Operating System Homework 2

Jinyan Xu, 3160101126, Information Security

1. Operating System Concept Chapter 2 Exercises: 2.9, 2.10, 2.12. 2.17

Answer:

2.9) The services and functions provided by an operating system can be divided into two main categories. Briefly describe the two categories, and discuss how they differ.

System services, also known as system utilities, provide a convenient environment for program development and execution.

One kind of services is dealing with physical information, like Status information, File modification, Programming-language support, Program loading and execution. These functions operate on the data that actually exists on hardware, such as device status information, text content, compilers, assemblers, debuggers, interpreters, linkers, loaders, and so on.

And the other kind is dealing with logical virtual information, like File management, Program loading and execution, Background services. These functions manage what the user abstracts logically, like file system, pipes, virtual memory, subsystems, and daemons.

2.10) Describe three general methods for passing parameters to the operating system.

<1> Pass the parameters in registers.

<2> Parameters are stored in a block, or table, in memory. The address of the block is passed as a parameter in a register.

<3> Parameters can be pushed onto a stack by the program, popped off the stack by the operating system.

2.12) What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices?

Advantage: Once the device has been requested, we can read, write, and reposition the device, just as we can with files. Device can be identified by special file names, directory placement, or file attributes, so it’s quite easy for us to add new devices.

Disadvantage: Because there are more operations on the device than on the file, the efficiency of using the file system API to control the device becomes lower.

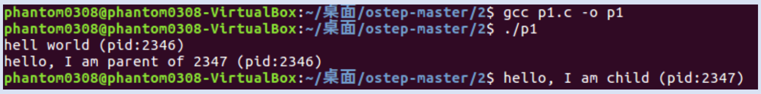
2.17) Why is the separation of mechanism and policy desirable?

Mechanisms determine *how to do something*; policies determine *what will be done*. So, the separation of policy and mechanism is important for flexibility. Policies are likely to change across places or over time. Without the separation, each change in policy would require a change in the underlying mechanism. With the separation, the mechanism can remain unchanged when the policy needs to change.

2. Compile and run the following code and capture the running results.

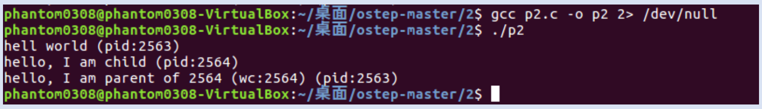
Answer:

p1.c:



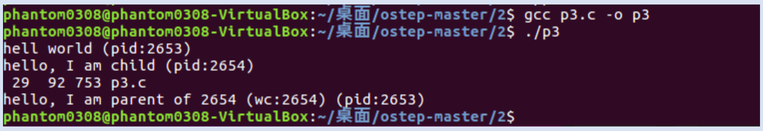
```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ gcc p1.c -o p1
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ ./p1
hell world (pid:2346)
hello, I am parent of 2347 (pid:2346)
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ hello, I am child (pid:2347)
```

p2.c:



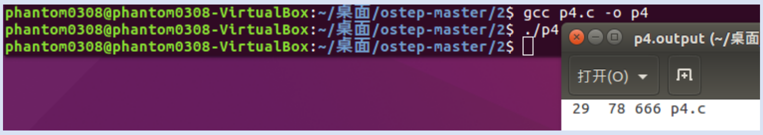
```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ gcc p2.c -o p2 2> /dev/null
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ ./p2
hell world (pid:2563)
hello, I am child (pid:2564)
hello, I am parent of 2564 (wc:2564) (pid:2563)
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ 
```

p3.c:



```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ gcc p3.c -o p3
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ ./p3
hell world (pid:2653)
hello, I am child (pid:2654)
 29  92 753 p3.c
hello, I am parent of 2654 (wc:2654) (pid:2653)
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$
```

p4.c:



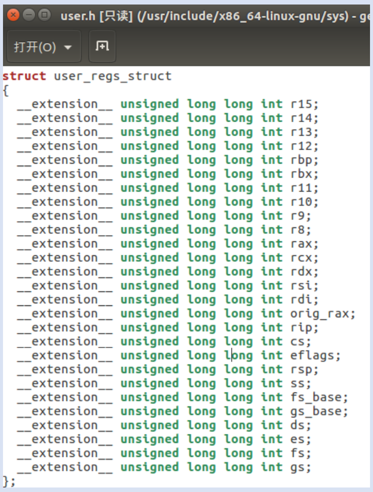
```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ gcc p4.c -o p4
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ ./p4
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ 
```

```
29  78 666 p4.c
```

3. Expand the ptrace sample code used in the class to display the pathname parameters of the open system call.(40 points)

Answer:

The system call number of open() is 2, this function has 3 parameters: const char *filename, int flags, int mode. Before changing to Kernel Model, system call number is saved in rax, flags is saved in rsi, mode is saved in rdx, filename is saved in block, its address is saved in rdi.

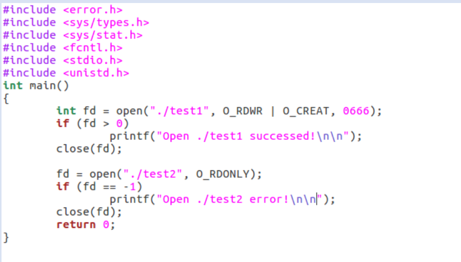


```
struct user_regs_struct
{
  __extension__ unsigned long long int r15;
  __extension__ unsigned long long int r14;
  __extension__ unsigned long long int r13;
  __extension__ unsigned long long int r12;
  __extension__ unsigned long long int rbp;
  __extension__ unsigned long long int rbx;
  __extension__ unsigned long long int r11;
  __extension__ unsigned long long int r10;
  __extension__ unsigned long long int r9;
  __extension__ unsigned long long int r8;
  __extension__ unsigned long long int rax;
  __extension__ unsigned long long int rcx;
  __extension__ unsigned long long int rdx;
  __extension__ unsigned long long int rsi;
  __extension__ unsigned long long int rdi;
  __extension__ unsigned long long int orig_rax;
  __extension__ unsigned long long int rip;
  __extension__ unsigned long long int cs;
  __extension__ unsigned long long int eflags;
  __extension__ unsigned long long int rsp;
  __extension__ unsigned long long int ss;
  __extension__ unsigned long long int fs_base;
  __extension__ unsigned long long int gs_base;
  __extension__ unsigned long long int ds;
  __extension__ unsigned long long int es;
  __extension__ unsigned long long int fs;
  __extension__ unsigned long long int gs;
};
```

The main task of the program is to extract the file name based on the address in rdi, but we can't just dereference the pointer to obtain the data, because of the operating system's protection, we can't directly access the memory of another processes, so I use PTRACE_PEEKDATA to get the data, the return value is long. Notice that data is saved as Little-endian in Intel CPU.

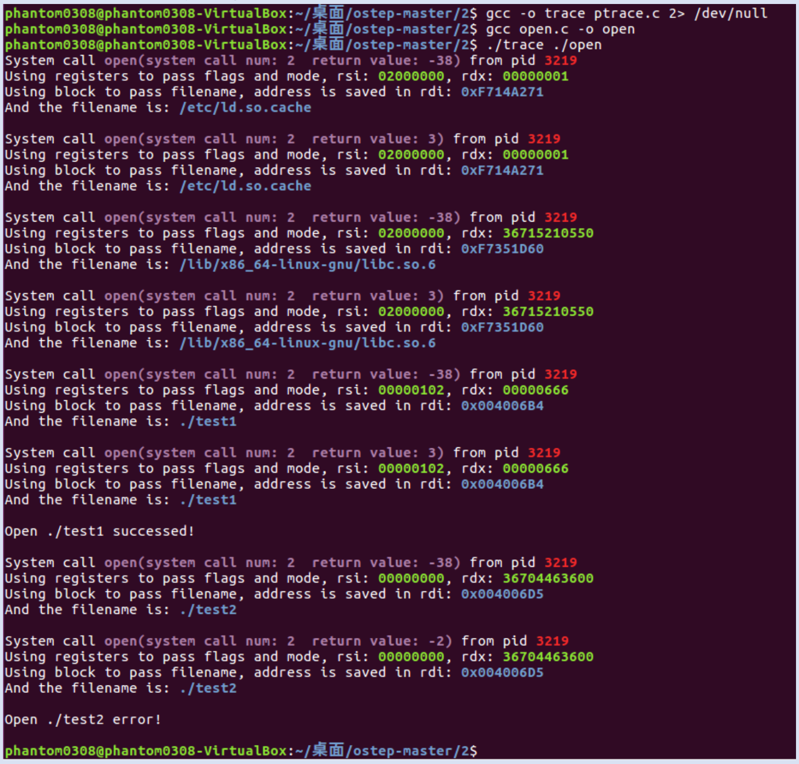
And I get the body of struct user_regs_struct, here has rax & orig_rax, the value in rax is return value after operation, the value in orig_rax is system call number before operation.

