lab2：Xv6 system calls

该实验由System call tracing 、Sysinfo 两部分组成。

1.System call tracing

实验目的

添加一个系统调用跟踪功能，该功能可以在以后的实验中为你提供帮助。你将创建一个新的 `trace` 系统调用来控制跟踪。

它应该有一个参数，一个整数`mask(掩码)`，其指定要跟踪的系统调用。例如，为了跟踪 `fork` 系统调用，程序调用 `trace (1 << SYS\_fork) `，其中` SYS\_fork` 是来自` kernel/syscall.h `的系统调用号。

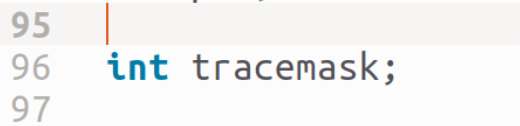
如果掩码中设置了系统调用的编号，则必须修改 xv6 内核以在每个系统调用即将返回时打印出一行。

该行应包含 进程 ID 、 系统调用名称 和 返回值 ；您不需要打印系统调用参数。 trace 系统调用应该为调用它的进程和它随后派生的任何子进程启用跟踪，但不应影响其他进程。

实验步骤

1.在 Makefile 的 UPROGS 环境变量中添加 $U/\_trace

2.在/kernel/proc.h文件的proc结构体中加入一个新的变量，让每个进程都有一个自己的mask：



3. 定义 trace 系统调用的原型, entry 和系统调用号

在 user/user.h中添加 trace 系统调用原型：

int trace(int);

在 user/usys.pl 脚本中添加 trace 对应的 entry

entry("trace");

在 kernel/syscall.h 中添加 trace 的系统调用号

#define SYS\_trace 22

4.编写 trace 的系统调用函数

结构体 struct proc 的定义在 kernel/proc.h 中，该结构体记录着进程的转态。需要为 trace 系统调用添加一个变量 tracemask 来记录其参数。因为 trace 只会在本进程发挥作用，所以 tracemask 应该作为进程的私有变量。



5. 修改 fork() 函数，将父进程的跟踪掩码复制到子进程。

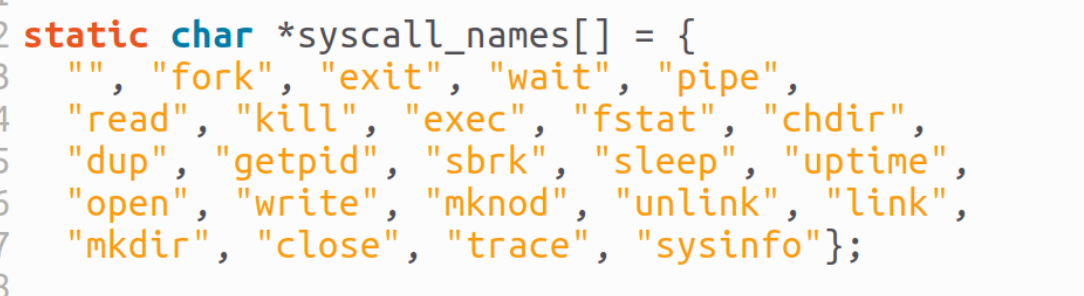
 np->tracemask = p->mask;

