**Student Name: Demetrius Johnson**

**Course: CIS-450-002**

**Professor: Dr. Jinhua Guo**

**Date: February 11, 2022**

**Due Date: February 11, 2022**

**Project 2 Report**

# Task 1: using the *find* command

* Graphical user interface, text

  Description automatically generated
  + Above: I learned how to use the find command so that I can use that and the grep functions to effectively search for files and directories and text within files.
  + I have to type: find [file path to the file I want to search within, with the default being the current directory and all directories under that directory] [-name] [the name of my file or directory] [-type f == search for a file type; I can also specify directory type with -type d, and I can also not specify type and it will by default search for files and directories].

# Task 2: creating *date.c* file

* Text

  Description automatically generated
  + Above: I have created a file called date.c inside xv6 directory and implemented the initial code that we are required to have in the user program.
  + I note from P2 discussion that the first parameter in the print function has integer value (1), and it tells the function to print/send output to the standard output.
  + If our system call function that we are creating – date() – fails it will return 0, indicating that the system call failed; then for the print function we use integer (2) for the first parameter (output parameter) in order to send error messages to cerr/stderr (standard error); note that by default stdoutput and stderr both stream output to the console window.
  + For xv6 and many other operating systems: 0 = kernel mode, 3 = user mode, and the levels in between have mixed privileges and are often not used.

# Task 3: using the *grep* command

* Text

  Description automatically generated
  + Above, calling the grep command, searching for the string “uptime” in any file with either .c, .h, or .S file extension in their file name (assembly language file extension types).
  + The actual implementation of my system code for my new system call function date() should be in the system process c file: sysproc.c

# Task 4: add SYSCALL(date) to usys.S file

* A screenshot of a computer

  Description automatically generated
* Text

  Description automatically generated
  + Above we look inside the usys.S (user system call file) assembly language file; SYSCALL is a macro function which is defined in our usys.S function; see examples the expansion of a few of the macro function implemented with specific names below after we compile usys.S code:
    - Text

      Description automatically generated
    - Notice the identification integer value for write, close, and kill being passed into SYSCALL macro function is 16, 21, and 6, respectively. We will add an identifier to our date function system call in another file (syscall.h), and assign it a unique integer value.
  + movl simply moves a value in the eax register to do a specific system call based on the value placed into the register; T\_SYSCALL is an integer constant (64) 🡪 int in assembly language is not “integer” but “interrupt”; so we do a system interrupt 64 to interrupt the system and call the given system call placed into the eax register.
  + In xv6 this is how we implement a system call: we do an interrupt; we know that the interrupt is normally a hardware interrupt – (such as when you have a disk operation complete) hardware will generate an interrupt; they are between 32 and 63. 0 to 31 are the software interrupts (such as divide by 0 error or segmentation fork; voluntarily give up CPU); 64 is the system call interrupt (also known as trap) and is also technically a software interrupt. The system interrupt such as syscall allows us to get into kernel mode (and implement the system call code; such as the uptime function, or the new date system call function we are implementing).
  + Finally, we add the date system call that we are implementing in this project.

# Task 5: compile usys.S to view all the macro functions generated

* Text

  Description automatically generated
* Text

  Description automatically generated
  + Above, using gcc compiler with option S to compile the usys.S assembly language file, showing the code defines a macro function from the defined macro template for each particular system call instance.
  + As an example: for the uptime syscall function, $14 is the specific number used to identify a specific system call such as uptime and is placed into the eax register; then int $64 is the SYSCALL interrupt that is generated to capture the trapped value and allow uptime to get into kernel mode.
    - Note to self: $ followed by a number means decimal integer. The movl means “move long”; as in an integer of size long.
    - The eax register not only stored the user system call value, but it also stores the return value of the system call (0 = success, 1 or -1 = fail, usually in this format since integer value 1 is used for standard output, and -1 is used for stderr output.)

# Task 6: add declaration for the date() function in the user.h file

* Text

  Description automatically generated
  + Above, adding the declaration for the date() function in the user.h header file, so that the signature exists for any function that calls date() during compilation; otherwise compiler will complain there is no existing function defined called date(); of course definition will be done in another file. So this is basically a function signature interfaced in the system call with the user applications.

# Task 7: Define another integer, named SYS\_date in the syscall.h file

* Text

  Description automatically generated
  + Above, we enter the syscall.h assembly language header file. We define another integer, named SYS\_date, and assign it an unused syscall identification number; for this case, we assign SYS\_date the integer value 22 (decimal by default; remember I could write 22d to specify decimal).
  + Now, SYSCALL macro function from the usys.S file will know what to replace the defined but unassigned identifying integer SYS\_ ## name (name = date, ## = 22):
    - A screenshot of a computer

      Description automatically generated
    - So now the template macro function will look like this:
    - .globl date; \
    - movl $SYS\_date, %eax; \
    - int $T\_SYSCALL; \
    - Ret
    - ;so for the above code, int = interrupt in assembly, T\_SYSCALL = integer value 64, SYS\_date = 22. Also note: .globl means this is a global function.
    - Here is a screenshot from how it looks after compilation, as shown earlier in tasks 4 and 5:
    - Text

      Description automatically generated

# Task 8: viewing the cmostime() function in the lapic.c file

* Text

  Description automatically generated
* Text

  Description automatically generated
  + In the above two screenshots: showing the cmostime() function from the lapic.c xv6 system file. This is the function that actually does the work/implementation of acquiring the system run time clock value and converting it to integer format.

# Task 9: