

程序运行的主要界面和实验结果

为了方便修改程序，本次在程序调试阶段主要通过 Vscode 进行本地调试：

1 操作系统启动情况

输入命令 启动操作系统:

```
make run platform=gemu
```

如下图：即为操作系统的双核启动成功，运行环境为 `qemu`，启动成功并打印 “HELLO HFUT!”，同时双核都已初始化完成，下面两行打印（`init.c` 文件中定义的输出）表示双核都成功进入了 `init.c` 文件，接下来进入 `shell` 模式。

[illegible]

2 编写测试程序进行系统调用的测试（代码见附录）

由于时间有限，编写的测试程序主要选取了以下几个系统调用：（其余通过封装进 `initcode` 进行测试，详看下文介绍）

首先 `ls` 命令查看目录下所有的文件:

```

-> / $ ls
bin                DIR      0
README             FILE     2059
cat                FILE     25656
echo               FILE     24472
find               FILE     26672
grep               FILE     28960
init               FILE     25032
kill               FILE     24440
ls                 FILE     26776
mkdir              FILE     24568
mv                 FILE     26360
rm                 FILE     24496
sh                 FILE     51096
sleep              FILE     24496
strace             FILE     24856
test               FILE     24200
usertests          FILE     135600
wc                 FILE     26864
xargs              FILE     26296
forktest           FILE     14304
pwd                FILE     23984
openat             FILE     25384
uname              FILE     24768
getppid            FILE     24224
autotest           FILE     25328
rename             FILE     24224
gettimeofday       FILE     24360
getdents64         FILE     24544
times              FILE     24472

```

接着输入下面命令创建 hello 文件，并写入 hello world! (为了验证 openat 和 rename)

```
Echo hello world! > hello
```

接着通过 cat 命令可以查看是否写入成功：

```

-> / $ echo hello world! > hello
-> / $ cat hello
hello world!
-> / $ █

```

接下来可以选择单独进行测试，如 getdents64:

```

-> / $ getdents64
get root: bin
test getdents64 success!
-> / $ █

```

结果显示测试成功，可以看到当前启动的操作系统的工作目录为 bin 目录，输出为 bin，成功。

也可以进行多个命令一起测试（通过 autotest.c 文件封装，依次运行所有测试程序）

```

-> / $ autotest
===== start test uname =====
System Information:
Operating System: hello-hfut
Node Name: BootLoader
Release: 0.0.0
Version: 0.0.0
Machine: riscv-64
Domainname: unknown
test uname success!
===== start test getppid =====
pid = 16
ppid = 14
getppid test success!
===== start test times =====
User time: -200904327
Kernel time: -50068205
test times success!
===== start test openat =====
open success! fdd = 6
now in kernel's openat!
openat success! fd = 7
open & openat success!
this file's content is: hello world!
openat test success!
===== start test rename =====
test rename success!
===== start test gettimeofday =====
Seconds: 1056735
Microseconds: 282562
test gettimeofday success!
===== start test getdents64 =====
get root: bin
test getdents64 success!
-> / $ █

```

这里主要说明一下 `rename` 的验证，其余测试可以通过输出验证成功。输入 `ls` 命令，如下图所示。发现 `hello` 文件变成了 `HelloWorld` 文件，同时查看该文件内容发现仍然为 `hello world!`，说明重命名成功。

```

autotest          FILE    25328
rename            FILE    24224
gettimeofday      FILE    24360
getdents64        FILE    24544
times             FILE    24472
HelloWorld        FILE     13
-> / $ cat HelloWorld
hello world!
-> / $ █