6. 在syscall.c中加上sys\_trace的声明

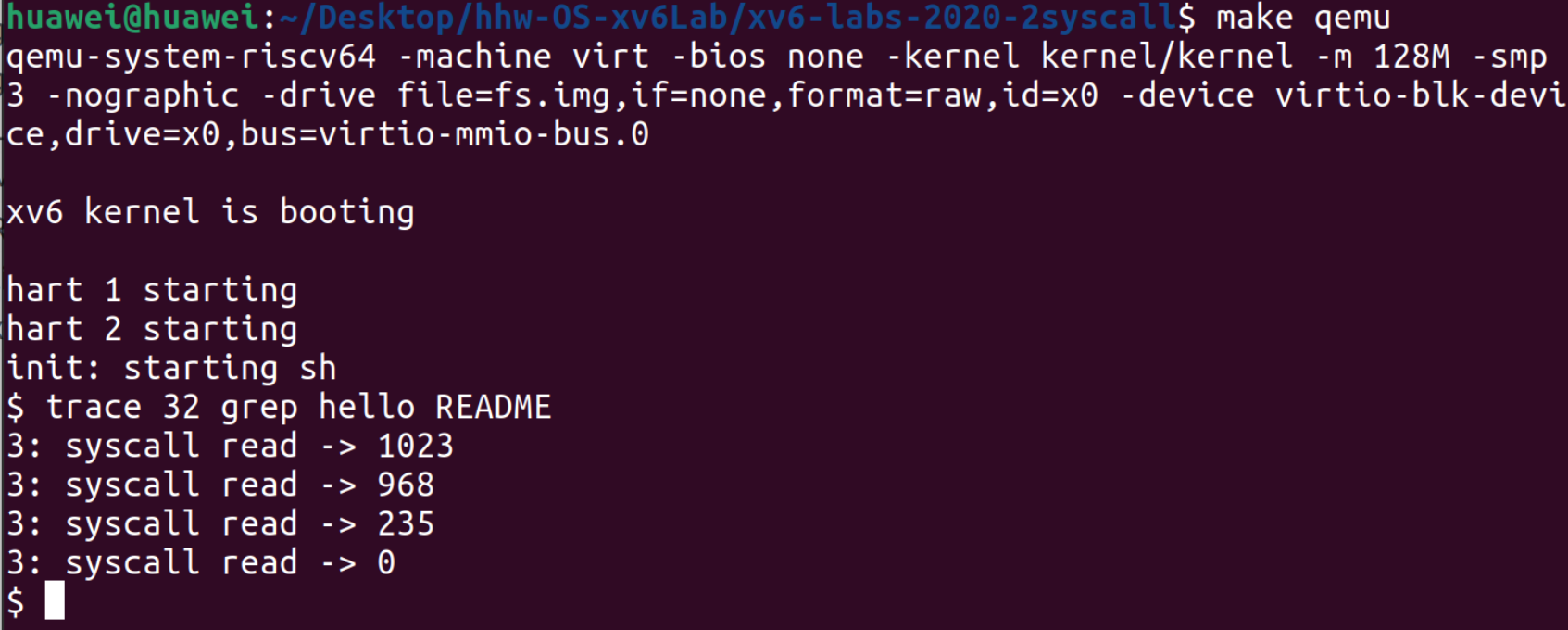
extern uint64 sys\_trace(void);

