# System – Shell – Java
# ( Linux – Bash – Jdk )

## Understanding Shell internals to masterize Shell Commands

this document:
http://arnaud.nauwynck.chez-alice.fr/
devPerso/Pres/Pres-System-Shell-Java.pdf

arnaud.nauwynck@gmail.com

# Plan

- Hello World Overview
  - Main = args + stdin/out + env...
- Kernel, System Resources
  - System Calls, Kernel/User Mode
  - Process, Thread, Memory Managment
  - Files
- Shells
  - Variables, Env, Evaluation
  - Files Redirection
  - Utility Bins

# Hello World

```
public class Hello {
    public static void main(String[] args) {
        System.out.println(
            “Hello World ” + args);
        System.exit(0);
    }
}
```

Launching from shell:

```
# java -cp hello.jar Hello Mr Gosling > log.out
```

# Hello World... Outside Java

- Kwnows what's really happens ?
- The shell process does:
  - read command line
  - split text line as exe filename + args
  - lookup java in PATH
  - fork itself (bash)
  - open files to redirect in/out to console
  - exec as java
  - get exit code... loop to read next cmd line

# System Calls
# Kernel Mode / User Mode

- “read”, “write”, “open”, “fork”, “exec”, ...

  are kernel system calls

  = entry-point to **Kernel-Mode** (Intel x86: “ring 0”)
  = have access to ALL hardware resources, without restrictions

- By Opposition to **User-Mode** programs (the shell, java..) which have many restrictions

# Global Hardware Resources

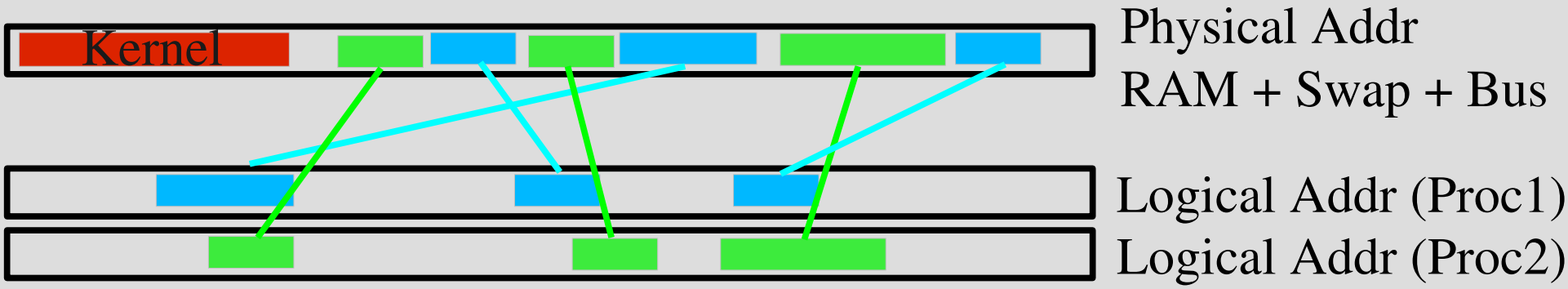
- Keyboard+Mouse (input), Screen (output)
- CPU (1 or several, multi-core..)
  = assembly interpreter, using a stack + curr stack pointer + instruction pointer + registers
- Memory bus : Read/Write access to
  - RAM (all physical memory)
  - Hard drives, pci, scsi, ide, ...
  - Video card, eth card...
  - Bios (motherboard sub-programs)

# Process Logical Resources

- All ressources are handled by the OS
  - Resources are wrapped in Object-Oriented services : Hardware Abstraction Layer
- Process are isolated / protected from others (separate sandbox / vm)
  - ... a Process should not crash the PC
  - or crash another process
- Most important isolation = Memory spaces

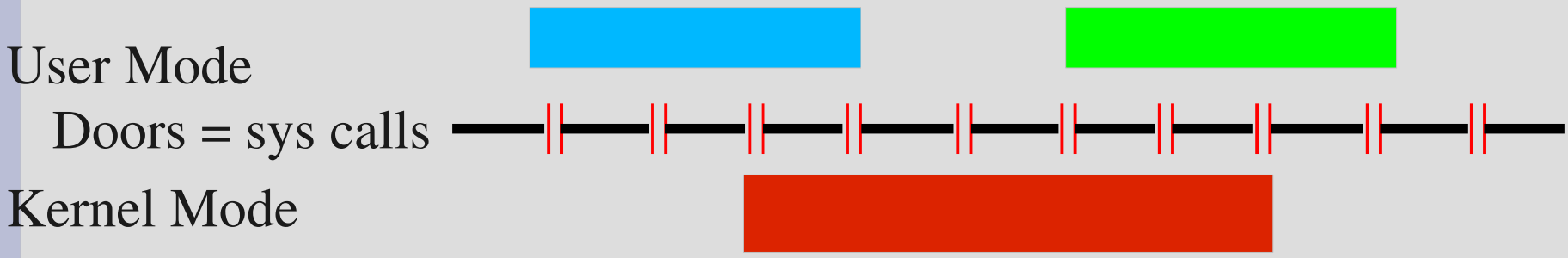
# Memory Isolation : MMU

- Memory Management Unit
  convert logical <-> physical address
  - use memory Paging
  - every process owns its “page table”
- page handlers (call back per page types / markers)
  - ex: access to page for exe, heap, stack => ok
    - access to page “0”, or unallocated => “page fault”
    - access to swap => transparent read/write to disk