```

3 利用 `initcode` 封装进行系统调用的测试

3.1 `qemu` 单个调试系统调用

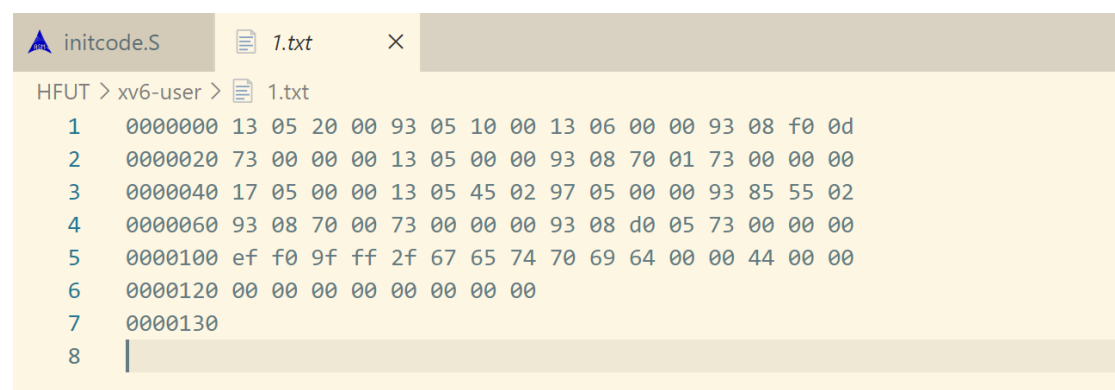
这里以 `getpid` 为例，在 `initcode.S` 文件中 `init` 下把 `init` 改为 `gitpid`，保存

之后进行编译。

```
.globl start
start:
    li a0, 2
    li a1, 1
    li a2, 0
    li a7, SYS_dev
    ecall
    li a0, 0
    li a7, SYS_dup
    ecall
    la a0, init
    la a1, argv
    li a7, SYS_exec
    ecall
# for(;;) exit();
exit:
    li a7, SYS_exit
    ecall
    jal exit
# char init[] = "/init\0";
init:
    .string "/getpid\0"
# char *argv[] = { init, 0 };
.p2align 2
argv:
    .long init
    .long 0
```

在 xv6-user 目录下使用以下命令将 initcode 转换为二进制文件并保存：

```
od -t xC initcode > 1.txt
```



```
HFUT > xv6-user > 1.txt
1  0000000 13 05 20 00 93 05 10 00 13 06 00 00 93 08 f0 0d
2  0000020 73 00 00 00 13 05 00 00 93 08 70 01 73 00 00 00
3  0000040 17 05 00 00 13 05 45 02 97 05 00 00 93 85 55 02
4  0000060 93 08 70 00 73 00 00 00 93 08 d0 05 73 00 00 00
5  0000100 ef f0 9f ff 2f 67 65 74 70 69 64 00 00 44 00 00
6  0000120 00 00 00 00 00 00 00 00
7  0000130
8  |
```

接下来输入下面的命令，将上述得到的二进制文件改为十六进制文件：

```
python3 1.py
```

其中 1.py 即为读取文件并将其中的二进制数转换为十六进制数并保存到另一个文件中，如下所示：

```
f = open('1.txt', 'r')
while True:
    line = f.readline()
    if len(line) > 8:
        newline = ""
        line.replace("\n", "")
        line.replace("\r", "")
        line_list = line.split(" ")

        for value in line_list:
            if len(value) <= 3:
                newline = newline + "0x" + value + ", "
        newline = newline[:-3]
        newline = newline + ",\n"
        with open("2.txt", "a") as mon:
            mon.write(newline)
    else:
        break
f.close()
```

```
initcode.S 2.txt x 1.txt
HFUT > xv6-user > 2.txt
1 0x13, 0x05, 0x20, 0x00, 0x93, 0x05, 0x10, 0x00, 0x13, 0x06, 0x00, 0x00, 0x93, 0x08, 0xf0, 0x0d,
2 0x73, 0x00, 0x00, 0x00, 0x13, 0x05, 0x00, 0x00, 0x93, 0x08, 0x70, 0x01, 0x73, 0x00, 0x00, 0x00,
3 0x17, 0x05, 0x00, 0x00, 0x13, 0x05, 0x45, 0x02, 0x97, 0x05, 0x00, 0x00, 0x93, 0x85, 0x55, 0x02,
4 0x93, 0x08, 0x70, 0x00, 0x73, 0x00, 0x00, 0x00, 0x93, 0x08, 0xd0, 0x05, 0x73, 0x00, 0x00, 0x00,
5 0xef, 0xf0, 0x9f, 0xff, 0x2f, 0x67, 0x65, 0x74, 0x70, 0x69, 0x64, 0x00, 0x00, 0x44, 0x00, 0x00,
6 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
7
```

