بسمه تعالی



دانشکده مهندسی کامپیوتر

خرداد **1400**

نام استاد: دکتر انتظاری

سیستم های عامل

فاز اول پروژه‌ی اضافه کردن فراخوان سیستمی به **xv6**

آرمان حیدری

شماره دانشجویی: **97521252**

**Table of Contents**

[What are system calls? 3](#_Toc75298202)

[overview 3](#_Toc75298203)

[Syscall.h 3](#_Toc75298204)

[syscall.c 4](#_Toc75298205)

[proc.h 6](#_Toc75298206)

[defs.h 6](#_Toc75298207)

[types.h 7](#_Toc75298208)

[proc.c 7](#_Toc75298209)

[sysproc.c 7](#_Toc75298210)

[usys.S 8](#_Toc75298211)

[user.h 9](#_Toc75298212)

[userTestProcDump.c 9](#_Toc75298213)

[Makefile 10](#_Toc75298214)

[Run example 12](#_Toc75298215)

[Resources 12](#_Toc75298216)

# What are system calls?

A **system call** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.

# overview

I want to add a system call named proc\_dump to xv6 operating system. First I cloned the repository as explained in the project document and then I edit these files to add this system call:

syscall.h

syscall.c

sysproc.c

usys.S

user.h

proc.h

defs.h

types.h

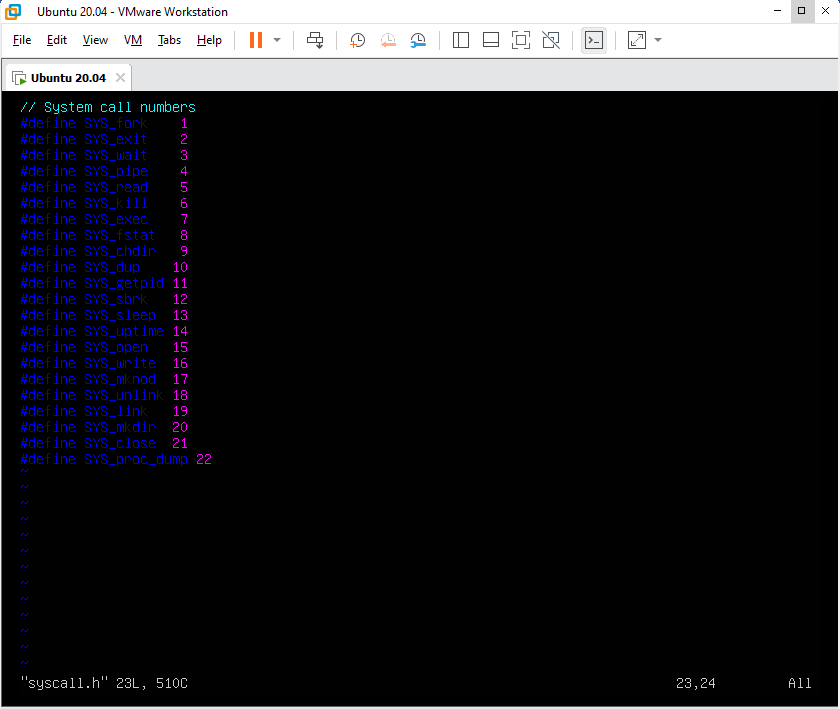
proc.c

And I create userTestProcDump.c to test the program, then add it to Makefile. I’ll show you step by step that what I have done in each of these files and why.

# Syscall.h

I started with this file  where number is assigned to every system call in this Xv6 system. As you can see, there are 21 system calls already defined in this file. Let’s go ahead and add following line to reserve system call number 22 for your own system call.

