Make and Gprof

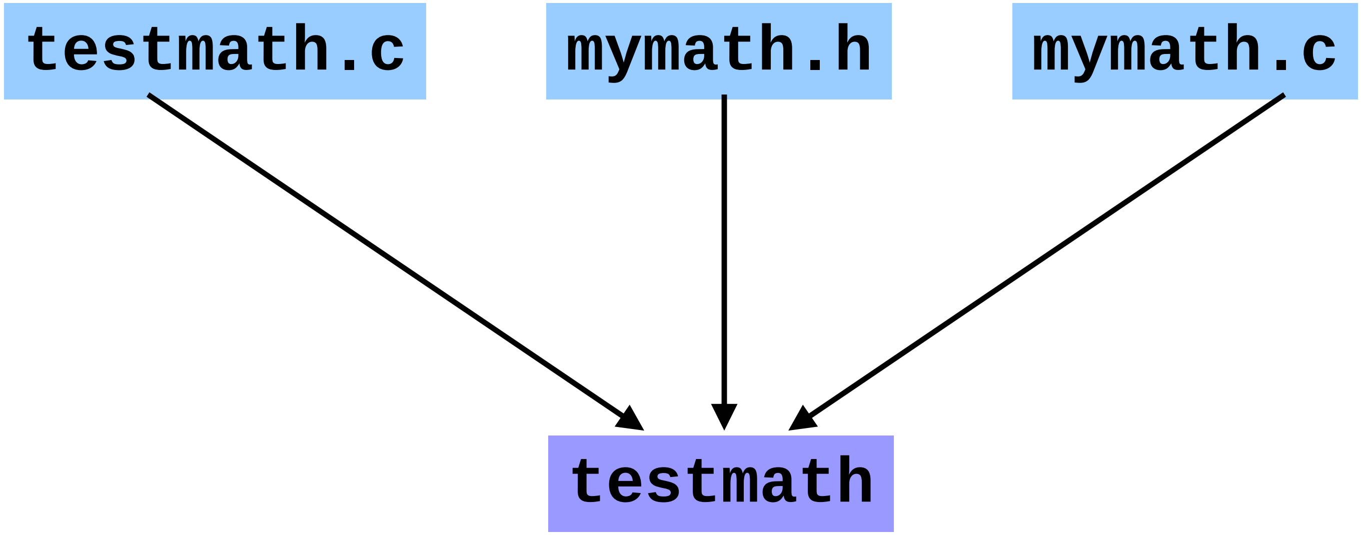
COS 217

Goals of Today’s Lecture

- Overview of two important programming tools
  - Make for compiling and linking multi-file programs
  - Gprof for profiling to identify slow parts of the code
- Make
  - Overview of compilation process
  - Motivation for using Makefiles
  - Example Makefile, refined in five steps
- Gprof
  - Timing, instrumenting, and profiling
  - GNU Performance Profiler (Gprof)
  - Running gprof and understanding the output

Example of a Three-File Program

- Program divided into three files
  - **`mymath.h`**: interface, included in **`mymath.c`** and **`testmath.c`**
  - **`mymath.c`**: implementation of math functions
  - **`testmath.c`**: implementation of tests of the math functions
- Creating the **`testmath`** binary executable



```
gcc –Wall –ansi –pedantic –o testmath testmath.c mymath.c
```

Many Steps, Under the Hood

- Preprocessing (`gcc –E mymath.c > mymath.i`)
  - Removes preprocessor directives
  - Produces **`mymath.i`** and **`testmath.i`**
- Compiling (`gcc –S mymath.i`)
  - Converts to assembly language
  - Produces **`mymath.s`** and **`testmath.s`**
- Assembling (`gcc –c mymath.s`)
  - Converts to machine language with unresolved directives
  - Produces the **`mymath.o`** and **`testmath.o`** binaries
- Linking (`gcc –o testmath testmath.o mymath.o –lc`)
  - Creates machine language exectutable
  - Produces the **`testmath`** binary