# Process Isolation : System Calls

- mechanism to go out out the MMU sandbox... execute code with ALL read/write access
- System Call steps:
  - put system call code number in special register
  - put extra args in other special registers
  - call interruption
  - => switch to ring 0, and execute system callback
  - read returned values in special registers



# System Calls in Libc / Java

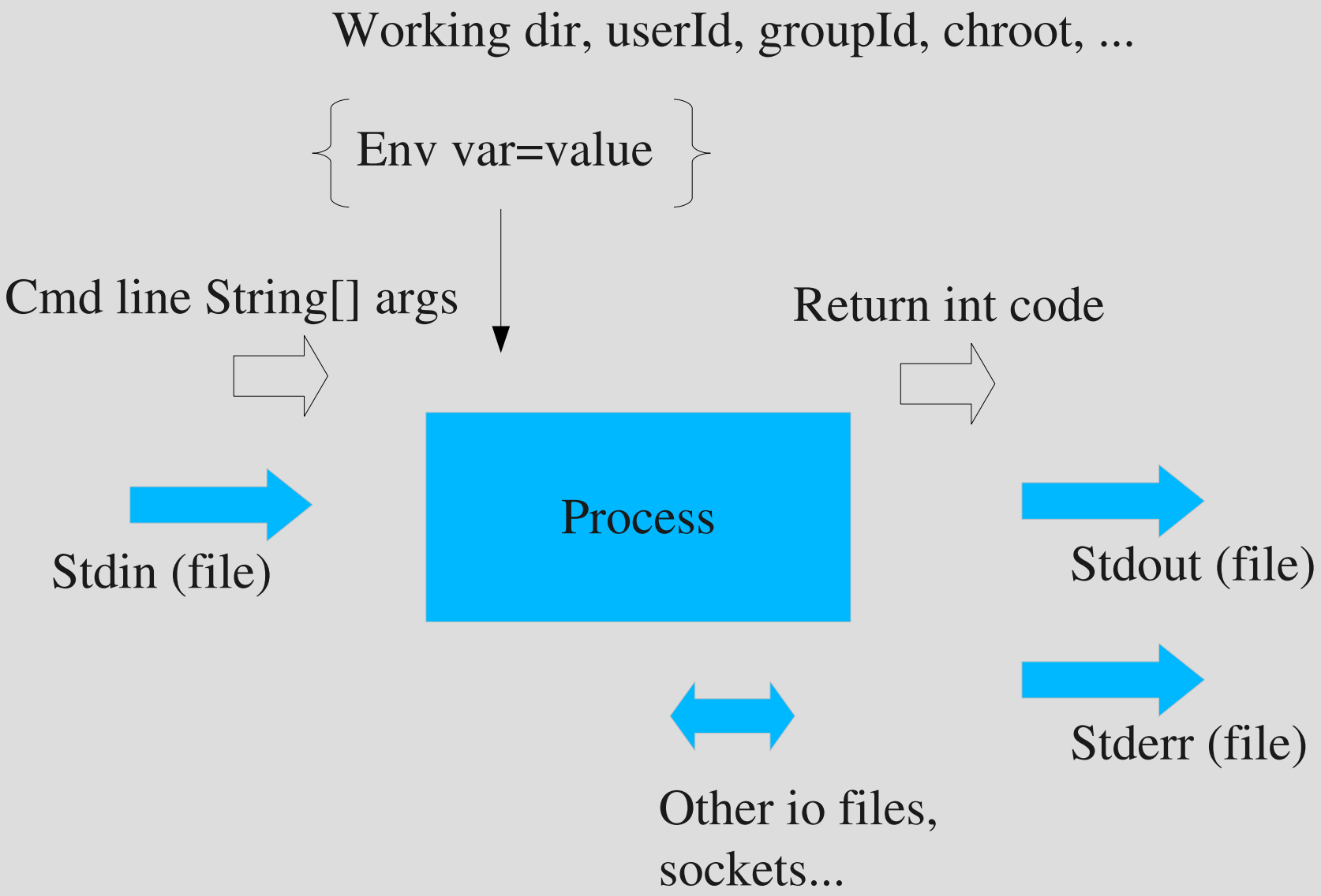
- painful to write low level assembly code reused the “libc”...
  - 1 system call <-> 1 C function in libc
  - libc also has high-level helper functions
- In Java, most System calls of interest are wrapped in a native method
  - 1 C system call <-> 0..1 java native
  - JVM = portability: only smallest common denominator of os

# Process/Thread Definition

- Definition: a process is a task with a (mmu) page table ... isolated from other process
  - Thread : when sharing the same page table!!
  - In Linux... no real difference between process and thread (both called task)!

- Other Process Resources:
  - File Descriptors, Environment variables, ...
  - Working dir, chroot, ...

# Graphical Process Representation



# Launch Process = Fork + Exec

- Pid “1” = process “init” (boot)
- Start a new one = **“fork”**
  - parent continue... child = clone of parent
  - Child inherit all from parent (files, env, etc...)
- Switch executable : **”exec”**
  - child mute to be different of parent
  - Child can close files, add setenv, ...
- Not a real parent-child tree:
  - Child with parent killed = “orphan”

