Lab2 SYSCALL： 系统调用

操作系统实验指导书 - 2024秋季 [实验概述 (https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu)](https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu) | [常见问题汇总 (https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong)](https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong) | [Lab1 UTIL (https://elearning.fudan.edu.cn/courses/78523/pages/lab1-util-xv6yu-unixying-yong-cheng-xu)](https://elearning.fudan.edu.cn/courses/78523/pages/lab1-util-xv6yu-unixying-yong-cheng-xu) | **Lab2 SYSCALL** | [Lab3 Scheduling (https://elearning.fudan.edu.cn/courses/78523/pages/lab3-scheduling-jin-cheng-diao-du)](https://elearning.fudan.edu.cn/courses/78523/pages/lab3-scheduling-jin-cheng-diao-du) | [Lab4 Page table (https://elearning.fudan.edu.cn/courses/78523/pages/lab4-page-table-ye-biao)](https://elearning.fudan.edu.cn/courses/78523/pages/lab4-page-table-ye-biao)

一、实验目的

本节实验的目的是对操作系统的系统调用模块进行修改，了解 xv6 内核的一些内部机制。

1. 了解xv6系统调用的工作原理。
2. 熟悉xv6通过系统调用给用户程序提供服务的机制。

二、实验准备

首先确保你已经拥有远程仓库的最新代码，然后切换到syscall分支后进行开发。

```
$ git fetch
$ git checkout syscall
$ make clean
```

在切换分支之前，记得通过`git commit -m`保存实验一的内容。 如果你希望舍弃实验一的更改，也可以使用-f选项强制切换分支，例如`git checkout -f syscall`。

三、实验内容

3.1 系统调用信息打印

实验目标

实现具有**系统调用跟踪**功能的`trace`系统调用，它可以打印系统调用信息。

预期结果

```
$ make qemu

/* 一大波输出 …… */

xv6 kernel is booting

hart 2 starting
```

```
hart 1 starting
init: starting sh
$

/* 例子1，手动输入:trace 32 grep hello README */

$ trace 32 grep hello README
3: sys_read(3) -> 1023
3: sys_read(3) -> 966
3: sys_read(3) -> 70
3: sys_read(3) -> 0
$

/* 例子2，手动输入:trace 2147483647 grep hello README */

$ trace 2147483647 grep hello README
4: sys_trace(2147483647) -> 0
4: sys_exec(12240) -> 3
4: sys_open(12240) -> 3
4: sys_read(3) -> 1023
4: sys_read(3) -> 966
4: sys_read(3) -> 70
4: sys_read(3) -> 0
4: sys_close(3) -> 0
$

/* 例子3，手动输入:grep hello README */

$ grep hello README
$


/* 例子4，手动输入:trace 2 usertests forkforkfork */

$ trace 2 usertests forkforkfork
usertests starting
test forkforkfork: 407: syscall fork -> 408
408: sys_fork(-1) -> 409
409: sys_fork(-1) -> 410
410: sys_fork(-1) -> 411
409: sys_fork(-1) -> 412
410: sys_fork(-1) -> 413
409: sys_fork(-1) -> 414
411: sys_fork(-1) -> 415
...
$
```

1. 在 **第一个例子** 中，`trace 32 grep hello README`，其中，trace表示我们希望执行用户态应用程序trace（见user/trace.c），后面则是trace应用程序附带的入参，其中，`32`是"1 << SYS_read"，表示只追踪系统调用read。该命令的作用是使用grep程序（见 user/grep. c）查找README文件中匹配"hello"的行，并将其所使用到的read系统调用的信息打印出来，打印的格式为：PID: sys_read(read系统调用的arg0) -> read系统调用的return_value。
2. 在 **第二个例子** 中，trace也是启动了`grep`程序，同时追踪所有的系统调用。其中`2147583647`是`31`位bit全置一的十进制整型。可以看出，打的第一条信息就是系统调用trace，其第一个参数即命令行中输入2147583647。
3. 在 **第三个例子** 中，启动了`grep`程序，但是没有使用trace，所以什么trace都不会出现。
4. 在 **第四个例子** 中，trace启动了`usertests`程序中`forkforkfork`（见 user/usertests.c），追踪系统调用了fork，每次fork后代都会打印对的进程id。