Motivation for Makefiles

- Typing at command-line gets tedious
  - Long command with compiler, flags, and file names
  - Easy to make a mistake
- Compiling everything from scratch is time-consuming
  - Repeating preprocessing, compiling, assembling, and linking
  - Repeating these steps for every file, even if just one has changed
- UNIX Makefile tool
  - Makefile: file containing information necessary to build a program
    - Lists the files as well as the dependencies
  - Recompile or relink only as necessary
    - When a dependent file has changed since command was run
    - E.g. if mymath.c changes, recompile mymath.c but not testmath.c
  - Simply type “make”, or “make –f <makefile_name>”

Main Ingredients of a Makefile

- Group of lines
  - Target: the file you want to create
  - Dependencies: the files on which this file depends
  - Command: what to execute to create the file (after a TAB)

- Examples

```
testmath: testmath.o mymath.o

   gcc –o testmath testmath.o mymath.o
```

```
mymath.o: mymath.c mymath.h

    gcc -Wall -ansi -pedantic -c -o mymath.o mymath.c
```

Complete Makefile #1

- Three groups
  - testmath: link testmath.o and mymath.o
  - testmath.o: compile testmath.c, which depends on mymath.h
  - mymath.o: compile mymath.c, which depends on mymath.h

```
testmath: testmath.o mymath.o
   gcc –o testmath testmath.o mymath.o

testmath.o: testmath.c mymath.h
    gcc -Wall -ansi -pedantic -c -o testmath.o testmath.c

mymath.o: mymath.c mymath.h
    gcc -Wall -ansi -pedantic -c -o mymath.o mymath.c
```

Adding Non-File Targets

- Adding useful shortcuts for the programmer
  - “**make all**”: create the final binary
  - “**make clobber**”: delete all temp files, core files, binaries, etc.
  - “**make clean**”: delete all binaries
- Commands in the example
  - “**rm –f**”: remove files without querying the user
  - Files ending in ‘**~**’ and starting/ending in ‘**#**”’ are temporary files
  - “**core**” is a file produced when a program “dumps core”

```
all: testmath

clobber: clean
    rm -f *~ \#*\# core

clean:
    rm -f testmath *.o
```

Complete Makefile #2

```
# Build rules for non-file targets

all: testmath

clobber: clean

    rm -f *~ \#*\# core

clean:

    rm -f testmath *.o


# Build rules for file targets
testmath: testmath.o mymath.o

   gcc –o testmath testmath.o mymath.o

testmath.o: testmath.c mymath.h

    gcc -Wall -ansi -pedantic -c -o testmath.o testmath.c

mymath.o: mymath.c mymath.h

    gcc -Wall -ansi -pedantic -c -o mymath.o mymath.c
```

Useful Abbreviations

- Abbreviations
    - Target file: $@
    - First item in the dependency list: $<
- Example

```
testmath: testmath.o mymath.o

   gcc –o testmath testmath.o mymath.o
```

↓

```
testmath: testmath.o mymath.o

   gcc –o $@ $< mymath.o
```

Complete Makefile #3

```
# Build rules for non-file targets

all: testmath

clobber: clean
    rm -f *~ \#*\# core

clean:
    rm -f testmath *.o


# Build rules for file targets
testmath: testmath.o mymath.o

   gcc –o $@ $< mymath.o

testmath.o: testmath.c mymath.h

    gcc -Wall -ansi -pedantic -c -o $@ $<

mymath.o: mymath.c mymath.h

    gcc -Wall -ansi -pedantic -c -o $@ $<
```

Useful Pattern Rules: Wildcard %

- Can define a default behavior
  - Build rule: **`gcc -Wall -ansi -pedantic -c -o $@ $<`**
  - Applied when target ends in **“.o”** and dependency in **“.c”**

```
%.o: %.c
    gcc -Wall -ansi -pedantic -c -o $@ $<
```