#define SYS\_proc\_dump 22



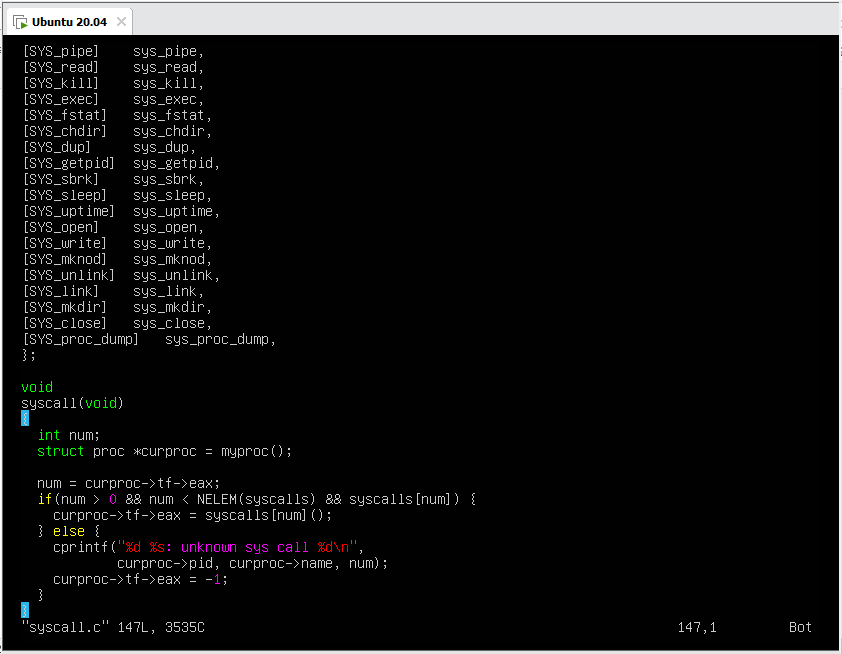
# syscall.c

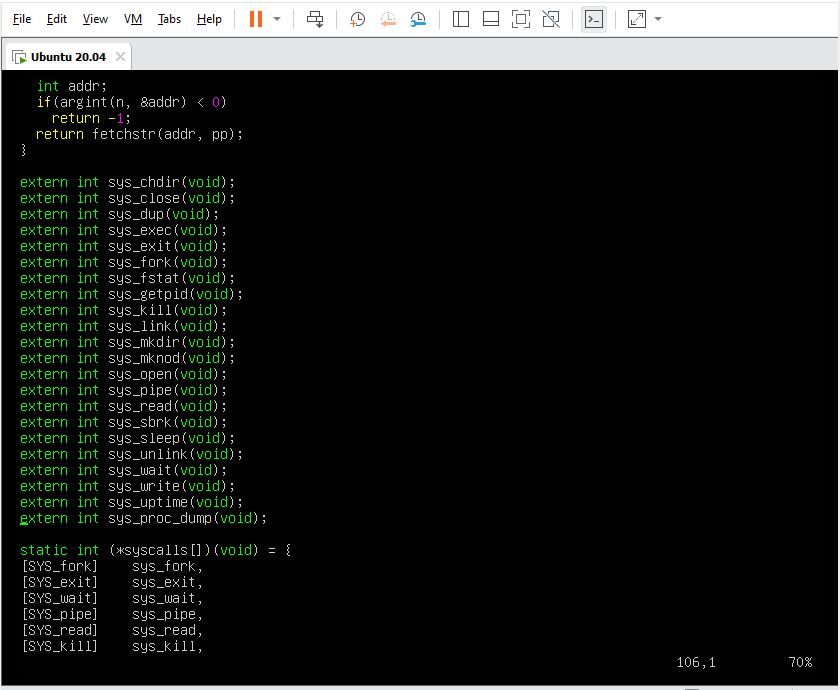
Next, I need to add pointer to system call in syscall.c file. This file contains an array of function pointers which uses indexes as pointers to system calls which are defined in different location. You can see it in the first photo.

[SYS\_proc\_dump] sys\_proc\_dump,

This means, when system call occurred with system call number 22, function pointed by function pointer sys\_proc\_dump will be called. So we implement it here just like other 21 systemcalls. You can see it in the second photo of this section.

