Systems Programming with Java

By
Ngatchu Damen

This course assumes the following

- That the reader has a basic understanding of computer architecture
- That the reader has a beyond introductory level of programming skills with Java programming language
- That the reader has a suitable Java programming environment and He/She is familiar with the environment
- That the reader is working on a Linux Operating system platform

Outline

- Input / Output Programming
  - The concepts of interrupts
  - Keyboard and mouse / display and audio
- Device Drivers
  - Interfacing with kernel and working with DLLs
  - APIs and JNI
- Processes and Threads
  - Interprocess Communication and Concurrent programming with Threads
- Network Programming
  - Sockets and Client – Server Programming

Input / Output Programming

In this section we shall Examine the following

- The Concept of interrupts as applied to device programming
- Programming the Keyboard
- The mouse and its events
- Switching between display modes
- Audio output

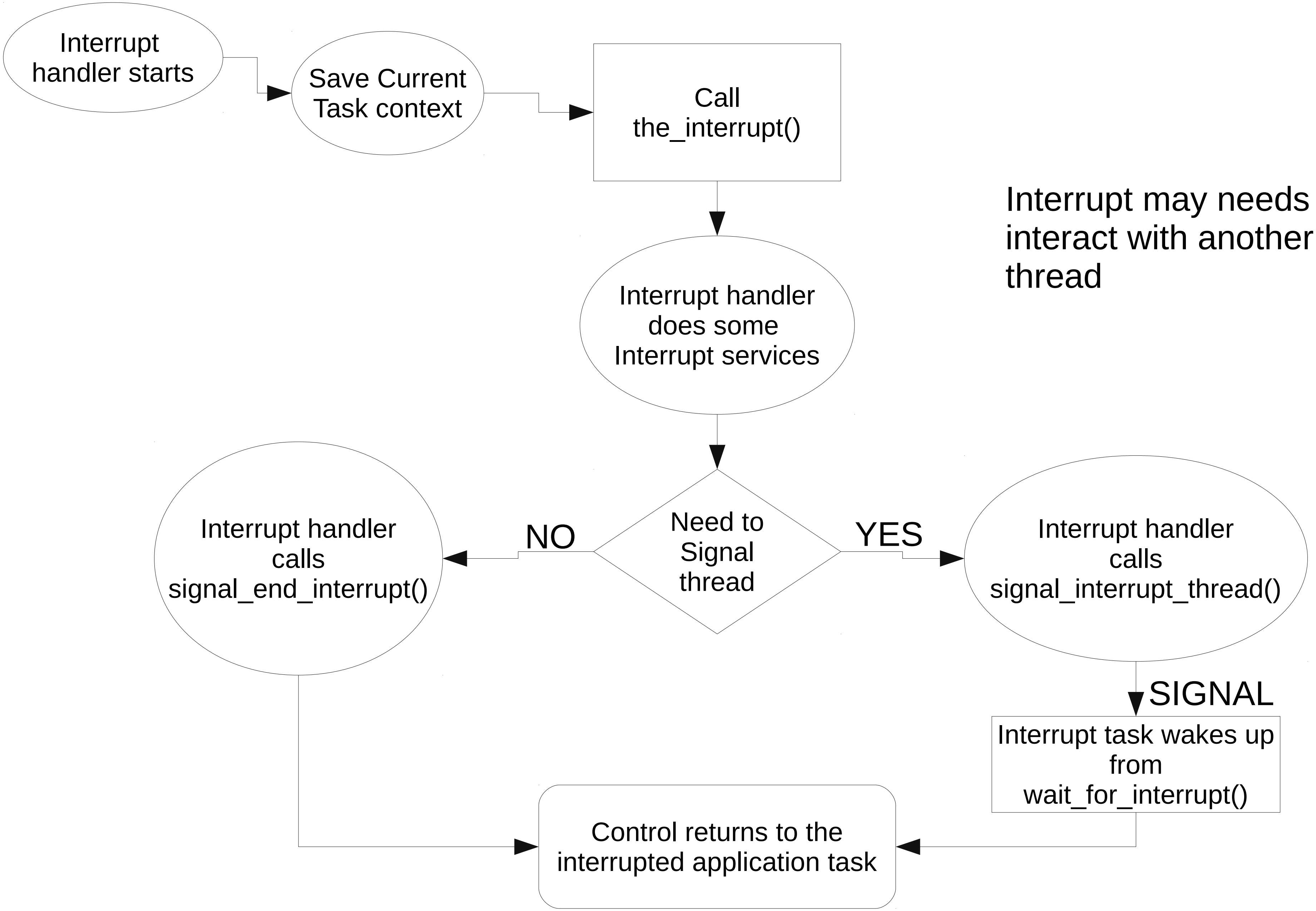
Input / Output Programming

- Most I/O processes (esp to HDD) could be very time consuming and rather than having the CPU wait for completion, It could be assigned to other processes and later interrupted when the I/O process is completed.
- Some CPU Interruptions are hard wired (eg through the clock) to occur on a regular basis while others are programmable and configurable.
- Although handling interrupts is Architecture dependent, the OS kernel provides a generic mechanism that works for most systems.
- When a hardware interrupt occurs the CPU stops what ever task it was processing and jumps either to a predefined location in memory or to the location that contains the interrupt handling code.