- Can omit command clause in build rules

```
testmath: testmath.o mymath.o
   gcc –o $@ $< mymath.o
testmath.o: testmath.c mymath.h

mymath.o: mymath.c mymath.h
```

Macros for Compiling and Linking

- Make it easy to change which compiler is used
  - Macro: **`CC = gcc`**
  - Usage: **`$(CC) -o $@ $< mymath.o`**
- Make it easy to change the compiler flags
  - Macro: **`CFLAGS = -Wall -ansi –pedantic`**
  - Usage: **`$(CC) $(CFLAGS) -c -o $@ $<`**

```
CC = gcc
# CC = gccmemstat

CFLAGS = -Wall -ansi -pedantic
# CFLAGS = -Wall -ansi -pedantic -g
# CFLAGS = -Wall -ansi -pedantic -DNDEBUG
# CFLAGS = -Wall -ansi -pedantic -DNDEBUG -O3
```

Sequence of Makefiles (see Web)

1. Initial Makefile with file targets

   testmath, testmath.o, mymath.o

2. Adding non-file targets

   all, clobber, and clean

3. Adding abbreviations

   $@ and $<

4. Adding pattern rules

   %.o: %.c

5. Adding macros

   CC and CFLAGS

References on Makefiles

- Brief discussion in the King book
  - Section 15.4 (pp. 320-322)
- GNU make
  - http://www.gnu.org/software/make/manual/html_mono/make.html
- Cautionary notes
  - Don’t forget to use a TAB character, rather than blanks
  - Be careful with how you use the “**`rm –f`**” command

Timing, Instrumenting, Profiling

- How slow is the code?
  - How long does it take for certain types of inputs?
- Where is the code slow?
  - Which code is being executed most?
- Why is the code running out of memory?
  - Where is the memory going?
  - Are there leaks?
- Why is the code slow?
  - How imbalanced is my hash table or binary tree?

Input
Program
Output

Timing

- Most shells provide tool to time program execution
  - E.g., bash "**time**" command

```
bash> time sort < bigfile.txt > output.txt
real     0m12.977s
user     0m12.860s
sys      0m0.010s
```

- Breakdown of time
  - Real: elapsed time between invocation and termination
  - User: time spent executing the program
  - System: time spent within the OS on the program's behalf
- But, which *parts* of the code are the most time consuming?

Instrumenting

- Most operating systems provide a way to get the time
    - e.g., UNIX “**gettimeofday**” command

```
#include <sys/time.h>

struct timeval start_time, end_time;

gettimeofday(&start_time, NULL);
  <execute some code here>
gettimeofday(&end_time, NULL);

float seconds = end_time.tv_sec - start_time.tv_sec +
     1.0E-6F * (end_time.tv_usec - start_time.tv_usec);
```

Profiling

- Gather statistics about your program's execution
  - e.g., how much time did execution of a function take?
  - e.g., how many times was a particular function called?
  - e.g., how many times was a particular line of code executed?
  - e.g., which lines of code used the most time?
- Most compilers come with profilers
  - e.g., **`pixie`** and **`gprof`**
- Gprof (GNU Performance Profiler)

  -
    ```
    gcc –Wall –ansi –pedantic –pg –o mymath.o mymath.c
    ```

Gprof (GNU Performance Profiler)

- Instrumenting the code
  - `gcc –Wall –ansi –pedantic –pg –o mymath.o mymath.c`
- Running the code (e.g., **`testmath`**)
  - Produces output file **`gmon.out`** containing statistics
- Printing a human-readable report from **`gmon.out`**
  - `gprof testmath > gprofreport`

Two Main Outputs of Gprof

- Call graph profile: detailed information per function
  - Which functions called it, and how much time was consumed?
  - Which functions it calls, how many times, and for how long?
  - We won’t look at this output in any detail…
- Flat profile: one line per function
  - name: name of the function
  - %time: percentage of time spent executing this function
  - cumulative seconds: [skipping, as this isn’t all that useful]
  - self seconds: time spent executing this function
  - calls: number of times function was called (excluding recursive)
  - self ms/call: average time per execution (excluding descendents)
  - total ms/call: average time per execution (including descendents)

Call Graph Output

