**Homework #2 Xv6**

Course

**CSE 460 Operating Systems**

Instructor

**Dr. Yan Zhang**

Meeting Time

**Mon. & Wed., 4:00 p.m. - 5:15 p.m.**

Due Date

**March 11, 2020**

Authors

**Kevin T. Vo**

**Esdras Lopez**

**Joseph Gonzales**

**Trevor Shortlidge**

**Brian Ayala**

# Group Members’ Information

|  |  |  |
| --- | --- | --- |
| **Name** | **Coyote ID** | **Responsibility** |
| Kevin T. Vo | 006316930 | * Implemented date system call & command (Part a) into xv6 * Implement part of uid, gid, ppid (Part c) * Contributed to this report/documentation |
| Esdras Lopez | 006198864 | * Implemented part of uid, gid, ppid (Part c) * Contributed to this report/documentation |
| Joseph Gonzales | 006242648 | * Implemented ps command and Ctrl+P (Part d) * Contributed to this report/documentation |
| Trevor Shortlidge | 006310209 | * Implemented part of Ctrl+P (Part b) * Reformatted output of ps command (Part d) * Contributed to this report/documentation |
| Brian Ayala | 006191688 | * Implemented part of Ctrl+P (Part b) * Contributed to this report/documentation |

# Table of Contents

1. Demonstration of the date() System Call ……………………………………………4
2. Demonstration of the Ctrl-P Special Control Sequence………………………….......6
3. Demonstration of UIDs, GIDs, and PPIDs………………………………………......7
4. Demonstration of the “ps” Command……………………………………………….13

# **Demonstration of the date() System Call**

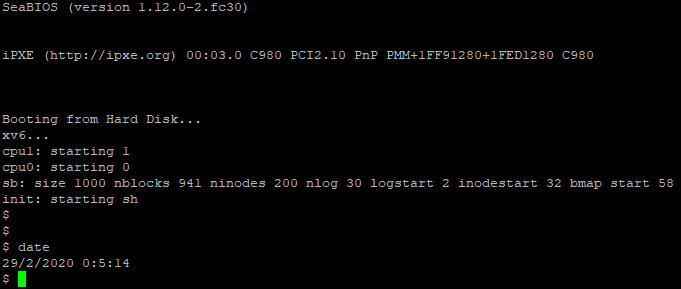
The system call “date” has been implemented to the Xv6 operating system where it would display the current day, month, year, hour, minute, and second in the format below:

*DAY/MONTH/YEAR HOUR:MINUTE:SECOND*

During the addition of this system call the following files have been created/modified:

|  |  |  |  |
| --- | --- | --- | --- |
| **Filename** | **Created**  **/ Modified** | **Line Numbers** | **Code** |
| Makefile |  |  |  |
| syscall.h | Modified | 23  &  248 |  |
| user.h | Modified | 26 |  |
| sysproc.c | Modified | 95-106 |  |
| usys.S | Modified | 32 |  |
| syscall.c | Modified | 106  &  136 |  |
| date.c | Created | 1-18 |  |

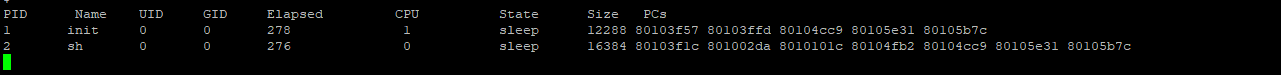
### Result from Execution of date() System Call:



Results from the syscall “date” comes from implementing the above into a full fledge syscall. Modified syscall.h that contains the mapping of the call number linked to SYS\_date. Modified defs.h to hold the kernerl wide function call for date. Modified user.h that contains the functions prototype for date that is required to run user programs. Implemented the function for date inside of sysproc.c by getting the arguments off the stack. This implementation was provided to us. Next we modified the usys.S that exports the system call. Modified syscall.c to add the extern int for being in another file. Then the last step is to add the date.c file where we invoke the syscall implementation.

