Assignment 5 Report

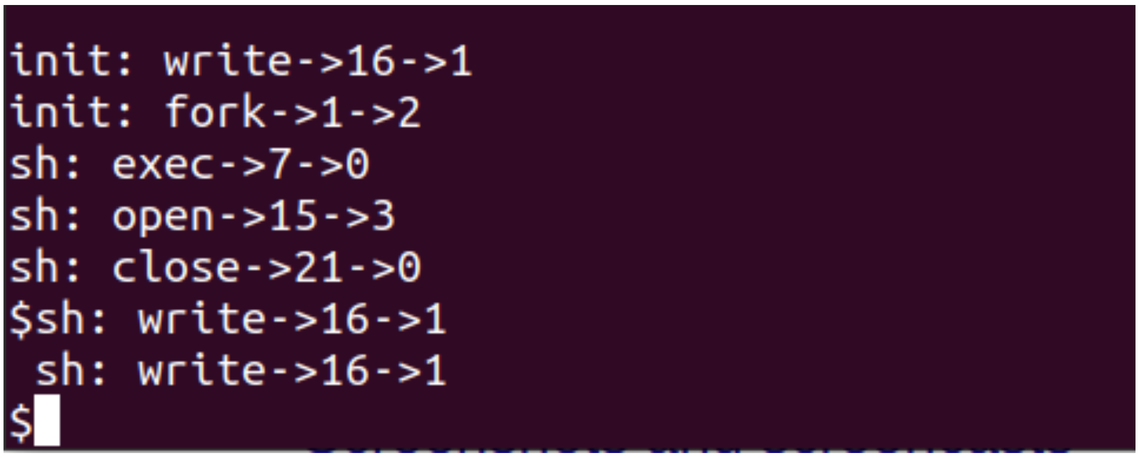
Tanmay Goyal
AI20BTECH11021

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.



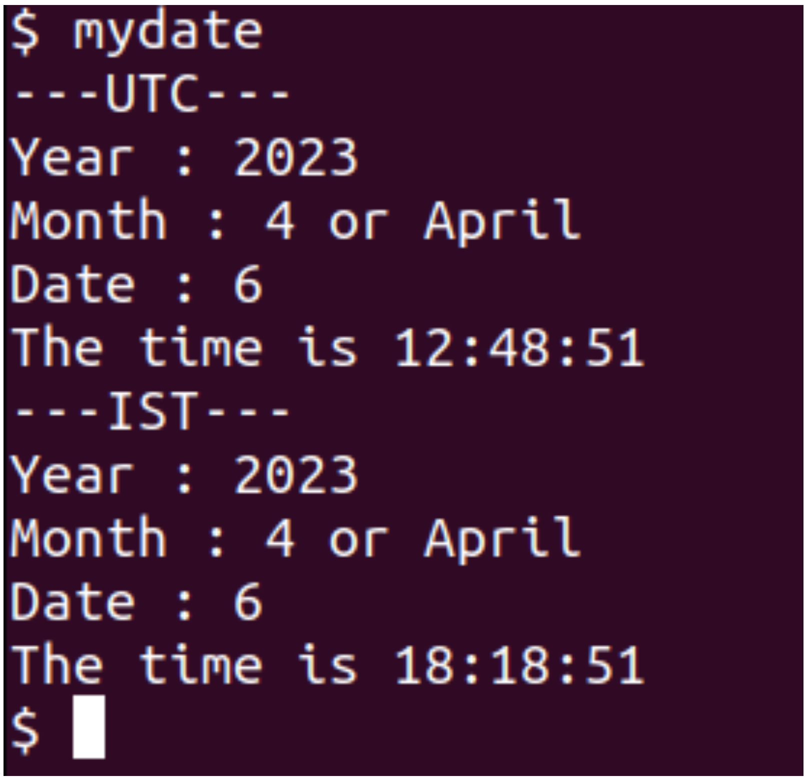
```
init: write->16->1
init: fork->1->2
sh: exec->7->0
sh: open->15->3
sh: close->21->0
$sh: write->16->1
 sh: write->16->1
$
```

Tracing the system calls upon booting xv6

2. <u>Implementing date system call</u>

   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*



```
$ mydate
---UTC---
Year : 2023
Month : 4 or April
Date : 6
The time is 12:48:51
---IST---
Year : 2023
Month : 4 or April
Date : 6
The time is 18:18:51
$ 
```