```
 called/total       parents
index  %time    self descendents  called+self    name               index
                                  called/total       children

                                                     <spontaneous>
[1]     59.7   12.97        0.00                 internal_mcount [1]
                0.00        0.00       1/3           atexit [35]
-----------------------------------------------
                                                     <spontaneous>
[2]     40.3    0.00        8.75                 _start [2]
                0.00        8.75       1/1           main [3]
                0.00        0.00       2/3           atexit [35]
-----------------------------------------------
                0.00        8.75       1/1           _start [2]
[3]     40.3    0.00        8.75       1         main [3]
                0.00        8.32       1/1           getBestMove [4]
                0.00        0.43       1/1           clock [20]
                0.00        0.00       1/747130      GameState_expandMove [6]
                0.00        0.00       1/1           exit [33]
                0.00        0.00       1/1           Move_read [36]
                0.00        0.00       1/1           GameState_new [37]
                0.00        0.00       6/747135      GameState_getStatus [31]
                0.00        0.00       1/747130      GameState_applyDeltas [25]
                0.00        0.00       1/1           GameState_write [44]
                0.00        0.00       1/1           sscanf [54]
                0.00        0.00       1/1698871     GameState_getPlayer [30]
                0.00        0.00       1/1           _fprintf [58]
                0.00        0.00       1/1           Move_write [59]
                0.00        0.00       3/3           GameState_playerToStr [63]
                0.00        0.00       1/2           strcmp [66]
                0.00        0.00       1/1           GameState_playerFromStr [68]
                0.00        0.00       1/1           Move_isValid [69]
                0.00        0.00       1/1           GameState_getSearchDepth [67]
-----------------------------------------------
                0.00        8.32       1/1           main [3]
[4]     38.3    0.00        8.32       1         getBestMove [4]
                0.27        8.05       6/6           minimax [5]
                0.00        0.00       6/747130      GameState_expandMove [6]
                0.00        0.00      35/4755325     Delta_free [10]
                0.00        0.00       1/204617      GameState_genMoves [17]
                0.00        0.00       5/945027      Move_free [23]
                0.00        0.00       6/747130      GameState_applyDeltas [25]
                0.00        0.00       6/747129      GameState_unApplyDeltas [27]
                0.00        0.00       2/1698871     GameState_getPlayer [30]
-----------------------------------------------
                                  747123             minimax [5]
                0.27        8.05       6/6           getBestMove [4]
[5]     38.3    0.27        8.05       6+747123  minimax [5]
                0.63        3.56  747123/747130      GameState_expandMove [6]
                0.25        1.82 4755290/4755325     Delta_free [10]
                0.07        0.69  204616/204617      GameState_genMoves [17]
                0.02        0.36  945022/945027      Move_free [23]
                0.23        0.00  747123/747130      GameState_applyDeltas [25]
                0.22        0.00  747123/747129      GameState_unApplyDeltas [27]
                0.10        0.00 1698868/1698871     GameState_getPlayer [30]
                0.09        0.00  747129/747135      GameState_getStatus [31]
                0.01        0.00  542509/542509      GameState_getValue [32]
                                  747123             minimax [5]
-----------------------------------------------
                0.00        0.00       1/747130      main [3]
                0.00        0.00       6/747130      getBestMove [4]
                0.63        3.56  747123/747130      minimax [5]
[6]     19.3    0.63        3.56  747130         GameState_expandMove [6]
                0.47        2.99 4755331/5700361     calloc [7]
                0.11        0.00 2360787/2360787     .rem [28]
-----------------------------------------------
                0.00        0.00       1/5700361     Move_read [36]
                0.00        0.00       1/5700361     GameState_new [37]
                0.09        0.59  945028/5700361     GameState_genMoves [17]
                0.47        2.99 4755331/5700361     GameState_expandMove [6]
[7]     19.1    0.56        3.58 5700361         calloc [7]
                0.32        2.09 5700361/5700362     malloc [8]
                0.64        0.00 5700361/5700361     .umul [18]
                0.43        0.00 5700361/5700361     _memset [22]
                0.10        0.00 5700361/5700363     .udiv [29]
-----------------------------------------------
                0.00        0.00       1/5700362     _findbuf [41]
                0.32        2.09 5700361/5700362     calloc [7]
[8]     11.1    0.32        2.09 5700362         malloc [8]
                0.62        0.62 5700362/5700362     _malloc_unlocked                <cycle 1> [13]
                0.23        0.20 5700362/11400732    _mutex_unlock [14]
                0.22        0.20 5700362/11400732    _mutex_lock [15]
```

Complex format at the beginning… let’s skip for now.

Flat Profile