# Process Admin Tool

# Pstree -p
# Ps -aux
# kill -9 $pid

Terminal

File Edit View Terminal Tabs Help

```
arnaud@laptop:~/devPerso/CoursIUT$ pstree -p
init(1)-+-acpid(2780)
        |-battstat-applet(3326)
        |-bonobo-activati(3275)---{bonobo-activati}(3276)
        |-cron(3120)
        |-cupsd(2974)
        |-dbus-daemon(2820)
        |-dbus-daemon(3236)
        |-dbus-launch(3235)
        |-famd(3008)
        |-gconfd-2(3238)
        |-gdm(2849)---gdm(2852)-+-Xorg(2855)
        |                       `-x-session-manag(3167)-+-gnome-panel(3267)
        |                                               |-gnome-settings-(3249)---{gno
        |                                               |-metacity(3265)
        |                                               |-nautilus(3269)
        |                                               |-seahorse-agent(3244)
        |                                               |-ssh-agent(3231)
        |                                               |-ssh-agent(3232)
        |                                               `-{x-session-manag}(3250)
        |-getty(3146)
        |-getty(3147)
        |-getty(3148)
        |-getty(3149)
        |-getty(3150)
        |-getty(3151)
```

Java proc admin:
# jps
# jstack
# kill -3 $pid  (send signal != “kill”)
# jconsole, jvmstat

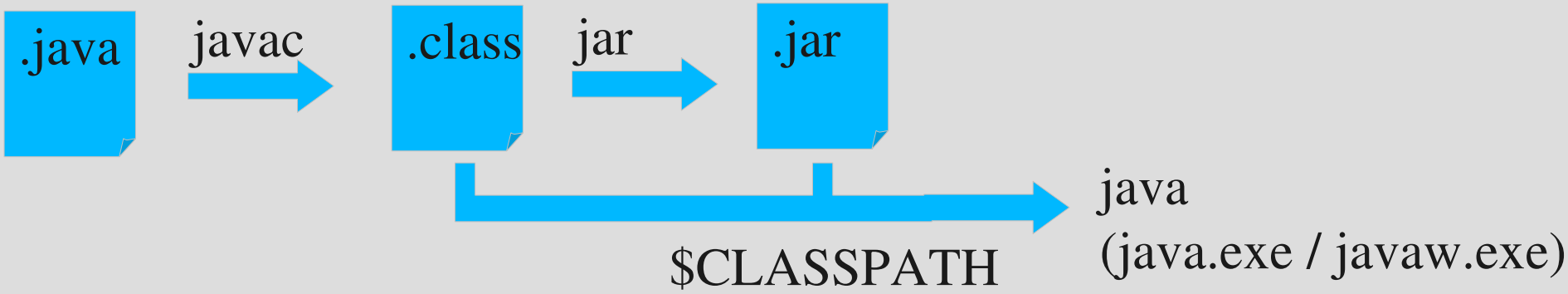
# Start/Stop Process&Thread in Java

- Start Process
  java.lang.System.exec(“prog”);
  - => fork + exec!
  - asynchronously .. can wait for exit

- start Thread: new Thread(runnable).start();
  - ... not recommended!
  - Either use SwingWorker (ui code),
  - or use j2ee threads (thread pool, workmanager, ...)

- stop Thread: other.setInterrupted();
  ...if (curr.isInterrupted()) throw new Excep();

# Launch Java Main Program




- # myjre/bin/java
  -cp $CLASSPATH ... for jars files
  -Xargs ... for jvm settings
  -Dvar=value ... for env vars
  fr.iut.MainClass ... the main!
  Arg1 ... argN ... main args

# Jvm Args -Xmx... -XX..

- Typical arguments:
  - java -Xmx512m -Xms100m -X
- see doc:
- # java -help,  java -X,  java -XX:...

  -Xms<size>  set initial Java heap size
  -Xmx<size>  set maximum Java heap size
  -XX:MaxPermSize=<size> ... classloader size

# System Env Variables

- set value
  - # export VAR=”VALUE”
  - ... ou  #VAR=”VALUE”; export VAR
  - VAR=”VALUE” : “locale” variable for shell
- get value ... both shell and env value!
  - $VAR ou ${VAR}
  - In C: getenv(“VAR”);
  - In Java: “java.lang.System.getProperty(“VAR”)

# Main Argument Evaluation

- 1 String line => N args : main(String[] args)
  - Split (tokenize) using whitespaces
  - Eval ${} shell variables
  - Eval *, ? as file regexps
  - Sub-eval `cmd`, $(cmd), $((expr))
  - Protect chars with \, ' ', and “ “

# Protecting Whitespaces Args

- “ “ and ' ' : both to avoid splitting args ws

```
# Cmd a b        => args[2]: { “a”, “b” }
# Cmd “a b”      => args[1]: { “a b” }
# Cmd a\ b       => args[1]: { “a b” }
```

- Difference “ ” vs ' ' ?
  ... in “ “, $ are evaluated

```
# Cmd “$a b”     => args[2]: { “123”, “b” }
# Cmd '$a b'     => args[1]: { “$a b” }
# Cmd \$a\ b     => args[1]: { “$a b” }
```

# Sub-Cmd Eval Details

- 3 forms to evaluate a string, and get result:

```
# eval cmd args
# `cmd args`
# $(cmd args)
```

- $() is non ambiguous, compared to eval or ``
  - ex: ``a`` => $()a$() or $( $(a) ) ??!!

