Proj 1：带参数的系统调用，子进程优先的fork，命令解释器mysh

> 现场验收：2025.5.11晚上18:00-21:00，实验楼103
>
> 报告提交截止日期：验收通过后、2025.5.13晚上22:00前

本项目你需要添加一个带参数的系统调用、子进程优先的fork以及命令解释器mysh。

1. 带参数的系统调用(30%)

proj0中添加的系统调用不带任何参数，但是后续实验中需要实现大量带参数的系统调用。你需要修改proj0中的系统调用为：

```
int shutdown(int a);
```

其中a为传入的参数。在执行此系统调用时，操作系统需要在屏幕上打印一行字后退出QEMU。如果用户调用的是shutdown(20)，则打印的字符串为：

```
Leaving with code 20.
```

注意，字符串需要由内核利用cprintf打印，不允许使用printf打印。

发布的源码中已包含`shutdown`的命令以及一个粗略的内核实现，你只需要修改这个实现即可。

如果shutdown没有携带参数，则默认传入参数-1。

验收要求

2个测试用例

```
$ shutdown
Leaving with code -1.

$ shutdown 400
Leaving with code 400.
```

实验报告要求

- 参考一个带参数的系统调用，如`sleep`。在该系统调用的内核实现中，内核通过`argint`获取用户传递的参数，请在实验报告中解释argint获取第n个参数（n从0开始）位置的方法。

实验提示

> xv6用户库中有atoi函数。
>
> 需要寻找一个函数的具体实现在哪个文件中时，可以使用`grep`命令。例如，你不知道`argint`在哪个文件中，可以在命令行中输入`grep argint *`（意思是：从当前目录下的所有文件中查找字符串`argint`）。输出类似如下：

```
$ grep argint *
defs.h:int             argint(int, int*);
syscall.c:argint(int n, int *ip)
syscall.c:  if(argint(n, &i) < 0)
syscall.c:  if(argint(n, &addr) < 0)
...
```

> 经过简单的排除，会发现函数是在`syscall.c`中实现的。
>
> `myproc()->tf`中存储了用户进程在陷入内核前的状态，包括各个寄存器的值，这些值存储在内核空间中，即，`myproc()->tf`指向了内核的一块空间，这块空间的组织方式是一个`trapframe`结构体（通过`grep`查找这个结构体的定义，了解有哪些域）。
> 你可以阅读xv6-book的Chapter 1和Chapter 3中关于系统调用的部分。
> 你可以阅读这个材料了解函数调用时用户的栈发生了哪些变化。

2. 子进程优先的fork（30%）

阅读proc.c中的int fork(void)函数。fork涉及到两个进程，新的进程称为子进程，原始的进程称为父进程。xv6执行完fork后，无形中让父进程先运行，然后才是子进程运行。你需要修改这个行为，允许子进程优先运行。

实现以下系统调用，以控制是否启用你的算法。此系统调用已经分配了系统调用号，你只需要在内核中实现即可。

```
void fork_winner(int winner);
```

- 若winner=1，表示启用你的算法；若winner=0，表示不启用你的算法。
- 默认情况下不启用你的算法。
- 如果你的实现是正确的，运行forktest命令后你应当能观察到，设置child为winner时，大多数情况是先打印child后打印parent；设置parent为winner时，大多数情况是先打印parent后打印child。
- 不允许修改forktest.c文件。
- 进程在运行过程中不允许修改pid的值，例如，你不应该交换进程的父子关系。

验收要求

运行forktest，典型输出如下

```
$ forktest
Fork test
Set child as winner
Trial 0:  child!  parent!
Trial 1:  parent!  child!
Trial 2:  child!  parent!
Trial 3:  child!  parent!
Trial 4:  child!  parent!
Trial 5:  child!  parent!
Trial 6:  child!  parent!
```

```
Trial 7:  child!  parent!
Trial 8:  child!  parent!
Trial 9:  child!  parent!

Set parent as winner
Trial 0:  parent!  child!
Trial 1:  parent!  child!
Trial 2:  parent!  child!
Trial 3:  parent!  child!
Trial 4:  parent!  child!
Trial 5:  parent!  child!
Trial 6:  parent!  child!
Trial 7:  parent!  child!
Trial 8:  parent!  child!
Trial 9:  parent!  child!
```