```
      %  cumulative    self              self     total
   time   seconds   seconds    calls  ms/call  ms/call name
   57.1      12.97    12.97                             internal_mcount [1]
    4.8      14.05     1.08  5700352     0.00     0.00  _free_unlocked [12]
    4.4      15.04     0.99                             _mcount (693)
    3.5      15.84     0.80 22801464     0.00     0.00  _return_zero [16]
    2.8      16.48     0.64  5700361     0.00     0.00  .umul [18]
    2.8      17.11     0.63   747130     0.00     0.01  GameState_expandMove [6]
    2.5      17.67     0.56  5700361     0.00     0.00  calloc [7]
    2.1      18.14     0.47 11400732     0.00     0.00  _mutex_unlock [14]
    1.9      18.58     0.44 11400732     0.00     0.00  mutex_lock [15]
    1.9      19.01     0.43  5700361     0.00     0.00  _memset [22]
    1.9      19.44     0.43        1   430.00   430.00  .div [21]
    1.8      19.85     0.41  5157853     0.00     0.00  cleanfree [19]
    1.4      20.17     0.32  5700366     0.00     0.00  _malloc_unlocked  [13]
    1.4      20.49     0.32  5700362     0.00     0.00  malloc [8]
    1.3      20.79     0.30  5157847     0.00     0.00  _smalloc          [24]
    1.2      21.06     0.27        6    45.00  1386.66  minimax [5]
    1.1      21.31     0.25  4755325     0.00     0.00  Delta_free [10]
    1.0      21.54     0.23  5700352     0.00     0.00  free [9]
    1.0      21.77     0.23   747130     0.00     0.00  GameState_applyDeltas [25]
    1.0      21.99     0.22  5157845     0.00     0.00  realfree [26]
    1.0      22.21     0.22   747129     0.00     0.00  GameState_unApplyDeltas [27]
    0.5      22.32     0.11  2360787     0.00     0.00  .rem [28]
    0.4      22.42     0.10  5700363     0.00     0.00  .udiv [29]
    0.4      22.52     0.10  1698871     0.00     0.00  GameState_getPlayer [30]
    0.4      22.61     0.09   747135     0.00     0.00  GameState_getStatus [31]
    0.3      22.68     0.07   204617     0.00     0.00  GameState_genMoves [17]
    0.1      22.70     0.02   945027     0.00     0.00  Move_free [23]
    0.0      22.71     0.01   542509     0.00     0.00  GameState_getValue [32]
    0.0      22.71     0.00      104     0.00     0.00  _ferror_unlocked [357]
    0.0      22.71     0.00       64     0.00     0.00  _realbufend [358]
    0.0      22.71     0.00       54     0.00     0.00  nvmatch [60]
    0.0      22.71     0.00       52     0.00     0.00  _doprnt [42]
    0.0      22.71     0.00       51     0.00     0.00  memchr [61]
    0.0      22.71     0.00       51     0.00     0.00  printf [43]
    0.0      22.71     0.00       13     0.00     0.00  _write [359]
    0.0      22.71     0.00       10     0.00     0.00  _xflsbuf [360]
    0.0      22.71     0.00        7     0.00     0.00  _memcpy [361]
    0.0      22.71     0.00        4     0.00     0.00  .mul [62]
    0.0      22.71     0.00        4     0.00     0.00  ___errno [362]
    0.0      22.71     0.00        4     0.00     0.00  _fflush_u [363]
    0.0      22.71     0.00        3     0.00     0.00  GameState_playerToStr [63]
    0.0      22.71     0.00        3     0.00     0.00  _findbuf [41]
```

Second part of profile looks like this; it's the simple (i.e.,useful) part; corresponds to the "prof" tool

Overhead of Profiling

