省去了我们安装虚拟机并自行配置环境的精力，能够更好地完成实验。

# 实验二 lab3\_challenge1：进程等待和数据段复制



## 实验目的

给定应用app\_wait.c，该应用存在父进程以及fork得到的子进程和孙子进程，flag分别为0、1、2，应用使用wait函数进行进程等待控制，使得按照孙子进程、子进程、父进程的顺序依次退出并输出对应信息。

按照实验要求，需要修改PKE内核和系统调用从而提供wait函数功能，该函数根据输入的参数pid进行不同的对应操作。

此外，为实现fork之后父子进程数据段相互独立，需要在基础实验lab3\_1的基础上进一步修改do\_fork函数，实现数据段的复制。

## 实验内容

首先对于do\_fork函数的修改，数据段的复制与lab3\_1中对于代码段复制的实现相类似，但在进行映射时需要映射到新的物理页帧，也即需要分配新的物理页。在do\_fork函数中的switch语句新增case DATA\_SEGMENT，使用实验所提供的alloc\_page函数进行新的物理页的分配，并使用user\_vm\_map函数进行物理地址到虚拟地址的映射，如代码2.1所示。

**代码2.1 数据段的复制**

case DATA\_SEGMENT:{

user\_vm\_map(child->pagetable,parent->mapped\_info[i].va,PGSIZE, (uint64)alloc\_page(),prot\_to\_type(PROT\_READ | PROT\_WRITE, 1));

// after mapping, register the vm region (do not sdelete codes below!)

child->mapped\_info[child->total\_mapped\_region].va = parent->mapped\_info[i].va;

child->mapped\_info[child->total\_mapped\_region].npages++;

child->mapped\_info[child->total\_mapped\_region].seg\_type = DATA\_SEGMENT;

child->total\_mapped\_region++;

break;

}

完成了do\_fork函数中数据段的复制之后，需要实现wait函数，首先需要在user\_lib.h文件中添加wait函数原型，并在user\_lib.c文件中添加wait函数实现。wait函数接收int pid作为参数，返回一个int值，其具体实现可以参照user\_lib.c文件中其他函数的实现，调用并返回do\_user\_call函数的返回值即可，对于参数值a0传入SYS\_user\_wait，a1传入pid，剩下六个参数均设置为0。如代码2.2所示。

**代码2.2 user\_lib部分的修改**

//以下为位于user\_lib.h文件中的wait函数原型

int wait(int pid);

//以下为位于user\_lib.c文件中的wait函数实现

//

// lib call to wait

//

int wait(int pid){

return do\_user\_call(SYS\_user\_wait, pid, 0, 0, 0, 0, 0, 0);

}

在声明和定义wait函数之后，需要对于syscall.h和syscall.c进行修改使得do\_user\_call函数可以对wait函数相关操作进行相应与实现。首先需要在syscall.h文件中对于SYS\_user\_wait进行定义，使得do\_user\_call函数可以正确识别并处理。之后需要在syscall.c文件中对do\_syscall函数进行修改，在switch语句中添加case SYS\_user\_wait，并增加sys\_user\_wait函数进行处理，该函数接收pid作为参数，调用内核函数do\_wait并返回该内核函数的返回值。如代码2.3所示。

**代码2.3 syscall部分的修改**

//以下为位于syscall.h文件中的新增定义

// added @challenge3\_1

#define SYS\_user\_wait (SYS\_user\_base + 6)

//以下为位于syscall.c文件中的函数实现与修改

ssize\_t sys\_user\_wait(long pid){

return do\_wait(pid);

}

long do\_syscall(long a0, long a1, long a2, long a3, long a4, long a5, long a6, long a7) {

switch (a0) {

case SYS\_user\_print:

return sys\_user\_print((const char\*)a1, a2);

case SYS\_user\_exit:

return sys\_user\_exit(a1);

// added @lab2\_2

case SYS\_user\_allocate\_page:

return sys\_user\_allocate\_page();

case SYS\_user\_free\_page:

return sys\_user\_free\_page(a1);

case SYS\_user\_fork:

return sys\_user\_fork();

case SYS\_user\_yield:

return sys\_user\_yield();

case SYS\_user\_wait://新增的case

return sys\_user\_wait(a1);

default:

panic("Unknown syscall %ld \n", a0);

}

}

在修改完syscall部分后，需要新增内核函数do\_wait。首先需要在process.h文件中添加do\_wait函数原型，并在syscall.c文件中添加do\_wait函数实现。该函数接收一个参数pid，并根据pid进行不同的操作，具体如代码2.4所示。

**代码2.4 process部分的修改**

//以下为位于process.h文件中的do\_wait函数原型

// challenge3\_1所需要实现的wait内核函数

int do\_wait(int pid);

//以下为位于process.c文件中的do\_wait函数实现

int do\_wait(int pid) {

if(pid==-1){

process\* child=NULL;

for(int i=0;procs[i].status!=FREE;i++){

if(current==procs[i].parent){child=&procs[i];}

}

if(child->status==ZOMBIE){return child->pid;}

current->status=BLOCKED;

schedule();

return child->pid;

}

if(pid>=0&&pid<NPROC){

//pid大于0且pid对应的进程不是当前进程的子进程，返回-1

if(current!=procs[pid].parent){return -1;}

if(procs[pid].status==ZOMBIE){return pid;}

current->status=BLOCKED;

schedule();

return pid;

}

return -1;

}

该函数功能如下：

1. 当pid为-1时，首先寻找一个当前进程的子进程，若该子进程状态为ZOMBIE，即运行结束，则返回其pid；否则将其状态修改为BLOCKED并调用schedule函数，最后返回pid。
2. 当pid大于等于0并小于NPROC时，首先需要判断pid对于的进程是否为当前进程的子进程，若不是则返回-1。同样地，若pid对应的子进程状态为ZOMBIE，则返回其pid；否则将其状态修改为BLOCKED并调用schedule函数最后返回pid。
3. 当pid不符合以上两种情况时，即为非法的pid，返回-1即可。

至此，完成了对于PKE内核和系统调用的修改，完成了对wait函数的实现，使所给定应用的app\_wait.c能够按照预期运行并输出。

## 实验调试及心得

本次挑战实验虽然有了上一个挑战实验的基础，有了心理准备，但是仍然可以感觉到较为困难复杂。本次实验需要在深刻透彻地理解基础实验1和基础实验3的数个实验的原理上进行，因此也花费了我较多时间回过头来重新阅读基础实验的指导并进行理解，诸如系统调用与内核之间的联系、syscall的实现等，都是需要从头再阅读并理解一次的。在实现期间也多次使用educoder所提供的命令行进行测试，让我再一次感谢老师为我们提供了这一方便的实验环境。在经历多次报错之后能够成功实现并通关还是十分有成就感的。

本次操作系统实验由9个基础实验和2个挑战实验组成，基础实验的讲解和指导十分详细明确，因此完成的过程中最大的工作就是理解，整体而言难度适中。对于挑战实验也是要求做6个挑战实验中最简单的2个，因此能够有从基础实验到挑战实验的较为顺利的过渡，也为下个学期的课设夯实了基础并让我们做好了准备。

这次实验是我所做过的最具特色的实验之一，以十分详尽的讲解为我们介绍了操作系统底层的组成与结构，让人不禁感叹计算机操作系统的精致巧妙以及严谨性，为我们展示了理论课之外更加细致具体的部分。十分感谢课程组的老师们对于本次实验的设计以及实验指导的撰写和实验平台和资料的部署，为我们提供了一个十分方便的实验环境和详尽的指导。