复制得到的十六进制数，替换 proc.c 文件中的 uchar initcode，如下

```
// a user program that calls exec("/init")
// od -t xC initcode
uchar initcode[] = {
    0x13, 0x05, 0x20, 0x00, 0x93, 0x05, 0x10, 0x00, 0x13, 0x06, 0x00, 0x00, 0x93, 0x08, 0xf0, 0x0d,
    0x73, 0x00, 0x00, 0x00, 0x13, 0x05, 0x00, 0x00, 0x93, 0x08, 0x70, 0x01, 0x73, 0x00, 0x00, 0x00,
    0x17, 0x05, 0x00, 0x00, 0x13, 0x05, 0x45, 0x02, 0x97, 0x05, 0x00, 0x00, 0x93, 0x85, 0x55, 0x02,
    0x93, 0x08, 0x70, 0x00, 0x73, 0x00, 0x00, 0x00, 0x93, 0x08, 0xd0, 0x05, 0x73, 0x00, 0x00, 0x00,
    0xef, 0xf0, 0x9f, 0xff, 0x2f, 0x67, 0x65, 0x74, 0x70, 0x69, 0x64, 0x00, 0x00, 0x44, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
}
```

然后退出中断，make clean 后再次运行即可得到测试结果：

```
hart 0 init done
hart 1 init done
===== START test_getpid =====
getpid success.
pid = 1
===== END test_getpid =====
```

结果显示测试成功。

3.2 实现一次性多个系统调用

为了连续跑多个评测程序，新增一个系统调用

```
\#define SYS\_runtest 800
```

接下来定义这个函数，加入我们需要测试的系统调用，实现代码如下：

```
uint64
sys_runtest(void)
{
    char *argv1[] = { "brk", 0 };
    char *argv2[] = { "chdir", 0 };
    char *argv3[] = { "close", 0 };
    char *argv4[] = { "dup", 0 };
    char *argv5[] = { "dup2", 0 };
    char *argv6[] = { "execve", 0 };
    char *argv7[] = { "fork", 0 };
    char *argv8[] = { "fstat", 0 };
    char *argv9[] = { "getcwd", 0 };
    char *argv10[] = { "getdents", 0 };
    char *argv11[] = { "getpid", 0 };
    char *argv12[] = { "mkdir_", 0 };
    char *argv13[] = { "open", 0 };
    char *argv14[] = { "openat", 0 };
    char *argv15[] = { "pipe", 0 };
    char *argv16[] = { "read", 0 };
    char *argv17[] = { "uname", 0 };
    char *argv18[] = { "unlink", 0 };
    char *argv19[] = { "write", 0 };
    fork();
    exec("brk", argv1);
    fork();
    exec("chdir", argv2);
    fork();
```

```
exec("close", argv3);
fork();
exec("dup", argv4);
fork();
exec("dup2", argv5);
fork();
exec("execve", argv6);
fork();
exec("fork", argv7);
fork();
exec("fstat", argv8);
fork();
exec("getcwd", argv9);
fork();
exec("getdents", argv10);
fork();
exec("getpid", argv11);
fork();
exec("mkdir_", argv12);
fork();
exec("open", argv13);
fork();
exec("openat", argv14);
fork();
exec("pipe", argv15);
fork();
exec("read", argv16);
fork();
exec("uname", argv17);
fork();
exec("unlink", argv18);
fork();
exec("write", argv19);
return 0;
}
```

接下来修改 initcode.S,如下:

```

.global start
start:
    li a0, 2
    li a1, 1
    li a2, 0
    li a7, SYS_dev
    ecall
    li a0, 0
    li a7, SYS_dup
    ecall
    li a7, SYS_runtest
    ecall
    #la a0, init
    #la a1, argv
    #li a7, SYS_exec
    #ecall
# for(;;) exit(),

```

接着按着上面的方法得到 `initcode` 的十六进制数并替换 `proc.c` 中的 `uchar initcode`，重新运行后得到测试结果：

```

===== START test_brk =====
Before alloc,heap pos: 24576
After alloc,heap pos: 24640
Alloc again,heap pos: 24704
===== END test_brk =====
===== START test_chdir =====
chdir ret: 0
    current working dir : /test_chdir
===== END test_chdir =====
===== START test_clone =====
    Child says successfully!
clone process successfully.
pid:5
===== END test_clone =====

```



```

===== START test_close =====
close 3 success.
===== END test_close =====
===== START test_dup2 =====
from fd 100
===== END test_dup2 =====
===== START test_dup =====
new fd is 3.
===== END test_dup =====
===== START test_execve =====
I am test_echo.
execve success.
===== END main =====
===== START test_exit =====
exit OK.
===== END test_exit =====
===== START test_fork =====
child process.
parent process. wstatus:0
===== END test_fork =====
===== START test_fstat =====
can't read registers
fstat ret: -1

--- Assert Fatal ! ---
===== START test_getcwd =====
getcwd: / successfully!
===== END test_getcwd =====
===== START test_getdents =====
open fd:3
getdents fd:56
getdents success.
/

```

```

===== END test_getdents =====
===== START test_getpid =====
getpid success.
pid = 17
===== END test_getpid =====
===== START test_getppid =====
getppid success. ppid : 1
===== END test_getppid =====

```

```
===== START test_gettimeofday =====
gettimeofday success.
start:2343, end:2430
interval: 87
===== END test_gettimeofday =====
===== START test_mkdir =====
mkdir ret: 0
    mkdir success.
===== END test_mkdir =====
===== START test_mmap =====
file len: 27
mmap content:  Hello, mmap successfully!
===== END test_mmap =====
===== START test_mount =====
Mounting dev:/dev/vda2 to ./mnt
mount not found filemount return: -1
```

```
    --- Assert Fatal ! ---
===== START test_munmap =====
file len: 27
munmap return: 0
munmap successfully!
===== END test_munmap =====
===== START test_openat =====
open dir fd: -1
openat fd: 3
openat success.
===== END test_openat =====
===== START test_open =====
```

```
    --- Assert Fatal ! ---
===== START test_pipe =====
cpid: 27
```

```
cpid: 0
    Write to pipe successfully.

===== END test_pipe =====
===== START test_read =====

    --- Assert Fatal ! ---
===== START test_times =====
mytimes success
{tms_utime:0, tms_stime:0, tms_cutime:0, tms_cstime:0}
===== END test_times =====
===== START test_umount =====
Mounting dev:/dev/vda2 to ./mnt
mount not found filemount return: -1
===== END test_umount =====
```

```
===== START test_uname =====
Uname: domainname machine nodename release sysname version
===== END test_uname =====
===== START test_unlink =====
unlink success!
===== END test_unlink =====
===== START test_wait =====
This is child process
wait child success.
wstatus: 0
===== END test_wait =====
===== START test_waitpid =====
This is child process
waitpid successfully.
wstatus: 3
===== END test_waitpid =====
===== START test_write =====
Hello operating system contest.
===== END test_write =====
===== START test_yield =====
I am child process: 39. iteration 0.
I am child process: 40. iteration 1.
I am child process: 41. iteration 2.
I am child process: 39. iteration 0.
I am child process: 40. iteration 1.
I am child process: 41. iteration 2.
I am child process: 39. iteration 0.
I am child process: 40. iteration 1.
I am child process: 41. iteration 2.
I am child process: 39. iteration 0.
I am child process: 40. iteration 1.
I am child process: 41. iteration 2.
I am child process: 39. iteration 0.
I am child process: 40. iteration 1.
I am child process: 41. iteration 2.
===== END test_yield =====
===== START test_sleep =====
sleep success.
===== END test_sleep =====
```