- 该例中的fork实际上并没有参数，方便起见，你可以直接打印用于传该参数的寄存器的值，它可能是任意值。
- forkforkfork 会一直不停的fork子进程，直到进程数超过`NPROC`，其定义见kernel/param.h。
- usertests是实验提供的用于测试xv6的系统调用，详见user/usertests.c。

提示

关于`trace`具体实现：

- `trace`接收一个`int`类型参数，代表`trace`要跟踪的系统调用功能。具体的系统调用对应索引请参考`kernel/syscall.h`。
- 进程启动`trace`后，如果fork，子进程也应该开启trace，并且继承父进程的`mask`，也就是trace系统调用的入参。（需要修改`kernel/proc.c`中fork()的代码）
- 可以在PCB （`struct proc`）中添加成员`int mask`，这样我们可以记住trace告知进程的mask。PCB定义于`kernel/proc.h`。
- 为了让每个系统调用都可以输出信息，我们应该在`kernel/syscall.c`中的`syscall()`添加相应逻辑。 **关于流程：**
- 记得在Makefile中给`UPROGS`加`$U/_trace`。
- 执行`make qemu`，你会发现无法编译 user/trace.c，这是因为还没有在用户态包装好`trace()`，因此我们需要添加一个系统调用的接口：
  - 在`user/user.h`加入函数定义；
  - 在`user/usys.pl`加入用户系统调用名称；
  - 在`kernel/syscall.h`加入系统调用号SYS_trace，以给trace做一个标识，该调用号的取值可自行决定。
- 然后启动xv6，在shell输入`trace 32 grep hello README`，会发现xv6崩溃了，因为这条系统调用没有在内核中实现，我们应该在内核部分进行以下步骤：
  - 在`kernel/sysproc.c`中添加`sys_trace()`。
  - 在`kernel/syscall.c`中加入对应的系统调用分发逻辑。
  - 开始实现`sys_trace()`对应的逻辑，同时该系统调用还需要修改其他函数的逻辑。

3.2 添加系统调用sysinfo

实验目标

实现具有**收集xv6运行信息**功能的`sysinfo`系统调用。sysinfo只需要一个参数，这个参数是结构体`sysinfo`的指针， **这个结构体在kernel/sysinfo.h** 可以找到。xv6内核的工作就是把这个结构体填上应有的数值。

```
$ struct sysinfo {
$     uint64 freemem; // amount of free memory (bytes)
$     uint64 nproc; // number of process
$ };
```

- `freemem`：当前剩余的内存 **字节** 数

- `nproc`： **状态为UNUSED** 的进程个数

预期结果

实验提供了一个`sysinfotest`用户级应用程序（见`user/sysinfotest.c`），依次测试剩余的内存字节数与UNUSED的进程个数。

本地测试

完成任务后，你可以在xv6中运行`sysinfotest`程序，通过测试会显示如下内容：

```
$ sysinfotest
sysinfotest: start
sysinfotest: OK
$
```

流程

在用户部分：

- 切至 `syscall` git分支
- 记得在`Makefile`中给 `UPROGS` 加 `$U/_sysinfotest`
- 然后 `make qemu` 会发现无法编译 `user/sysinfotest.c` ，问题和前面一样，一顿操作猛如虎。

```
/* 你需要在user/user.h添加如下定义 */
struct sysinfo;  // 需要预先声明结构体，参考fstat的参数stat
int sysinfo(struct sysinfo *);
```

