IIT BOMBAY

Operating Systems Lab
Assignment V

Ranveer Aggarwal (120050020)   Bhargav Chippada (120050053)

February 13, 2015

1 PART I

1. The source code file should be put in *<path to geekOS>/GeekOS/src/user*. To execute, we build GeekOS again (i.e., `make`), run `geek` and execute it by keying in the file name and pressing return in the GeekOS terminal. The files under the *include/geekos* and *include/libc* directories can be included in the user space.

2. a) *syscall.h* contains enums defining the various syscalls and also defines the `DEF_SYSCALL` macro. *syscall.c* contains the syscall methods. *conio.c* includes the required header files. Using `DEF_SYSCALL`, it defines wrapper methods for three syscalls, `SYS_PRINTSTRING`, `SYS_GETKEY` and `SYS_GETCURSOR`. Also, it defines some more methods and *conio.h* contains declaration of all the methods.

   b) **.h* contains the declarations of the methods implemented by **.c*. For a new system call, we have to add **SYS_<syscallName>** to the enums in *syscall.h* header file. For the new syscall, we will define its method inside *syscall.c* and add the method name to the `g_syscallTable` then, we'll define a wrapper source file for the new system call and also define the new header file for it. The wrapper source file has `DEF_SYSCALL` which attaches a syscall to a method name.

3. The number of parameters passed in a system call is given as the last parameter (`regs`) in the `DEF_SYSCALL` macro. In *syscall.h*, the macros for `regs` is defined as `SYSCALL_REGS_0-5`, and from there, we get the parameters for each call depending on which macro is used in the wrapper for that syscall.

4. The **pid** is stored in the `Kernel_Thread` struct defined in the file *kthread.h*. It is referenced through a pointer (used as `CURRENT_THREAD->pid`) in the *syscall.c* file in the method `Sys_GetPID`.

2 Part II: Using kernel facilities to accept input from the keyboard

For this part, we first modified the function `Read_line` as was defined in *syscall.c* and redefined it to `Read_line_New` in a user file *a5q2.c*. We changed the part where it terminated on getting newline as an input to it not terminating until it got as an input. Here's the modified code segment:

```c
if (k == '@') done = true;
if (n < bufSize)
{
  if (k != '@') *ptr++ = k;
  ++n;
}
```

The main function looked like:

```c
int main(){
   char inp[1000];
   Read_Line_New(inp, 1000);
   Print("\n");
   Print_String(inp);
   return 0;
}
```

3 Part III: Adding a new system call "Get_NewToD"

We added a new syscall called `SYS_NEWTOD` to the `Enums` in *syscall.h*. We defined its handler method called `Sys_NewTod` in *syscall.h* and added its name to `g_syscallTable` in the following way:

```c
static int Sys_NewTod(struct Interrupt_State *state) {
    int tod = g_numTicks;
    if (!Copy_To_User(state->ebx, &tod, sizeof(int)))
        return -1;
    return 0;
}
```

We then defined a wrapper function for system call `SYS_NEWTOD` inside *sched.c* using `DEF_SYSCALL` macro in the following way:

```
DEF_SYSCALL(Get_NewToD, SYS_NEWTOD, int, (int *value),int* arg0 = value;, ↩
    SYSCALL_REGS_1)
```

Then we declared the function `Get_NewTod` inside *sched.h*.
An example code(*q2.c*) would be:

```
#include <conio.h>
#include <sched.h>
int main(){
    int xyz;
    Get_NewToD(&xyz);
    Print("%d\n",xyz);
    return 0;
}
```

4 Part IV: Adding some system calls to collect information about execution of a process

Note: We have done part (a) of this question.
First we added a new integer variable called `syscallsCount` to the structure `Kernel_Thread`. Then, inside *trap.c* we added the following line to `Syscall_Handler` function:

```
CURRENT_THREAD->syscallsCount = CURRENT_THREAD->syscallsCount+1;
```

We then added a new syscall called `SYS_SYSCOUNT` to the `Enums` in *syscall.h*. We defined its handler method called `Sys_SysCount` in *syscall.h* and added its name to `g_syscallTable` in the following way:

```
static int Sys_SysCount(struct Interrupt_State *state __attribute__ ((unused))) {
    return CURRENT_THREAD->syscallsCount;
}
```

We then defined a wrapper function for system call `SYS_SYSCOUNT` inside *sched.c* using `DEF_SYSCALL` macro in the following way:

```
DEF_SYSCALL(Get_Syscalls_Count, SYS_SYSCOUNT, int, (void),, SYSCALL_REGS_0)
```

Then we declared the function `Get_SysCount` inside *sched.h*.

An example code(*q2.c*) would be:

```c
#include <conio.h>
#include <sched.h>
int main(){
    int xyz;
    Get_NewToD(&xyz);
    Print("Time of Day: %d\n",xyz);
    Get_NewToD(&xyz);
    Print("Time of Day: %d\n",xyz);
    int count;
    count = Get_Syscalls_Count();
    Print("Number of syscalls: %d\n",count);
    return 0;
}
```

We get **6** syscalls as our output.
Note: The modified code has been included with this document.