Systems Programming and Computer Architecture - Lecture

Prof. Timothy Roscoe
TA: Christoph Erdmann

Contents

The following was presented in the lecture on 18. September 2019.

2 Introduction to C

```
#include <studio.h> /* header file */

int main(int argc, char *argv[]) {
    printf("Hello World\n");
    return 0; /* Returning 0 indicates everything is ok */
}
```

2.1 Control flow statements

```
if (Expression) Statement_when_true
    else Statement_when_false

switch (Expression) {
    case Constant_1: Statement; break;
    ...
    case Constant_n: Statement; break;
    default: Statement; break;
}

return (Expression)

for (Initial; Test; Increment) Statement

while (Expression) Statement

do Statement while (Expression)

break

continue

goto Label
```

```
int i = 22;
const char s[] = "Yannick";
printf("My name is %s and I am %d years old.", s, i);
int a[3] = {3, 7, 9};
```

The following was presented in the lecture on 24. September 2019.

3 Integers in C

Note 1. If in C one value is unsigned in a comparison then both values get evaluated unsigned. ex. $-1 > 0U$ equals true

The following was presented in the lecture on 25. September 2019.

4 Pointers

Note 2. When the OS loads a program it creates an address space, inspects the executable file, copies regions of the file and finally does any linking or relocation.

Note 3. With &p one can print out the address of a variable.

Note 4. Dereferencing a pointer means to read the value behind it.

Note 5. Pointer arithmetic: If you add a value to a pointer it adds the value times the length of the type.

The following was presented in the lecture on 01. October 2019.

5 Dynamic memory allocation

Program explicitly requests new block of memory at runtime. C requires manual memory management. In C you can allocate memory with malloc and deallocate with free:

```
typedef unsigned long size_t;
void *malloc(size_t sz);
long *arr = (long *)malloc(10*sizeof(long)); // does not change the memory
if (arr == NULL) {
    return ERRCODE;
}
arr[0] = 5L;
long *arr = (long *)calloc(10, sizeof(long)); // zeroes the memory
if (arr == NULL) {
    return ERRCODE;
}
// Do your stuff
free(arr);
arr = NULL;
```

Note 6. One can also use realloc to get more or less memory.

```
struct Point {int x; int y;};
struct Point p1 = {0, 2};
struct Point p2 = {4, 6};
p->x // is nicer than
(*p).x
p1 = p2 // copies the entire content!
```

Note 7. One needs to pass a reference to a function to manipulate a struct, otherwise it creates a copy of the struct and manipulates the new one.

Note 8. Unions are like structs, but hold only one of a set of alternative values.

The following was presented in the lecture on 02. October 2019.

Note 9. One needs to save meta information to know where and how much free space there is

The following was presented in the lecture on 08. October 2019.

7 Implementing dynamic memory allocation

7.1 Fragmentation

For a given block, internal fragmentation occurs if the payload is smaller than the block size. This is needed to maintain the heap structure and for padding purposes.
It occurs when there is enough aggregate heap memory, but no single free block is large enough.
One has different options to store the header file: In the beginning of each free block with a pointer to the next free block, at the beginning of each block the length or a separate list with all the free blocks or to store the block in a sorted Red-Black tree.
There are different way of finding a free block: First fit, Next fit, Best fit. Start at the beginning of the list until found, start at the previous search finished until found, search list and choose a block which keeps the leftover bytes small.
While freeing a block it is important to check if the previous block or the next block and then we join the block so malloc can find the whole block. This is called coalescing. In order to coalescing on both sides (bidirectional) one needs to have a flag also on both sides.
One can differ in the coalescing policy: Immediate coalescing to free immediately or deferred coalescing, to do it first when needed.

7.2 Garbage Collection

Definition 1. Garbage collection is the automatic reclamation of heap-allocated storage, so an application never has to free.

7.3 Memory as a graph

One can see memory as a graph. All pointers are roots and show to the memory. Everything that is connected to a pointer is memory. Everything else is garbage.
A garbage collector can be implemented using mark and sweep. It marks everything that is reachable from the root and deletes everything else.

The following was presented in the lecture on 09. October 2019.

8 Basic x86 architecture

Definition 2. RISC: Reduced Instruction Set: uses fewer and simpler instructions than before. All Operations perform highly uniform; need the same amount of time. But x86 is not RISC. On embedded system RISC still makes sense today.

The following was presented in the lecture on 15. October 2019.

8.1 Moving Data

```
movq Source, Dest // Quad Word (8-byte)
movl Source, Dest // Long Word (4-byte)
movw Source, Dest // Word (2-byte)
movb Source, Dest // Byte (1-byte)
```

8.2 Condition codes

There are Carry Out flags, Overflow flags, Zero flags and Signed flags.

The following was presented in the lecture on 16. October 2019.

9 Compiling C Control Flow

Note 10. Large switch statements use jump tables

Note 11. Sparse switch code implements a binary decision tree.

