UNIT 5

- **Development Tools, ssh**
- **Kernel Debugging Techniques**
- **Debugging Embedded Linux Applications**
- **Stepper Motor Controller interfacing using BBB**
- **Embedded Graphics and Multimedia Tools and Applications.**

- GNU debugger (GDB)
  - Debugging a core dump
- Data Display Debugger (DDD)
- Cbrowser / cscope
- Tracing and profiling tools.
- SSH

Gnu debugger GDB

- The GNU debugger is a complex and powerful debugger with many capabilities.

- Its capabilities include low level hardware specific debugging support for a wide variety of architectures and microprocessors.

- One of the most common reasons to use GDB is to evaluate a **core dump**

- A core dump is a file generated by the kernel when an application program generates a fault, such as accessing a memory location it does not own.
- Many conditions can trigger a core dump, but SIGSEGV (segmentation fault) is by far the most common.
- A SIGSEGV is a Linux kernel signal that is generated on illegal memory accesses by a user process.
- When this signal is generated, the kernel terminates the process. The kernel then dumps a core image if it is so enabled

- To enable generation of a core dump, the user process must have the authority to enable a core dump.
- This is achieved by setting the process’s resource limits using the setrlimit() C function call, or from a BASH shell command prompt using ulimit

```
 $ ulimit -c unlimited
 $ ulimit
unlimited
```

- The core dump is a snapshot of the running process at the time the segmentation fault occurred.

```
$ Gdb filename core
(gdb) run
```

Data Display Debugger (DDD)

- The Data Display Debugger (DDD), shown in, is a graphical front end to GDB and other command-line debuggers. DDD has many advanced features beyond simply viewing source code and stepping through a debug session.
- DDD is a powerful graphical front end for GDB. It is relatively easy to use and widely supported for many development hosts.
- www.gnu.org/software/ddd/

Cbrowser / cscope

- Cbrowser is a simple source code browing tool that makes it easy to navigate around a large source tree following symbols.

```
$ sudo apt-get install cbrowser
$ make ARCH=    CROSS_COMPILE= cscope
```

The above line produces cscope symbol database that cbrowser uses.

Tracing and Profiling Tools

- Many useful tools can provide with various views of the system. Some tools offer a high-level perspective. It is possible to discover what processes are running on the system and which processes are consuming the most CPU bandwidth.
- Other tools can provide detailed analysis, such as where memory is being allocated or, even more useful, where it is being leaked.
    - **For example : Strace ltrace top ps**

- strace is a useful trace utility found in virtually all Linux distributions.
- strace captures and displays information for every kernel system call executed by a Linux application program.
- strace is especially handy because it can be run on programs for which no source code is available.

```
$ strace ./testprogram
```

- The ltrace and strace utilities are closely related. The ltrace utility does for library calls what strace does for system calls. It is invoked in a similar fashion. You precede the program to be traced by the tracer utility, as follows:

```
$ ltrace ./example
```

Binary Utilities :

- These are cross-utilities and must be built to execute on development host while operating on binary files targeted to chosen architecture.
    - **readelf**
- The readelf utility examines the composition of target ELF binary file. This is particularly useful for building images targeted for ROM or Flash memory where explicit control of the image layout is required.
- It is also a great tool for learning how our toolchain builds images and for understanding the ELF file format.
- For example, to display the symbol table in an ELF image, following commands can be used :

```
$ readelf -s <elf-image>
$ readelf -e <elf-image>
```

SSH : Secure shell

- SSH, which is an acronym for Secure SHell, was designed and created to provide the best security when accessing another computer remotely.
- It encrypts the session, and it also provides better authentication facilities, as well as features like secure file transfer, X session forwarding, port forwarding and more so as to increase the security of other protocols.

The first thing we'll do is simply connect to a remote machine. This is accomplished by running 'ssh hostname' on our local machine.

```
ssh username@remotemachinename
```

Kernel Debugging

- The Linux kernel has matured into a very high-performance operating system that can compete with the best commercial operating systems.
- Many areas within the kernel do not lend themselves to easy analysis by simply reading the source code.
- Knowledge of the architecture and detailed design are often necessary to understand the code flow in a particular area