在内核部分：

- 请参考之前的流程自行完成设计。

提示

- `sysinfo` 需要在内核地址空间中填写结构体，然后将其复制到用户地址空间。可以参考 `kernel/file.c fstat()` 以及 `kernel/sysfile.csys_fstat()` 中通过 `copyout()` 函数对该过程的实现。
- 计算剩余的内存空间的函数代码，最好写在文件 `kernel/kalloc.c` 里。
- 计算空闲进程数量的函数代码，最好写在文件`kernel/proc.c` 里。
- 在添加上述两个函数后，可以在`kernel/defs.h`中声明，以便在其他文件中调用这些函数。
- 查阅《xv6 book》`chapter1`和`chapter2`中相关的内容。

3.3 问答题

1. 简述`trace`全流程。
2. 阅读`kernel/syscall.h`，试解释该文件的作用，以及它如何发挥作用?
3. 阅读`kernel/syscall.c`，试解释函数`syscall()`如何根据系统调用号调用对应的系统调用处理函数（例如`sys_fork`）？`syscall()`将具体系统调用的返回值存放在哪里?
4. 阅读`kernel/syscall.c`，哪些函数用于传递系统调用参数？试解释argraw()函数的含义。

请在实验报告中回答以上问题。

四、实验结果提交

将实验报告和实验代码打包为压缩文件提交到eLearning平台。

4.1 实验报告

参照[课程文件 (https://elearning.fudan.edu.cn/courses/78523/files/folder/%E5%AE%9E%E9%AA%8C%E7%9B%B8%E5%85%B3%E6%96%87%E6%A1%A3)](https://elearning.fudan.edu.cn/courses/78523/files/folder/%E5%AE%9E%E9%AA%8C%E7%9B%B8%E5%85%B3%E6%96%87%E6%A1%A3)中的[《OS实验报告模板》 (https://elearning.fudan.edu.cn/files/4915949/)](https://elearning.fudan.edu.cn/files/4915949/)，按如下要求书写实验报告，力争规范、简洁。

1. 对于每个实验，详细描述实验过程，对于你认为的关键步骤附上必要的截图。
2. 有需要写代码的实验，必须配有代码、注释以及对代码功能的说明。
3. 鼓励实验报告中包括但不局限于以下内容：实验过程中碰到了什么问题？如何解决这些问题？实验后还存在哪些疑问或者有什么感想?
4. 如果实验附有练习，请在每个练习之后作答，这是实验报告评分的重要部分。

4.2 实验代码

不需要提交完整的代码包，**只需要提交**`commit.patch`**文件即可**，操作步骤如下：

- 在完成实验之后，将当前分支上的所有更改进行提交（`commit`，具体方法参考[git使用教程 (https://os-labs.pages.dev/lab1/part4/#3-git)](https://os-labs.pages.dev/lab1/part4/#3-git)）。
- 在仓库的目录下使用`make diff`命令导出更改文件`commit.patch`。

4.3 提交平台

请将**生成的**`commit.patch`**文件**与**实验报告**一起打包提交到eLearning平台Lab2 System Calls。

操作系统实验指导书 - 2024秋季 [实验概述 (https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu)](https://elearning.fudan.edu.cn/courses/78523/pages/shi-yan-gai-shu) | [常见问题汇总 (https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong)](https://elearning.fudan.edu.cn/courses/78523/pages/chang-jian-wen-ti-hui-zong) | [Lab1 UTIL (https://elearning.fudan.edu.cn/courses/78523/pages/lab1-util-xv6yu-unixying-](https://elearning.fudan.edu.cn/courses/78523/pages/lab1-util-xv6yu-unixying-)

yong-cheng-xu) | **Lab2 SYSCALL** | Lab3 Scheduling (https://elearning.fudan.edu.cn/courses/78523/pages/lab3-scheduling-jin-cheng-diao-du) | Lab4 Page table (https://elearning.fudan.edu.cn/courses/78523/pages/lab4-page-table-ye-biao)