

0.

首先将LAB1-7的代码填充（包括整个default_sched要替换，priority.c的MAX_TIME要改大），然后grep查找LAB8：

```
→ lab8 git:(main) # grep -R LAB8 .
./kern/fs/sfs/sfs_inode.c: //LAB8:EXERCISE1 YOUR CODE HINT: call sfs_bmap_load_nolock, sfs_rbuf, sfs_rblock,etc. read different kind of blocks in file
./kern/process/proc.c: //LAB8:EXERCISE2 YOUR CODE HINT:need add some code to init fs in proc_struct, ...
./kern/process/proc.c://copy_files&put_files function used by do_fork in LAB8
./kern/process/proc.c: //LAB8:EXERCISE2 YOUR CODE HINT:how to copy the fs in parent's proc_struct?
./kern/process/proc.c:bad_fork_cleanup_fs: //for LAB8
./kern/process/proc.c: put_files(current); //for LAB8
./kern/process/proc.c://load_icode_read is used by load_icode in LAB8
./kern/process/proc.c: /* LAB8:EXERCISE2 YOUR CODE HINT:how to load the file with handler fd in to process's memory? how to setup argc/argv?
./kern/process/proc.c:// this function isn't very correct in LAB8
./kern/trap/trap.c: // There are user level shell in LAB8, so we need change CMD/KBD interrupt processing.
```

可以发现sfs_inode.c为excise1，proc.c为excise2，trap.c已经改好，即此时没有需要额外改的，make qemu执行可得：

```
check_swap() succeeded!
sfs: mount: 'simple file system' (262/32506/32768)
vfs: mount disk0.
++ setup timer interrupts
not valid addr c, and can not find it in vma
trapframe at 0xc0337e30
  edi 0x00000000
  esi 0x00000000
  ebp 0xc0337eac
  oesp 0xc0337e50
  ebx 0xc010d14e
  edx 0xc0334448
  ecx 0x00000016
  eax 0x0000000c
  ds 0x---0010
  es 0x---0010
  fs 0x---0000
  gs 0x---0000
  trap 0x0000000e Page Fault
  err 0x00000000
  eip 0xc0109ad5
  cs 0x---0008
  flag 0x00000006 PF,IOPL=0
kernel panic at kern/trap/trap.c:211:
  handle pgfault failed in kernel mode. ret=-3

Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K> 
```

可以看出设置好timer interrupts之后准备执行exec启动用户程序的时候程序才崩掉。

1.

注释说的很清楚，大概就是要分三段来做，第一段是开头未对齐部分，接着是中间对齐部分，最后是末尾的未对齐部分。然后可以看到sfs_buf_op和sfs_block_op分别在前面已经根据是读入还是输出设定为对应的函数了。sfs_rbuf和sfs_rblock的参数意思也很明确，需要注意的是，由于整体是采取索引的结果，一个文件的所有的block都不一定会相邻，故而在读入中间对齐部分的时候，sfs_rblock参数中的nblks不能直接设为整个总数，而应设为1，一个个的依次读入（或输出）。

2.

首先alloc_proc中初始化，由于使用的memset，故不需要特别修改。

然后在do_fork需要将父进程相关的文件信息传递给子进程，通过调用copy_files函数实现。

最后实现load_icode，这个函数要做的事比较多，参考lab7中的load_icode来进行修改实现，此处与之前lab实现上的唯一区别应该就是读取的位置不同，本lab中要通过调用load_icode_read来读文件

；
首先是ELF文件头：

struct elfhdr *elf = (struct elfhdr *)binary改为load_icode_read(fd, elf_buf, sizeof(struct elfhdr), 0)从文件开始读入文件头大小的字符;

接着是程序段头:

struct proghdr *ph = (struct proghdr *) (binary + elf->e_phoff)改为load_icode_read(fd, ph, sizeof(struct proghdr), elf->e_phoff + sizeof(struct proghdr) * i), 需要注意的是, 这里lab7中的写法相当于是把一次把所有的程序段头一次性读入在内存中了, 而我们这里为了不动态分配空间, 一次就只读入一个段头, 故而是每次循环中读取一次;

最后是复制各段内容到内存:

memcpy(page2kva(page) + off, from, size)改为load_icode_read(fd, page2kva(page) + off, size, offset)直接读入到对应位置。

此时运行会发现page fault, 仔细阅读说明可发现, 本lab中还需要为运行的程序传递参数, 设置uargc和uargv:

此处我们采取Linux的参数放置格式, 即首先将参数字符串放入栈中, 然后将指向这些字符串的指针压入栈中当做参数, 最后再压入argc;

而对于计算字符串长度的函数strlen, 第二个参数为最大长度, 我们可以将其设为0x7fffffff(后grep -R "#define .MAX" .发现在libs/unistd.h有宏定义EXEC_MAX_ARG_LEN, 可以在此处使用);

然后再者, 为了保证运行的速度, 参数最好能够4字节对齐, 故在计算参数位置的时候, 可以and一个0xffffffff来做对齐。

至此, 整个程序已经完成, make qemu运行可以得:

```
Iter 3, No.4 philosopher_sema is eating
Iter 4, No.4 philosopher_sema is thinking
Iter 4, No.4 philosopher_sema is eating
No.4 philosopher_sema quit

$ ls
@ is [directory] 2(hlinks) 23(blocks) 5888(bytes) : @'.'
[d] 2(h) 23(b) 5888(s) .
[d] 2(h) 23(b) 5888(s) ..
[-] 1(h) 10(b) 40382(s) badarg
[-] 1(h) 10(b) 40386(s) badsegment
[-] 1(h) 10(b) 40404(s) divzero
[-] 1(h) 10(b) 40406(s) exit
[-] 1(h) 10(b) 40385(s) faultread
[-] 1(h) 10(b) 40391(s) faultreadkernel
[-] 1(h) 10(b) 40410(s) forktest
[-] 1(h) 10(b) 40435(s) forktree
[-] 1(h) 10(b) 40381(s) hello
[-] 1(h) 11(b) 44640(s) ls
[-] 1(h) 11(b) 44584(s) matrix
[-] 1(h) 10(b) 40381(s) pgdir
[-] 1(h) 11(b) 44571(s) priority
[-] 1(h) 11(b) 44694(s) sh
[-] 1(h) 10(b) 40404(s) sleep
[-] 1(h) 10(b) 40385(s) sleepkill
[-] 1(h) 10(b) 40383(s) softint
[-] 1(h) 10(b) 40380(s) spin
[-] 1(h) 10(b) 40408(s) testbss
[-] 1(h) 10(b) 40516(s) waitkill
[-] 1(h) 10(b) 40381(s) yield

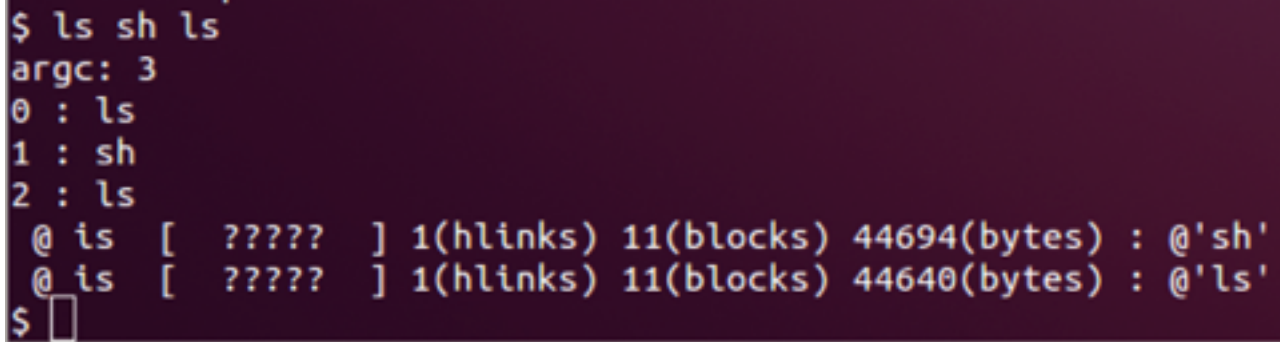
lsdir: step 4
$ hello
Hello world!!
I am process 15.
hello pass.
$
```

其中ls、hello为输入运行的用户程序。

最后，通过修改用户程序，增加：

```
printf("argc: %d\n", argc);  
int i = 0;  
for (i = 0; i < argc; ++i) {  
    printf("%d : %s\n", i, argv[i]);  
}
```

将argc和argv打印出来可发现参数正确：



```
$ ls sh ls  
argc: 3  
0 : ls  
1 : sh  
2 : ls  
@ is [ ?????? ] 1(hlinks) 11(blocks) 44694(bytes) : @'sh'  
@ is [ ?????? ] 1(hlinks) 11(blocks) 44640(bytes) : @'ls'  
$
```