- example:
  - # res=`cat file.txt | wc -l`
  - # echo “file has ${res} lines”

# Eval Arithmetic Expressions

```
# $((expr))            ... for arith expressions
# test arg1..argN      ... for boolean test
# [ arg1..argN ]       ... idem!
```

... /usr/bin/[ is an alias for /usr/bin/test !!!
... strange magic for using “if [ cond ]”

- Example:
  - # echo “diff $(( ${res} - ${comp} ))  lines”
  - # if [ ${res} > 5 ]; then echo “$res >5”; fi
  - # for ((i=0; $i < 10; i=$i + 1 ));do echo $((i * 2));done

# Difference Eval / Exec Sub-Shells

- file with header line
  #!/bin/bash
  => recognized by shell as executable
  (with “chmod u+x” )

- to exec : shell fork a sub-shell
  # ./myscript.sh
- to source = do NOT fork a sub-shell:
  # source myscript.sh
  # . myscript.sh <= using DOT as shortcut

- ex: using # . setenv.sh
  (..var not lost in forked shell)

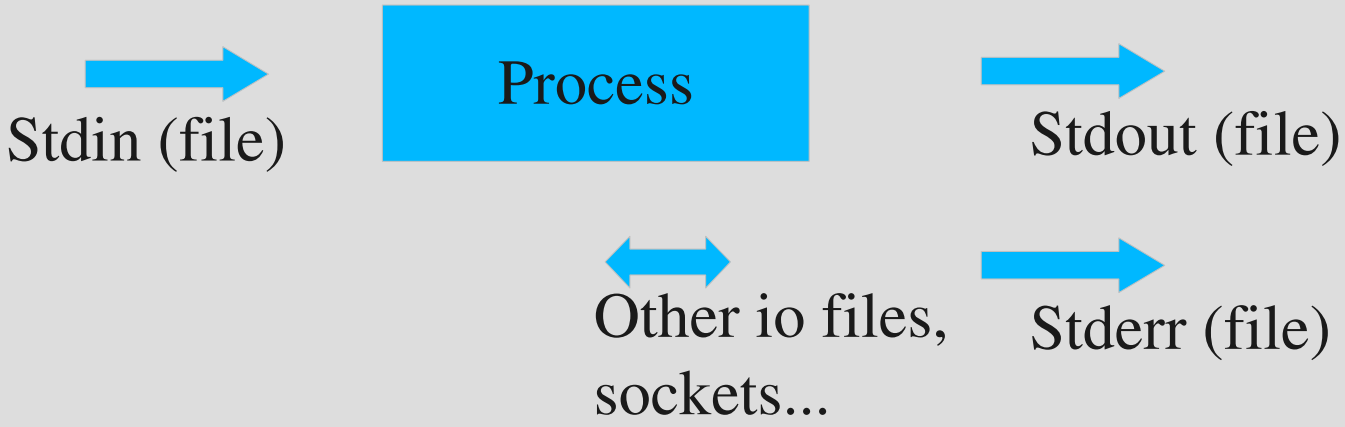
# Next Part: Stdin / out / err Redirections

- 5 minutes exercises break

- Next Sub Part:
  - FileDescriptors and forking
  - Shell stdin, stdout, stderr redirection < | > >>
  - Unix Bin Utilities for in-out: echo,cat,xargs...

# Process Std In / Out / Err

- Process have an array of open files
- Fork process => fork file descriptors
- In particular, processes have 3 std files:
  - FILE[0] = stdin (~keyboard device / file)
  - FILE[1] = stdout (~console/file, for logs)
  - FILE[2] = stderr (~console/file, for errors)



# Shell Std in/out/err Redirection

- Keywords for redirecting: <, |, >, >>, 2>&1
  - # cmd < file.txt
    process can read on stdin the content of “file.txt”
  - # cmd > file.txt
    the process can write... to file.txt (overwrite)
  - # cmd >> file.txt
    - to append to file.txt
  - # cmd1 | cmd2
    output of 1 = input of 2

# Shell File Redirections

- How it works ? ... after forking itself
  - the shell open/close/ioctl files differently in parent and forked child !
  - Then call exec for the child

- Ex:
  - < f.txt : child.FILE[0]= fopen(“f.txt”, “r”);
  - > f.txt : child.FILE[1]= fopen(“f.txt”, “w”);
  - >> f.txt : child.FILE[1]= fopen(“f.txt”, “a”);
  - 2> f.txt : child.FILE[2]= fopen(“f.txt”, “w”);
  - 2>&1 : child.FILE[2]= child.FILE[1];

