**TEAM -09**

Project 1: Demo Question and Answers.

**Q1. Explain (with xv6 code) what your user-level wrapper routines of the two system calls are, and where these user-level wrapper routines are deﬁned.**

**Solution:**

* The user level wrapper routine of the two system calls are **int shutdown(void)** and **int shutdown2(char \*)**
* **Shutdown :** int shutdown(void) is defined in the user.h file and is linked to the system call method named int sys\_shutdown(void)
* In xv6, shutdown is mentioned in Uprogs=\ which provides access to the shutdown file
* Qemu-nox is executed to point to the system call of the user wrapper routine (SYS\_shutdown)

User-level wrapper routine code:

#include “types.h”

#include “stat.h”

#include “user.h”

Int

main(int argc, char \* argv[])

{

printf(1,”BYE~\n”);

shutdown();

exit(); //return 0;

}

* **Shutdown2:** shutdown2 uses int shutdown2(char\*) as a user level wrapper routine
* This is defined in the user.h file which points to the system call method named int sys\_shutdown 2(void) in sysfile.c file
* Character argument type prints the shutdown message
* In xv6, shutdown2 is mentioned in Uprogs=\ which provides access to the shutdown2 file and creates an object.
* Qemu-nox is executed to point to the system call of the user wrapper routine (SYS\_shutdown2)

User-level wrapper routine code:

#include “types.h”

#include “stat.h”

#include “user.h”

Int

main(int argc, char \* argv[])

{

char msg[128] = {0};

char \* p = msg;

int i = 0;

for(i=1; I < argc; i++)

{

strcpy(p, argv[i]);

p += strlen(argv[i]) + 1;

if (p >= msg + sizeof(msg))

{

p = msg + sizeof(msg);

break;

}

else

{

\*(p - 1) = ‘ ’;

}

}

\*(p - 1) = 0;

//”msg” now holds the shutdown message provided by the user

shutdown2(msg);

exit();//return 0;

}

**Q2. Explain (with xv6 code) how the system call wrapper routine used in the shutdown2 program triggers the corresponding system call (i.e., the user-level operations of calling a system call).**

**Solution:**

In this process, the control shifts from user mode to kernel mode.

Shutdown2 is defined in a c function called syscall.c as

extern int sys\_shutdown2(void); and also in the static int()

static int(\*syscall[](void))={

[SYS\_shutdown2] sys\_shutdown2

};

For an user program to call shutdown2, an interface is to be added. So, SYSCALL(shutdown2) is added in the file usys.S

Now the system call service routine invokes SYS\_shutdown2, which is implemented in sysfile.c file as:

int sys\_shutdown2(void)  
{

outw(0XB004, 0X0|0X2000);

outw(0X604, 0X0|0X2000);

return 0;

}

System call service routine is called and shutdown2 is executed from the kernel.

**Q3. Using the above system as an example, explain how kernel locates the system call and calls it (i.e., the kernel-level operations of calling a system call).**

**Solution:**

In User-level wrapper routine, user invokes the system call that starts the calling process related to system call in the kernel.

**For Instance,** Shutdown system call.

**Stage 1.** User calls the c program shutdown.c, which contains the function shutdown() that invokes the system call shutdown2.

#include "types.h"

#include "stat.h"

#include "user.h”

int

main(int argc, char \* argv[])

{

printf(1, "BYE~\n");

shutdown();

exit(); //return 0;

}

**Stage 2.** Calling function shutdown in the user.h file is defined with following argument:

int shutdown(void);

**Stage 3.** From user.h the syscall.h is invoked where the system call is defined with its system call number (pointer to the system call in sysfile.c).

#define SYS\_shutdown 22

**Stage 4.** Shutdown implementation is written in syscall.c :

extern int sys\_shutdown(void);

static int(\*syscalls[](void))={

[SYS\_shutdown] sys\_shutdown

};

**Stage 5.** From syscall.c, the call is invoked from int SYS\_shutdown(void) function which is in sysfile.c and will execute the system call which contains code of terminating the xv6 machine.

int sys\_shutdown(void)  
{

outw(0XB004, 0X0|0X2000);

outw(0X604, 0X0|0X2000);

return 0;

}

This way a system call is executed by a kernel.

**Q4. Explain (with xv6 code) how arguments are passed to a system call in kernel space.**

**Solution:**

Arguments cannot be passed from user to kernel space in xv6, because xv6 has its own inbuilt functions to pass arguments to kernel. Built-in function for string and integer in xv6 are “argstr()” and “argint()” respectively.

**Example:**

Int sys\_shutdown2(void)

{

char \* alpha;

argstr(0,&alpha);

cprintf("%s\n", alpha);

outw(0xB004, 0x0 | 0x2000);

outw(0x604, 0x0 | 0x2000);

return 0;

}

Here, argstr() function gets the message from shutdown2(msg) function called in shutdown2.c