CSE13S notes
Week of 2/24/23

Gprof (“man gprof”)

- Profiling with gprof
    - Run program by itself
    - Then run gprof
    - Gprof will tell you which functions/parts of the program are taking the most time

Introduction to Make

- A utility on most UNIX systems that automatically builds executable programs and libraries from source code
- Has several derivatives, one of which is GNU Make (gmake), the standard make implementation on Linux/OSX

What is in a Makefile

- Plaintext file that contains instructions
    - Has a syntax like any programming language, can be thought of as a script
- Resides in the same directory as executables

Targets

- Name of a rule
- Users specify which target to make running “make <target>”
- Usually the name of the file that is being built

What’s a phony target

- A target that, when its rule is executed, doesn’t produce a file with the same name
- Important since it prevents make from erroneously checking for files since real and phony targets are now differentiated
- Clean, debug are some conventional phony targets

Useful flags

- See slides

Variables in makefiles

- Four types of variable assignment in a Makefile
- “=” - lazy assignment (literal assignment of text)

    - Recursively expands
- ":=" - immediate assignment (value assignment)
    -
- "?=" - conditional assignment
    - Lazy assignment but only works iff the variable hasn't been assigned yet
- "+=" - concatenation

Dependency?

- Either a target or a filename (this includes source and header files)
- If a rule has dependency that has been modified or if its target doesn't exist, make tries to fill in the dependency by executing the rule with the dependency name
- Else if the dependency has been made, then make ignores/skips the rule
- Dependencies are topologically ordered

What is a command?

- An action (see slides)

Commands, compilers, and compiler flags

- Variables are used to factor makefiles to make them easier to maintain
- Some convention Makefile variables used for compiling C programs:
    - CC - the C compiler to use (typically gcc or cc or clang)
    - CFLAGS - a list of compiler flags
    - Etc. Variables can be assigned
    - A good practice is to declare all variables at the top of the Makefile so it is easy to change

Automatic variables

- Make automatically defines some special variables within the context of an individual rule
- Some useful automatic variables include
    - "$@": the name of the target
    - "$^": list of all dependencies for target
    - "$^" : list of dependencies more recent than the target
    - "$<" : the name of the first dependency

The shell function

- Communicates with the world outside of make
- Performs command expansion - takes a shell command and evaluates to the output of the command

- Newlines in command output are converted to spaces
- A useful example of the shell function:
    - SRC := $(shell ls *.c)
- The above example can also be done using wildcard function

Wildcard function

- Can be used in rules as the “*” operator, where it is expanded by the shell
    - A rule using a phony target to delete object files using wildcard:
- If used for a variable assignment, wildcard expansion doesn’t occur using “*” unless the wildcard function is explicitly specified

The patsubst function

- Read up on your own, very complicated file

Makefile characteristics

- You can include makefiles in makefiles

SECTION NOTES

- no file pointers for this assignment
- Figure out his cool auto compile error stuff
- Use flags instead of integers/numbers
- Pay attention to endianness
- open() will return -1 if it fails
- Use errno, it tells you what went wrong
- Static functions only exist within the current “translation unit” (file). Basically tells linker to ignore functions.
- Static variables within a function will persist within multiple calls of the **same** function. That is, it’s not global, but if each call to a function increments a variable, then the value of that variable would persist and get continuously incremented.
- TA says use OPUS and AV1 for compression
- Entropy: measure of randomness in a program
    - Means random data is compressed less than repeated data

Graphs (3/3/2023)

- When talking about graphs, we are not talking about plots (think stock trading “graphs”)
- Network routing
    - The internet (originally called ARPAnet) is a graph
    - Think social network analysis

Formal definition of graph:

- G = <V, E>
    - V is set of vertices = {V_1, V_2, …, V_n}
    - E is the set of edges, whereas each edge is a tuple of vertices = {<V_i,V_j>, …}
    - A graph is defined by its vertices and edges

Directed and undirected graphs

- Edges may have a direction, $n_1 \rightarrow n_2$, and we call that a directed graph
- Edges may have no direction (or both directions), and we call that an undirected graph
- The edges may have weights, which represents capacity, strength, or cost

Representing a graph

- Adjacency matrix useful for complete graphs
- Adjacency lists especially useful for incomplete graphs

Adjacency list

- Each node is represented as an entry in a column vector
    - Each entry is the head of a linked list
- The list elements contain:
    - The destination node, and
    - The weight of the edge
- Why would you prefer this over an adjacency matrix?
    - An adjacency matrix is $O(n^2)$ space,
    - An adjacency list will be more space efficient for sparse graphs

Basic graph algorithms

- Two ways of searching a graph:
    - Breadth-first search (BFS):
        - Uses a queue
        - Explore the set of vertices immediately reachable
            - Repeat process for each vertex in the set
        - Also known as level order traversal
    - Depth-first search (DFS)
        - Uses recursion or a stack
        - Search as far as possible before backing up
        - We will showcase iterative DFS using a stack

Single-Source shortest paths

- Assuming some graph G = <V,E> and source vertex s is element of V
    - We want to find shortest path from s to any vertex in V
- SSSP algorithms

    - Bellman-Ford
    - Dijkstra's (our focus)

Eulerian path

- A path in an undirected or directed graph that visits each edge exactly once
    - Must start from an origin vertex and end up back at the origin

Summary

- Graphs pervade computer science
    - Shortest path finding
    - Graph coloring
    - Network flow
    - Dependency ordering
    - And so much more
- Come in undirected and directed forms
- Used generally to indicate relationships between entities
- Can be represented using either an adjacency matrix or using adjacency lists