Monday, Feb 13

look at resource files for asgn5, test-prime-ss.c, ss.py

Code in c for 64s, then translate to gnu multi-precision keeping c as comments

Language Translators, Compilers

What is it: maps input language to output language, meaning preserved

Roles

- Converting high-level source language to machine-level target language
- Provide diagnostic messages if the programmer violates specifications of the program

Hardware functions controlled by compatible software

- Hardware interpreted in binary format

Compilation of source code (.c) to executable (look at slides for more details)

- Pre-processor, including macros and macro processor, produces .i code
- Compiler, produces .s assembly code
- Assembler, produces .o object code
- Linker, produces executable

Compiler - lexical phase

- Break source code into lexemes, generate sequence of tokens
- Tokens: int () {} []

Parse trees and e-t-f grammar

- Syntax analysis phase
    - Parse tree to check if exp is syntactically correct
    - Based on CFG (context free grammar), formal specification

Abstract syntax tree

- Representation of tokens
    - Similar to concrete syntax tree structure
    - Generating symbol tables and representing constructs and rules in language

The assembler

- Convert assembly language to machine language (binary)
- Conversion in two ways
    - One pass
    - Two pass

Forward referencing in assembly

- One pass forwarding referencing
    - Can’t resolve forward reference, so backpatching used to fill sports of unknown address
- Two pass forward referencing
    - Tables maintained
    - Second translates to machine language

The linker

- Links .o with libraries
- Ensure dependencies are resolved, throw error if it can’t find something
- Merge all into one executable file

Loaders

- Puts things in RAM to prepare for execution
- 4 basic functions
    - Allocation: allocate space in memory for program
    - Linking: resolve symbolic references between programs
    - Relocation: fix all dependent locations and point to newly allocated space
    - Loading: place machine code and date directly into processor

Memory layout

- Stack and heap grow towards each other in the empty void

Compilers vs interpreters

- Compiler
    - Translator programming language to executable
    - All at once
- Interpreter
    - Directly execute code without compiling
    - Translate one line at a time
    - First part of compiling and does what the tree says
    - Slower than compiler
    - bash

Wednesday, Feb 15 Intro to Files

Interoperability - my code works with other people's code based on specs, compatibility

Long-term storage

- Store large amounts of data
- Info must survive the termination of the process using it
    - Memory dram does not survive turning the computer off
- Files accessed using names, memory is accessed using addresses

People can remember names not numbers

File names

- Trillions of files on the internet
- Has to be unique to find it
- Files have the base name and the extension .whatever
- Unix uses the extension by convention only, it doesn't really mean anything, c compiler is what cares about that

File structure

- Sequence of bytes, sequence of records
- May have internal structure
- Executable file and archive file (see slides for diagram)

File access

- Sequential access
    - Read from beginning
    - Don't jump around, backwards or forwards
    - Convenient with magnetic tape
- Random access
    - bytes/records read in any order

        - Essential for data base systems
        - Read can be
            - Move file marker then read (seek)
            - lseek
            - Read and then move file marker
        - Can't really do data bases without random access

File attributes *see slide for table of meanings*

- Protection: who can access file and in what way
- Password: for the file

File operations

- Create, delete, open, close, read, write
- Append, seek, get attributes, set attributes rename

Directories

- We like to group things together, hierarchy (how we deal with complexity)
- This is done with directories
- Easier to
    - Find files
    - Locate related files
    - Determine which are related

Single-level directory system

Two-level directory system

- Root dir → user dir → files

Hierarchical dir sys

- Root dir, user dir, user sub dir, user files

Path names

- Check slides

Dir operations

- Create (mkdir), delete, opendir, closedir
- Readdir, rename, link, unlink

Shared files

Unix V7 file system

File system implementation

Friday, Feb 17 processes

What is it

- Running program
- Code, data, stack
    - Usually has own address space
- Program state
    - Cpu registers
    - Program counter, current location in code
    - Stack pointer
- Only one process can be running in single cpu core
    - Mulit core can support multiple

What is address space

- Each program instruction has an address
- Each byte of data the program accesses has an address
- Address space is the region of a computer's mem where the program executes
    - Protected from other programs
- Want to make it like the program is the only one happening (usally other stuff happening)

How??

- Loader could relocate instructions, base address
- Base register to set beginning and end
- Processor and operatins system can provide virtual mem

Ideal world

- Memory is very large, fast, affordable, non-volatile (doesn't go awat when power is off)
- Real world: large, fast, affordable

Mem hierarchy

- Cache: small amount of fast expensive mem
    - L1: on CPU chip
    - L2: on or off chip
    - L3: off chip, made of SRAM
- Main mem: medium speed, medium price DRAM, 100+ times slower than CPU
- Disk: many gigs of slow cheap non-volatile storage
- Goal is to keep it in L1 but that's not rly possible

Basic mem management

- Components
    - Operating system
    - Single process
- Goal: lay out in mem

Fixed partitions: multiple programs

- Divide mem into fixed spaces
- Assign process to a space when free
- Mechanisms
    - Separte input queues for partitions
    - Single input queue: better ability to optumize CPU usage

Multiprogrammed syst performance

- Cpu utilization for 1-4 jobs with 80% I/O wait

Mem and multiprograming

- Needs two things: relocation and protection
- Program doesn't know where it will be running
    - Var and procedutes can't be in absolute locations
- Keep processes' mem separate
    - Protect them from others reading or modifying

Base and limit registers

- Special cpu registers
    - Base: start of memory partition
    - Limit: length of the mem partition

    - Go outside of it = seg fault

Allocation

- Search through region list to find a large enough space
- Several choices which one to use
    - First fit: first big enough hole
    - Next fit: next big enough hole
    - Best fit: smallest whole larger (this actually is bad), doesn't waster mem?? But it actually does it leaves a small fragment that nothing else will prob be able to use
    - Worst fit: largest availabel hole, leaves largest fragment

Free mem

- Fragmentation is bad!! → hard to find a good place to run program

Buddy allocation

- Allocate in powers of two
- Split larger chunks to two smaller chunks
- When chunk is freed see if it can be combined with buddy to rebuild larger chunk

Virtual mem

- Hand out more mem than exists in the sys
- Keep recently used stuff in physical mem
- More less recent to disk
- All hidden from processes
    - Process sees whole address space 0 - max_address
    - Movement of info to from disk done by OS only
- Helpful in multiprogrammed systems
    - Schedules b while a waits for mem to be retrieved from disk

Virtual and physical addresses

- Program uses virtual address
- Translation to physical done by special hardware MMU

Paging and page tables

- Virtual mapped to physical
- Unit of mapping a called a page
- Table translate virtual page num to physical page num
- Not all virtual has phys
- Not every phys needs to be used

Page table entry

- Contains
    - Valid bit: set if it has corresponding phys
    - Page frame num: page in phys
    - Referenced bit: set if data has been accessed
    - Dirty (modified) bit: set if data has been modified
    - Protection info

Mapping process

- Split CPU address into two pieces: page num p and page offset d
- Page num
    - Index into page table, which has the address in physical memory

- Go to phys address plus offset

Process created

- Two ways
    - Sys initialization
    -