Operating Systems

Tutorial 2 & 16

Michael Tänzer

os-tut@nhng.de
http://os-tut.nhng.de

Calendar Week 44

Outline I

Outline II

- User vs. Kernel Mode
- Memory Hierarchy
- Cache Organisation

6 Finish

About Me

Me, Myself and I

- Student of Computer Science (Diploma)
- 7th semester
- Interests:
  - Operating systems
  - Cryptography and security
  - Telematics

Theory Assignments

- Appear every two weeks
- 20 points each $\Rightarrow$ 120 points total
- Have a deadline (don't miss it), usually Wednesday at 12:00 (noon)
- Will be marked and returned by me (not always on time)
- You may work in groups but everyone has to hand in separately
- If you only want to get your results and leave, you may do so
- Otherwise please be patient until the end of the tutorial, when I will return them

Programming Assignments

#	Topic	Release	Deadline	Points
0	First Steps		None	0
1	Synchronisation	09.11.09	25.11.09	20
2	Memory Management	23.11.09	23.12.09	25
3	Syscalls	21.12.09	03.02.10	25

- You will work in pairs
- Register until 18.11.09 12:00 by sending an email to one of your tutors containing
  - Your names
  - Your matriculation numbers
  - A fancy nickname for your group
- Send your solution to the tutor chosen (compressed)
- We'll agree on a meeting where you show me what you've done

Programming Assignments

Theory

- Will not be marked
- Give an overview of the source needed for the assignment
- Basically you should do them but as you don't have to hand it in, you don't have to formulate the answer

Programming Assignments

Tips

- Do assignment 0 this will make sure you have a working system for the coming assignments
- Don't use `gdb` via sockets (as proposed in some eclipse tutorials on the net) it's way too slow
- Use a VCS (Version Control System) to allow concurrent programming without headaches
  - I personally recommend `git`
  - I'll write a short `git`-Intro which you'll find on the website (`http://os-tut.nhng.de` – once it's done)

‘Schein’ & Bonus Points

- You can get up to 190 points
  - 120 from theory assignments
  - 70 from programming assignments
- For a ‘Schein’ you need $\geq$ 100 points
- If you are a bachelor student you need the ‘Schein’ to pass the module
- If you are a diploma student you don’t need it
- Bonus points will be added to the points you get in your exam if you pass it $\Rightarrow$ if you fail the exam the bonus points can’t save you
  - $\geq$ 110 points $\Rightarrow$ 1 bonus point
  - $\geq$ 130 points $\Rightarrow$ 2 bonus point
  - $\geq$ 150 points $\Rightarrow$ 3 bonus point
  - $\geq$ 170 points $\Rightarrow$ 4 bonus point

Distro Wars

Now the fun part

- Your given name
- The Operating System and Distribution/Version you use most of the time
- For how long you've been using it

What is a Shell?

The shell is a CLI (Command Line Interface) for the Operating System

- In Linux the shell runs as a normal user programme
- Usually a command starts another programme which does the work
- The shell provides mechanisms to combine commands (pipes, control flow statements)
- And manipulate (environment) variables
- Allows manipulation of multiple files through wild cards
- Sophisticated shells (e. g. `bash`) also have convenience features like tab completion for almost everything

Basic Commands

`cd`	Change the working directory
`ls`	List the contents of a directory (like `dir` in Windows)
`mkdir`	Make a new directory
`rmdir`	Remove directory (only works if the directory is empty)
`cp`	Copy file or directory
`mv`	Move/Rename file or directory
`rm`	Delete a file
`cat`	Concatenate inputs to output
`less`	Scroll through input
`grep`	Print the line containing a specified pattern in the input
`xargs`	Execute a command with the input as parameters

How can you get help for a programme?

No, ‘Use Google’ is not the answer I’ve hoped for

- The command `man` shows the (hopefully complete and understandable) manual
- Sometimes more than one programme has the same name (e. g. there is the command, system call and function `exit`) then you also have to specify which section you want to see
- `man man` shows how to use `man` and which sections are available

How can you search for a file?