The following was presented in the lecture on 22. October 2019.

10 Compiling C Data Structures

Note 12. Because of the alignment is different on different machines it can save space if one puts large data types first in a struct. Arrays in C have contiguous allocation of memory, are aligned to satisfy every element's alignment requirement, the pointer points to the first element and there is no bound checking.
Structs allocate bytes in order declared and pad in middle and at the end to satisfy alignment. Unions overlay declarations and are a way to circumvent the type system.

The following was presented in the lecture on 30. October 2019.

11 Linking

Definition 3. Linker associates each symbol reference with exactly one symbol definition. Symbol definitions are stored by the compiler in the symbol table. Programs define and reference symbols.

11.1 Pipeline

This steps are executed if one calls the gcc

1. Preprocessing (Macros)
2. Compiling
3. Assembler (Zu Maschinencode)
4. Linking

11.2 Linking rules

```
int x = 2; // Strong symbol
int y; // Weak symbol
int p1() {} // Strong symbol
```

1. Multiple strong symbols are not allowed
2. If there is one strong symbol and the other are weak it takes the strong one.
3. If all the symbols are weak it takes an arbitrary one,

11.3 Object files

Relocatable Object file (.o) Contains data that cannot be executed.

Executable object file Contains code that can be copied directly into memory

Shared object file (.so) File that can be loaded into memory and linked dynamically at either load time or run-time.

Definition 4. ELF object file means Executable and Linkable Format

Note 13. Local linker symbols are not local program variables.

11.4 Linker Symbols

Global symbols Symbols defined by module m that can be referenced by other modules.

External symbols Global symbols that are referenced by module m but defined by some other module.

Local Symbols Symbols that are defined and referenced exclusively by module m.

The following was presented in the lecture on 05. November 2019.

11.5 Static libraries

Definition 5. A static library (.a archive file) concatenate related relocatable object files into a single file with an index.

Note 14. The order in which one links the object is really important, because as soon as the compiler reads a library it discards all the unused ones.

11.6 Shared libraries

Object files that contain code and data that are loaded and linked into an application dynamically, at either load-time or run-time. Shared library routines can be shared by multiple processes.

12 Code Vulnerabilities

Definition 6. A worm is a program that can run by itself and can propagate a fully working version of itself to other computers

Definition 7. A virus is a code that adds itself to other programs, but cannot run independently

12.1 Stack-Overflow Bug

This can be used if the buffer is too small and one can write out of the buffer. The input string contains byte representation of executable code and one overwrites the return address with the address buffer.

The following was presented in the lecture on 6. November 2019.

13 Floating Point

There are normalized and denormalized floating point numbers.

The following was presented in the lecture on 12. November 2019.

Note 15. Course: 263-2300 How to Write Fast Numerical Code by Markus Püschel

14 Optimizing Compilers

One can get a speedup with multiple threads, with vector instructions, and with memory hierarchy and other optimizations.
One needs to optimize at all those levels:

1. Algorithm
2. Data representations
3. Procedures
4. Loops

The compiler is good at finding:

1. register allocation
2. code selection and ordering
3. dead code elimination
4. eliminating minor inefficiencies

The compiler is not good at

1. improving asymptotic efficiency
2. select best algorithm
3. overcoming optimization blockers

Definition 8. Code motion: Reduce frequency with which computation is performed. e.g. Moving code out of a loop (Precomputation)

Definition 9. Strength reduction: Replace costly operation with simper one e.g. $16x \rightarrow x << 4$

Definition 10. Share common subexpressions: Reuse portions of expressions.

The following was presented in the lecture on 13. November 2019.

14.1 Optimize Blockers

Procedure (function calls) may have side effects, as they could return different values. As it doesn't know what it does; it blocks the compiler from moving it out of loops.

Note 16. There is a tradeoff between readable and fast code. Early optimization is the root of all evil!

15 Architecture and Optimization

Definition 11. A superscalar processor can issue and execute multiple instructions in a single machine cycle.

The following was presented in the lecture on 19. November 2019.

16 Caches

16.1 Software caches

There are also software caches, like the file system buffer caches and the web browser caches. They are much more flexible than hardware caches.

The following was presented in the lecture on 20. November 2019.

17 Exceptions

17.1 Exceptional control flow

Definition 12. A hardware exception is a transfer of control to the OS in response to some event. Exceptions cause a switch to kernel mode. It is also called supervisor mode, privileged mode or ring 0. In kernel mode the following is different.

- Access to system state
- Some exceptions are disabled
- Some new instructions and registers
- MMU behavior may change

17.2 Kernel

- A set of trap handling functions
- A set of threads in a special address space
- Code to create the illusion of user-space processes

17.3 Synchronous exceptions

Traps Intentional and returns control to the next instruction. E.g. system calls, breakpoint traps or special instructions.

