CS3523: Operating Systems-2

Programming Assignment 5

Syscall implementation

a)Part-1:Understanding and tracing system call:

We can observe the processes and the system calls that are invoked while booting xv6 by printing the process name, system call name, system call number and its return value. We give character strings to each system call to get the name of the system call before the function syscall(void).

Part-2:Implementing system call(date):

We can understand the different pieces of system call machinery in the way of making a new system call by modifying the xv6 kernel.

The steps followed to implement the new system call-date:

- Give a number to system call in the file syscall.h

  #define SYS_date 22

- Give a pointer that will point to that particular system call number in the file syscall.c

  [SYS_date] sys_date,

- Implement a prototype for the function sys_date in the file syscall.c

  extern int sys_date(void);

- Implement the function in the file sysproc.c
- Build a interface for the user program that we write to test the new system call in the file usys.s

  SYSCALL(date)

- For the user to access the system call we need to modify another header file, user.h

  int date(struct rtcdate*);

- Write a user program to test the new system call and name it mydate.c and save it in the same directory as the make file.

- Make modifications in the make file. In UPROGS,

  _mydate\

- Another modification in the make file. In extras, give the name of the source file

  mydate.c\

- Typing mydate to xv6 shell prompt will print the current date and time.

b) Part-3:Implementing system call(pgtPrint):

The steps mentioned for part 2 are followed to implement the pgtPrint system call.

Observations:

- For no array declared, there are two entries whose physical address changes every time we execute but the virtual address and the number of entries remains the same.
- For an globally declared array, the physical address changes every time we execute but the virtual address and the number of entries remains the same.
    1. For an array of size 100, there are 2 entries.
    2. For an array of size 1000, there are 3 entries.
    3. For an array of size 10000, there are 12 entries.
    4. For an array of size 100000, there are 100 entries.
    5. For an array of size 1000000, there are 979 entries.
    6. For an array of size 10000000, there are 9768 entries.
    7. For an array of size 10000000, alloc uvm out of memory, exec mypgtPrint failed.
- Global variables are stored in the data section of virtual address space. This is later mapped to physical address space that may not be present initially leading to page fault. The OS brings those pages into physical memory, resulting in the entries being added to the page table.
- For a locally declared array, there are 2 entries whose physical address changes every time we execute but the virtual address and the number of entries remains the same.
- The entries for the local array may remain the same because it is allocated on the stack, which is typically pre-allocated with a fixed size when the process is created. As long as the size of the local array does not exceed the pre-allocated stack size, the page table entries for the stack should remain the same.