Operating Systems Labs Environment

http://d3s.mff.cuni.cz/osy

CHARLES UNIVERSITY
Faculty of Mathematics and Physics

Department of Distributed and Dependable Systems

Vojtěch Horký

horky@d3s.mff.cuni.cz

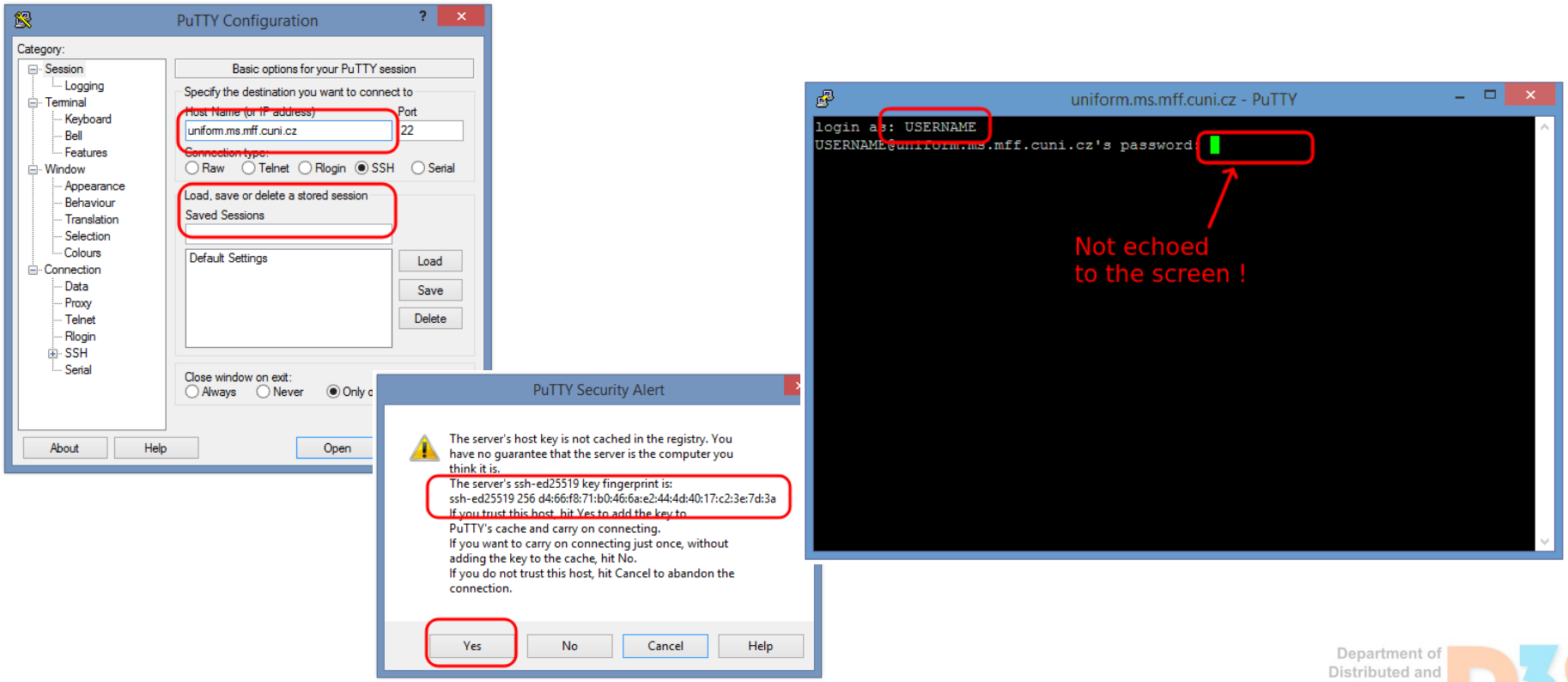
Platform Used In the Labs

- **MIPS 32b on R4000 CPU**
- **Cross-compiler toolchain for MIPS**
  - GCC, binutils
- **MIPS R4000 simulator**
  - MSIM
- **Kalisto**
  - Educational kernel for MSIM/MIPS R400
- **uniform.ms.mff.cuni.cz**
  - Linux machine with prepared environment (toolchain, MSIM, ...)
  - You should have received e-mail with account information
  - Used for in-labs submission