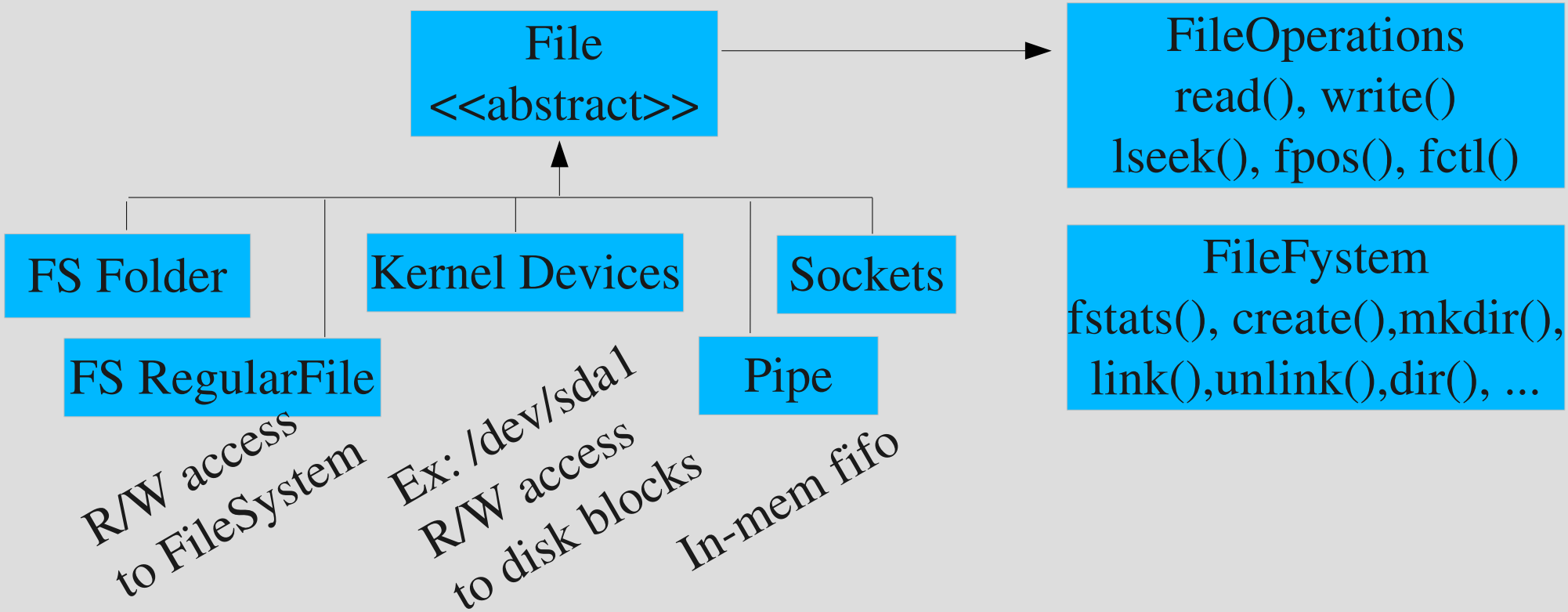
# Pipes Redirection

- # cmd1 | cmd2
  the output of cmd1 ... is the input of cmd2
  pr,pw = pipe();  // see also mkfifo
  cmd1.FILE[1] = pw;  cmd2.FILE[0] = pr;
- A pipe is a special “file”
  - Internally : a fifo  (in memory buffer of 4ko)


- Process synchronization:
  - Cmd1 is blocked on writing when pipe is full
  - Cmd2 - reading - empty

# Unix Unified File Descriptors

- Legend: “On Unix, everything is a file”
- Most commands do simple things (kiss), with
  - Input = stdin or filename(s) as arguments
  - Output = stdout or filename(s) as arguments



# Usage Example of Redirections

- Most unix commands are silent (!=DOS)
  - only result goes to stdout (or use -v)
  - errors are logged to stderr

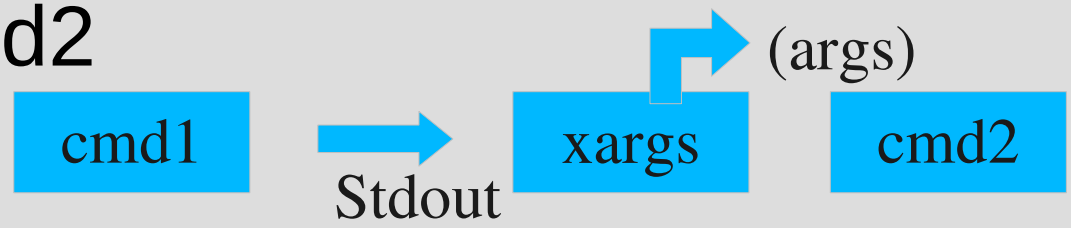
- Ignore stdout logs:
  - # cmd > /dev/null
- Redirect stderr to stdout, both to file
  - # cmd 2>&1 > log.txt
- duplicate stdout to console and file
  - # cmd | tee log.txt | less

# Bin Utils for cascading in/out: cat, echo, xargs, eval

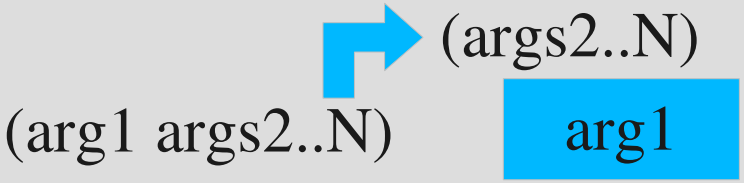
- # **cat** file.txt


- # **echo** “arg1 .. argN”


- # cmd1 | **xargs** cmd2




- (reminder) # **eval** cmd, `cmd`, $(cmd)



# Other Bin Utils for in/out: grep, sed, cut, tail, head, awk...

- # grep patttern [file] ... select matched lines
- # grep -v pattern [file] ... select all but lines

- # sed 's/replaceThis/byThat/g'
  ... replace per lines

- # cut -f1 -d: ... extract column 1, separated by ':'

- # head -N : select N first lines
- # tail -N : select N end lines

- # awk, perl, python, java... for more

# Bin Utils for Interactive in/out

- # less  ... is more
- # tail -f file ... realtime reader (see also itail,mtail)

- # CNTRL+S / CNTRL+Q
  suspend / resume console inout
- # cat >> file.txt  ... type ... CNTRL+D
  write to file from console, without text editor

- # read var...  for reading/prompting in scripts
- # yes, expect  ... automatic read/answer to interactive cmds