- Assume you want to edit a file named `syscall.c`, but you have forgotten in which subdirectory of our project it resides. What can you do?
- ```
  $ find ./ -name 'syscall.c'
  ./kern/arch/mips/mips/syscall.c
  $
  ```
  ```

Combine commands to get complex things done

- Assume you have a directory that contains multiple C source files in multiple subdirectories. How can you search for all occurrences of the macro `FOOBAR` in these C files?
- `$ find ./ -name '*.c'| xargs grep 'FOOBAR'`
  `./foo/bar.c:#define FOOBAR 42`
- `$ grep -r --include='*.c' 'FOOBAR'./`
  `./foo/bar.c:#define FOOBAR 42`

Creating Binaries

- What steps are necessary to create an executable program from multiple C source files?
  1. Compilation – that's `gcc`'s job, each file is compiled on its own $\Rightarrow$ multiple object files (*.o)
     - Object files contain machine code and not yet resolved symbols which reference to 'things' in other object files
  2. Linkage – that's what `ld` does, it resolves the references and puts everything together $\Rightarrow$ one binary
- Usually `gcc` will also call `ld` but you could give the `-c` option
- Manual linking is a little nightmare

What is `make`?

- `make` is an automated build tool (an ancestor of `ant`)
- The build process is separated into steps called targets
- Targets can depend on other targets
- Reduces overhead:
  - Targets will only run once per invocation of `make`
  - Sources will only be compiled if they were modified since the last build
- Executes `gcc` and other tools to do the dirty work
- If something doesn't work try a `make clean` followed by `make` to get a clean build

Example Makefile

```
coffee: water powder filter
        mv machine/water machine/filter/ && \
        mv machine/filter/water machine/coffee
water:
        touch machine/water