# **Demonstration of the Ctrl-P Special Control Sequence**

Result for Ctrl-P Special Control Sequence:



Note: UID,GID, CPU are from part D, Dr.Zhang said it was ok to leave elapsed time in milliseconds.

Proc.h line 56 -> added uint start\_ticks to structure proc



Modified procdump in proc.c to display our results in the ptable to output elapsed time, & size

Elapsed time result is done by subtracting ticks minus start\_ticks do give us the delta in milliseconds.

However, changing it to a float and dividing by 1000 to give us the result in seconds was not working with cprintf. Refer to to the comment I made before about Dr.Zhang letting our group use milliseconds instead.

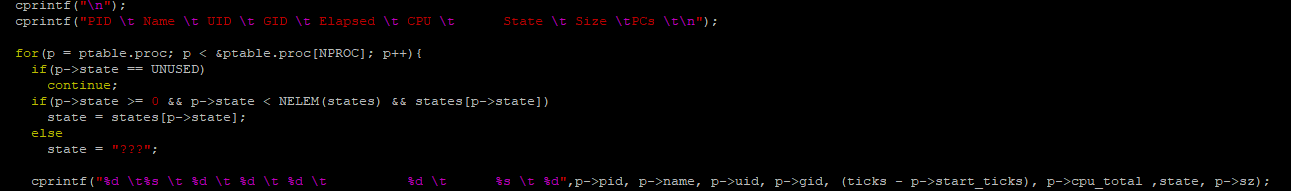
Also p->sz is a built in variable in proc that deals with the size of each block of memory.

We just outputted the size : p->sz to display.

Proc.c line 131 -> initialized start\_ticks to ticks that is a global counter in milliseconds.



proc.c procdump() - line 540



1. **Demonstration of** **UIDs, GIDs, and PPIDs**

The implements the feature of users and groups into xv6 through uid and gid where it be used to store ID unsigned integers for tracking the ownership of a process for a user or group. By typing “id” into the terminal when executing xv6, the system call will invoke and demonstrate this feature by displaying the UID, GID and PPID that has been established by the test function within “id.c”.

|  |  |  |  |
| --- | --- | --- | --- |
| **Filename** | **Created**  **/ Modified** | **Line Numbers** | **Image** |
| Makefile |  | 178  &  248 |  |
| syscall.h | Modified | 25-29 |  |
| user.h | Modified | 28-32 |  |
| proc.c | Modified | 597-691 |  |
| sysproc.c | Modified | 114-162 |  |
| usys.S | Modified | 34-38 |  |
| syscall.c | Modified | 108-112  &  138-142 |  |
| id.c | Created | 1-26 |  |

*Description of changes made within the file “syscall.h”:*

* The system calls responsible for setting and returning the UID, GID, and returning PPID are added so that it system call name can be mapped to a system call number so that it can be invoked within xv6.

*Description of changes within the file “user.h”:*

* The prototype to the getter and setter functions for setting and returning the unsigned integers of UID and GID including one getter function for the PPID.
* The prototypes included are to the functions that holds the code to perform the task that a user uses a system call to request for.

*Description of changes within the file “proc.c”:*

* proc.c contains the functions:

uint getuid (void) // UID of the current process

uint getgid (void) // GID of the current process

uint getppid ( void) // process ID of the parent process

int setuid (uint) // set UID

int setgid (uint) // set GID

to actually sets the UID and GID using the setuid(uint) and setgid(uint) functions.

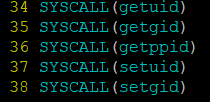
Note that user/xv6 will use these two functions to set the UID and GID. The getter functions uint getuid(void), uint getgid(void), and uint getppid(void) that returns the UID, GID, and PPID respectively. It is within setter functions where it checks if the value UID and GID are within range of [0, 32727].

*Description of changes within the file “sysproc.c”:*

* Included the sys functions that returns the actual functions listed above to perform the process associated its system call.

