Course Overview

CSE 238/2038/2138: Systems Programming

Instructor:

Fatma CORUT ERGİN

Slides adapted from Bryant & O'Hallaron's slides

Overview

- Course logistics
- Course outcomes
- Course coverage

Fatma CORUT ERGİN

- e-mail: fatma.ergin@marmara.edu.tr
- Office: MB 449
- Zoom link: https://cutt.ly/4lLV0ci
  - Meeting ID: 255 133 3127
  - Passcode: 238495

Teaching Assistants

- Zuhal ALTUNTAŞ
  - zuhal.altuntas@marmara.edu.tr

- Lokman ALTIN
  - lokman.altin@marmara.edu.tr

Logistics

- Lecture Hours
  - Mondays 14.00 – 15.50 (2 sessions)
  - Tuesdays 13.00 – 13.50 (1 session)

- Course Content
  - Canvas https://canvas.instructure.com/enroll/CWRL68
    - You can enroll in the course with the link
  - ues.marmara.edu.tr

- Lecture Videos
  - ues.marmara.edu.tr

- Announcements
  - Canvas https://canvas.instructure.com/courses/2609650
  - ues.marmara.edu.tr

Textbook

- Randal E. Bryant and David R. O'Hallaron,
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu

  - **This book really matters for the course!**
    - How to solve labs
    - Practice problems typical of exam problems

Grading (tentative)

- 70% attendance is a must to pass the course
- Scheduled quizzes – each 2% (on Canvas)
  - **You may have scheduled quizzes.**
  - **You will take quizzes on Canvas**
  - **The quiz date will be announced at least 1 day before.**
  - **These quizzes will take between 10-20 minutes**
- Pop quizzes – 10% (on Zoom lecture)
  - **Pop quizzes can happen any time during the lecture hours.**
  - **The questions will be asked on Zoom lecture meeting.**
  - **They will be on the topic of the corresponding lecture.**
  - **These quizzes will be very-short (1-2 minutes)**
- 3 Projects – 30% (may change according to scheduled quiz number)
- Midterm Exam – 20% (on Zoom)
- Final Exam – 40% (on Zoom)

Canvas Participation

- You will receive an e-mail from “Instructure Canvas” as an invitation
- You should accept the invitation
- Your scheduled quizzes will be prepared on Canvas, so it is very important for you to enroll in the class as soon as possible

Cheating: Description

- What is cheating?
  - Sharing code: by copying, retyping, looking at, or supplying a file
  - Describing: verbal description of code from one person to another.
  - Coaching: helping your friend to write a code, line by line
  - Searching the Web for solutions
  - Copying code from any kind of source
    - You are only allowed to use code we supply

- What is NOT cheating?
  - Explaining how to use systems or tools
  - Helping others with high-level design issues

Cheating: Consequences

- Penalty for cheating:
  - “0” grade for both parties
- Detection of cheating:
  - We have sophisticated tools for detecting code plagiarism

Start early

Ask the staff for help when you get stuck

Welcome and Enjoy!

Course Theme:
Systems Knowledge is Power!

- Systems knowledge
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - How you (as a programmer) can best use these resources