[SYS\_trace]   sys\_trace,

7. 在 syscall.c 中添加一个syscall\_names数组，使用系统调用名称来索引。

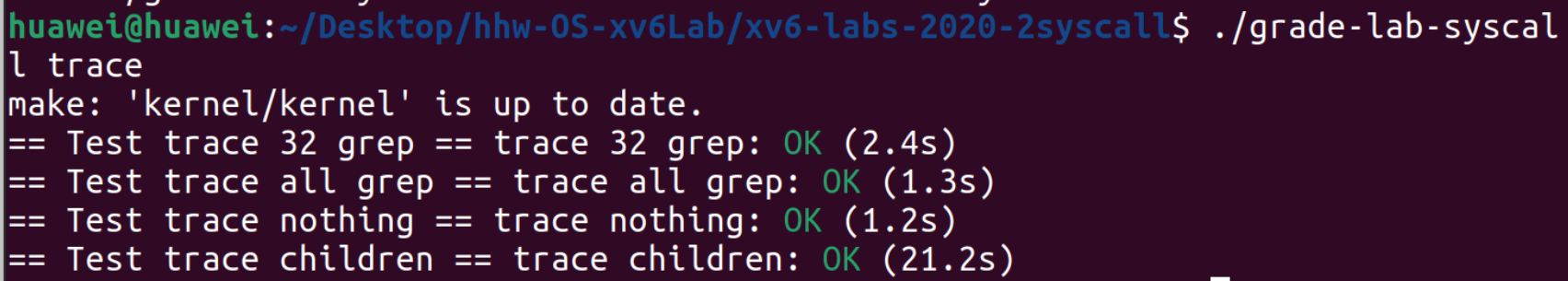


8.结果测试：





在终端里运行 ./grade-lab-syscall trace 可进行评分：



遇到的问题与心得

遇到的问题：

用户态和内核态之间的数据传递：在系统调用的实现中，需要正确地传递参数和数据，包括用户态和内核态之间的切换的问题。

通过仔细阅读系统调用相关的文档和源代码，找到了正确使用数据传递方法，如使用合适的系统调用参数读取函数等。

1. **系统调用参数传递：** 在实验中，用户程序通过调用 sys\_trace 系统调用来传递一个整数参数 mask 给内核，用于指定要追踪的系统调用。用户程序通过 argint 函数将参数从用户空间传递到内核空间。
2. **内核态与用户态切换：** 当用户程序调用 sys\_trace 系统调用时，会触发用户态切换到内核态。这个切换由系统调用机制完成。
3. **数据传递和复制：** 用户程序传递给内核的 tracemask 参数需要在内核中进行数据复制。当用户程序传递参数给内核时，内核需要正确地从用户空间复制数据到内核空间，以确保内核能够访问到正确的数据。
4. **数据验证和权限检查：** 在实验中，内核需要验证用户传递的 tracemask 参数是否合法，并确保用户程序有权限调用 sys\_trace 系统调用。内核可以通过对 tracemask 进行合法性检查来防止恶意传递错误的参数。

实验心得

从这个实验中，我学到了如何在xv6内核中添加新的系统调用，如何修改进程控制块以支持跟踪掩码，并且理解了如何在内核中实现系统调用的功能。

2. Sysinfo

实验目的

本实验添加一个系统调用sysinfo，用于收集有关正在运行的系统的信息。该系统调用接受一个参数：指向struct sysinfo结构体的指针（见kernel/sysinfo.h）。内核应填充该结构体的字段：freemem字段应设置为空闲内存的字节数，nproc字段应设置为状态不是UNUSED的进程数量。

提供了一个名为sysinfotest的测试程序；如果该程序打印出"sysinfotest: OK"，则表示通过此任务。

实验步骤

1. 在 Makefile 的 UPROGS 中添加 $U/\_sysinfotest

2.在 user/user.h中添加 sysinfo()调用原型：

struct sysinfo；

int sysinfo(struct sysinfo \*)；

3.在 kernel/syscall.h 中添加 trace 的系统调用号(为trace 分配一个系统调用的编号)

