华中科技大学

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

1.1 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。

1.2 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;
```

1.3 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

2.1 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。

2.2 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。

2.3 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。