Input / Output Programming

- In order to prevent every elementry I/O process from interrupting the CPU, controllers are used to manage I/O processes and the CPU is only interrupted when the controller completes its job.
- While in interrupts mode, some CPUs do not permit the occurrence of other interrupts. Others classify their interrupts following a predefined priority scale
- When the interrupt has been handled, the CPU is restored to continue processing the task it had before the interrupt occurred.
- Interrupt codes need to be very efficient to prevent the OS from blocking too many interrupts for too long.

Input / Output Programming



Interrupt handler starts

Save Current Task context

Call the_interrupt()

Interrupt handler does some Interrupt services

Need to Signal thread

NO

Interrupt handler calls signal_end_interrupt()

YES

Interrupt handler calls signal_interrupt_thread()

SIGNAL

Interrupt task wakes up from wait_for_interrupt()

Control returns to the interrupted application task

Interrupt may needs to interact with another thread

Input / Output Programming

- The programmable interrupt controller uses registers to enable and disable interrupts.
- Linux kernel provides services that are used to request for interrupts, enable and disable them.
- Device drivers call these routines to register their interrupt handling routine addresses.
- Working with the keyboard, mouse, display and audio require hardware interrupt!!
- The JVM environment provides an abstraction from the hardware interruption mechanism.

Input / Output Programming

See tutorials for working with KeyListener

See tutorials for working with MouseListening

See tutorials on display mode

See tutorial on Java Sound

Device Drivers

In this section we shall examine the following

- Types of device Drivers
- Interfacing device drivers with Kernel
- Working with Dynamic Link Libraries (DLLs)
- Java Native Interface (JNI) and Application Programming Interface (API)

Device Drivers

- Each physical device on a computing system has its own controller
- The codes for managing these controllers are kept in the OS kernel and are called Device Drivers
- Device drivers are, essentially, a shared library of privileged, memory resident, low level hardware handling routines.
- Device drivers therefore abstract devices to the point where they are viewed as files.

Device Drivers

- Linux supports three types of hardware devices: **Character, Block** and **Network**
- Character devices are read and written to without any buffering (the serial ports)
- Block devices can only be written to OR read from, in multiples of the block size. Usually 512 of 1024 bytes. They are accessed through the buffer cache either randomly of in series. (files)
- Network devices are accessed through the BSD socket interface and the networking subsystem.

Device Drivers

- Most device drivers form a modular part of the kernel
- They must provide a standard interface to the kernel (I/O interface) and make use of standard kernel services (memory allocation and sevice delivery)
- Most device drivers are dynamically loadable and configurable.
- Device drivers registers themselves with the kernel as they are initialized.
- The record contains pointers to routines and supporting information for interfacing with the specified class of device.

Device Drivers

- Most device drivers are written in native programming languages and stored in libraries.

See Tutorial for working with JNI

- Most third party software for working with devices which are written in Java are bundled up as a set of classes called APIs
- Some are integrated into the development Environment (Java Data Object, Java Media Framework, Java Naming and Directory Interface, …) while others are downloaded separately (Java API for XML-Based RPC, XQuery API for Java).

See Tutorial for working with Xquery and Oracle

Processes and Threads

In this section we shall examine the following

- The Concept of processes
- Interprocess Communication
- Introduction to Concurrent Programming with threads
- Remote Procedure Call (RPC)

Processes and Threads

- A process is a program in execution. i.e it contains the program's instruction and data, as well as the values of the CPU's registers.
- Each process runs in its own virtual address space and can only interact with another process through secure kernel managed mechanisms.
- Each process has a state (running, waiting, stopped, zombie), an identifier (user – UID and group – GID identifier), links to other processes (except the initial process), timer (creation time and elapsed running time) and more.
- In Java, processes are created, executed and destroyed with the help of (java.lang.process) API

See tutorial for working with processes

Processes and Threads

- Both Processes and threads are independent path of execution.
- The main difference between Thread and Process in Java is that Threads are part of process. i.e. one process can spawn multiple Threads.
- Threads uses the memory space of their parent process and can therefore easily communicate amongst themselves.
- In Java, processes are created, executed and destroyed with the help of (java.lang.thread) API

See tutorial for working with threads

Processes and Threads

- Sharing information, distributing tasks for processing and creating hierarchical privileges are some of the reasons for implementing interprocess communication.
- inter-process communication (IPC) is the activity of sharing data across multiple and commonly specialized processes using communication protocols.
- The availability of one or another IPC mechanism depends on the vendor of the OS and the variety of mechanisms address different scenarios for communication between processes.