I prepare a program, which has both open success and fail example.



```
#include <error.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
int main()
{
        int fd = open("./test1", O_RDWR | O_CREAT, 0666);
        if (fd > 0)
                printf("Open ./test1 successed!\n\n");
        close(fd);

        fd = open("./test2", O_RDONLY);
        if (fd == -1)
                printf("Open ./test2 error!\n\n");
        close(fd);
        return 0;
}
```

Following is the result of my experiment result:

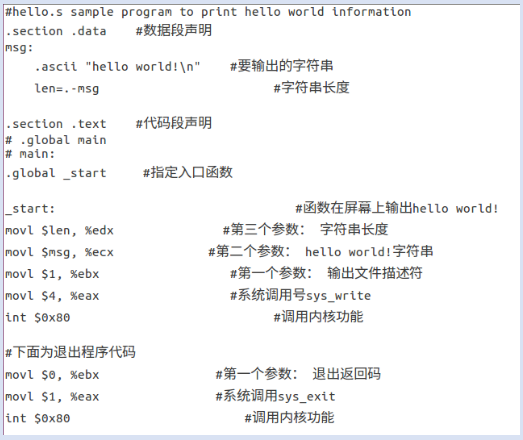


```
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ gcc -o trace ptrace.c 2> /dev/null
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ gcc open.c -o open
phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$ ./trace ./open
System call open(system call num: 2  return value: -38) from pid 3219
Using registers to pass flags and mode, rsi: 02000000, rdx: 00000001
Using block to pass filename, address is saved in rdi: 0xF714A271
And the filename is: /etc/ld.so.cache

System call open(system call num: 2  return value: 3) from pid 3219
Using registers to pass flags and mode, rsi: 02000000, rdx: 00000001
Using block to pass filename, address is saved in rdi: 0xF714A271
And the filename is: /etc/ld.so.cache

System call open(system call num: 2  return value: -38) from pid 3219
Using registers to pass flags and mode, rsi: 02000000, rdx: 36715210550
Using block to pass filename, address is saved in rdi: 0xF7351D60
And the filename is: /lib/x86_64-linux-gnu/libc.so.6

System call open(system call num: 2  return value: 3) from pid 3219
Using registers to pass flags and mode, rsi: 02000000, rdx: 36715210550
Using block to pass filename, address is saved in rdi: 0xF7351D60
And the filename is: /lib/x86_64-linux-gnu/libc.so.6

System call open(system call num: 2  return value: -38) from pid 3219
Using registers to pass flags and mode, rsi: 00000102, rdx: 00000666
Using block to pass filename, address is saved in rdi: 0x004006B4
And the filename is: ./test1

System call open(system call num: 2  return value: 3) from pid 3219
Using registers to pass flags and mode, rsi: 00000102, rdx: 00000666
Using block to pass filename, address is saved in rdi: 0x004006B4
And the filename is: ./test1

Open ./test1 successed!

System call open(system call num: 2  return value: -38) from pid 3219
Using registers to pass flags and mode, rsi: 00000000, rdx: 36704463600
Using block to pass filename, address is saved in rdi: 0x004006D5
And the filename is: ./test2

System call open(system call num: 2  return value: -2) from pid 3219
Using registers to pass flags and mode, rsi: 00000000, rdx: 36704463600
Using block to pass filename, address is saved in rdi: 0x004006D5
And the filename is: ./test2

Open ./test2 error!

phantom0308@phantom0308-VirtualBox:~/桌面/ostep-master/2$
```

The data in registers is represented by octal notation, the words in blue are the filename and its address.

