1-25 – lecture 2 – Computer memory

CSAPP 6.1.1 (pg 581-582) - Random Access Memory

* RAM comes in two types – static and dynamic
  + Static is faster and more expensive (money wise)
  + SRAM is used for cache memories, both on and off the CPU chip
  + DRAM is used for the main memory plus the frame buffer of a graphics system.
  + Desktops typically have less than a 10-20 megabytes of SRAM

CSAPP 6.1.4 – Storage Technology trends

* Faster storage technologies are always more expensive than slower storage
  + Order of speed and price: SRAM -> DRAM -> SSD -> Rotating disk
* Since 1985, both cost and performance have improved, but at dramatically different rates depending on the type of storage were talking about
* DRAM and disk performance increases are lagging behind CPU performance

1-30 – lecture 3 – Computer memory

CSAPP 6.4.2 - direct-Mapped Caches, conflict misses

* Caches are grouped into different classes based on E – the number of cache lines per set. A cache with 1 line per set (E = 1) is a direct-mapped cache
* Direct mapped caches are the simplest to implement and understand.
* Suppose we have a CPU that executes an instruction that reads a memory word w. It requests the word from the cache. If the cache has a cached copy of w, it returns it to the CPU. Otherwise we have a cache miss, and the CPU has to wait while the cache requests a copy of w from main memory.
* The process a cache goes through when getting a request has steps.
  + Set selection: the cache extracts the s set index bits from the middle of the address for w.
  + Line matching: determine if a copy of word w is stored in one of the cache lines contained in set i. this is easy and fast in a direct-mapped cache because there is only 1 line per set. If the valid bit is set and the tag in the line matches the tag in the address of w, it is a hit, otherwise it is a miss.
  + Word extraction: once we have a hit, we know w is somewhere in the block. The block offset tells us where the word starts in the block (a block is an array of bytes). If it was a miss, the cache retrieves the block from the next level in the memory hierarchy, and overwrites an entire line in the cache. It will try to overwrite an empty line first (in direct-mapped, there is only one line to overwrite in a given set though).
* Conflict misses are common in real programs and can cause baffling performance problems.
* Why use set index with the middle bits? If the high-order bits are used as the index, then some contiguous memory blocks will map to the same cache set. This is an inefficient use of the ache. With middle-bit indexing, adjacent blocks always map to different cache sets. This way, the cache can hold an entire C-size chunk of the array, where C is the cache size.

2-6 – lecture 5 - Linking

CSAPP 7.4 - ELF file format (Relocatable object files)

* The ELF header consists of a description of word size and byte ordering of the system that generated the file, and information that allows a linker to parse and interpret the object file, including the size of the ELF header, the object file type (relocatable, executable, shared), the machine type (x86, x64), the file offset and the size and number of entries in the section header table.
* The section header table describes the locations and sizes of the various sections of the object file.
* The actual sections are sandwiched between the ELF header and the section header table.
* The following sections are typically included:
  + .text – the machine code of the compiled program.
  + .rodata – read-only data such as the format strings in printf and jump tables for switch statements.
  + .data – initialized global and static C variables. Local C variables are maintained at runtime on the stack and do not appear here.
  + .bss – uninitialized global and static C variables, and those initialized to 0. These variables do not occupy any actual disk space.
  + .symtab – a symbol table with info about functions and global variables that are defined and referenced in the program.
  + .rel.text – a list of locations in the .text section that will need to be modified when the linker combines this object file with others. In general, these are instructions that call an external function or reference a global variable.
  + .rel.data – relocation information for any global variables that are referenced or defined by the module.
  + .debug – a debugging symbol table with entries for local variables and typedefs defined in the program, global variables defined and referenced in the program, and the original C source file. This is only present if the -g flag is used.
  + .line – a mapping between line numbers in the original C source program and machine code instructions in the .text section. Only present if the -g flag is used.
  + .strtab – a string table for the symbol tables I the .symtab and .debug sections and for the section names in the section headers.

CSAPP 7.7 – Relocation

* Once the linker has completed the symbol resolution step, it knows the exact sizes of the code and data sections in its input object modules. It is now ready for the relocation step, where it merges the input modules and assigns run-time addresses to each symbol.
* Relocation consists of 2 steps:
  + Relocating sections and symbol definitions. The linker merges all sections of the same type into a new aggregate section of the same type. The linker then assigns run-time memory addresses to the new aggregate sections. After this step, each instruction and global variable in the program has a unique run-time memory address.
  + Relocating symbol references within sections. The linker modifies every symbol reference in the bodies of the code and data sections so they point to the correct run-time addresses.

Run code examples

2-8 – lecture 6 – Linking

CSAPP 7.14 – Tools for manipulating object files

* There are many tools available on Linux systems to help understand and manipulate object files, like the GNU binutils package.
* AR. Creates static libraries, inserts, deletes, lists, and extracts members.
* STRINGS. Lists all of the printable strings contained in an object file.
* STRIP. Deletes symbol table info from an object file.
* NM. lists the symbols defined I the symbol table of an object file.
* SIZE. Lists the names and sizes of the sections in an object file.
* READELF. Displays the complete structure of an object file.
* OBJDUMP. The mother of all binary tools. Can display all of the info in an object file. Its most useful function is disassembling the binary instructions in the .text section.
* LDD. Lists the shared libraries that an executable needs at run time.

CSAPP 7.15 – linking summary

* Linking can be performed at compile time by static linkers and at load time and run time by dynamic linkers.
* Linkers manipulate binary files called object files, which come in three different forms: relocatable, executable, and shared.
* Relocatable object files are combined by static linkers into an executable object file that can be loaded into memory and executed.
* The two main tasks of linkers are symbol resolution, where each global symbol in an object file is bound to a unique definition, and relocation, where the ultimate memory address for each symbol is determined and where references to those objects are modified.
* Static linkers are invoked by compiler drivers such as GCC. They combine multiple object files into a single exe file. If multiple object files define the same symbol, the linker has rules for resolving this, which can lead to bugs in programs.
* loaders map the contents of exe files into memory and run the program. Linkers can produce partially linked exe object files with unresolved references, which can be completed with a dynamic linker at runtime.

Run code examples

2-13 – lecture 7 – Scheduling

OSTEP 6.1-6.3 (excluding fig. 6.4)