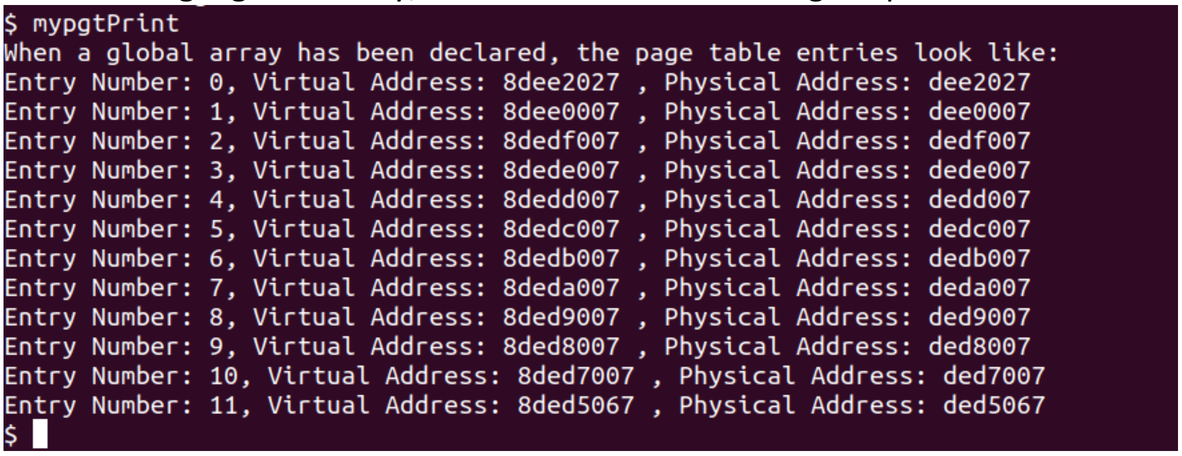
3. <u>Printing the Page Table Entries</u>

   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:

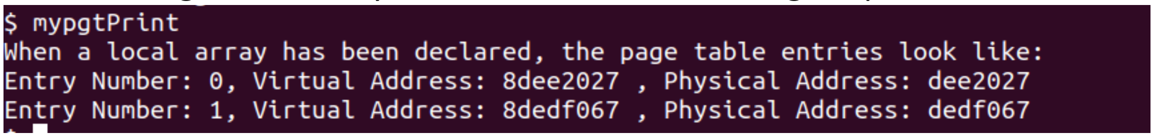
On declaring a global array, we obtained the following output:



```
$ mypgtPrint
When a global array has been declared, the page table entries look like:
Entry Number: 0, Virtual Address: 8dee2027 , Physical Address: dee2027
Entry Number: 1, Virtual Address: 8dee0007 , Physical Address: dee0007
Entry Number: 2, Virtual Address: 8dedf007 , Physical Address: dedf007
Entry Number: 3, Virtual Address: 8dede007 , Physical Address: dede007
Entry Number: 4, Virtual Address: 8dedd007 , Physical Address: dedd007
Entry Number: 5, Virtual Address: 8dedc007 , Physical Address: dedc007
Entry Number: 6, Virtual Address: 8dedb007 , Physical Address: dedb007
Entry Number: 7, Virtual Address: 8deda007 , Physical Address: deda007
Entry Number: 8, Virtual Address: 8ded9007 , Physical Address: ded9007
Entry Number: 9, Virtual Address: 8ded8007 , Physical Address: ded8007
Entry Number: 10, Virtual Address: 8ded7007 , Physical Address: ded7007
Entry Number: 11, Virtual Address: 8ded5067 , Physical Address: ded5067
$
```

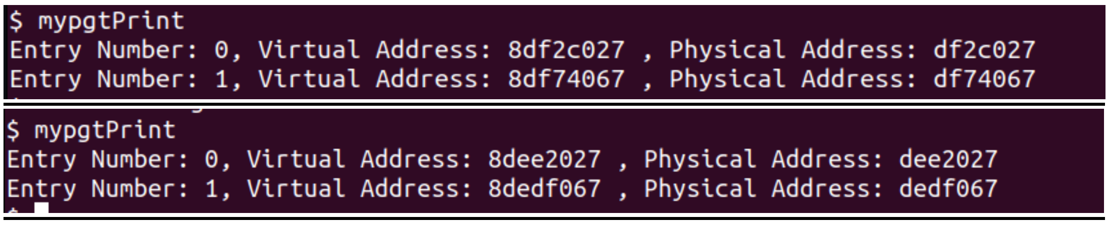
Defining a local array:

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



```
$ mypgtPrint
When a local array has been declared, the page table entries look like:
Entry Number: 0, Virtual Address: 8dee2027 , Physical Address: dee2027
Entry Number: 1, Virtual Address: 8dedf067 , Physical Address: dedf067
```

On repeating the command multiple times:



```
$ mypgtPrint
Entry Number: 0, Virtual Address: 8df2c027 , Physical Address: df2c027
Entry Number: 1, Virtual Address: 8df74067 , Physical Address: df74067
```

```
$ mypgtPrint
Entry Number: 0, Virtual Address: 8dee2027 , Physical Address: dee2027
Entry Number: 1, Virtual Address: 8dedf067 , Physical Address: dedf067
```

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.