# Next Part: Exec, Dynamic Linking

15 minutes exercises break

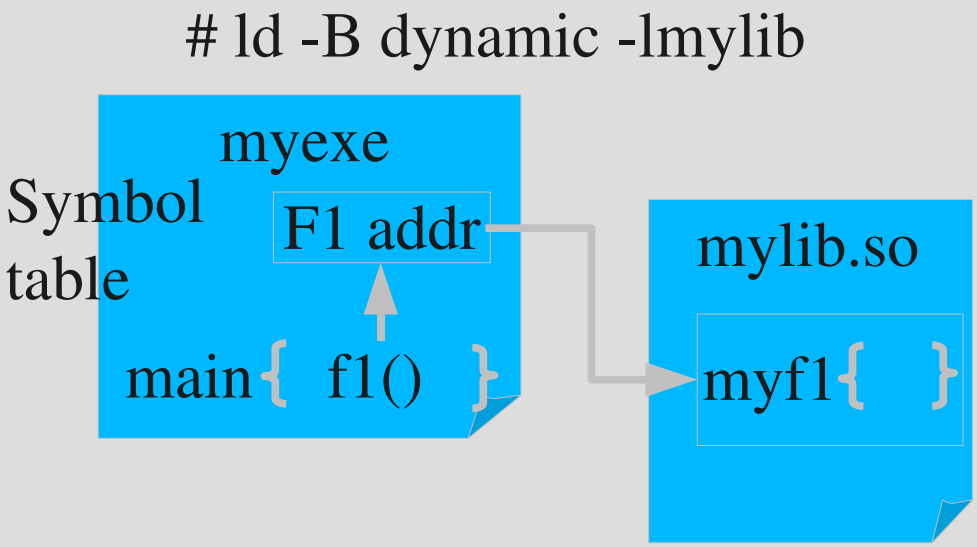
- Next Sub Part:
  - PATH, resolving exe filename
  - Dynamic Linking .so LD_LIBRARY_PATH
  - Ldd, advanced code injection, ltrace
  - JNI, native methods

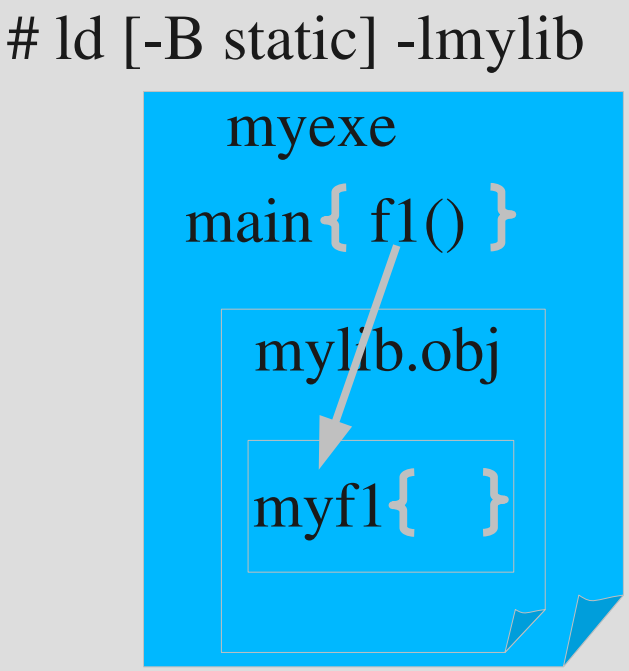
# Shell Process Resolution : $PATH

- $PATH is a built-in env var for shells
- # export PATH=/usr/bin:/usr/local/bin:~/scripts
  - ... tell the shell where to find exec in dirs
- # which filename
  - ... ask the shell where it would find exec
- Tip:
  - Usually `pwd` not in $PATH  ... too dangerous
  - To force finding exec in pwd: # ./myexe

# LD_LIBRARY_PATH, Dynamic Linking

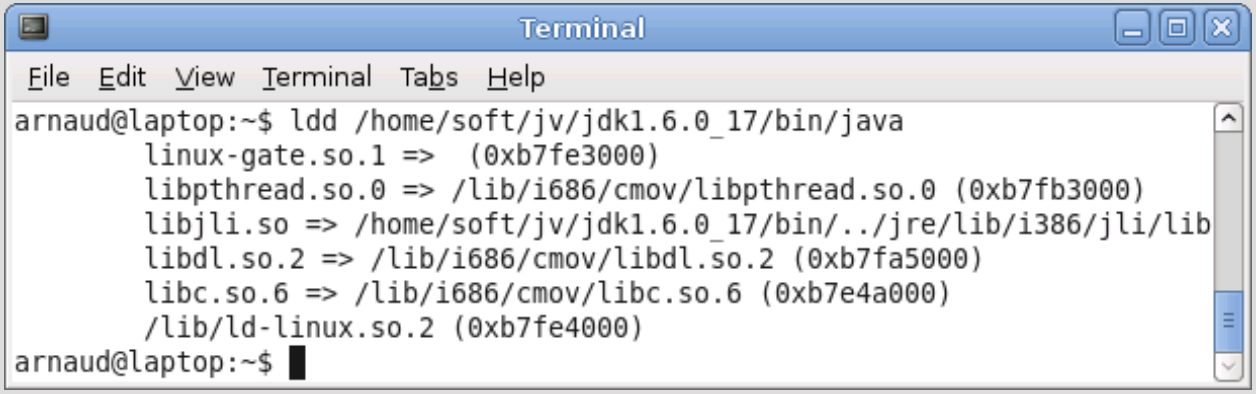
- $LD_LIBRARY_PATH ... like $PATH, but for finding shared libraries (.so), not exe
- Dynamic Linking versus Static Linking