#define SYS\_sysinfo 23

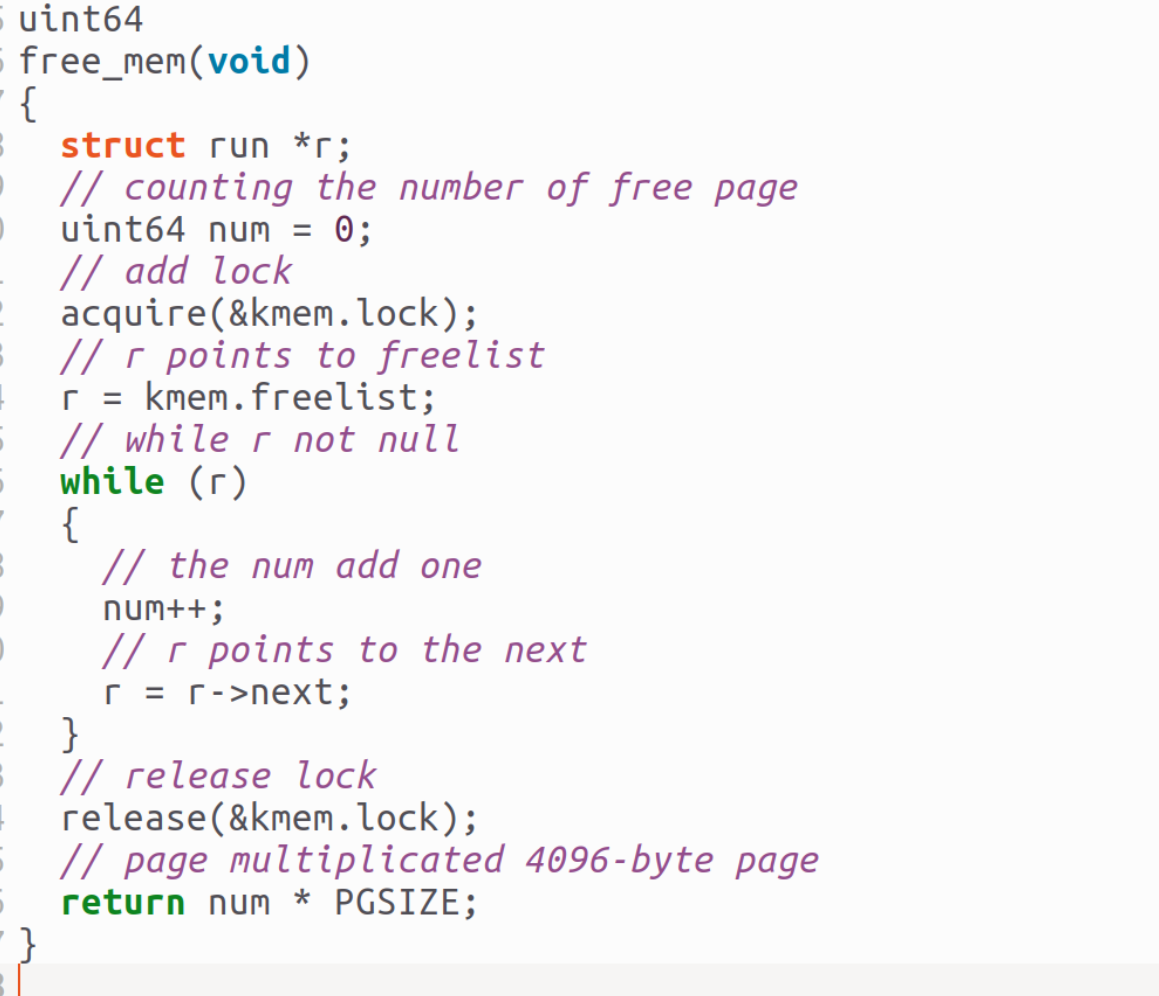
4. 在 syscall.c 中添加函数引用

#include "sysinfo.h"

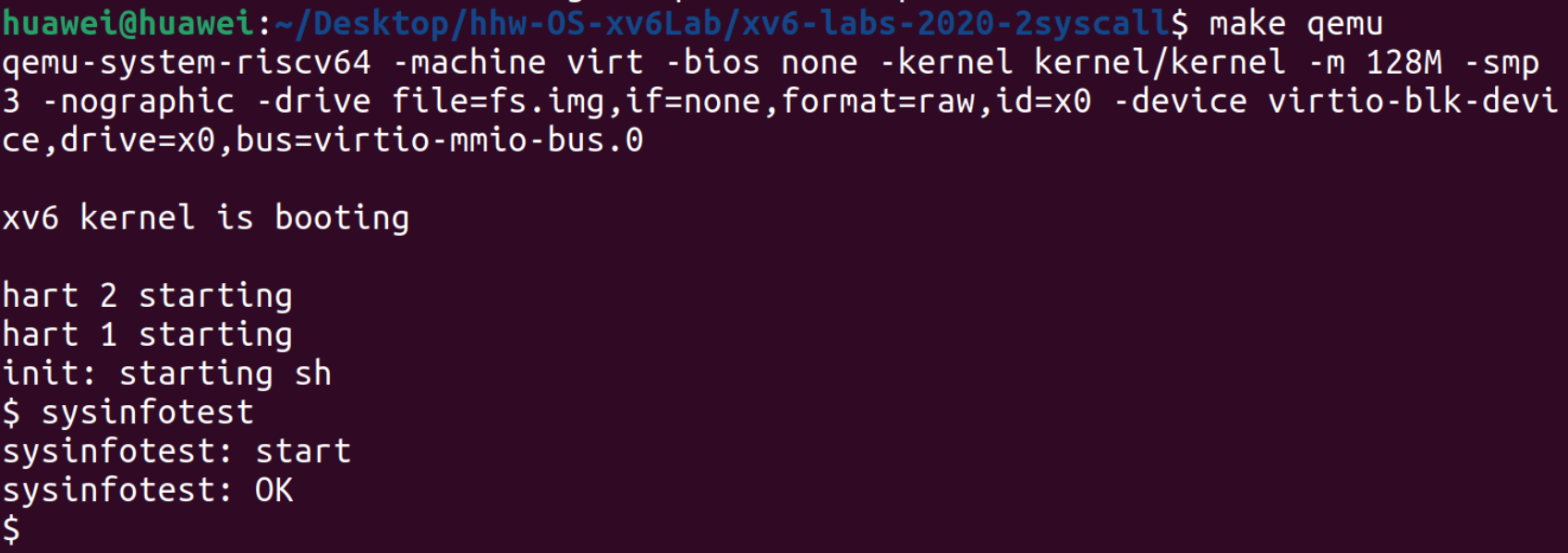
uint64 acquire\_freemem();

