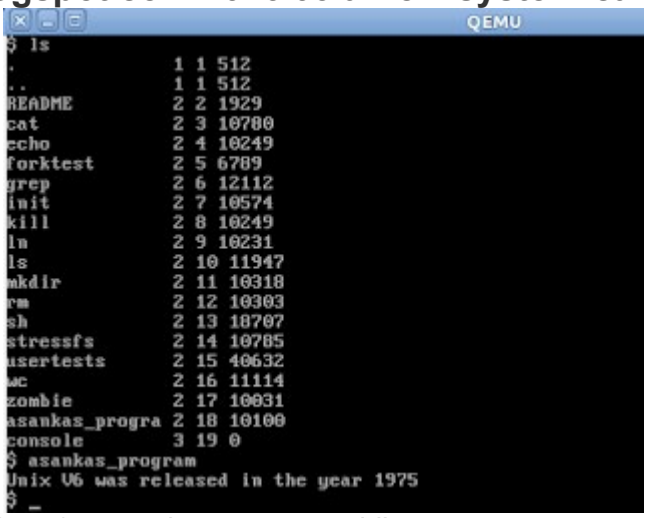
A New System Call to xv6
Link :
https://recolog.blogspot.com/2016/03/a-new-system-call-to-xv6.html



In our explorations with xv6 operating system, adding a new system call is one of the most important activities that helps us to learn the internal working of the system call handling functionality of the operating system. In this blog post, I'm going to implement a system call called **`getyear`** which will return **`1975`** from the kernel always. the Unix version 6 was released in that year. Even though this system call does not do any useful thing, it shows us the necessary places in the source code of xv6 which we have to touch in this kind of business.

We start this work from **`syscall.h`** file where a 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 the following line to reserver a system call number for our own system call.

```
#define SYS_getyear  22
```

Now we have to do some additional work in **`syscall.c`** file related to the line we added in the header file above. So, open the C file. There's an array of function pointers inside this file with the function prototype **`static int (*syscalls[])(void)`**. It uses the numbers of system calls defined above as indexes for a pointer to each system call function defined elsewhere. At the end of this function pointer array, let's add the following line.

```
[SYS_getyear]   sys_getyear,
```

This means, when a system call occurred with the system call number 22, the function pointed by the function pointer **`sys_getyear`** will be called. So, we have to implement this function. However, this file is not the place we are going to implement it. We will just put the function prototype here inside this file. So, find the suitable place inside this file and add the following line. You can see that all the other 21 system call functions are defined similarly.

```
extern int sys_getyear(void);
```

Now, let's implement this actual system call function. There are two files inside xv6 system where system calls are defined. **`sysproc.c`** and **`sysfile.c`** are those two places. If you open and check, you will see that many system calls related to file system are located in **`sysfile.c`**

while the rest is in **sysproc.c**. So, to implement our system call, we use the file **sysproc.c**. Open it and add the following function implementation at the end of the file.

```
// return the year of which the Unix version 6 was released
int
sys_getyear(void)
{
    return 1975;
}
```

Now we have just two little files to edit and these files will contain the interface for our user program to access the system call. Open the file called usys.S and add a line line the below at the end.

```
SYSCALL(getyear)
```

Then, open the file called user.h and add the following line. This is the function that the user program will be calling. As you know now, there's no such a function implemented in the system. Instead, a call to the below function from a user program will be simply mapped to the system call number 22 which is defined as **SYS_getyear** preprocessor directive. The system knows what exactly is this system call and how to handle it.

```
int getyear(void);
```

Having done all the above stuff, our xv6 operating system now contains a new system call. Let's write a user program which will try to make that new system call to see whether the xv6 kernel respond to it. I have previously written about how to add a user program to xv6. Refer this blog post to achieve it. Following those steps, add a user program but the content of the program should be as follows.

```
// #Asanka:
// A simple program which just print something on screen

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

int
main(void)
{
 printf(1, "Unix V6 was released in the year %d\n", getyear());
 exit();
}
```

After adding this user program properly into the xv6 source code directly and making the necessary changes in Makefile, we can attempt to compile and run it. So, issue the following commands inside the source code directory of xv6.

```
make clean
make
```

Now, start the xv6 system on QEMU and wait till it boots up to the shell prompt. There you can issue **`ls`** command to see whether your user program is available. If yes, just go ahead and run the program. It should give you an output like the one shown in the above picture if everything went smoothly.

References:

[1] http://www.fotiskoutoulakis.com/2014/04/28/introduction-to-xv6-adding-a-new-system-call.html

[2] http://moss.cs.iit.edu/cs450/assign01-xv6-syscall.html