Department of Distributed and Dependable Systems

uniform.ms.mff.cuni.cz

- Fedora server with environment for the labs
- Check login there NOW (use PuTTY)



MIPS R4000 Platform

MIPS R4000: 64bit RISC processor

- **Basic features**
  - Load/store instruction set model
  - 32-bit and 64-bit operation, little and big endian
  - Fixed instruction length (32 bits)
  - Orthogonal instruction set, explicit stack management
  - Simple pipeline processing
  - System Control Coprocessor (CP0)
- **Registers**
  - 32 general-purpose registers (32/64 bits wide)
    - Almost orthogonal, usage defined by ABI
  - Special registers
- **Memory management**
  - Virtual address space divided into hard-wired segments
  - TLB-only paging (TLB managed by OS)

Department of Distributed and Dependable Systems D3S

MIPS System V ABI (o32)

- **Application Binary Interface**
  - Set of conventions defining machine code interoperability
    - Hardware, compilers, assemblers, linkers, libraries, operating systems, etc.
- **MIPS hardware & System V ABI register designation**
  - R0 is hard-wired to zero
  - R31 is a link register
  - R26 (K0), R27 (K1) reserved for kernel use
  - R28 (GP) global pointer
  - R29 (SP) stack pointer (stack grows towards lower addresses)
  - R30 (FP) frame pointer
  - R4 (A0), R5 (A1), R6 (A2), R7 (A3) first 4 integer function arguments
    - Additional arguments on the stack, but the stack space is always reserved
  - R2 (V0), R3 (V3) integer function return values
- **Special registers**
  - PC is program counter
  - LO, HI store the results of multiplication and division

