Announcements

- Program #6 due Thursday
- Final Exam
  - **Saturday, May 12**
  - **4:00-6:00 pm**
  - in **PHY 1412**
- Reading
  - Notes

Time

- On a computer there are many ways measure time
  - Wall time
    - always running
  - Process time
    - time your process was running
    - doesn't count:
      - time when your process was stopped for others
      - time when your program stopped to wait for I/O
      - time spent running in OS kernel (system calls)
- What time to use depends on what you are measuring

Timing a Program

- To Time an entire program
  - time <program name> <program arguments>
  - prints out time in the form:
    - 2.23u 0.26s 0:06.52 38.1% 0+0k 0+0io 80pf+0w
    - first two numbers are user and system time
    - third number is wall time
    - fourth is percentage of wall time (user+system)/wall
    - remainder are paging and I/O statistics
- Provides Some idea about timing
  - hard to know what to do about it
    - what functions are taking most of the time?

Representing Time Of Day

- How to deal with
  - timezones
  - daylight savings times
  - computers moving timezones
  - leap seconds?
- Answer:
  - keep time internally a seconds + fractions of a second
    - 2 32 bits values work nicely for 1ns accuracy for > 100 years
  - use a reference starting point
    - called epoch - midnight 1/1/1970
  - keep all time in UTC form until printed
    - no time zones or daylight savings time to deal with

Adding Timing Calls to your program

- Wall Time
  - int gettimeofday(struct timeval *tv, struct timezone *tz);
  - tv is a struct of time tv_sec and tv_usec (10^{-6} seconds)
  - tz is no longer used (pass null)
- Process Time
  - int getrusage(int who, struct rusage *usage);
  - who is RUSAGE_SELF or RUSAGE_CHILDREN
    - children is all terminated children
  - rusage contains fields for
    - struct timeval ru_utime; /* user time used */
    - struct timeval ru_stime; /* system time used */
    - various other OS stats

Example Measuring Time

```
main()  {
    struct rusage startRU, endRU;
    struct timeval startWall, endWall;

    getimeofday(&startWall, NULL);
    getrusage(&startRU);
    /* code to time */

    getimeofday(&endWall, NULL);
    getrusage(&endRU);
    /* compute difference */
}
```

Review Lecture

- Final covers all material from the class
  - Comprehensive
  - relatively equally weighted between 1st, 2nd and after 2nd exam
- Today's lecture is set of highlights
  - Be aware that there are things not in today's review which will likely also be on the final!

Why Study Low-level Programming? and Why Study the C language?

- Provides Understanding of How Things Work
  - Compilers, Processors, Data Structures
- Allows access to Hardware when required
- Allows efficiency and control
  - Writing device drivers and operating systems
  - If you are careful and good it will be faster
  - BUT could also be Dangerous or Slower
- Widely Used for System Programming
- "Middle Aged" Language – created in late 1970's
- C's design goals Goals
- Major Differences from JAVA
  - Explicit Memory Allocation and Deallocation
  - Procedural rather than object oriented
  - *Compiles* directly to machine code (rather than byte codes)

Phases of Compilation

- Preprocessor
  - Not everything is in one file
    - Definitions of routines are in separate files
  - Function *Prototypes*
    - Define name, parameters, and return types
  - #include, #if, #endif, #ifndef
  - macros, constants, conditional compilation
- translation (compilation)
  - source to object code
  - of individual files
  - making sure prototypes match definitions and calls
- linking
  - combining object files
  - making sure all parts are there

Readability, Reliability and Efficiency of Your program

- Readability
  - Comments
    - Should be surrounded by /* and */
    - May span multiple lines
  - White Space
    - vertical and horizontal
- Testing
  - Write Tests as you write software
  - When the software is “done”, tests continue as part of it
  - Types of Testing
    - line coverage, white box and black box
- Efficiency
  - Code motion
  - Loop Unrolling
  - Dead-Code Elimination
  - human changes vs -O option on compiler

Make: A Tool to Compile Programs

- Problem: Compiling programs is tedious
- Solution: Automate it!

- Done in Unix by the make command
  - Uses a file called Makefile
    - A makefile is a file (script) containing :
      - Project structure (files, dependencies)
      - Instructions for files creation
- Make is not limited to C programs

Adapted from: www.cs.tau.ac.il/~dcor/Software1/Makefile.**ppt**