# HW2

# 赵洁-3160102525

## 1. Operating System Concept Chapter 2 Exercises:

### 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.

> *One set of services provided by an operating system is for ensuring the efficient operation of the system by managing the sharing of the computer resources among the different processes.*
> *Another set of services is to provide new functionality that is not supported directly by the underlying hardware, but help the users to better operate.*

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

> *Register, block, stack. a. Pass parameters in registers*
> *b. Registers pass starting addresses of blocks of parameters*
> *c. Parameters can be placed, or pushed, onto the stack by the program, and 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?

> *Each device can be accessed as though it was a file in the filesystem. Since most of the kernel deals with devices through this file interface,it is relatively easy to add a new device driver by implementing the hardware-specific code to support this abstract file interface. Therefore,this benefits the development of both user program code, which can be written to access devices and files in the same manner, and device driver code, which can be written to support a well-defined API.*
> *The disadvantage with using the same interface is that it might be difficult to capture the functionality of certain devices within the context of the file access API, thereby either resulting in a loss of functionality or a loss of performance. Some of this could be overcome by the use of ioctl operation that provides a general purpose interface for processes to invoke operations on devices.*

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

> *The separation of policy and mechanism is important for flexibility. Policies are likely to change across places or over time. In the worst case, each change in policy would require a change in the underlying mechanism. A general mechanism flexible enough to work across a range of policies is preferable. A change in policy would then require redefinition of only certain parameters of the system.*

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

“

*running result:*

```
root@DESKTOP-5N08FI4:/home/h2# ./p1
hell world (pid: 73)
hello, I am parent of 74 (pid: 73)
hello, I am child (pid: 74)
```

```
root@DESKTOP-5N08FI4:/home/h2# ./p2
hello world(pid:75)
hello,I am child (pid:76)
hello, I am parent of 76 (wc: 76)(pid: 75)
```

```
root@DESKTOP-5N08FI4:/home/h2# ./p3
hello world (pid:77)
hello, I am child (pid:78)
 29  52 610 p3.c
hello, I am parent of 78 (wc:78)(pid:77)
```

```
root@DESKTOP-5N08FI4:/home/h2# ./p4
root@DESKTOP-5N08FI4:/home/h2# cat p4.output
 30  44 551 p4.c
```

## 3. Expand the ptrace sample code used in the class to display the pathname parameters

“

*Test the program:*

```
root@DESKTOP-5N08FI4:/home/h2# ./ptrace date +%Y-%m-%d
open() opened:/etc/ld.so.cache
open() opened:/etc/ld.so.cache
open() opened:/lib/x86_64-linux-gnu/libc.so.6
open() opened:/lib/x86_64-linux-gnu/libc.so.6
open() opened:/usr/lib/locale/locale-archive
open() opened:/usr/lib/locale/locale-archive
open() opened:/etc/localtime
open() opened:/etc/localtime
2018-10-10
root@DESKTOP-5N08FI4:/home/h2#
```

”

“

*Source code:ptrace.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<string.h>
#include<stdlib.h>

#if __WORDSIZE==64
#define REG(reg) reg.orig_rax
#define WORDLENGTH 8
#else
#define WORDLENGTH 4
#define REG(reg) reg.orig_eax
#endif

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 reg;
            ptrace(PTRACE_GETREGS,child,NULL,&reg);
            if(REG(reg)==SYS_open){
                char file[255];
                union{
                    long word;
                    char str[WORDLENGTH];
                }data;
                long offset=0;
                int done=0,i;
                file[0]='\0';
                while(!done){
                    data.word=ptrace(PTRACE_PEEKDATA,child,reg.rdi+offset,NULL);
                    strncat(file,data.str,WORDLENGTH);
                    for(i=0;i<WORDLENGTH;i++)
```

```
                        if(data.str[i]=='\0')
                            done=1;
                        offset+=WORDLENGTH;
                    }
                    printf("open() opened:%s\n",file);
                }
                ptrace(PTRACE_SYSCALL,child,NULL,NULL);
            }
        }
        return 0;
}
```