powder: filter
        touch machine/filter/powder
filter:
        mkdir machine/filter
clean:
        rm -r machine/*
```

Bit Operators

C	Description
&	Bitwise $\wedge$
|	Bitwise $\vee$
~	Bitwise $\neg$
<<	Right shift
>>	Left shift (filled with zeros)
^	Bitwise XOR

Why do we need bit operators?

- Why might it be necessary to set or clear a single bit of an integer value?
- Most hardware has control registers where each bit has a special meaning
- If one only needs to store one bit, why use more space? Bit maps, bit fields, etc.

Exercises

- How can you set the `i`th bit of a given integer `value`?
- `result = value | (1 << i);`
- How can you clear the `i`th bit of a given integer `value`?
- `result = value & ~(1 << i);`
- How can you retrieve the contents of bit 2 to bit 5 of a given integer `value`?
- `result = value & (0xf << 2);`

Difference between policy and mechanism

- Policies specify how a certain task should be fulfilled, optimised for certain goals.
- Mechanisms are used to realise those policies
- Example:
  - Task: Get from the Mensa to the ‘Info-Bau’
  - Goals: Short, fast
  - Mechanisms: Turn left/right, slow down, etc.
  - Policies: Minimum deviation from linear distance, minimal number of obstacles

Example II

- Assume you are the leader of a supermarket
- Task: How many cash points should be occupied?
- Goal: Minimal personnel costs, minimal waiting time for the customers
- Mechanisms:
  - Bell to indicate another cash point should be occupied
  - Icon on the cash display to indicate this cash point should be closed
  - Light barriers
  - Buzzers
- Policies: Have a minimum and maximum queue length, let the customers hit a buzzer if they wait too long

Enumerate the major tasks of an operating system

- Abstraction/Standardisation
- Resource Management
- Security/Protection
- Providing an execution environment for applications

Differences between a processor in user and kernel mode

Why are they needed?

- User mode:
  - Only unprivileged instructions may be executed
  - Otherwise Exception is thrown
- Kernel mode:
  - All instructions may be executed
  - Entered if an interrupt, trap or exception occurs
- Needed to protect applications from each other and make sure everyone gets his fair share

Typical examples for privileged instructions

Why are they privileged?

- Manipulate control registers for memory address translation
- Disable or enable interrupts
- Access privileged plattform devices

If they weren't user software could

- manipulate critical system state
- elevate its privileges
- overcome protection
- ruin isolation
- impact stability
- impact trustworthiness
- ...

How could parameters for syscalls be passed to the kernel?

- Registers
  - Fast?
  - Limited number of registers
- Stack (or possibly another memory location)
  - Slower?
  - More flexible
  - Less hardware dependent

Principle and Benefits

- High speed memory is expensive
- Only the currently needed data is kept in high-speed memory
- If there is no space left for new data, data that hasn't been used for a long time is moved down in the hierarchy
- This is done recursively for L1, L2, (L3), RAM and hard disk
- Most of the time data can be found in high speed memory without a costly pure high-speed system

What typical behaviour is utilised?

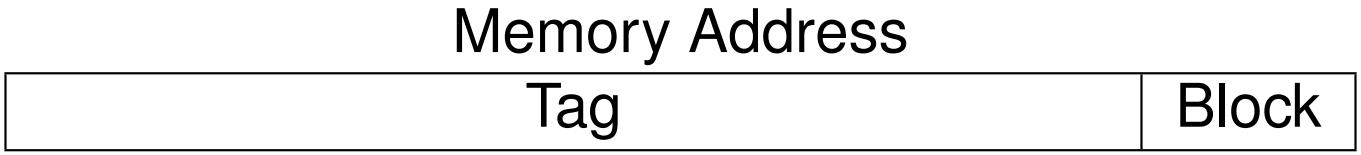
- Temporal locality: Data that has been accessed recently is likely to be accessed again (loops, etc.)
- Spatial locality: Data that is close together is likely to be accessed successively (arrays, etc.)
- Example: Stack

Why is the cache divided into cache lines?

Which size?

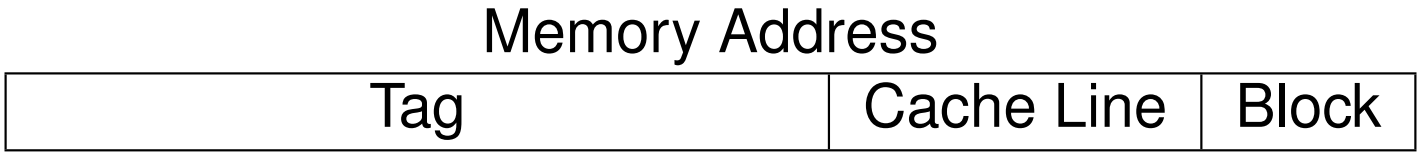
- Transferring big parts at one time (burst mode) is faster than transferring multiple little units
- Transferring parts that are never used is unneeded effort
- $\Rightarrow$ need to find the right balance

Fully Associative



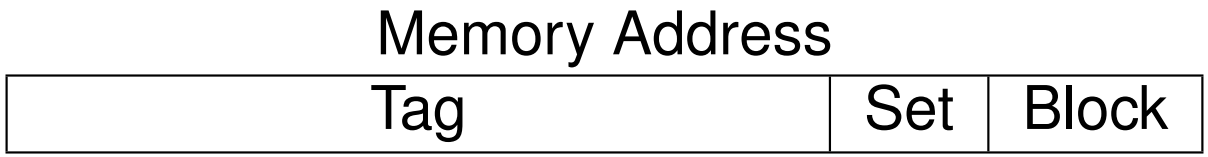
- Each memory block may be mapped to each cache line
- \+ Ideal utilisation of the given space
- – Lots of comparators needed (one per cache line) $\Rightarrow$ expensive

Direct Mapped



- Each memory block is mapped on the cache line corresponding to the lower bits of its address (without the bits used within each block)
- \+ Only one comparator needed
- – If more than one memory block that are mapped to the same cache line are in use they replace each other $\Rightarrow$ cache useless

n-way Set Associative



- Compromise of the other two
- Each memory block is mapped to a subset of n cache lines (like the direct mapped approach) inside these sets it's mapped like in the fully associative organisation

\+ n comparators needed

\+ It takes more than n memory blocks mapped to the same set to cause the blocks replacing each other

- $n = 1 \Rightarrow$ direct mapped, $n = m \Rightarrow$ fully associative

Questions & Comments

Any questions or comments?

A Quick Survey

Write on an anonymous piece of paper:

- At least one thing you liked
- At least one thing that could be improved about the tutorial

The End

The End