Ass 4 : don't worry about vulnerable code

CSS503A

Lecture #18

2019-05-29

Ass 3 ..?

Course eval : Say nice things!

Administrivia

*Final

*Start grading lab 3 Thurs/Friday   lab 4 -> ?

- update: grading (in progress)
  -> rule of thumb: more time I spend coding at it, the lower your grade
- update: final

Question: Single binaries file
vs separate files for each class ?

- Posting assignment solutions to public spaces (e.g. github, domyhomeworkforme.com):

  Academic Misconduct!!

  - if you're naive enough to do it during the quarter, you may be sanctioned
  - can't do anything about it after you graduate (barring copyright takedown notice), but it's a pretty crappy thing to do

=> please do not defecate on my hard work by posting your solution for someone next year to copy

* okay to share privately to people who will never take 503 (significant others, potential employers, ...)

Our Story So Far

- networking
    - layers of protocol (OSI : 7-layer model)
    - TCP/IP & UDP/IP won the wars
    - Berkeley Sockets - multiple protocol support
    - can use sockets for IPC on single machine
        - Unix Domain
        - TCP/UDP - IP on loopback address
    - socket

        stream - read() / write()

        datagram - send/recv

Client-Server Systems

- difference between client & server : networking level
  - server: passive
  - client active

- difference between client & server : application level

  server provides some well-defined service
  - map name to IP address ?
  - what is the current time ?
  - run user shell on this machine ?
  - give me the file
  - let's play global thermonuclear war

=> API

Remote Procedure Calls

- RPC: API that looks like a function call
  - 2 components } ⊕ Service discovery
    - server
    - client library

- server:
  - accept request
  - unmarshal request & args
  - perform request
  - marshal response
  - send marshaled response back

- client library: "stub" to manage communication with server
  - marshal: "serialize"
  - unmarshal: "de-serialize"

- RPC is an abstraction, but
  - network lag / timeout
  - network failures (partition)

⇒ abstraction is "leaky"

- XML / RPC

  XML : encoding

  HTTP: transport mechanism (layer)

  -> evolved into SOAP (Simplified Object Access Protocol)

· See also JSON-RPC, REST { REpresentational State Transfer

- Google Protobufs / Stubby

REST - CRUD

↪ Create
Read
update
delete

RPC (cont.)

RPC: generic term (abstraction)

- multiple implementations
  - RPC over HTTP (XML/JSON) REST
  - Erlang
  - Google stubby/protobufs
  - roll-your-own

Sun RPC (AKA ONC [open network computing] RPC)

- original (?) - developed by Sun
- developed for NFS
  - 1st (?) remote-mounted file system
  - abstraction: looks like local filesystem (mostly)
- protocol (exchanges messages) } based on unix/C calling conventions
- rpcgen tool
- interface definition language (IDL)
  - C-like syntax
- works with both tcp & udp (transparent)
- port mapper (port 111) } service discovery
  - rpc bind
  - (directory service)

External Data Representation (XDR)

*.x

- function library
- independent at transport layer

- base unit : 4 bytes (big endian)
    - smaller data types occupy 4 bytes after encoding
    - strings, etc padded to 4-byte boundary
    - floats: IEEE 754

RPCINFO

```
rpcinfo -p
```

rpcgen (ONC RPC)

- protocol compiler : generate server & client stubs
  - input: RPCL (RPC Language)

rpcgen prog.x

→ prog_clnt.c
  prog_svc.c
  prog_xdr.c
  prog.h

Stubs:
prog_client.c
prog_server.c

Server side : register callable procedures

rpc_reg()
rpc_call()

Client side

clnt_create() / clnt_create_timed()
svc_create
clnt_call

Memory Hierarchy

L1	cache	~ ~~1/2~~ 1~2 cycle
L2		~ 7 cycles
L3		
Main memory		~ 100 cycles
Disk		(Seek: 8 M cycles)

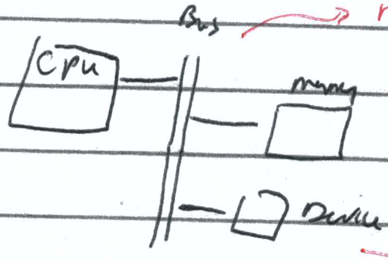
1 cycle ≈ 1 ns (Gigahertz processor)

- Cache aware programming } lecture #1
    - array processing

mutex lock/unlock : ~ 25 ~~instruction~~ cycles
(25 ns)

- numbers are subject to change

Main Memory

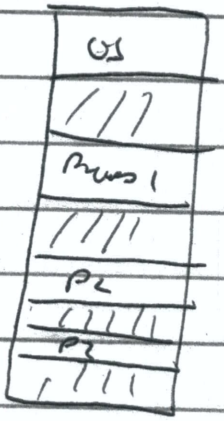


→ not unlike ETH

{ cpu write address lines
sets read/write line
~~sets~~ sets data

→ "memory-mapped I/O"

non-VM system:



- multiple processes loaded into memory at different locations

→ Concurrency
more efficient use of CPU

- processes waiting on I/O sleep while ready process run

- physical address
  - all memory (cached memory chaos) & I/O-mapped devices

problem

- main() - fixed address
- all functions - call <addr>
- global memory

- solution 1 : linkage editing
  - edit all addresses relative to start-of-process address

- solution 2 :
  - relative branch (current position + offset
  - doesnt work for global data

- solution 3
  - process-specific base address
  - kernel must reset "segment register" on context switches