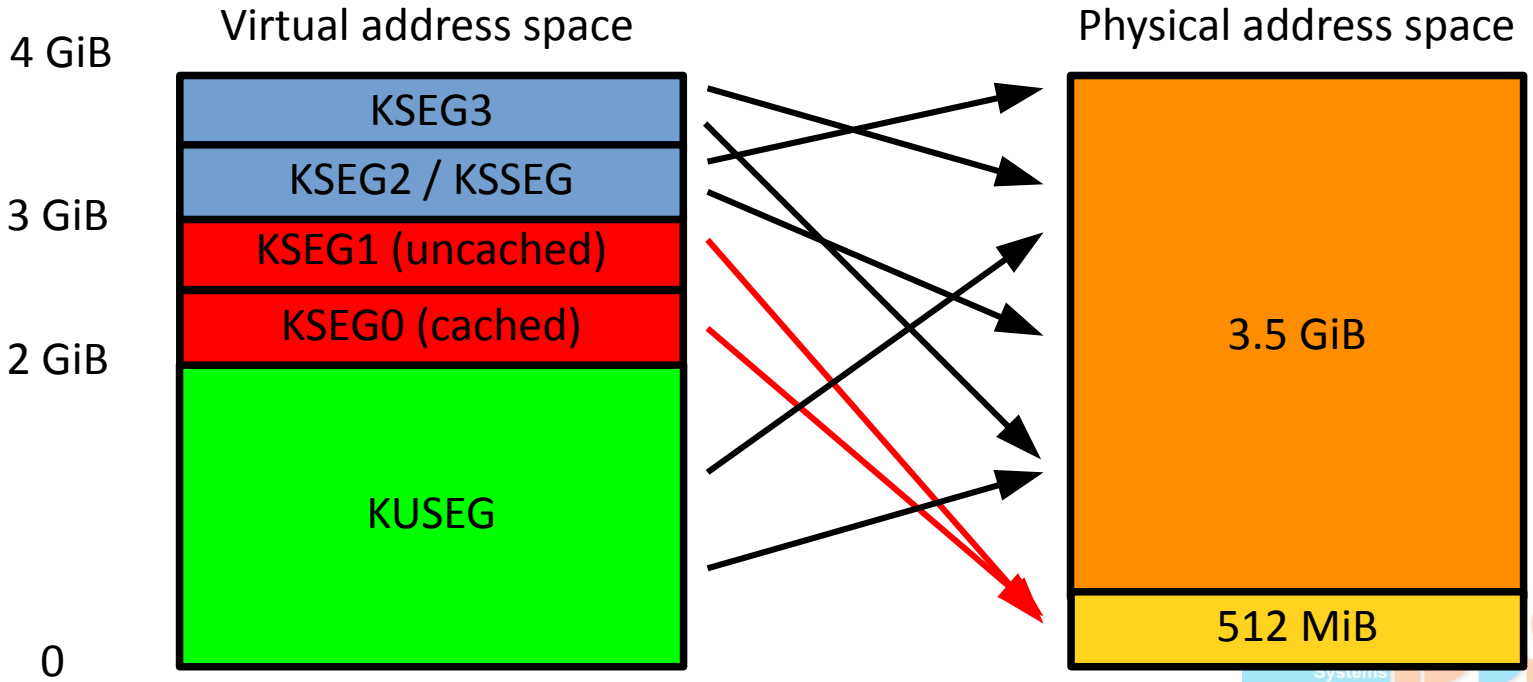
MIPS R4000 Memory Management

- **Virtual address space**
  - Hard-wired segments (top 3 bits)
- **Translation Lookaside Buffer (TLB)**
  - Software-managed
  - 48 records
  - Variable page size (from 4 KiB up to 16 MiB)