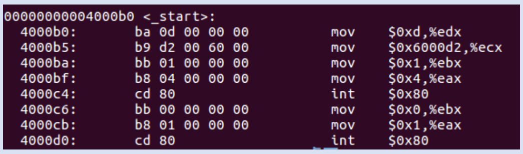
Here is something interesting, I find that before the main() there exits some open() operations, and each open() invokes twice sys_open().

To find out the reason, I wrote a assemble program with 32-bits (Notice that the system call number is different with 64-bits):



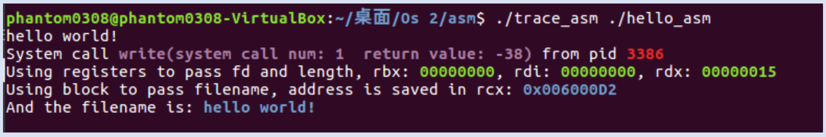
```
#hello.s sample program to print hello world information
.section .data    #数据段声明
msg:
    .ascii "hello world!\n"    #要输出的字符串
    len=.-msg                          #字符串长度

.section .text    #代码段声明
# .global main
# main:
.global _start     #指定入口函数

_start:                                    #函数在屏幕上输出hello world!
movl $len, %edx                #第三个参数: 字符串长度
movl $msg, %ecx              #第二个参数: hello world!字符串
movl $1, %ebx                  #第一个参数: 输出文件描述符
movl $4, %eax                  #系统调用号sys_write
int $0x80                            #调用内核功能

#下面为退出程序代码
movl $0, %ebx                 #第一个参数: 退出返回码
movl $1, %eax                 #系统调用sys_exit
int $0x80                          #调用内核功能
```

Generated assembly code:



```
00000000004000b0 <_start>:
  4000b0:       ba 0d 00 00 00          mov    $0xd,%edx
  4000b5:       b9 d2 00 60 00          mov    $0x6000d2,%ecx
  4000ba:       bb 01 00 00 00          mov    $0x1,%ebx
  4000bf:       b8 04 00 00 00          mov    $0x4,%eax
  4000c4:       cd 80                   int    $0x80
  4000c6:       bb 00 00 00 00          mov    $0x0,%ebx
  4000cb:       b8 01 00 00 00          mov    $0x1,%eax
  4000d0:       cd 80                   int    $0x80
```

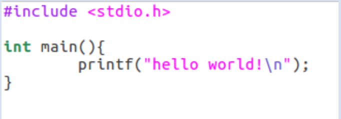
At 0x4000C4, there is a system call, using the program to trace, we get that:



```
phantom0308@phantom0308-VirtualBox:~/桌面/Os 2/asm$ ./trace_asm ./hello_asm
hello world!
System call write(system call num: 1  return value: -38) from pid 3386
Using registers to pass fd and length, rbx: 00000000, rdi: 00000000, rdx: 00000015
Using block to pass filename, address is saved in rcx: 0x006000D2
And the filename is: hello world!
```