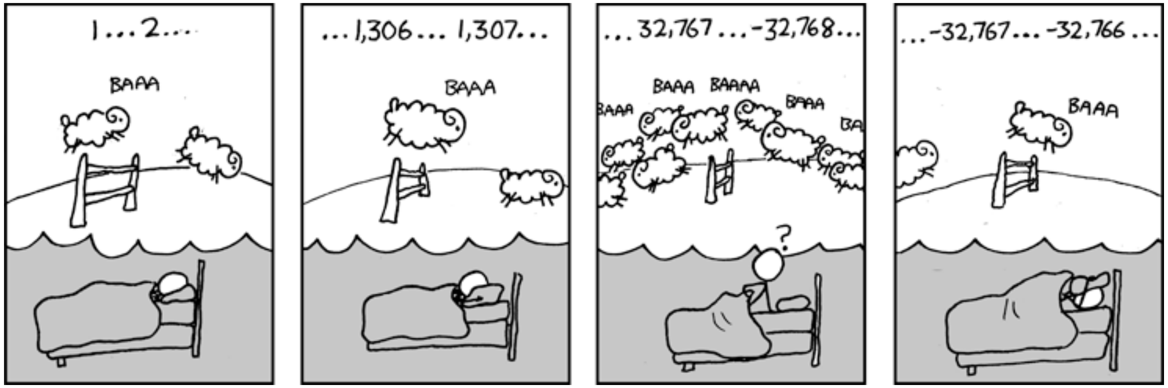
- Useful outcomes of the course
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CSE
    - Computer Organization, Operating Systems, etc.

Understand How Things Work

- Why do I need to know this stuff?
  - Abstraction is good, but don't forget reality
- Most CSE courses emphasize abstraction
  - Abstract data types
  - Asymptotic analysis
- These abstractions have limits
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
  - Sometimes the abstract interfaces don't provide the level of control or performance you need

Great Reality #1:
Ints are not Integers, Floats are not Reals

- Example 1: Is $x^2 \geq 0$?
  - Float's: Yes!
  - Int's:
    - 40000 * 40000
      = 1600000000
    - 50000 * 50000
      = ??



- Example 2: Is (x + y) + z = x + (y + z)?
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all “usual” mathematical properties**
  - Due to finiteness of representations
  - Integer operations satisfy “ring” properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy “ordering” properties
    - Monotonicity, values of signs
- **Observation**
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

Great Reality #2: You've Got to Know Assembly

- **You'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters
Random Access Memory Is an Unphysical Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially evil**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741121;
  return s.d;
}
```

```
fun(0) → 3.14                  (a[0]=1073741121, a[1]=0)
fun(1) → 3.14                  (a[0]=0, a[1]=1073741121)
fun(2) → 3.1399998664856       (a[0]=0, a[1]=0)
fun(3) → 2.00000061035156      (a[0]=0, a[1]=0)
fun(4) → 3.14                  (a[0]=0, a[1]=0)
fun(5) → 3.14                  (a[0]=0, a[1]=0)
fun(6) → 3.14                  (a[0]=0, a[1]=0)
fun(7) → Segmentation fault
```

- Result is system specific

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741121;
  return s.d;
}
```

`fun(0)`	→	`3.14`
`fun(1)`	→	`3.14`
`fun(2)`	→	`3.1399998664856`
`fun(3)`	→	`2.00000061035156`
`fun(4)`	→	`3.14`
`fun(5)`	→	`3.14`
`fun(6)`	→	`3.14`
`fun(7)`	→	Segmentation fault

Explanation:

Critical State	7
?	6
?	5
?	4
`d7 ... d4`	3
`d3 ... d0`	2
`a[1]`	1
`a[0]`	0

`struct_t`

Location accessed by `fun(i)`

What About This?

```
typedef struct {
  double d;
  int a[2];
} struct_t

double fun(int i) {
  struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741121;
  return s.d;
}
```

```
fun(0)  →   3.14
fun(1)  →   3.14
fun(2)  →   3.14
fun(3)  →   3.14
fun(4)  →   3.14
fun(5)  →   Segmentation fault
```

Explanation:

Critical State	7
?	6
?	5
?	4
a[1]	3
a[0]	2
d7 ... d4	1
d3 ... d0	0

`struct_t` (locations 0–3)

Location accessed by `fun(i)` (locations 0–7)

Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated
- **How can I deal with this?**
  - Program in Java, Ruby, Python, ML, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Memory System Performance Example

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms

81.8ms

2.0 GHz Intel Core i7 Haswell

Course Perspective

- Our Course is Programmer-Centric
  - Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to write programs that are more reliable and efficient