@[TOC](Mit6.S081-实验2-System calls)

# 一、System call tracing

## 1，实验准备

1）阅读xv6 book章节2、4.3、4.4；

2）system call的用户空间代码：user/user.h和user/usys.pl

3）system call的kernel空间代码：kernel/syscall.h和user/syscall.c

4）进程相关代码：kernel/proc.h和kernel/proc.c

## 2，实验要求

添加一个trace system call调用，可以实现跟踪system call。

此函数入参为一个数字，可以控制跟踪哪些system call。

如：

trace(1<<SYS\_fork)，trace(10b)，trace(2)表示跟踪fork调用；

trace(1<<SYS\_read)，trace(10 0000b)，trace(32)，表示跟踪read调用；

trace(10 0010b)，trace(34)，表示跟踪fork、read调用；

达到的效果：trace 32 grep hello README，表示执行grep hello README时，read system call调用时，进行打印。

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019192203428.png#pic\_center)

## 3，system call调用链路

1）在user/user.h做函数声明

2）Makefile调用usys.pl（perl脚本）生成usys.S，里面写了具体实现，通过ecall进入kernel，通过设置寄存器a7的值，表明调用哪个system call

3）ecall表示一种特殊的trap，转到kernel/syscall.c:syscall执行

4）syscall.c中有个函数指针数组，即一个数组中存放了所有指向system call实现函数的指针，通过寄存器a7的值定位到某个函数指针，通过函数指针调用函数

## 4，trace system call具体实现

1）在kernel/syscall.h中宏定义 #define SYS\_trace 22

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019194403408.png#pic\_center)

2）修改user/usys.pl中新增一个entry

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020101919470587.png#pic\_center)

3）在user/user.h中新增trace函数声明

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019194848917.png#pic\_center)

3）在kernel/proc.h的proc结构体定义一个字节数组，用于存放mask码

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019195200714.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

5）将trace函数的具体实现写在kernel/sysproc.c中

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019195257514.png#pic\_center)

6）kernel/syscall.c中新增sys\_trace函数定义

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019195615845.png#pic\_center)

7）kernel/syscall.c中函数指针数组新增sys\_trace

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019195811241.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

8）kernel/syscall.c中新建一个数组存放system\_call的名称

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019195930333.png#pic\_center)

9）kernel/syscall.c中根据mask值打印system call

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019202840589.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

10）kernel/proc.c中fork函数调用时，子进程复制父进程的mask

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019204848878.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 4，执行效果

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019200500547.png#pic\_center)

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201019200602392.png#pic\_center)

## 5，测试效果

在xv6-labs-2020中，执行下面指令，测试程序

```bash

./grade-lab-syscall trace

```

![在这里插入图片描述](https://img-blog.csdnimg.cn/202010192050014.png#pic\_center)

# 二、System call sysinfo

## 1，实验要求

添加一个sysinfo system call调用，可以实现打印可用空间(字节)、可用进程数。

达到的效果：sysinfo

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020150242506.png#pic\_center)

## 2，sysinfo system call具体实现

1）在kernel/syscall.h中宏定义 #define SYS\_sysinfo 23

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020150531173.png#pic\_center)

2）修改user/usys.pl中新增一个entry

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020102015072584.png#pic\_center)

3）在user/user.h中新增sysinfo结构体、sysinfo函数声明

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020150851683.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

4）kernel/syscall.c中新增sys\_trace函数定义

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020151121422.png#pic\_center)

5）kernel/syscall.c中函数指针数组新增sys\_trace；syscall\_names新增一个"sysinfo"元素

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020151302637.png#pic\_center)

6）kernel/proc.h中mask数组长度为24

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020151932128.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

7）kernel/kalloc.c中新增函数freemem\_size，获取空闲内存数量

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020152214215.png#pic\_center)

8）kernel/proc.c中新增函数proc\_num，获取可用进程数目

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020152341396.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

8）kernel/defs.h做出函数声明

![在这里插入图片描述](https://img-blog.csdnimg.cn/2020102015245763.png#pic\_center)

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020152539125.png#pic\_center)

8）kernel/sysproc.c加入sysinfo.h的header头，新增sysinfo的实现

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020152826857.png#pic\_center)

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020152915192.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

8）新写一个user/sysinfo.c的用户程序

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020153155627.png?x-oss-process=image/watermark,type\_ZmFuZ3poZW5naGVpdGk,shadow\_10,text\_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3UwMTM1Nzc5OTY=,size\_16,color\_FFFFFF,t\_70#pic\_center)

## 3，执行效果

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020153238856.png#pic\_center)

## 4，测试效果

在xv6-labs-2020中，执行下面指令，测试程序

```bash

./grade-lab-syscall sysinfo

```

![在这里插入图片描述](https://img-blog.csdnimg.cn/20201020153335226.png#pic\_center)