**Assignment 5 Report**

**Tanmay Goyal**

**AI20BTECH11021**

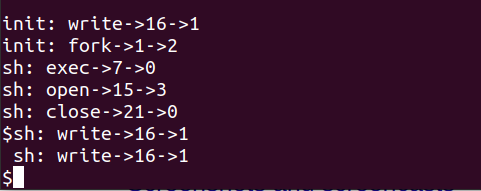
Implementation of System Calls:

User programs have certain limitations and restrictions on the actions they can carry out. This is to guarantee protection of the data in the OS. However, user programs can call System Calls, which are then implemented in the Kernel mode. These System Calls request a service from the operating system, which are executed in the Kernel mode, and then switches back to the User mode.

For implementing system calls in xv6, in the file *usys.S* , we have defined a macro called SYSCALL, which takes as input the name of the user call, and obtains the corresponding unique umber that the call has been mapped to. These user calls are declared in the file *user.h*. For example, when the user types in the command *mydate,* the code for mydate is given in *mydate.c*, which calls upon *mydate()*, which is the user call. In the file *syscalls.h*, we have mapped each system call to a unique number. These unique numbers are then used in *syscalls.c* to call upon the appropriate system calls by locating them off the stack.

1. Understanding and tracing system calls

We simply add a statement to print to the console. However, for the name, we are required to define a new array containing the names, else we would get the location of the stack pointer corresponding to the particular system call.



**Tracing the system calls upon booting xv6**

1. Implementing date system call

We modify the following files:

1. *syscall.c:* We add the key-value pair for mydate. Also, we do not define the function within the file, and hence, mention it is an *extern* function.
2. *syscall.h:* Add a unique number to map the call.
3. *user.h*: Add the function declaration for the user call.
4. *sysproc*.c: Add the function definition for the system call.
5. *usys*.s: Added the call for the macro SYSCALL.
6. *Makefile*
7. *mydate.c*

Text

Description automatically generated

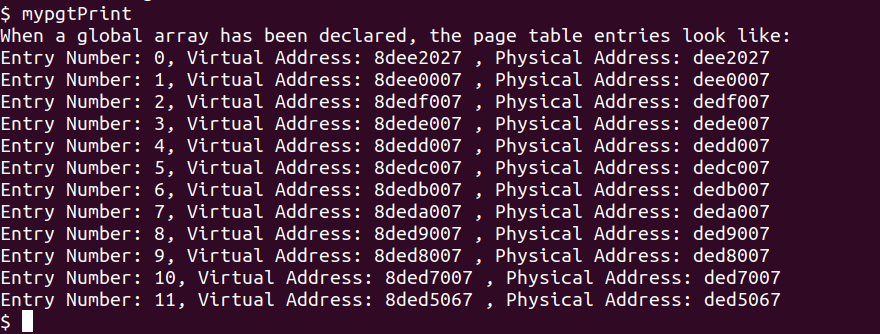
1. Printing the Page Table Entries

We modify the following files:

* 1. *syscall.c:* We add the key-value pair for mypgtPrint. Also, we do not define the function within the file, and hence, mention it is an *extern* function.
  2. *syscall.h:* Add a unique number to map the call.
  3. *user.h*: Add the function declaration for the user call.
  4. *sysproc*.c: Add the function definition for the system call.
  5. *usys*.s: Added the call for the macro SYSCALL.
  6. *Makefile*
  7. *mypgtPrint.c*

Defining a global array:

On declaring a global array, we obtained the following output:



Defining a local array:

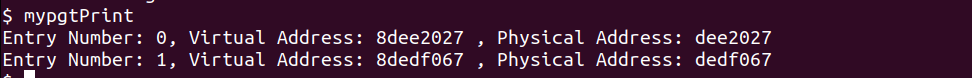
On declaring a local array, we obtained the following output:

Text

Description automatically generated

On repeating the command multiple times:

Text

Description automatically generated

We find that the number of page table entries in case of the declaration of global array. However, in case of a local array of the same size, it remains the same. This is because for a global array, memory is allocated in a fixed location as it is supposed to be accessible by the entire process and hence, belongs to the process’ address space. However, for local variables, they are dynamically created, given space on the stack, and then removed once their scope ends.

We also find that on repeating mypgtPrint multiple times, we obtain different answers. This is because every time different parts of the memory might be free. Also it depends on demand paging and victim replacement, etc.