unprivileged, TLB mapped

privileged, identity mapped

privileged, TLB mapped

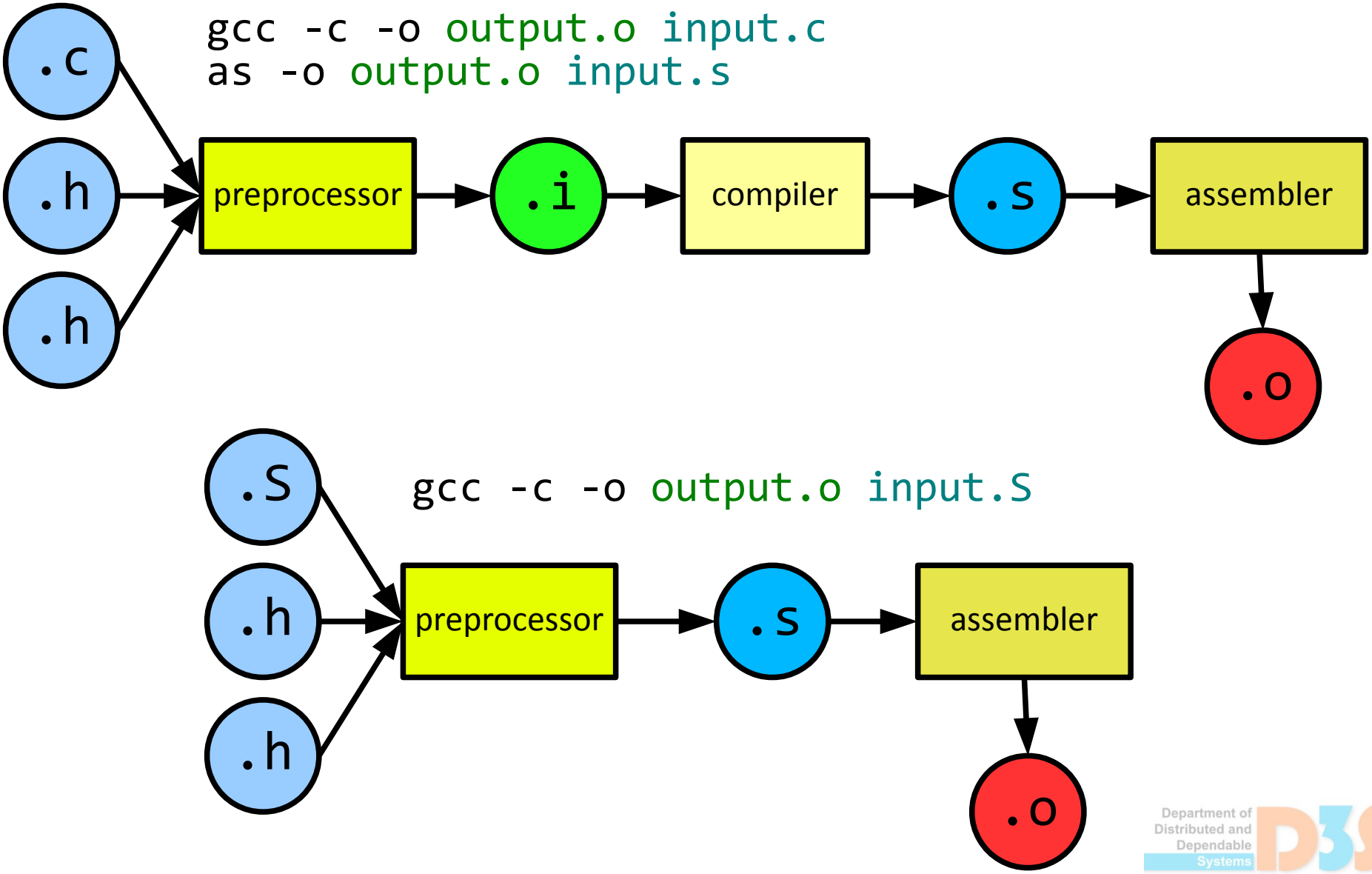


Cross-compiler Toolchain

Cross-compiler Toolchain for MIPS

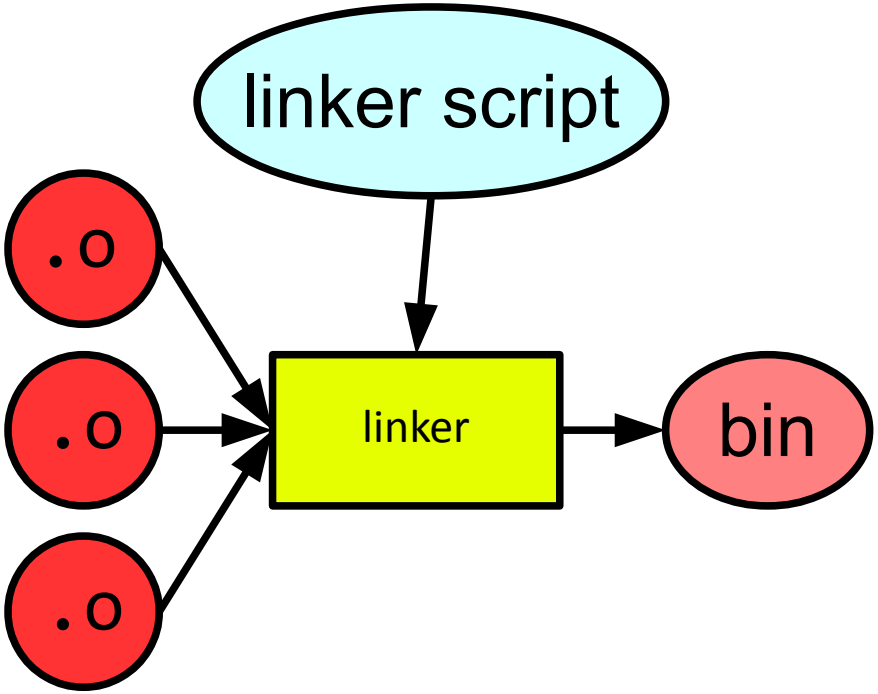
- **Cross-compiler**
  - Generates code for a different platform than it runs on
- **Components**
  - GNU GCC
    - The actual C language compiler
  - GNU binutils
    - Assembler, linker and other supplementary utilities
  - GNU GDB
    - Debugger

