

# Homework 2: Command Line Interface, System Calls, and System Utilities

## Objectives

At the end of this activity, you should be able to:

1. add a console command;
2. add a new system call service/function; and
3. invoke a system call from a system utility.

## 1 Introduction

The command line interface (CLI) is one way an operating system allows users to access its services such as program execution. The user enters a command string on the CLI prompt and the OS executes the command. The execution usually involves *system calls* which invoke some services provided by the kernel. These services are functions that execute in *kernel mode*. The CLI can be implemented as part of the kernel itself, as in the case of ICS-OS, or as a separate program, called a *shell*, which is outside of the kernel. Since shells (and other system utilities and user applications as well) are not in the kernel but may require system calls, they do so using software interrupts (Linux uses `int 80h`, DOS uses `int 21h`, and ICS-OS uses `int 30h`). Application Programming Interfaces (APIs), Software Development Kit (SDKs), and Runtime environments make it easy to write programs for operating systems by hiding the details of the system calls to the programmers.

## 2 Prerequisites

You should have completed Homework 1. Take note that the *top-level directory* of the source tree contains the following.

```
extras  ics-os  README.md  SEMESTER  wiki
```

However, most of the commands will be described relative to the `ics-os` directory. For consistency, we will refer to this folder as `$ICSOS_HOME`. Use the `cd` command to transfer from one directory to another. Several updates have been made in the remote repository. Update your local copy of the source using the following commands.

```
$cd ics-os
$git pull
```

## 3 Deliverables and Credit

Perform the tasks below and capture screen shots while you do them. Answer all questions. Submit a PDF file containing the screen shots with captions and answers to questions. Do not forget to put your name and laboratory section. Credit is ten (10) points.

## 4 Tasks

### Task 1: Add a new console command (3 points)

The CLI in ICS-OS is part of the kernel. Its implementation is located in `kernel/console/console.c`. The function `int console_execute(const char *str)` is where the command string (what you type in the % prompt) is processed. Study this function. The `strtok()` function is used to tokenize the command string to extract the command name and its arguments. The code fragment below is for the new `add` command with two integer arguments that we wish to include. Insert the code fragment in an appropriate location in the `console_execute()` function. Build and boot ICS-OS to test if the command works. Capture screen shots. Also show where you placed the code fragment.

**QUESTION:** What are the advantages/disadvantages of having the CLI in the kernel?

```
if (strcmp(u,"add") == 0){  //-- Adds two integers. Args: <num1> <num2>
    int a, b;
    u = strtok(0," ");
    a = atoi(u);
    u = strtok(0," ");
    b = atoi(u);
    printf("%d + %d = %d\n",a,b,a+b);
}else
```

### Task 2: Add a new system call service/function (3 points)

The list of functions/services accessible through system calls are placed in a *system call table*. In ICS-OS, it is the array of structures defined in `kernel/dexapi/dex32API.h`:

```
api_systemcall api_syscalltable[API_MAXSYSCALLS];
```

The function `api_init()` in `kernel/dexapi/dex32API.c` populates this table.

ICS-OS hooks to `int 30h` to handle system calls. This is set in `kernel/hardware/chips/irqhandlers.c`. Recall the interrupt system and the interrupt vector table (IVT) discussed in the lecture.

```
setinterruptvector(0x30, dex_idtbase, 0xEE, syscallwrapper, SYS_CODE_SEL);
```

You do not have to understand all the parameters of the above function for now. The parameter `syscallwrapper` is a function defined in `kernel/irqwrap.asm`. It calls the function `api_syscall(...)` from `dex32API.c` which processes the system call and invokes the appropriate service from the system call table. `api_syscall(...)` is called everytime `int 30h` is generated.

To add a system call, the `api_addsystemcall()` function is used. Its prototype is shown below.

```
int api_addsystemcall(DWORD function_number, void *function_ptr,
                     DWORD access_check, DWORD flags);
```

The important parameters are `function_number` and `function_ptr`. Say you want to implement the `kchown()`<sup>1</sup> system call function/service below that changes the owner of a file. Edit `dex32API.c` and add the function.

<sup>1</sup>Usually functions in the kernel are written with the letter `k` at the start

```
int kchown(int fd, int uid, int gid){
    printf("Changing owner of fd=%d to user id=%d and group id=%d\n", fd, uid, gid);
    //Actual code to change file ownership is placed here.
    return 0; //0-success
}
```

To add it to the system call table, add the following line in the `api_init()` function. The function number you will use is `9Fh`. Take note of this number.

```
api_addsystemcall(0x9F, kchown, 0, 0);
```

Capture screenshots where you placed the codes. Build ICS-OS. At this point, the new system call is added to the kernel but it is not doing anything yet. You will do that in Task 3.

### Task 3: Invoke a system call in a system utility (4 points)

In this task you are to make a system utility that invokes the system call service you created in Task 2. In ICS-OS, system utilities and user applications are placed in the `contrib` folder. There is an example application, `hello`, which you will use as template. Study the `Makefile`. Run the commands below to create your `chown` system utility.

```
$ cd contrib          #go to the contrib folder
$ cp -r hello/ chown/ #copy hello to chown
$ cd chown/           #go inside chown
$ mv hello.c chown.c  #rename hello.c to chown .c
$ sed -i 's/hello/chown/g' Makefile #replace hello with chown in the Makefile
$ make
$ make install
```

Go back to `$ICSOS_HOME`. Build and boot ICS-OS. Inside ICS-OS, run the following commands and capture screenshots.

```
% cd apps
% ls -l -oname
% chown.exe
```

**QUESTION:** What is the output of executing `chown.exe` inside ICS-OS?

Go back to the `contrib/chown` folder. Edit `chown.c` and replace the contents with the code below.

```
#include "../..//sdk/dexsdk.h"
int main(int argc, char *argv[]) {
    if (argc < 4){
        printf("Usage: chown.exe <fd> <uid> <gid>\n");
        return -1;
    }
    dexsdk_systemcall(0x9F, atoi(argv[1]), atoi(argv[2]),
                     atoi(argv[3]), 0, 0);
    return 0;
}
```

**QUESTION:** Study the function `dexsdk_systemcall()` defined in `sdk/tccsdk.c`. What does this function do?

Now build and install the `chown` system utility.

```
$ make
$ make install
```

Go back to `$ICSOS_HOME`. Build and boot ICS-OS. Inside ICS-OS, run the following commands and capture screenshots.

```
% cd apps
% ls -l -oname
% chown.exe
```

**QUESTION:** What is the output of executing `chown.exe` this time?

## 5 Tips

You can use the `grep` utility to quickly search for strings in files from `$ICSOS_HOME`.

```
$ grep -rn api_init
```