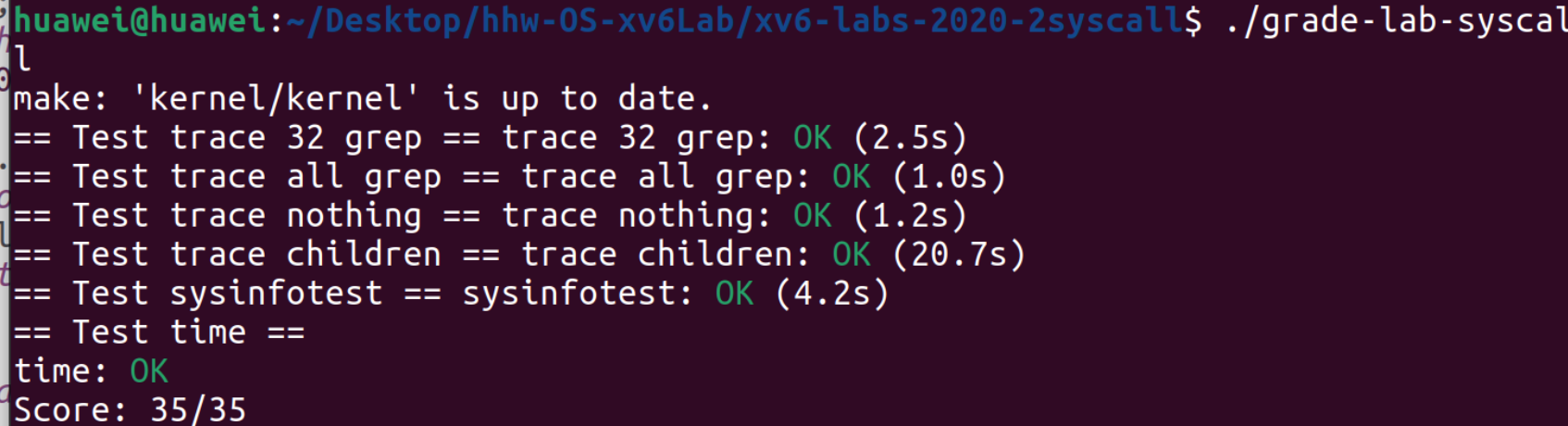
uint64 acquire\_nproc();

5. 编写函数, 获得空闲内存大小:



6.保存后在终端里执行make qemu编译运行xv6：





遇到的问题与心得

遇到的问题：

注意在实现共有变量的通信时要加一个锁来控制；不同文件使用相同函数要先声明

实验心得：

本实验让我掌握了一个新系统调用的添加过程，首先在user/user.h中加入系统调用在用户态的入口，其次在同级目录下的usys.pl中加入在用户态执行的汇编代码，然后在kernel/syscall.h中为系统调用分配一个系统调用的编号，最后深入内核代码，实现系统调用具体的逻辑，包括设计数据结构及其依赖的函数等。