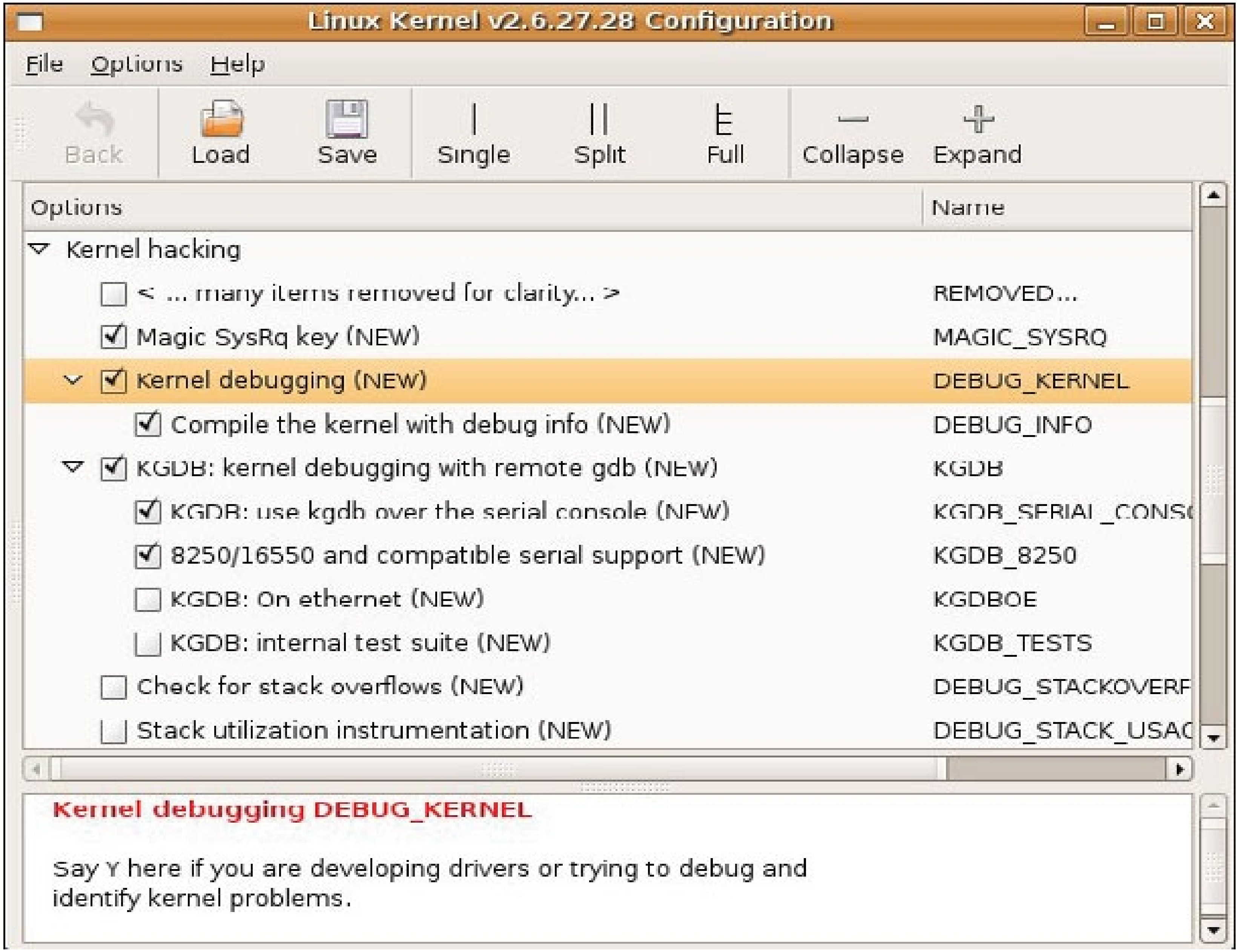
Two popular methods enable symbolic source-level debugging within the Linux kernel:

- Using KGDB as a remote GDB agent
- Using a hardware JTAG probe to control the processor

KGDB (Kernel GDB) is a set of Linux kernel patches that provide an interface to GDB through its remote serial protocol.

KGDB implements a GDB stub that communicates with a cross-gdb running on the host development workstation.

- Until recently, KGDB on the target required a serial connection to the development host. Some targets support KGDB connection via Ethernet and even USB.
- You need to port KGDB to your chosen target or obtain an embedded Linux distribution for your chosen architecture and platform that contains KGDB support. Most commercial embedded Linux distributions available today support KGDB.