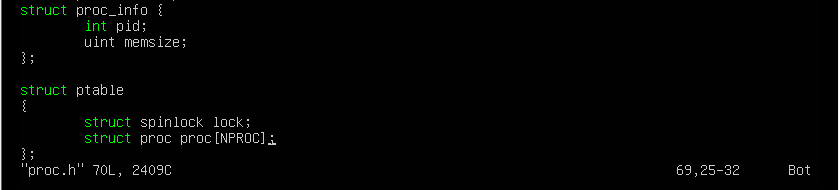
extern int sys\_getyear(void)





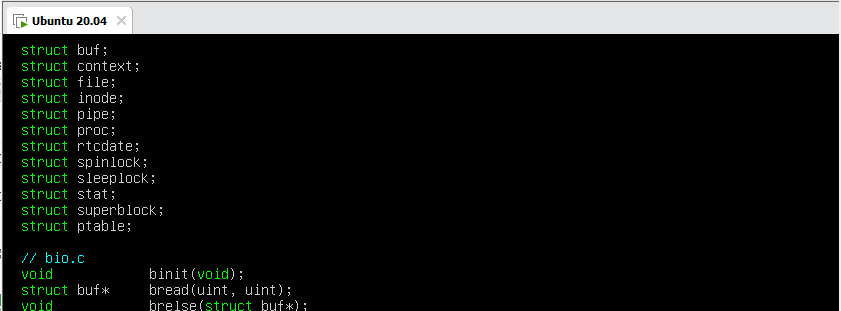
# proc.h

this is a kernel header file. I defined 2 type of structs (ptable and proc\_info) that I needed in my program.

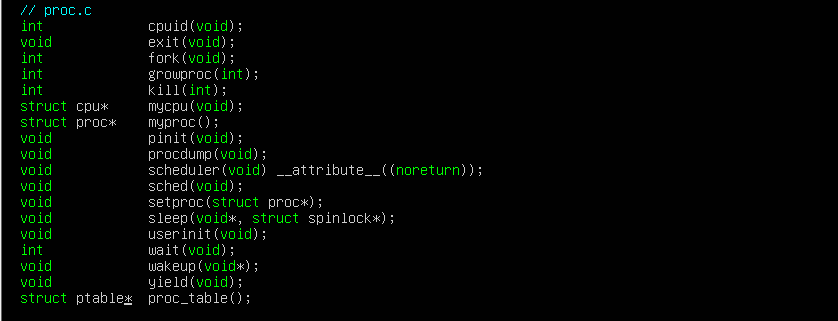


# defs.h

We need to add function prototype to this file. To use it on another files that include this header. I defined a function that returns a pointer to ptable struct. But of course we must implement this in the next step.



struct ptable;



struct ptable\* proc\_table()

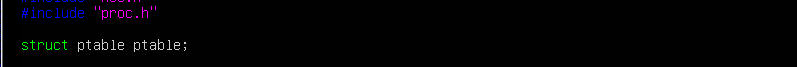
# types.h

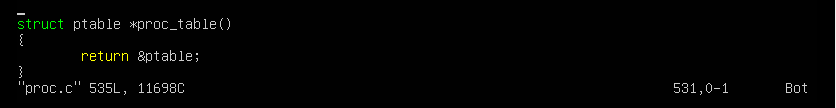
because xv6 doesn’t recognize ‘NULL’, I defined it here where we define our types.



# proc.c

here I just defined proc\_table function which returns the process table that we will work with that in our system call function.



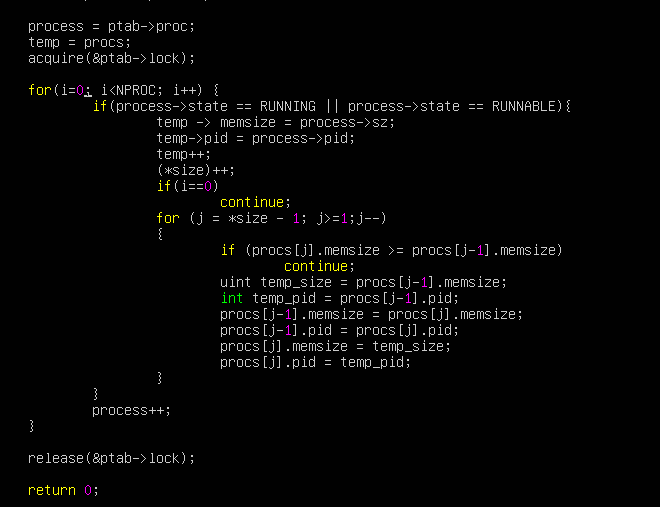


# sysproc.c

Next, I implement system call function. In order to do this, I did this in sysproc.c file where system call functions are defined.

