Hints for Computer System Design

Baris Kasikci

Why is System Design Hard?

- The external Interface is not well-defined
  - *Requirements are not clear*
- The system has much more internal structure
  - *Internal interfaces are complex*
- The measure of success is not very clear

After the second system comes the fourth

Butler Lampson

The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. Hence plan to throw one away; you will, anyhow.

Fred Brooks, Mythical Man Month

What are your favorite hints?

Why?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load End-to-end Safety first	End-to-end
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Interface Design - 1

- Conflicting Requirements
  - *Simple*
  - *Complete*
  - *Efficient*
- Interface design is like programming language design
  - *Objects and operations manipulating those objects*
- Kis(s): Keep it simple (and stupid)
  - *Do one thing at a time and do it well*

Interface Design - 2

- Interfaces should not over-promise
  - *Unless this comes for free*
- Get it right
  - *Beware of the dangers of abstraction (Word processor field search example with $O(n^2)$ complexity in Lampson's paper)*
- Make it fast rather than general and complete
  - *Tools matter (e.g., profilers)*
- Don't hide power
  - *Abstraction should not hide desirable properties*
  - *Multiplexing can have costs*

Interface Design - 3

- Use procedure arguments for flexibility
  - *In C you can use function pointers*

```
void qsort(void* base, size_t num, void size_t size, int (*compar)(const void *,const void *));
```

  - *In C++ you can use functors*

```
template< class RandomIt, class Compare > void sort( RandomIt first, RandomIt last, Compare comp );
```

- LD_PRELOAD trick

Interfaces Design - 4

- Leave it to the client (check the Exokernel paper in the schedule)
  - *Unix pipes*
- Keep interfaces stable
  - *Counterexample: LLVM's infamous backwards incompatibility*
- Keep a place to stand
  - *IBM 360/370' support of machines like 1401 and 7090*
  - *Virtualization*

Implementation - 1

- Plan to throw one away, you will anyhow
  - *Learn from your prototypes*
- Keep secrets
  - *Implementation details should be hidden from the clients*
  - *This can be a tradeoff as knowing secrets can help*
- Divide and conquer
  - *Concurrency (can be a struggle to get it right)*
- Use a good idea again
  - *Use the idea, not necessarily the implementation*

Implementation – 2 (Completeness)

- Handle normal and worst cases separately
  - *It might be OK to crash a few processes for the better of the many*
  - *Caches in processors are optimized for the common case (the principle of locality)*
  - *Paging is another classic example*
  - *Avoid premature optimizations*

https://cdn.instructables.com/F1G/FRJJ/I8UHFHXE/F1GFRJJI8UHFHXE.LARGE.jpg

Efficiency - 1

- Split resources
  - *It is faster to allocate dedicated resources and access them*
  - *Heterogeneous systems is a popular example*
    - Example: CPU + FPGA systems (Microsoft Azure example)
- Use static analysis (if you can)
  - *Compilers perform static analysis for optimization*
    - Example: LICM: Loop-Invariant Code Motion

```
for (int i = 0; i < n; i++) {
        x = y + z;
        a[i] = 6 * i + x * x;
}
```

Can be hoisted out of the loop

Efficiency - 2

- Cache answers
  - *Modifications should invalidate small portions of a cache*
  - *Examples: Memoization, TLB, processor caches*
- Use hints
  - *Saved results of some computation (can be wrong)*
  - *Examples: IP routing protocol, Ethernet*
- When in doubt, use brute force
  - *Pay attention to the constant factors*
- Compute in background
- Use batch processing
- Safety first
  - *Clever optimizations only for predictable workloads*

Reliability

- Design with reliability in mind
- End-to-end principle
  - *Error recovery at the application is necessary*
  - *Intermediate checks are for performance reasons*
  - *Examples: File Transfer, TCP/IP*
- Log updates
  - *Can be used to recover a system's state*
  - *Append-only -> Efficient*
  - *Log update procedures must be functional*
- Make actions atomic or restartable

Wrap-up

- When reading papers, think about which hints they apply/ignore
  - *The midterm will have questions about this*
- What other hints might we learn from the systems we study?
  - *Lampson updated his hints recently, perhaps we can add more to the list*
    - E.g., Approximation versus precision (Google Search vs. Microsoft Excel)
- What hints will you adopt/ignore in your projects?
  - *Think carefully for either decision*