```
   %  cumulative    self              self    total
time   seconds   seconds    calls  ms/call  ms/call name
57.1      12.97    12.97                            internal_mcount
 4.8      14.05     1.08  5700352     0.00     0.00  _free_unlocked
 4.4      15.04     0.99                            _mcount (693)
 3.5      15.84     0.80 22801464     0.00     0.00  _return_zero
 2.8      16.48     0.64  5700361     0.00     0.00  .umul [18]
 2.8      17.11     0.63   747130     0.00     0.01  GameState_expa
 2.5      17.67     0.56  5700361     0.00     0.00  calloc [7]
 2.1      18.14     0.47 11400732     0.00     0.00  _mutex_unlock
 1.9      18.58     0.44 11400732     0.00     0.00  mutex_lock
 1.9      19.01     0.43  5700361     0.00     0.00  _memset [22]
 1.9      19.44     0.43        1   430.00   430.00  .div [21]
 1.8      19.85     0.41  5157853     0.00     0.00  cleanfree [19]
 1.4      20.17     0.32  5700366     0.00     0.00  _malloc_unlo
 1.4      20.49     0.32  5700362     0.00     0.00  malloc [8]
 1.3      20.79     0.30  5157847     0.00     0.00  _smalloc
 1.2      21.06     0.27        6    45.00  1386.66  minimax [5]
 1.1      21.31     0.25  4755325     0.00     0.00  Delta_free [10]
 1.0      21.54     0.23  5700352     0.00     0.00  free [9]
 1.0      21.77     0.23   747130     0.00     0.00  GameState_appl
 1.0      21.99     0.22  5157845     0.00     0.00  realfree [26]
 1.0      22.21     0.22   747129     0.00     0.00  GameState_unAp
 0.5      22.32     0.11  2360787     0.00     0.00  .rem [28]
 0.4      22.42     0.10  5700363     0.00     0.00  .udiv [29]
 0.4      22.52     0.10  1698871     0.00     0.00  GameState_getPl
 0.4      22.61     0.09   747135     0.00     0.00  GameState_getSt
```

Malloc/calloc/free/...

```
   %  cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call name
 57.1     12.97    12.97                             internal_mcount [1]
  4.8     14.05     1.08  5700352     0.00     0.00  _free_unlocked [12]
  4.4     15.04     0.99                             _mcount (693)
  3.5     15.84     0.80 22801464     0.00     0.00  _return_zero [16]
  2.8     16.48     0.64  5700361     0.00     0.00  .umul [18]
  2.8     17.11     0.63   747130     0.00     0.01  GameState_expandMove
  2.5     17.67     0.56  5700361     0.00     0.00  calloc [7]
  2.1     18.14     0.47 11400732     0.00     0.00  _mutex_unlock [14]
  1.9     18.58     0.44 11400732     0.00     0.00  mutex_lock [15]
  1.9     19.01     0.43  5700361     0.00     0.00  _memset [22]
  1.9     19.44     0.43        1   430.00   430.00  .div [21]
  1.8     19.85     0.41  5157853     0.00     0.00  cleanfree [19]
  1.4     20.17     0.32  5700366     0.00     0.00  _malloc_unlocked [13]
  1.4     20.49     0.32  5700362     0.00     0.00  malloc [8]
  1.3     20.79     0.30  5157847     0.00     0.00  _smalloc      [24]
  1.2     21.06     0.27        6    45.00  1386.66  minimax [5]
  1.1     21.31     0.25  4755325     0.00     0.00  Delta_free [10]
  1.0     21.54     0.23  5700352     0.00     0.00  free [9]
  1.0     21.77     0.23   747130     0.00     0.00  GameState_applyDeltas
  1.0     21.99     0.22  5157845     0.00     0.00  realfree [26]
  1.0     22.21     0.22   747129     0.00     0.00  GameState_unApplyDeltas
  0.5     22.32     0.11  2360787     0.00     0.00  .rem [28]
  0.4     22.42     0.10  5700363     0.00     0.00  .udiv [29]
  0.4     22.52     0.10  1698871     0.00     0.00  GameState_getPlayer
  0.4     22.61     0.09   747135     0.00     0.00  GameState_getStatus
  0.3     22.68     0.07   204617     0.00     0.00  GameState_genMoves [17]
```

expandMove