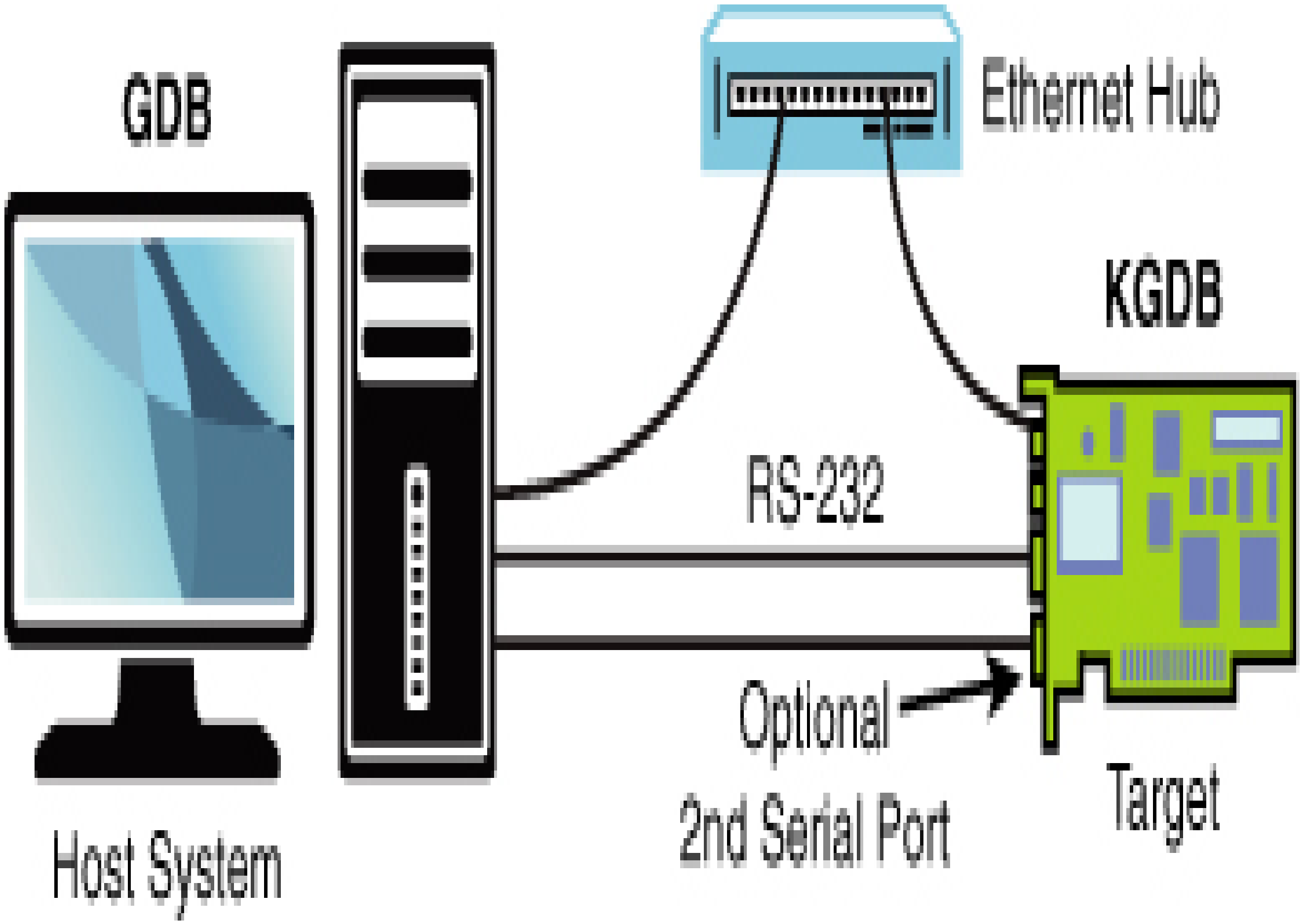


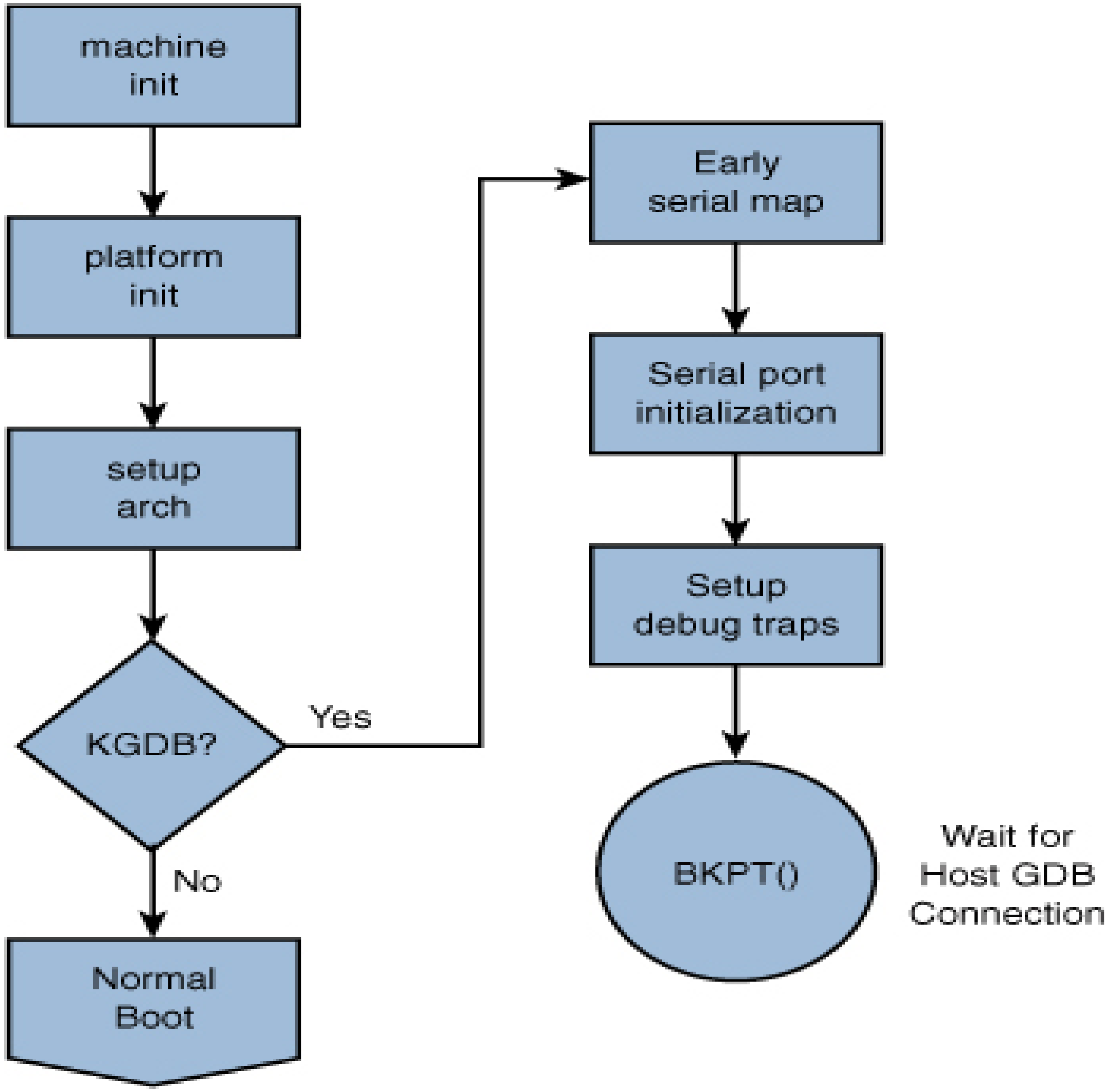
Options	Name
▽ Kernel hacking	
☐ < ... many items removed for clarity... >	REMOVED...
✓ Magic SysRq key (NEW)	MAGIC_SYSRQ
▽ ✓ Kernel debugging (NEW)	DEBUG_KERNEL
✓ Compile the kernel with debug info (NEW)	DEBUG_INFO
▽ ✓ KGDB: kernel debugging with remote gdb (NEW)	KGDB
✓ KGDB: use kgdb over the serial console (NEW)	KGDB_SERIAL_CONS
✓ 8250/16550 and compatible serial support (NEW)	KGDB_8250
☐ KGDB: On ethernet (NEW)	KGDBOE
☐ KGDB: internal test suite (NEW)	KGDB_TESTS
☐ Check for stack overflows (NEW)	DEBUG_STACKOVERF
☐ Stack utilization instrumentation (NEW)	DEBUG_STACK_USAC

Kernel debugging DEBUG_KERNEL

Say Y here if you are developing drivers or trying to debug and identify kernel problems.