This is the main part of our system call, where we store processes that are in running or runnable state in our struct proc\_info array. And then sort it using bubble sort algorithm by their memsize assending.

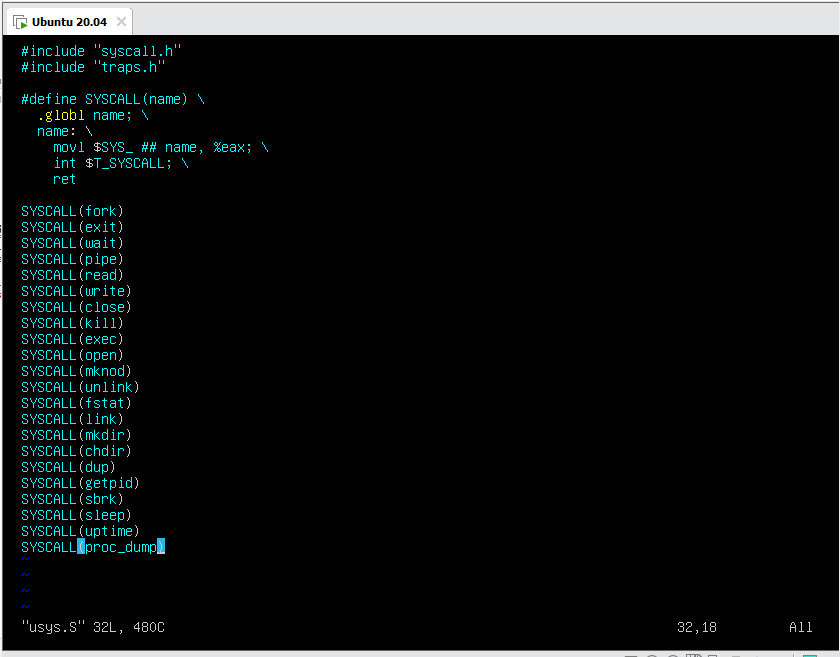
Here I use acquire and release the struct ptable because Acquire-release ordering add more synchronization between threads compared to relaxed ordering. It ensures synchronization between threads that are storing and loading the same atomic object.



# usys.S

this file contains interface for our user program(userTestProcDump.c) to access system call. So we just add this line like the other 21 system calls.

SYSCALL(proc\_dump)



# user.h

call to below function from user program will be simply mapped to system call number 22 which is defined as SYS\_proc\_dump preprocessor directive. So I just add this two line of program like the other system calls:

struct proc\_info;

int proc\_dump(struct proc\_info \*, int \*);

because the system call should give an array of struct proc\_info and the length that user wants.



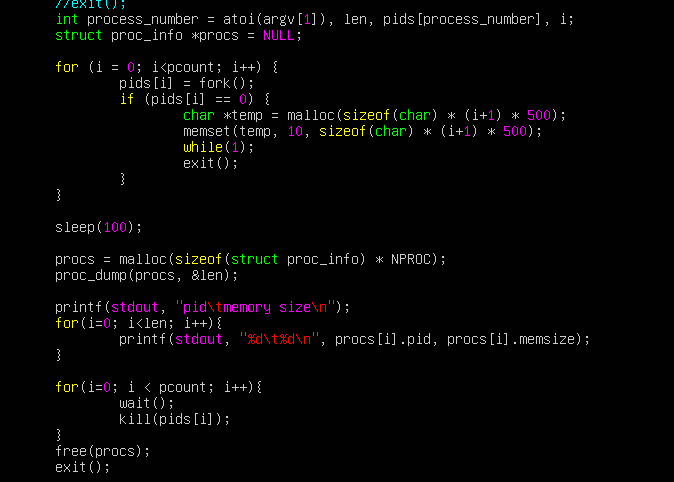
# userTestProcDump.c

Now I fully create our system call on xv6, then I wrote this test program for this system call. The user will enter number of processes that he wants to add to the array as an argument to this program.