特征：大部分情况下按照优先级输出，偶尔输出有交叉。

实验提示

> xv6使用的调度算法类似于时间片轮转，每个进程执行一个时间片（1个时钟中断间隔）后放弃CPU。在trap函数中找到相关判断语句，逻辑是：如果当前是时钟中断，则代替用户进程放弃CPU，让用户进程进入就绪态，这里调用了哪个函数放弃的CPU？是否可以用类似的方法，让父进程在系统调用返回前放弃CPU？

实验报告要求

请在实验报告中回答以下问题：

- 在父进程优先的情况下，偶尔会有子进程先于父进程打印到屏幕的情况出现；在子进程优先的情况下，偶尔也会有父进程先于子进程打印到屏幕的情况。解释可能的原因。

3. 编写命令解释器mysh（20%）

编写一个shell，接受特定的用户命令。

shell的名字为mysh，其中为学号的最后两位，比如，041221135同学的shell的名字为mysh35，以下以mysh35举例说明。

用户执行`mysh35`后，屏幕打印提示符

```
mysh35@
```

等待用户的输入。当用户输入命令，敲回车后，mysh35应当执行命令，结束后返回；如果用户输入了`quit`，则退出，返回原始的shell。

需要支持两类命令（类似于原始的shell），一类为"内置命令"，一类为“外部命令”。

内置命令：

- quit，退出shell，返回原始的shell。
- promise，屏幕打印“I am XXX. I will study OS hard."，其中XXX为学生的姓名拼音。

- 如果输入为空，则换行，重新打印提示符。

外部命令：

- ls，已经实现的ls命令。
- sh，即原始的命令解释器。
- mysh35，即新的命令解释器。（是的，shell本来就可以一直嵌套执行。）

简化约定：

- 如果不是以上命令，则返回继续接受输入，屏幕打印提示“mysh35: command not found：XXX”，其中，XXX为用户刚才的输入。
- 约定所有命令不带参数。
- 为了避免同学直接复制粘贴sh.c文件，约定mysh35.c的实现中，不允许自定义子函数，即，全文件只有一个main函数，不需要考虑缓冲区溢出问题，可以使用库函数gets和strcmp。

验收要求

- 未知命令，空白命令返回，继续等待输入
- promise命令打印特定字符串，并换行
- 执行quit，退出mysh35
- 顺利执行外部命令ls、sh、mysh35

  以下为一个典型的输出。

```
$ mysh35
mysh35@ hello
mysh35: command not found: hello
mysh35@
mysh35@ promise
I am Xiaojun. I will study OS hard.
mysh35@ quit
$ mysh35
mysh35@ ls
.              1 1 512
..             1 1 512
cat            2 2 16288
echo           2 3 15140
grep           2 4 18508
init           2 5 15728
kill           2 6 15168
ln             2 7 15024
ls             2 8 17652
mkdir          2 9 15268
rm             2 10 15244
sh             2 11 27888
wc             2 12 17020
zombie         2 13 14836
shutdown       2 14 15092
```

```
forktest        2 15 16016
mysh35          2 16 15868
console         3 17 0
mysh35@ mysh35
mysh35@ quit
mysh35@ sh
$ mysh35
mysh35@ quit
$ shutdown
Leaving with code -1
```

实验提示

> - 模仿sh.c，灵活应用fork+exec，目前sh.c只实现了一个内置命令，cd，其他均为外部命令。
> - 在最简单的实现中，可以使用strcmp逐一比对5个命令，分别处理；整个代码应该100行以内，或者50行
> - 注意，exec的输入参数中，argv包含了命令行参数列表，最后一个元素必须是0，比如命令“exit”，则argv应该长这样（argv[0]可以指向任意字符串，比如栈区的，现在这个字符串常量在只读数据区）：
>
> ```
> char * argv[2];
> argv[0]="exit";
> argv[1]=0;
> ```

实验报告要求

请在实验报告中回答问题：

- 你能否将两个内置命令quit和promise实现为外部命令？如果可以，怎么实现？（不需要真实现）

4. 实验报告和代码（20%）

在助教现场验收结束后，需要准备两份文件：

- 源代码：zip压缩包
  - 进入`proj1-revise`目录，执行`make clean`，然后在proj1-revise的父目录下执行`zip -r proj1-revise.zip proj1-revise`，此命令会创建一个源代码压缩文件`proj1-revise.zip`。提交此文件。
- 实验报告：pdf，
  - 在实验报告中回答上面提到的问题；
  - 给出参考资料（网址）和工具（哪个大模型）；
  - 报告中需要添加姓名学号。不限定模板，尽量整洁、美观。