OVERVIEW OF SYSTEM CALLS

A system call is how a program requests a service from an operating system's kernel. This may include hardware-related services (for example, accessing a hard disk drive), creation and execution of new processes, and communication with integral kernel services (like scheduling). System calls provide an essential interface between a process and the operating system.

The design of the microprocessor architecture on practically all modern systems involves a security model which specifies multiple privilege levels under which software may be executed. For instance, a program is usually limited to its own address space so that it can't access or modify other running programs or the operating system itself.

A. CATEGORIES OF SYSTEM CALLS

System calls can be roughly grouped into five major categories:

1. **Process Control**

- end, abort
- load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

2. **File management**

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

3. **Device Management**

- request device, release device
- read, write, reposition
- get device attributes, set device attributes
- logically attach or detach devices

4. **Information Maintenance**

- get time or date, set time or date

- get system data, set system data
- get process, file, or device attributes
- set process, file, or device attributes

5. **Communications**

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

INTRODUCTION OF KERNEL DEBUGGERS

The kernel has two different debugger front ends (kdb and kgdb) which interface to debug core. It is possible to use either of the debugger front ends and dynamically transition between them if you configure the kernel properly at compile and runtime.

Kdb is simplistic shell-style interface which you can use on a system console with a keyboard or serial console. You can use it to inspect memory, registers, process lists, dmesg, and even set breakpoints to stop in a certain location. Kdb is not a source level debugger, although you can set breakpoints and execute some basic kernel run control.

Kgdb is intended to be used as a source level debugger for the Linux kernel. It is used along with gdb to debug a Linux kernel.

A. GDB

You need to get GDB that is capable of understanding your target architecture. Often, this comes with your cross-compiler, but if you have to, you can compile it yourself, but you need to understand the difference between –target and –host configure options. GDB will be running on the host but will be able to understand the target.
The JTAG based debugging method described here is not intrusive. This means that besides debugging symbols you don't need to modify the kernel in any way. This is because we operate on the hardware, CPU core level.

EXAMPLE:
load vmlinuz
target remote :3333

B. LOADING KERNEL IN MEMORY

Once you are used to using gdb to debug kernels you will want to use gdb to directly load kernels onto your target.

KERNEL DEBUGGING IN WINDOWS XP

You will then need to start the virtual machine you want to debug and do the followings:

1. Go to My Computer –> Properties –> Advanced –> Settings under Startup and Recovery –> Edit
2. You should see a line that is something like 'multi(0)disk(0)rdisk(0)partition(1)WINDOWS="Microsoft Windows XP Professional" /fastdetect /NoExecute=OptIn'
3. You will need to add '/DEBUG /DEBUGPORT=COM1' to the end of this line
4. Once you have done all of this you should shut down the virtual machine

KERNEL DEBUGGING IN LINUX

For host based debugging run "kd –
ySRV*c:websymbols*http://msdl.microsoft.com/download/symbols -k com:pipe,port=\.pipedebugPipe,resets=0,reconnect" and then launch your target virtual machine.

For virtual machine based debugging run "kd –
SRV*c:websymbols*http://msdl.microsoft.com/download/symbols -k com:port=com1" and then launch your target virtual machine.