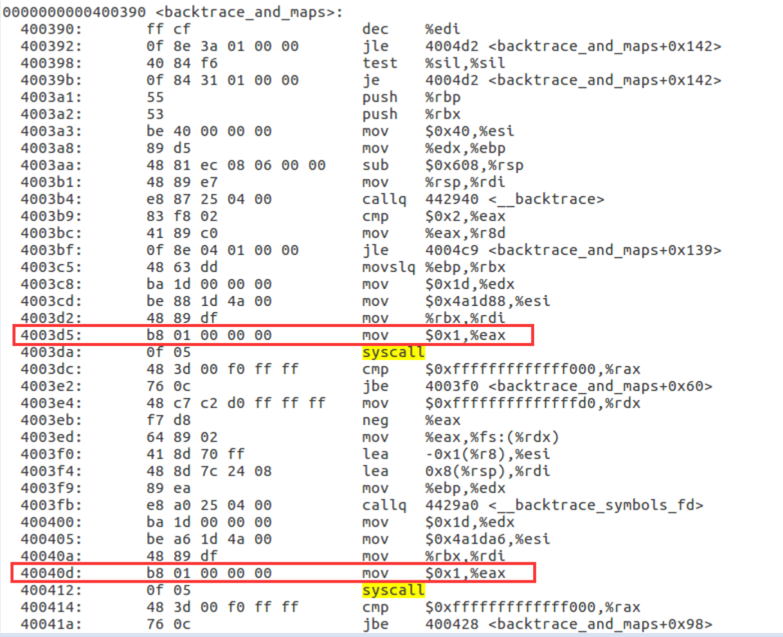
Only once sys_write().

And I code this same program in C:



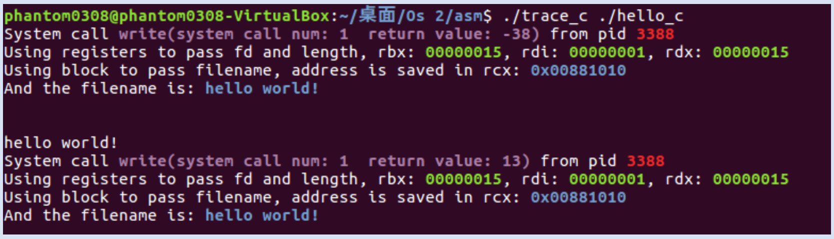
```c
#include <stdio.h>

int main(){
        printf("hello world!\n");
}
```

Generated assembly code:



```
0000000000400390 <backtrace_and_maps>:
  400390:       ff cf                   dec    %edi
  400392:       0f 8e 3a 01 00 00       jle    4004d2 <backtrace_and_maps+0x142>
  400398:       40 84 f6                test   %sil,%sil
  40039b:       0f 84 31 01 00 00       je     4004d2 <backtrace_and_maps+0x142>
  4003a1:       55                      push   %rbp
  4003a2:       53                      push   %rbx
  4003a3:       be 40 00 00 00          mov    $0x40,%esi
  4003a8:       89 d5                   mov    %edx,%ebp
  4003aa:       48 81 ec 08 06 00 00    sub    $0x608,%rsp
  4003b1:       48 89 e7                mov    %rsp,%rdi
  4003b4:       e8 87 25 04 00          callq  442940 <__backtrace>
  4003b9:       83 f8 02                cmp    $0x2,%eax
  4003bc:       41 89 c0                mov    %eax,%r8d
  4003bf:       0f 8e 04 01 00 00       jle    4004c9 <backtrace_and_maps+0x139>
  4003c5:       48 63 dd                movslq %ebp,%rbx
  4003c8:       ba 1d 00 00 00          mov    $0x1d,%edx
  4003cd:       be 88 1d 4a 00          mov    $0x4a1d88,%esi
  4003d2:       48 89 df                mov    %rbx,%rdi
  4003d5:       b8 01 00 00 00          mov    $0x1,%eax
  4003da:       0f 05                   syscall
  4003dc:       48 3d 00 f0 ff ff       cmp    $0xfffffffffffff000,%rax
  4003e2:       76 0c                   jbe    4003f0 <backtrace_and_maps+0x60>
  4003e4:       48 c7 c2 d0 ff ff ff    mov    $0xffffffffffffffd0,%rdx
  4003eb:       f7 d8                   neg    %eax
  4003ed:       64 89 02                mov    %eax,%fs:(%rdx)
  4003f0:       41 8d 70 ff             lea    -0x1(%r8),%esi
  4003f4:       48 8d 7c 24 08          lea    0x8(%rsp),%rdi
  4003f9:       89 ea                   mov    %ebp,%edx
  4003fb:       e8 a0 25 04 00          callq  4429a0 <__backtrace_symbols_fd>
  400400:       ba 1d 00 00 00          mov    $0x1d,%edx
  400405:       be a6 1d 4a 00          mov    $0x4a1da6,%esi
  40040a:       48 89 df                mov    %rbx,%rdi
  40040d:       b8 01 00 00 00          mov    $0x1,%eax
  400412:       0f 05                   syscall
  400414:       48 3d 00 f0 ff ff       cmp    $0xfffffffffffff000,%rax
  40041a:       76 0c                   jbe    400428 <backtrace_and_maps+0x98>
```