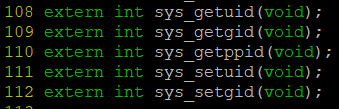
*Description of changes within the file “usys.S”:*

* “usys.S” contains the system calls to be made available by the kernel:



*Description of changes within the file “syscall.c”:*

* First contains the system call entry points:

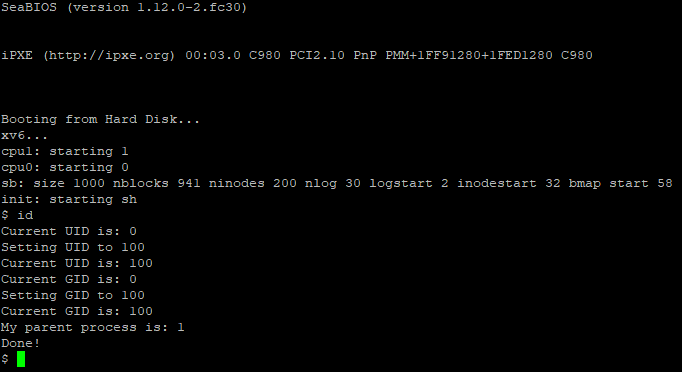


* Secondly, “syscall.c” contains a dispatch table that defines from the symbol name made earlier within files “usys.S” and “syscall.h” to its function name. All of this is done within the “int (\*syscalls[])(void)” within the “syscall.c”.

*Description of changes made within file “id.c”:*

* Contains the test function to run the system calls implemented in this part.

### Result from UIDs, GIDs, and PPIDs:



The following results derived from the test function within “id.c” for testing the uint getuid(void), uint getgid(void), uint getppid(void), int setuid(uint), and int setgid(uint) functions along with the system call to them. The setuid(uint) and setgid(uint) takes in an unsigned integer so assigns them the UID or GID for a process. The getters uint getuid(void) and uint getgid(void) returns value of the UID and GID that has been set. Lastly, uint getppid(void) returns back the parent process by first calling the sti() to interrupt the processor. Then, it acquires the p table which contains all the process. However, this function will only focus on the parent process with uint temp = p->pid; and then returns it. If the p->pid happens to be a negative value, then temp = 0.

1. **Demonstration of the “ps” Command**

The system call “cps” has been implemented to the Xv6 operating system in order to display the ptable for the current processes, through the use of the “ps” command. It outputs the following process information: pid, name, UID, GID, PPID, ELAPSED time (in ms), CPU time (in ms), size, and state.

During the addition of this system call the following files have been created/modified:

|  |  |  |  |
| --- | --- | --- | --- |
| **Filename** | **Created**  **/ Modified** | **Line Numbers** | **Image** |
| Makefile | Modified | 177  &  248 |  |
| ps.c | Created | 1-12 |  |
| syscall.h | Modified | 24 |  |
| defs.h | Modified | 123 |  |
| syscall.c | Modified | 107  &  137 |  |
| user.h | Modified | 27 |  |
| sysproc.c | Modified | 108-112 |  |
| usys.S | Modified | 33 | : |
| proc.c | Modified | 707-740 | A screenshot of a cell phone  Description automatically generated |

The total CPU time was completed as follows:

proc.h lines 56 & 57 -> added uint cpu\_total and uint cpu\_runtime to the proc structure.



proc.c line 132 -> cpu\_total initialized to 0.



proc.c line 376 -> cpu\_runtime set to ticks.



proc.c line 413 -> cpu\_total calculated by adding (ticks minus the cpu\_runtime) to give us the total CPU time in milliseconds.



### Result from Execution of cps() System Call:

****

Results from the system call “cps” comes from implementing it, as well as the ps command, into Xv6. The “ps” command was created by creating the ps.c file and modifying the Makefile, in order to add the command to Xv6. Modification of syscall.h was done, mapping the call number linked to SYS\_cps. Additionally, defs.h was modified to hold the function call for cps. Next, syscall.c was modified in order to include the extern int of sys\_cps. A function prototype for cps was also added to user.h, used by user programs. Next, sysproc.c was modified to implement the function for cps. Exporting the system call was done by modifying usys.S to include the cps system call. Finally, the implementation for cps was added to proc.c, allowing the various process information to be output in a ptable.