- the KGDB debug setup




machine init
platform init
setup arch
KGDB?
Yes
No
Normal Boot
Early serial map
Serial port initialization
Setup debug traps
BKPT()
Wait for Host GDB Connection

Hardware assisted debugging – Abatron BDI 3000 JTAG probe

- **Debugging by printing**
- Probably the simplest way to get some debug information from your kernel code is by printing out various information with the kernel's equivalent of printf - the printk function and its derivatives.
- printk works more or less the same way as printf in userspace, so if you ever debugged your userspace program using printf, you are ready to do the same with your kernel code,

e.g. by adding:

```
printk("My Debugger is Printk\n");
```

In order to see the kernel messages, just use the

```
$ dmesg
```

command in one of your shells - this one will print out the whole kernel log buffer to you.

Printk from userspace

- Sometimes, especially when doing automated testing, it is quite useful to insert some messages in the kernel log buffer in order to annotate what's going on.
- It is as simple as
- # echo "Hello Kernel-World" > /dev/kmsg
- Of course this messages gets the default log level assigned, if you want e.g. to issue a KERN_CRIT message you have to use the string representation of the log level - in this case "2"
- # echo "2 Writing critical printk messages from userspace" >/dev/kmsg

Debugging Embedded Linux Applications using gdbserver

- gdbserver

  On target machine:

  - gdbserver ip_of_target:port_of_target strippedfilename

- gdb
  - On host machine :

  $ xscale_be-gdb -q filename

  $ target remote ip_of_target:port_of_target

Debugging Embedded Linux Applications when shared libraries are used

A simple C program can use standard C libraries like printf, fopen, etc. Or libraries which are available outside std C libraries like audio libraries etc.

$ ldd filename : on target board it shows shared libraries path for given file

$ xscale_be-ldd filename : on development host, tool chain is aware of the path of the shared libraries

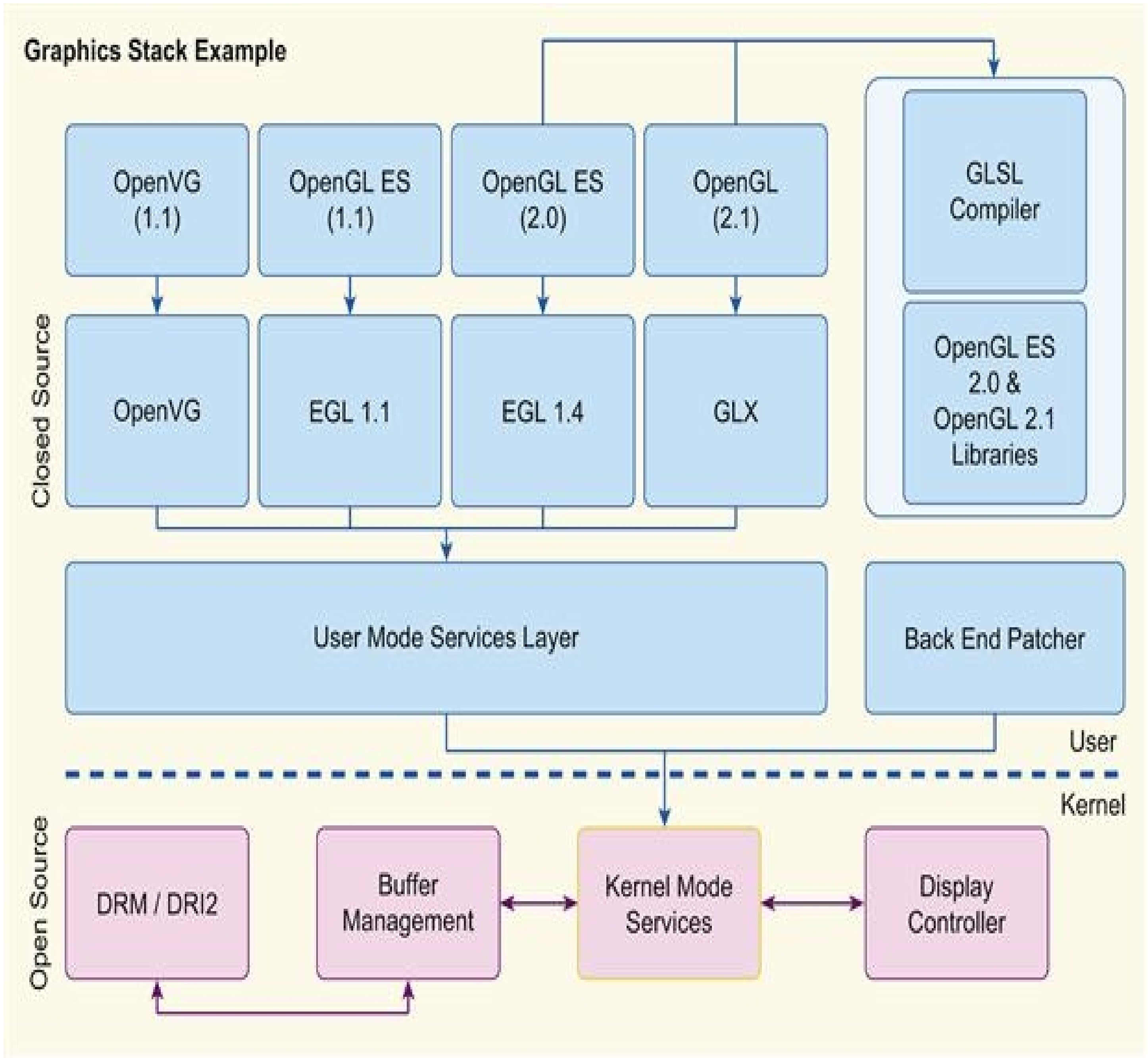
(gdb) i shared   <<< display loaded shared libraries