There are two calls, and the result also proves this point:



```
phantom0308@phantom0308-VirtualBox:~/桌面/Os 2/asm$ ./trace_c ./hello_c
System call write(system call num: 1  return value: -38) from pid 3388
Using registers to pass fd and length, rbx: 00000015, rdi: 00000001, rdx: 00000015
Using block to pass filename, address is saved in rcx: 0x00881010
And the filename is: hello world!


hello world!
System call write(system call num: 1  return value: 13) from pid 3388
Using registers to pass fd and length, rbx: 00000015, rdi: 00000001, rdx: 00000015
Using block to pass filename, address is saved in rcx: 0x00881010
And the filename is: hello world!
```

Acknowledgment

My deepest gratitude to TA Jiaqi Li.

Here is my ptrace.c:

链接：https://pan.baidu.com/s/1CZHR5Ney0NZCqOtar2D0-A 密码：lqai

```c
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/user.h>
#include <syscall.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#if __WORDSIZE == 64
    #define AX(reg) reg.orig_rax
#else
    #define AX(reg) reg.orig_eax
#endif

int ltoa(char** index, long data) { //convert long to string
    char* little = (char*)&data;    //in order to get each byte
    int flag = 0;
    for (int i = 0; i < 4; i++) {
        if (little[i] == 0){
            flag = 1;
            **index = '\n';
            break;
        }
        **index = little[i];
        (*index)++;
    }
    return flag;
}



int main(int argc, char* argv[]) {
    pid_t child;
    if (argc == 1)
        exit(0);
    char* chargs[argc];
    int i = 0;
    while (i < argc - 1) {
        chargs[i] = argv[i+1];
        i++;
    }
    chargs[i] = NULL;
    child = fork();
    if(child == 0) {
        ptrace(PTRACE_TRACEME, 0, NULL, NULL);
        execvp(chargs[0], chargs);
    }
    else {
        int status;
        while(waitpid(child, &status, 0) && ! WIFEXITED(status)) {
            struct user_regs_struct regs;
            ptrace(PTRACE_GETREGS, child, NULL, regs);
            if((size_t) (AX(regs)) == SYS_open){
                //print systemcalll num & return value
                fprintf(stderr, "System call \e[35;1mopen(system call num: %zd  return value: %lld)\e[0m from pid \e[31;1m%d\e[0m\n", (size_t) (AX(regs)), regs.rax, child);
                //print data in rsi & rdx
                printf("Using registers to pass flags and mode, rsi: \e[32;1m%08o\e[0m, rdx: \e[32;1m%08o\e[0m\n", regs.rsi, regs.rdx);
                char addr[10];
                //print the adress in rdi
                sprintf(addr, "0x%08X\n", regs.rdi);
                printf("Using block to pass filename, address is saved in rdi: \e[34;1m%s\e[0m", addr);
                //print the content in the adress
                long data = ptrace(PTRACE_PEEKDATA, child, regs.rdi, NULL);
                char buf[100];
                memset(buf, 0, 100);
                char* index = buf;
                int i = 1;
                while(!ltoa(&index, data)){
                    data = ptrace(PTRACE_PEEKDATA, child, regs.rdi+4*i, NULL);
                    i++;
                }

                printf("And the filename is: \e[34;1m%s\e[0m\n",buf);

            }
            ptrace(PTRACE_SYSCALL, child, NULL, NULL);
          }
    }
    return 0;
}
```