```
   %  cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call name
57.1      12.97    12.97                            internal_mcount [1]
 4.8      14.05     1.08  5700352     0.00     0.00  _free_unlocked [12]
 4.4      15.04     0.99                            _mcount (693)
 3.5      15.84     0.80 22801464     0.00     0.00  _return_zero [16]
 2.8      16.48     0.64  5700361     0.00     0.00  .umul [18]
 2.8      17.11     0.63   747130     0.00     0.01  GameState_expandMove
 2.5      17.67     0.56  5700361     0.00     0.00  calloc [7]
 2.1      18.14     0.47 11400732     0.00     0.00  _mutex_unlock [14]
 1.9      18.58     0.44 11400732     0.00     0.00  mutex_lock [15]
 1.9      19.01     0.43  5700361     0.00     0.00  _memset [22]
 1.9      19.44     0.43        1   430.00   430.00  .div [21]
 1.8      19.85     0.41  5157853     0.00     0.00  cleanfree [19]
 1.4      20.17     0.32  5700366     0.00     0.00  _malloc_unlocked [13]
 1.4      20.49     0.32  5700362     0.00     0.00  malloc [8]
 1.3      20.79     0.30  5157847     0.00     0.00  _smalloc       [24]
 1.2      21.06     0.27        6    45.00  1386.66  minimax [5]
 1.1      21.31     0.25  4755325     0.00     0.00  Delta_free [10]
 1.0      21.54     0.23  5700352     0.00     0.00  free [9]
 1.0      21.77     0.23   747130     0.00     0.00  GameState_applyDeltas
 1.0      21.99     0.22  5157845     0.00     0.00  realfree [26]
```

May be worthwhile to optimize this routine

Don’t Even *Think* of Optimizing These