Multimedia and Graphics :

The following open APIs are generally offered in embedded systems, although many embedded systems do not support the full OpenGL and the SOC vendors chose to implement a subset of OpenGL known as OpenGL ES.

OpenGL : This is the most widely used library providing 2D and 3D APIs. The functions typically make use of hardware capabilities in the GPU. The standard also includes a shader language. The OpenGL Shading Language allows application programmers to express processing that occurs at programmable points within the OpenGL processing pipeline.

OpenGL ES : Standard for Embedded Accelerated 3D Graphics
A royalty-free, cross-platform API for full-function 2D and 3D graphics on embedded systems. It consists of well-defined subsets of desktop OpenGL. OpenGL ES includes profiles for floating-point and fixed-point systems in the EGL specification. OpenGL ES 1.X is for fixed function hardware and offers acceleration, image quality, and efficient embedded performance. OpenGL ES 2.X enables full programmable 3D graphics.
EGL (*Khronos* Native Platform Graphics Interface) : EGL provides mechanisms for creating rendering surfaces onto which client APIs such as OpenGL ES and OpenVG can draw, creates graphics contexts for client APIs, and synchronizes drawing by client APIs as well as native platform rendering APIs. This enables seamless rendering using both OpenGL ES and OpenVG for high-performance, accelerated, mixed-mode 2D and 3D rendering.


Graphics Stack Example
OpenVG (1.1)
OpenGL ES (1.1)
OpenGL ES (2.0)
OpenGL (2.1)
GLSL Compiler
OpenVG
EGL 1.1
EGL 1.4
GLX
OpenGL ES 2.0 & OpenGL 2.1 Libraries
Closed Source
User Mode Services Layer
Back End Patcher
User
Kernel
Open Source
DRM / DRI2
Buffer Management
Kernel Mode Services
Display Controller

Accelerated Media Decode

- Raw video content is quite voluminous in terms of the data required to fully represent it, as a result in order to efficiently transport and store video content, it is generally compressed. There are two aspects to the transport/storage of video content. The first is the container format, and the second is the encoder format. A media encoder takes an uncompressed source and compressed format.
- • MPEG4—.MP4
- • 3GPP—.3GP
- • WMV, Windows Media Video—.WMV
- • MPEG Transport Streams—Streaming format used by Digital Video Broadcast (DVB standards)
- • Adobe Systems Flash Video (FLV) —.F4V

– Media Frameworks

- A number of media frameworks are in common use in embedded platforms today, namely, Gstreamer, OpenHelix and frameworks based on the OpenMax defined APIs.
- **Gstreamer** is an open source multimedia framework for constructing filter graphs and media processing components using a plug-in architecture.
- **OpenMax** is a standard for portable media libraries. OpenMax is an API definition; however, both commercial and open source implementations of each layer are available.

- References :
- Embedded Linux Primer Christopher Hallinan.
- Modern Embedded Computing : Peter Barry Patrick Crowley.
- www.elinux.org
- www.gnu.org/software/ddd
- www.ziplink.net
- www.eclipse.org/
- www.kernel.org
- www.beagleboard.org