Processes and Threads

- Some of the approaches used for implementing IPC include File (accessed by different processes), Signal (system message sent from one process to another), Socket (a data stream sent over a network), Message queue (an asynchronous data stream), Pipes (a two way data stream through standard I/O), Named pipes (a two way data stream through a file), Semaphore (a structure for synchronizing multiple processes acting on the same resource), Shared memory (multiple processes are given access to the same memory block), Message passing (commonly used in concurrency models), Memory Mapped file (A file mapped to RAM).
- The implementation of each of these concepts vary from one programming language to the next and are completely absent in some languages.

See tutorials for Interprocess Communication

Processes and Threads

- Threads are more popular in Java programming and the concept of inter-thread communication is self explicit.

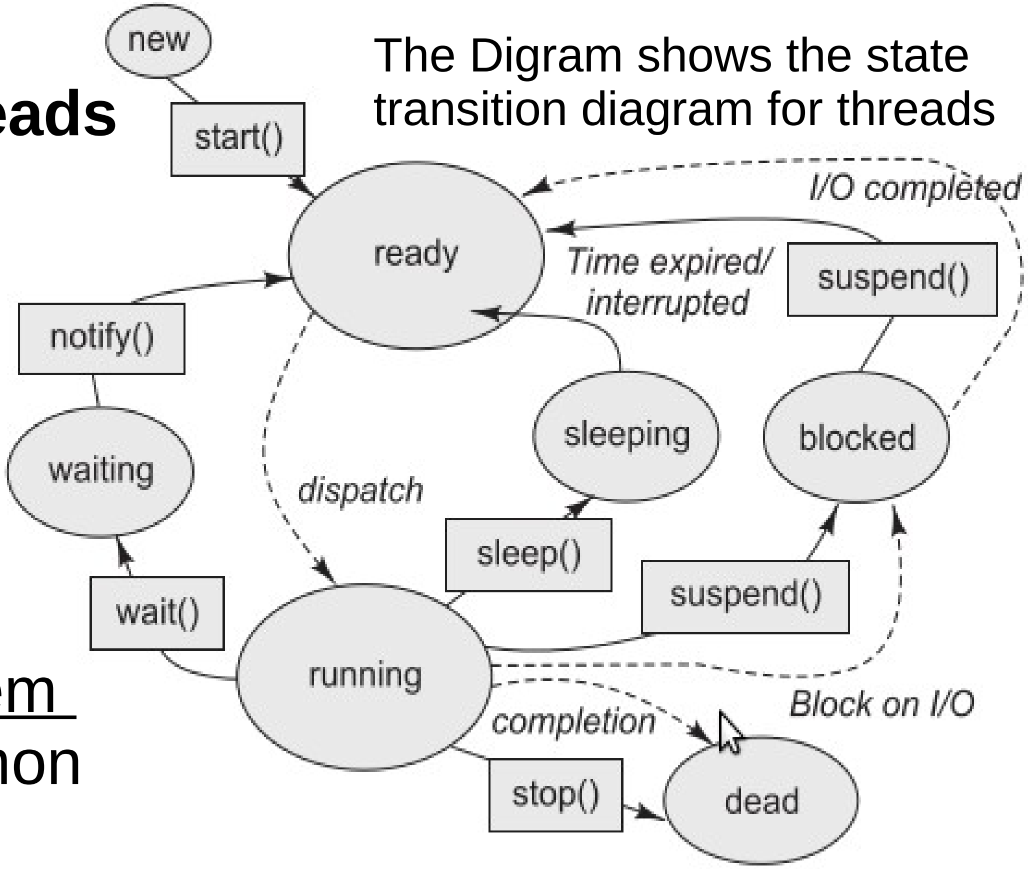
Some current issues with threads

The read / write problem

Reading data on one thread while the same data is being updated on another

The producer / consumer problem

Two threads sharing a common fixed-sized buffer.



The Digram shows the state transition diagram for threads

See tutorials for working with Threads

Network Programming

- The Proliferation of interconnected computing devices makes it paramount for program developers to have a good appreciation of network programming.
- Sockets are the primary mechanism used for network programming and their popularity is linked to that of the TCP/IP protocol stack
- Java provides the opportunity to create two kinds of sockets (ServerSocket – for listening to clients' requests and simple client Sockets) which use IP addresses or host names and ports to communicate in a TCP/IP network

See tutorials for working with Sockets