# Dynamic Linking, ldd

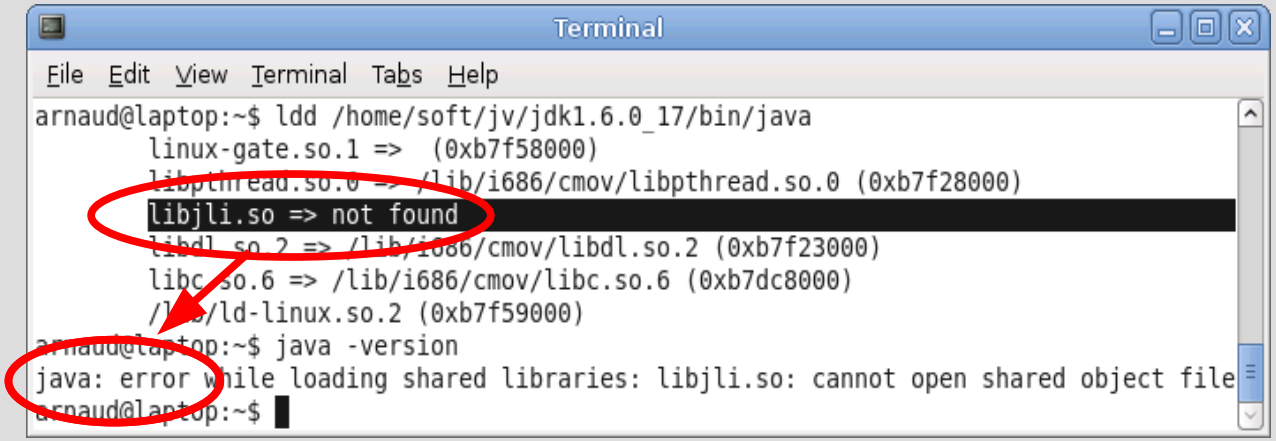
- Pros:
  - Library Code not HARD-linked with exe
  - Can upgrade library
  - Exe files size much smaller
  - Can inject/replace code, call code at runtime
- Cons: more complex dependency, slower?
- Dump:
  - # ldd myexe
  - # nm myexe



# Exec Internals with Dynamic Links

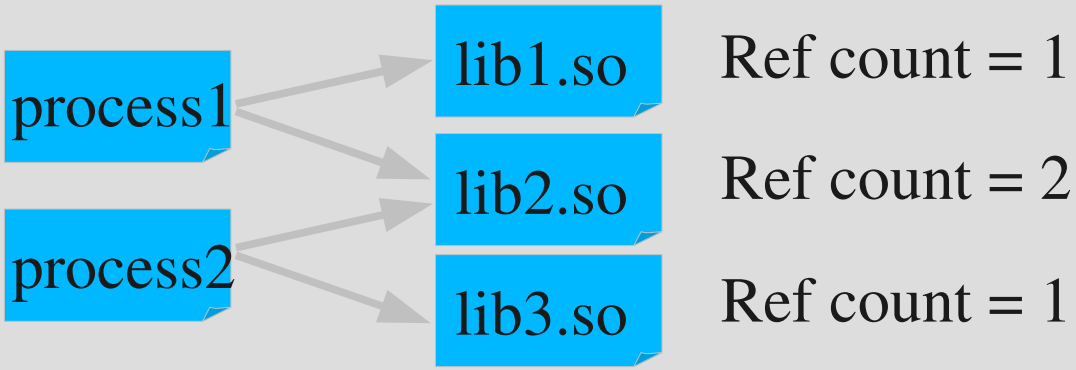
- At exec time,
  - .so are resolved and linked  ... like # ldd
  - Load .so libraries files
  - Linked : real address in symbol dispatch table

- If not found => exec failed..



# Memory Mapping .so, Sharing Maps

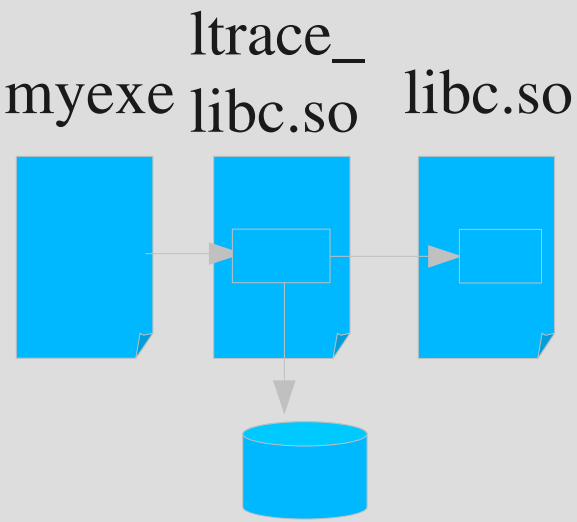
- Mapping file in memory
  - File content locked to RAM, “read lazily” (cf MMU, memory pages, callback handler)
- Ref Count optimization
  - File already mapped => simply incr counter !!
  - Mapped as Read-Only (or “clone on write”)
  - Save RAM ... Save Time to exec
- # ps ... process resource usage: size != vsize

