# Mini-Project 1 (System Calls) Building a Light System Call Part A, B, and C

****

**Rubric for each part**

Report 30%

Presentation 20 %

Code and documentation 50%

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Participants name-Part A** | **Code Section** | **Report Section** | **Documentation Sec** | **Presentation Sec** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Participants name-Part B** | **Code Section** | **Report Section** | **Documentation Sec** | **Presentation Sec** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Participants name-Part c** | **Code Section** | **Report Section** | **Documentation Sec** | **Presentation Sec** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Submission deadline: Tuesday, Sep 11 at 11:59 pm Presentation deadline: Tuesday, Sep 11 at class time.

## NOTE: NO PRESENTATION NO GRADE (Every member of the team should be present to receive a grade, otherwise that member will get zero)

**Every student submits a report and xv6 (I know a lot of redundancy)**

Deliveries:

* Zip file with XV6 already modified.

Report: Provide a description explaining the reason for the kernel modifications and how they were made. Include screenshots of the modifications and screenshots of the output.

# General description

In chapter 6 (Direct Execution) we talked about system calls, and how the OS provides this mechanism for allowing process in user mode to execute privileged instructions.

### PART A:

In this part your team will add a simple system call to xv6. Your new syscall should look like this: int FirstPart(void) Your system call returns the value of a counter (perhaps call it int *partAcount* or something like that) which is incremented every time any process calls the getpid() system call. That's it!

### PART B:

Here, we want **to count the total number of systems calls** that has been executed by you or by the system so far. For instance, when you start the operating system (XV6) several system calls take place, like the init() system call, when you type >>ls you execute *ls* and the system execute other associated system calls.

Here, you will create a system call, name it SecondPart(void) This system call will print the accumulated number of systems calls already executed in the system.

Here you need a counter *partBcount* which will be incremented by the SecondPart system call before any system call is issued. Remember, This count should incremented before a system call is issued, not after.

### PART C:

Similar to SecondPart(void) system call the ThirdPart(void) system call will return the number of system calls that have completed successfully (i.e., did not have a return value of -1).

Here, the counter *partCcount* will be incremented after the system call has returned, if and only if the system call did not return a -1 (this counter is returned by ThirdPart(void)).

# Hint (I will be giving a practical intro to sycall in XV6)

You can start by taking a look and using another system call, like **getpid()** to get an idea of how the code works . Most of the time will be spent on understanding the code. There shouldn't be a whole lot of code added you can use any editor and gcc as a compiler.

Check this Lab tutorial for C programming in three easy pieces website: [http://pages.cs.wisc.edu/~remzi/OSTEP/lab-tutorial.pdf](http://pages.cs.wisc.edu/%7Eremzi/OSTEP/lab-tutorial.pdf)

It will be of a lot of benefit to learn how to use gdb (the debugger) it may be helpful in understanding code, doing code traces, and is helpful for later projects too. Get familiar with this fine tool!

# Details

### Setting up the environment (Installing XV6 if you haven’t following the next set of instructions).

Download XV6 from Canvas Modules->Resources-XV6

Unzip the file

tar xvzf FILE-NAME.tgz

Modify file Makefile line 56: QEMU:=qemu-system-i386 or qemu-system-x86\_64 (accordingly)

//compile the kernel

>>make

if the system doesn’t find gcc (get it)

>>sudo apt update

>>sudo apt install build-essential

Try again

>>make

if the system doesn’t find qemu

>>sudo apt install qemu

Run qemu by running the following instruction

make qemu-nox

Just for this time, these are the files you will have to modify

|  |  |
| --- | --- |
| **File** | **Description** |
| *include/syscall.h* | Choose an appropriate number for your new system call and add it to  syscall.h with a corresponding SYS\_ prefix  **The name of the program is is “seconds”**  **$ vim xv6\_patched/kernel/syscall.c**  **int counterB;**  **counterB++;**  **$ vim defs.h**  **extern int CounterB;**  **int\_third(void) { return counterB; }**  **Screenshot courtesy of Aidan:** |
| *kernel/sysproc.c* | This is the place where you create the prototypes for system calls (then the actual implementation in proc.c), or you can implement a complete small system call her. |
| *user/usys.S* | user-space function to call the new system call  **SYSCALL(seconds)** |
| *user/user.h* | user-space function to call the new system call  **int seconds(void);** |
| *kernel/sysfunc.h* | Register your system call there  **int sys\_seconds(void);**  **$ make** |
| *kernel/syscall.c (lastone)* | Add pointer to the new system call and then a prototype to the system call function  **[SYS\_seconds] sys\_seconds,** |
| *user/makefile.mk* | Add the tester program to list of user programs so we can run the test |
| *Run your program* | Run the user-program that calls the system call  **$ cp first.c mytester.c # Because then you don’t have to memorize the header files**  **$ vim mytester.c**  **printf(1, “The number of times I called getpid is: %d\n”, seconds());**  **$ make qemu-nox** |

* syscall.h
* syscall.c
* user.h
* usys.S
* sysproc.c
* Makefile
* sysfunc.h

**PART B,** you may need to modify defs.h depending of your implementation. You may also need to extern variables and functions

**Example**

extern int variable; extern in function(void);

Extern system calls in syscall.c and sysproc.c

Share variables between syscall.c and sysproc.c through defs.h

**Resources:**

1. Installing Linux Ubuntu 18 as a subsystem of Windows 10 https://[www.windowscentral.com/install-windows-subsystem-linux-windows-10](http://www.windowscentral.com/install-windows-subsystem-linux-windows-10)
2. Installing XV6 and intro to system calls

<https://www.youtube.com/watch?v=vR6z2QGcoo8>

1. Some Background about system call in XV6 [https://github.com/remzi-](https://github.com/remzi-arpacidusseau/ostep-projects/blob/master/initial-xv6/background.md) [arpacidusseau/ostep-projects/blob/master/initial-xv6/background.md](https://github.com/remzi-arpacidusseau/ostep-projects/blob/master/initial-xv6/background.md)
2. Check for the two videos in Canvas->Resources->Mini project 1

Check in class how to create the program that use the system call. Your driver program should looks somehow like that

## Project Testing

Here, some programs to test your project. You can add more, or modify them if you want.

## Testing Part A

#include "types.h" #include "stat.h" #include "user.h"

// running system call FirstPart() int

main(void)

{

printf(1, "I’m a process with ID:%d\n", getpid()); exit();

}

#include "types.h" #include "stat.h" #include "user.h"

int main(void)

{

printf(1, "There are %d getpid() system calls.\n", **FirstPart()**); exit();

}

## Testing Part B

#include "types.h" #include "stat.h" #include "user.h"

// running system call PartB() int

main(void)

{

printf(1, "There are %d system calls.\n", **SecondPar**t()); exit();

}

## Testing Part C

#include "types.h" #include "stat.h" #include "user.h"

// running system call PartC() int

main(void)

{

printf(1, "There are %d system calls.\n", **ThirdPar**t()); exit();

}