Compiler Toolchain



Object Linking

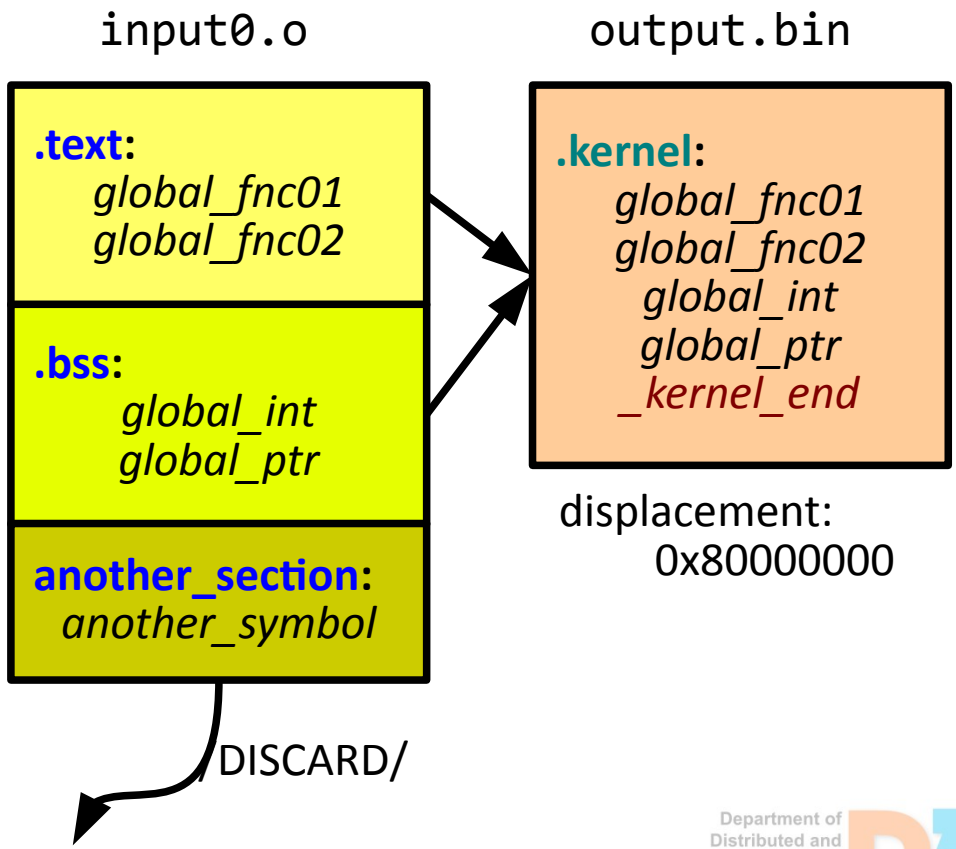
```
ld -T link.ld -o output.bin input0.o input1.o
```



Linker Script

```
OUTPUT_FORMAT(binary)

OUTPUT_ARCH(mips)

SECTIONS {

      .kernel 0x80000000 : {

            *(.text)

            *(.data)

            *(.rodata .rodata.*)

            *(.bss)

            *(COMMON)

            _kernel_end = .;

      }


      /DISCARD/ : {

            *(another_section)

      }

}
```

Symbol accessible from C code but placement (after all the code) specified in the linker script.



MSIM Emulator

MSIM 1.3.5 – MIPS R4000 Simulator

- Simplified (but faithful) model of MIPS R4000 CPU
  - Only 32 bit operation
  - No FPU, no CPU cache emulation
- Peripheral hardware
  - Extremely simplified, does not follow any actual hardware
  - RAM and ROM memory, keyboard, console, timer, disk
- Basic command-line debugging features
- http://d3s.mff.cuni.cz/~holub/sw/msim/

Department of Distributed and Dependable Systems

MSIM Basic Usage

- **Invocation**

  `msim [-c <config file>] [-i] [-t]`

- **Basic configuration/prompt commands**
  - `add` add device/memory
  - `dd` dump configured devices
  - `mbd` dump memory regions
  - `set` set internal control variables
    - `iaddr`, `iopc`, `icmt`, `iregch`, `ireg`, `trace`

MSIM Basic Usage (2)

- **Basic configuration/prompt commands** (2)
  - `step [n]` execute *n* instructions
  - `continue` execute until stopped
  - `md` memory dump (physical address)
  - `id` instruction dump (physical address)
  - `stats` dump run-time statistics
  - `echo` echo a string
  - `help`
  - `quit`
- **Non-standard debugging instructions**
  - DTRC, DTRO, DINT, DRV, DHLT, DVAL

Sample MSIM Configuration

```
add dcpu cpu0
add rwm main 0
main generic 16M
add rom bios 0x1fc00000
bios generic 32k
bios load “bios.img“
add dprinter output 0x10000000
add dkeyboard input 0x10000008 3
cpu0 info                          stat tlbd md id
output redir “dump.log“
```

Kalisto

Kalisto

- Simple educational operating system for MIPS R4000
  - All necessary functionality but no optimizations or advanced features
- Base implementation for all assignments
  - Some parts are distributed as precompiled object files .o to obscure implementation
- Mostly in C, several assembler files for low-level routines (context switch, bootloader, atomics, ...)