Faults Unintentional and either re-executes faulting instruction or aborts. E.g. Page faults, protection faults or floating point exceptions

Aborts Unrecoverable and aborts the current program. E.g. parity error or machine check

17.4 Asynchronous exceptions

Caused by events external to the processor, for example I/O interrupts, hard or soft reset interrupt. There are those interrupts

- CPU interrupt-request line triggered by I/O device
- Interrupt handler receives interrupts
- Maskable to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
- Interrupt mechanism also used for exceptions

17.5 Interrupt controllers

Definition 13. A PIC (Programmable Interrupt Controller) maps physical interrupt pins to interrupt vectors, it buffers simultaneous interrupts, prioritizes interrupts and selectively masks any individual device's interrupts.

The following was presented in the lecture on 26. November 2019.

18 Virtual Memory

- Efficient use of limited main memory
- Simplifies memory management for programmers
- Isolates address spaces

Virtual memory works because of locality.

Definition 14. Trashing: Performance meltdown where pages are swapped in and out continuously.

- Caching disk contents in RAM
- Performing memory management
- Protecting memory
- Simplifying linking and loading

The following was presented in the lecture on 27. November 2019.

Note 17. A cache can be with all combinations of virtually/physically indexed/tagged. But most of the time a cache is virtually indexed and physically tagged.

18.1 VV caches with keys

- Add address space identifier (ASID) part of tag
- On access compare with CPU's ASID register
- Removes homonyms, creates synonyms
- Doesn't solve synonym problem
- Doesn't solve write-back problem

18.2 Write buffers

- Store operations take long to complete
- Can avoid stalling CPU by buffering writes
- Write buffer is FIFO queue of incomplete stores
- Can read intermediate values out of the buffer
- Implies that memory contents are temporarily stale

The following was presented in the lecture on 3. December 2019.

Definition 15. Buffering is to copy a block to contiguous memory access.

19 Multiprocessing and Multicore

19.1 Mesi Protocol

Modified This is the only copy, it's dirty

Exclusive this is the only copy, it's clean

Shared This might be one of several copies, all clean

Invalid Other copies have been modified

The following was presented in the lecture on 4. December 2019.

Note 18. In the Mesi protocol the dirty data is always written through memory and there are no cache-cache transfers. The data is always either dirty in one cache and must be written back before a remote read or it is clean.
AMD uses MOESI and Intel uses MESIF. The O stands for Owner and gives the owner the right to modify the dirty copy. The F stands for Forward and is the same as shared but is the designated responder for requests.

19.2 Processor Consistency (PRAM)

All processors see writes from one processor in the order they were issued. Processors can see different interleavings of writes from different processors.

19.3 Simultaneous Multithreading

The question is if one can do anything useful when the processor is waiting for memory. One can label all the instructions in hardware with a thread id. Superscalar techniques are used to do fine-grained or coarse-grained multithreading. One can get a 10-20% performance improvement. This is the 6 cores and 12 threads thing.

The following was presented in the lecture on 10. December 2019.

19.4 Non-Uniform Memory Access (NUMA)

Caches are interconnected and the advantage is that one can access now the memory from different locations. This if done correctly can make RAM access even faster.

19.5 NUMA cache coherence

NUMA uses a message-passing interconnect. The solution to solve cache coherence is to use a cache directory. For all the memory, each cache line data has one has table with who is the owner and all the nodes who have access to this memory. It requires up to a third of the cache space but it is worth it to make it faster.

19.6 False sharing

If two different variables are in the same cache line false sharing can appear, as the processor thinks a variable is dirty, even if it is not, because the only the other has been modified which is not needed by the other processor.

Definition 16. MCS locks: When acquiring, a processor enqueues itself on a list of waiting processors, and spins on its own entry in the list. When releasing, only the next processor is awakened.

20 Devices

Note 19. Device registers don't behave like RAM.

The following was presented in the lecture on 11. December 2019.

20.1 Registers

Device registers are memory locations. Reads and writes also signal events to the hardware. Frequently sets of bitfields. Definitions provided in a datasheet.

20.2 Direct Memory Access (DMA)

Decoupling of data transfer from processing. CPU does not need to copy data to or from the device. DMA means memory becomes inconsistent with CPU caches. DMA addresses are physical.

20.3 Descriptors

Definition 17. Descriptor is a pointer to areas of memory. It is a level of indirection.

The following was presented in the lecture on 17. December 2019.

20.4 Device Initialization

Definition 18. Initialization are events to ensure state transistors are synchronized.

20.5 DMA transactions

1. DMA Read: Descriptor
2. If descriptor.owner == "OS" then enter state "stopped"
3. DMA Read: buffer
4. DMA Writer: descriptor.owned ← "OS"
5. Calculate next descriptor address
6. Goto 1

20.6 Discoverable buses: PCI

Definition 19. PCI means Peripheral Component Interconnect
PCI tries to solve the following problem:

- Device discovery
- Address allocation
- Interrupt routing
- Intelligent DMA