```
 %   cumulative    self              self    total
 time    seconds   seconds    calls  ms/call  ms/call name
57.1      12.97    12.97                              internal_mcount [1]
 4.8      14.05     1.08  5700352     0.00     0.00   _free_unlocked [12]
 4.4      15.04     0.99                              _mcount (693)
 3.5      15.84     0.80 22801464     0.00     0.00   _return_zero [16]
 2.8      16.48     0.64  5700361     0.00     0.00   .umul [18]
 2.8      17.11     0.63   747130     0.00     0.01   GameState_expandMove [6]
 2.5      17.67     0.56  5700361     0.00     0.00   calloc [7]
 2.1      18.14     0.47 11400732     0.00     0.00   _mutex_unlock [14]
 1.9      18.58     0.44 11400732     0.00     0.00   mutex_lock [15]
 1.9      19.01     0.43  5700361     0.00     0.00   _memset [22]
 1.9      19.44     0.43        1   430.00   430.00   .div [21]
 1.8      19.85     0.41  5157853     0.00     0.00   cleanfree [19]
 1.4      20.17     0.32  5700366     0.00     0.00   _malloc_unlocked  <cycle 1> [13]
 1.4      20.49     0.32  5700362     0.00     0.00   malloc [8]
 1.3      20.79     0.30  5157847     0.00     0.00   _smalloc          <cycle 1> [24]
 1.2      21.06     0.27        6    45.00  1386.66   minimax [5]
 1.1      21.31     0.25  4755325     0.00     0.00   Delta_free [10]
 1.0      21.54     0.23  5700352     0.00     0.00   free [9]
 1.0      21.77     0.23   747130     0.00     0.00   GameState_applyDeltas [25]
 1.0      21.99     0.22  5157845     0.00     0.00   realfree [26]
 1.0      22.21     0.22   747129     0.00     0.00   GameState_unApplyDeltas [27]
 0.5      22.32     0.11  2360787     0.00     0.00   .rem [28]
 0.4      22.42     0.10  5700363     0.00     0.00   .udiv [29]
 0.4      22.52     0.10  1698871     0.00     0.00   GameState_getPlayer [30]
 0.4      22.61     0.09   747135     0.00     0.00   GameState_getStatus [31]
 0.3      22.68     0.07   204617     0.00     0.00   GameState_genMoves [17]
 0.1      22.70     0.02   945027     0.00     0.00   Move_free [23]
 0.0      22.71     0.01   542509     0.00     0.00   GameState_getValue [32]
 0.0      22.71     0.00      104     0.00     0.00   _ferror_unlocked [357]
 0.0      22.71     0.00        4     0.00     0.00   _thr_main [367]
 0.0      22.71     0.00        3     0.00     0.00   GameState_playerToStr [63]
 0.0      22.71     0.00        2     0.00     0.00   strcmp [66]
 0.0      22.71     0.00        1     0.00     0.00   GameState_getSearchDepth [67]
 0.0      22.71     0.00        1     0.00     0.00   GameState_new [37]
 0.0      22.71     0.00        1     0.00     0.00   GameState_playerFromStr [68]
 0.0      22.71     0.00        1     0.00     0.00   GameState_write [44]
 0.0      22.71     0.00        1     0.00     0.00   Move_isValid [69]
 0.0      22.71     0.00        1     0.00     0.00   Move_read [36]
 0.0      22.71     0.00        1     0.00     0.00   Move_write [59]
 0.0      22.71     0.00        1     0.00     0.00   check_nlspath_env [46]
 0.0      22.71     0.00        1     0.00   430.00   clock [20]
 0.0      22.71     0.00        1     0.00     0.00   exit [33]
 0.0      22.71     0.00        1     0.00  8319.99   getBestMove [4]
 0.0      22.71     0.00        1     0.00     0.00   getenv [47]
 0.0      22.71     0.00        1     0.00  8750.00   main [3]
 0.0      22.71     0.00        1     0.00     0.00   mem_init [70]
 0.0      22.71     0.00        1     0.00     0.00   number [71]
 0.0      22.71     0.00        1     0.00     0.00   scanf [53]
```

Using a Profiler

- Test your code as you write it
  - It is very hard to debug a lot of code all at once
  - Isolate modules and test them independently
  - Design your tests to cover boundary conditions
- Instrument your code as you write it
  - Include asserts and verify data structure sanity often
  - Include debugging statements (e.g., #ifdef DEBUG and #endif)
  - You'll be surprised what your program is really doing!!!
- Time and profile your code **only** when you are done
  - Don't optimize code unless you have to (you almost never will)
  - Fixing your algorithm is almost always the solution
  - Otherwise, running optimizing compiler is usually enough

Summary

- Two valuable UNIX tools
  - Make: building large program in pieces
  - Gprof: profiling a program to see where the time goes
- How does gprof work?
  - Good question!
  - Essentially, by randomly sampling the code as it runs
    - And seeing what line is running, and what function its in
  - More on this for the last programming assignment of the course!
- Rest of this week
  - Exam on Thursday 10:00-10:50am in this room
  - Open books and open notes, but not open laptop/PDA
  - No precept on Wednesday/Thursday in honor of midterms