Compiling and running Kalisto

- **Default configuration (3 printing threads)**
  - `make` (compilation & linking)
  - `msim` (configuration from `msim.conf`)

```
This is Kalisto 0.9.11,
built by horky at 14:23:14 Oct  1 2018.
cpu0: Address translation ... OK
cpu0: Frame allocator ... OK
cpu0: Heap allocator ... OK
cpu0: Threading ... OK
cpu0: Scheduler ... OK
cpu0: Timers ... OK
cpu0: Disk ... OK
 [Thread 0] -------- …… ****** [Thread 2] **********
Creating user space process ...

User space: Hello world!

Cycles: 956433
```

Compiling and running Kalisto (2)

- **Running Kalisto self-tests**
  - Both kernel and userspace
    - Usually ends with `Test passed`…
  - Do not forget to `make clean` first
    - Needed for recompilation with different test
    - For incremental building `make` is sufficient
  - `make KERNEL_TEST=tests/basic/timer1/test.c`
  - `make USER_TEST=tests/thread/uspace1/test.c`
  - Notice how the path is specified
- **Running test suites**
  - `./tests-malloc.sh` (depending on lab topic)

Department of Distributed and Dependable Systems

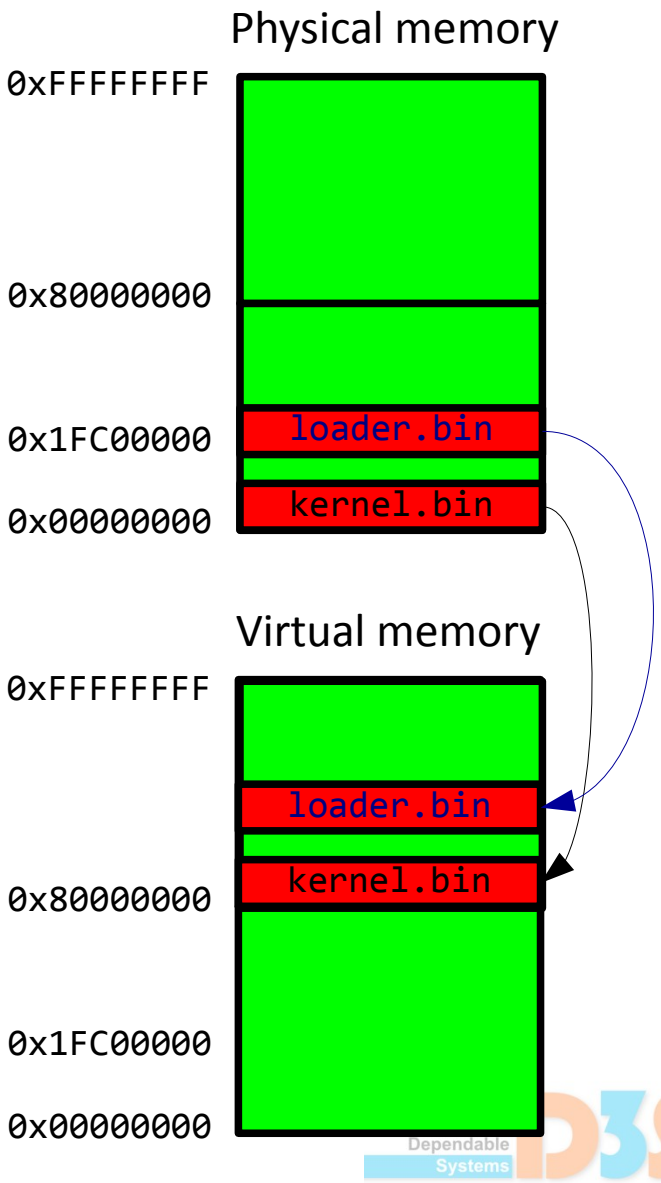
Kalisto Source Code

- **kernel/** – Kalisto kernel: scheduling, memory management, drivers etc.
  - **include/** – shared headers
  - **adt/** – data structure (list, bitmap, RB-tree)
  - **boot/** – bootloader
  - **drivers/** – screen, keyboard, disk, CPU enumeration
  - **exc/** – exception and interrupt handling
  - **lib/** – basic functions (`printk`, `memcpy` ...)
  - **mm/** – frame allocator, virtual memory management (paging, TLB)
  - **proc/** – threads and processes
  - **sched/** – scheduler
  - **synch/** – semaphore, mutex, conditional variable
  - **time/** – kernel timers
  - **tests/** – kernel tests
- **user/** – Kalisto userspace: libc etc.
  - **librt/** – basic userspace C library
  - **tests/** – userspace tests
- **contrib/toolchain.mips.sh** – cross-compilation toolchain setup

Department of Distributed and Dependable Systems

Kalisto Memory Layout

- **Kernel**
  - `kernel.bin`
  - Loaded into RAM starting at physical address `0`
- **Loader (firmware)**
  - `loader.bin`
  - Loaded into ROM at physical address `0x1FC00000`
  - Jumps to kernel entry point



Kernel Initialization

- `kernel/main.c`
  - `bsp_start()`
    - Global kernel initialization (on the bootstrap CPU)
      - `tbl_init()` setup of the TLB
      - `frame_init()` frame allocator setup (physical memory size detection)