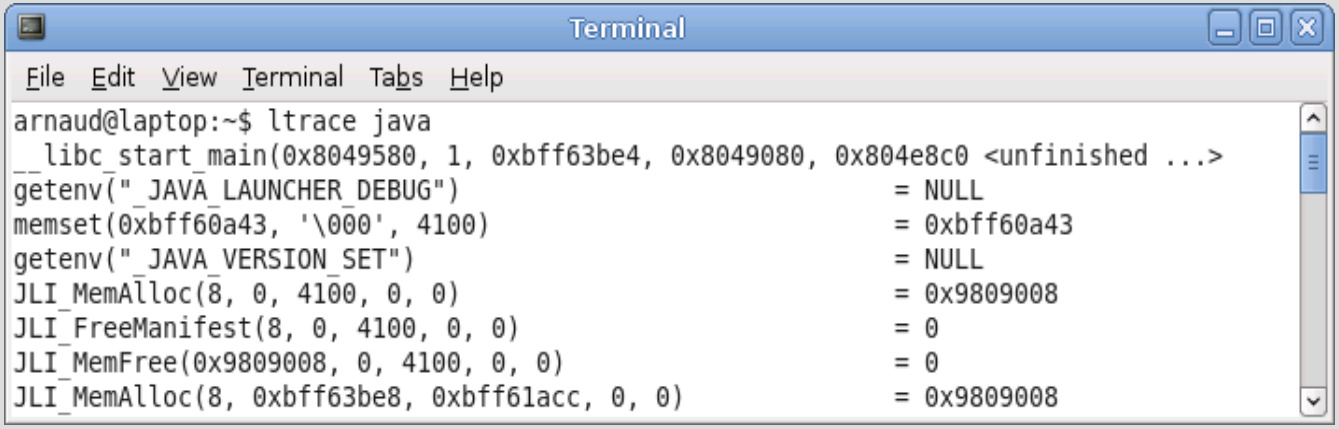


# Advanced Dynamic Link Usages
# Code Injection, ltrace

- Override $LD_LIBRARY_PATH for 1 process => change its bindings

- Ex: ltrace (see also strace for syscalls logs)
  - Override “libc”, and rebind with proxy loggers
  - # ltrace myexe args



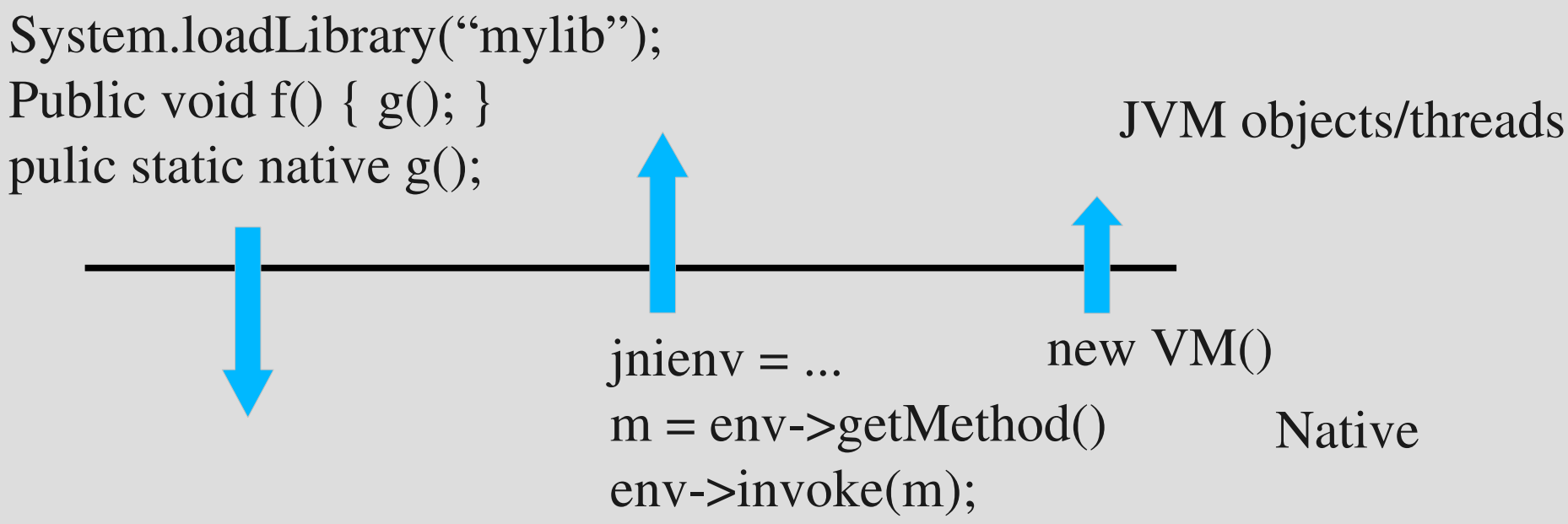



# Dynamic Link for RT Execution

- dynamic reflection (execution) is magic:
  - In shell, call “# eval cmd”
  - In java: call “Method m = ...; m.invoke();”
- In C ... More complex, but possible
  - “mmap(); f = &...; (*f)();”
  - System calls: mmap, exec ...
  - Lookup &.. from symbol name (like nm)
  - Libelf : utility library to read/write ELF files (analogy : bcel to read/write .class files)

# JNI : Java Native Interface

- JNI contains 3 mecanisms:
  - Calling native code from java code
  - Calling java code from native
  - Starting embedded jvm in process



# Example of JNI Usage

- Wrap OS specific system calls
  - Link, Symbolic link, mount, inotify, ...
    ... not supported in poor os => not in jvm!
- Use real Java UI : SWT (+Jface + Eclipse RCP)
  - Swing “lightweight” ui components is ugly:
    home made look&feel => not respecting os, not homogeneous with others languages!
- Swig (Simple Wrapper Generator)
  migrate simply .h files to java natives

# More on linux,bash,java...

http//www.google.fr

http//wikipedia.fr

... or rtfm

```
# man bash
# less HOWTO/*
```

... or read sources

```
~/jdk/src.zip
~/jsdk/sources/...
```

# Questions

Questions ??

arnaud.nauwynck@gmail.com

This document: http://arnaud.nauwynck.chez-alice.fr/devPerso/Pres/Pres-System-Shell-Java.pdf