It forks some processes and malloc different amount of memory on each of them. And before these processes termination calls the proc\_dump system call to fill the array of proc\_info. Then prints this array (that sorted by the function of system call).

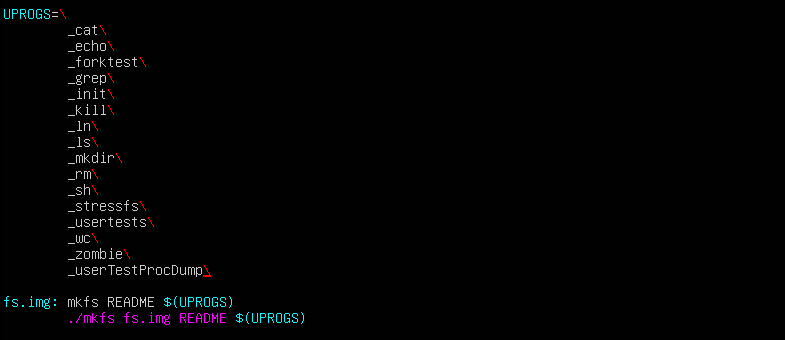
Of course we must kill and wait for child process to prevent zombie processes.

At the end we free the memory of our array.



# Makefile

As explained in the project document I added my test file name to UPROGS and EXTRA of Makefile so the user can enter “userTestProcDump” and then the test program runs.

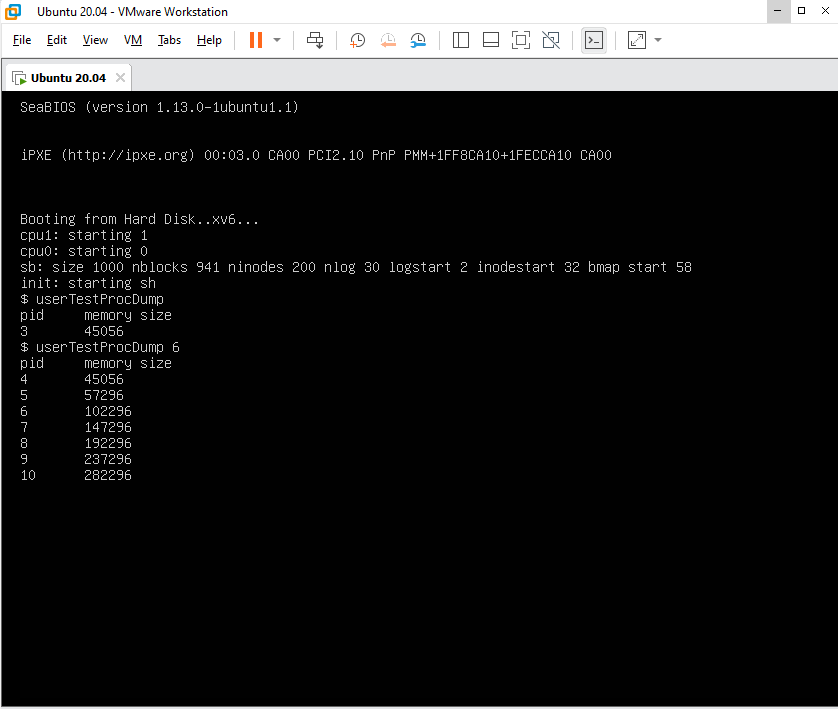


\_userTestProcDump\



userTestProcDump.c\

# Run example



# Resources

* <https://www.geeksforgeeks.org/xv6-operating-system-adding-a-new-system-call/>
* <https://www.guru99.com/system-call-operating-system.html>
* https://stackoverflow.com/questions/47744641/how-to-initialize-a-globe-struct-in-xv6