      - `heap_init()` kernel heap allocator setup
      - `threads_init()` thread support setup
      - `scheduler_init()` scheduler setup (run queue setup)
      - `timers_init()` timers setup
      - `disk_init()` disk driver setup
      - Creation of the idle thread
      - Creation of the main kernel thread (with `example()` as the thread body)
      - Activation of the next CPU (atomic variable)
      - Context switch to the main kernel thread
        - The kernel starts scheduling threads
  - ap_start()
    - Local kernel initialization (on application CPUs)

Useful Functions

- `printk()`
  - Simplified kernel implementation of `printf`
    - Only basic modifiers are implemented (%d, %p, %s, …)
- `dprintk()`
  - Debugging version, prints calling function and source line
- `assert()`
- `msim_stop()`
  - Enter interactive mode of the simulator
- `msim_reg_dump()`
  - Dump CPU registers
- `msim_trace_on()`
  - Trace executed instructions
- `msim_trace_off()`
  - Stop tracing executed instructions

Department of Distributed and Dependable Systems D3S

Using Lists

```c
#include <adt/list.h>

/* Structures that are in a list. */
struct my_struct {
    link_t link;
    int value;
}

/* List declaration and initialization. */
static list_t my_list;
list_init (&my_list);

/* Adding to a list. */
struct my_struct *x = malloc(sizeof(struct my_struct));
link_init (&x->link);
x->value = 42;
list_append (&my_list, &x->link);

/* Iterate through items, it points to my_struct */
list_foreach (my_list, struct my_struct, link, it) {
    printk("Value is %d\n", it->value);
}
```

Using Unix Environment, C language, etc.

Using Unix, Knowing C, etc.

- **User knowledge of Unix at the level of NSWI095 (*Introduction to UNIX*) is sufficient**
- **C knowledge at the level of NPRG041 (Programming in C++) should be sufficient**
  - Just mind some differences between C++ and C
    - No STL, no classes and namespaces, no RTTI and exceptions, no streams and overloading
- If you think otherwise
  - There are extra slides available on the web with short recap
  - Or contact us for advice (the sooner the better)

Power-on Self-test

Verify You Understand the Basics

- Complete the following pseudo-test before coming to next labs.

  https://docs.google.com/forms/d/e/1FAIpQLSeMJpkExgkrq4LGH3sUj-Ks9n5E4-ETZZsC9S0ATl-k1nH2Dw/viewform

- It is not mandatory but unless you are well versed with kernel development, it is **highly recommended** to do it.

Q&A