Working in Unix environments: the shell

Juan Pablo Mallarino

Universidad de los Andes

jp.mallarino50@uniandes.edu.co

January 21, 2020

Facultad de Ciencias
Universidad de los Andes

the big question

Computation

What is computation?

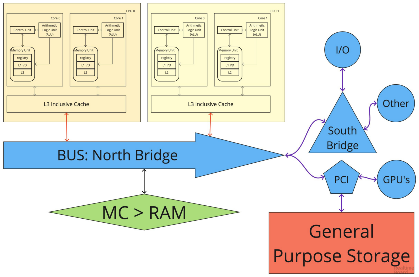


Figure: Von Neumann Architechture

the big question

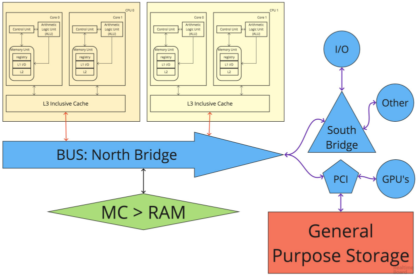


Figure: Von Neumann Architechture

Computation

What is computation?

Key infrastructure components

- Storage
- RAM
- Processing block: registries, instruction sets and clock
- FPGA's, GPU's, accelerators and other alternate processing units (RaspBerries, portable devices . . . ARM)
- Compilers - Machine language

the big question

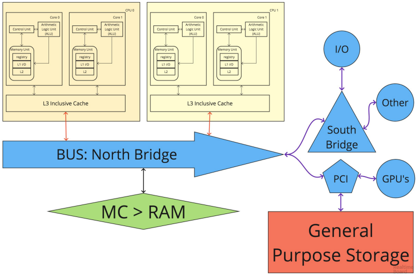


Figure: Von Neumann Architechture

Computation

What is computation?

Key infrastructure components

- Storage
- RAM
- Processing block: registries, instruction sets and clock
- FPGA's, GPU's, accelerators and other alternate processing units (RaspBerries, portable devices . . . ARM)
- Compilers - Machine language

Limitations & Complications

1. All of the above
2. Education: infrastucture topology, coding strategies, profiling & optimization
3. Interpreted languages
4. Unix like systems
5. Time - accelerating technologies and real-time applications

the big question

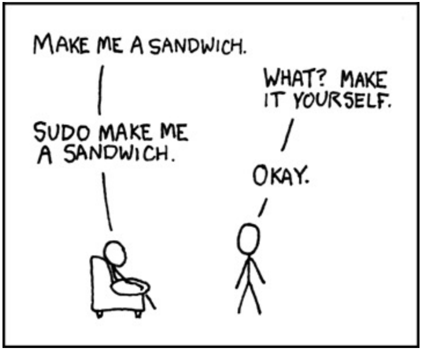
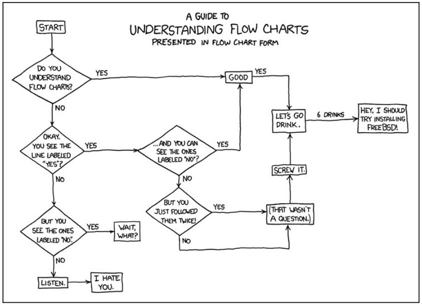


Figure: Not my jokes

Here comes UNIX

the big question



A GUIDE TO
UNDERSTANDING FLOW CHARTS
PRESENTED IN FLOW CHART FORM
START
DO YOU UNDERSTAND FLOW CHARTS?
YES
NO
GOOD
YES
LET'S GO DRINK.
6 DRINKS
HEY, I SHOULD TRY INSTALLING FREEBSD!
OKAY. YOU SEE THE LINE LABELED "YES"?
YES
NO
...AND YOU CAN SEE THE ONES LABELED "NO"?
YES
NO
SCREW IT.
BUT YOU JUST FOLLOWED THEM TWICE!
YES
NO
(THAT WASN'T A QUESTION.)
BUT YOU SEE THE ONES LABELED "NO".
YES
NO
WAIT, WHAT?
LISTEN.
I HATE YOU.

Figure: Not my jokes

Here comes UNIX

Advantages?

- † Uniform access to components
- † Kernel designed for administrating tasks, managing resources: the kernel space
- † Intuitively transparent for the user. Everything is accessible
- ⋆ Security
- † The shell: "One shoe fit for all"
- ‡ Software: C (Dennis Ritchie)
- ‡ Propietary Licensing to "Open Source" (BSD, FreeBSD & Linux)
- △ People, science & culture

⋆'s and △ is HPC

the big question

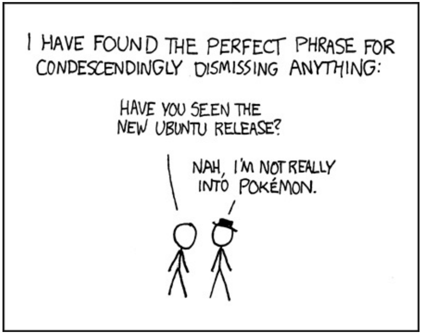

I HAVE FOUND THE PERFECT PHRASE FOR CONDESCENDINGLY DISMISSING ANYTHING:
HAVE YOU SEEN THE NEW UBUNTU RELEASE?
NAH, I'M NOT REALLY INTO POKÉMON.

Figure: Not my jokes

Here comes UNIX

Until we have. . . Ubuntu

Why a Linux system?

Advantages of Unix-like systems

- Filesystems (*df -hT*): NFS, Journaled, EXT, XFS, <long filenames>, we hate spaces
- Advanced Kernel
- Everything is a file! Even RAM, procs, eth, CPU, . . . (*lscpu, lspci, /dev/*?, /proc/*?*)
- Free / OpenSource software / Package Managers
- one shoe fit for all - terminal concept (Command Line Interpreter or CLI)
- Highly configurable - steep learning curve
- Your best friend: StackOverflow

the CLI

- Command interpreter: applications and builtin instructions, keywords, . . . [*type, which*]
- Screening and piping processes! [use of |, &, &&]
- Identifying processes: $$, $!, $?, *jobs --help* [type jobs?], *ps aux*
- Logging to stdout and stderr
- Custom outputting [use of > "file", >&1, 2>&1 > file, . . .]
- Environment Variables with scoping [*printenv, env*]
- Important Variables: PATH , LD_LIBRARY_PATH , CPATH , MANPATH , PYTHONPATH [use *echo $VAR*]
- Other Info Variables: HOME , USER , GROUPS , SHELL , HISTSIZE

the CLI

- Command interpreter: applications and builtin instructions, keywords, . . . [*type, which*]
- Screening and piping processes! [use of |, &, &&]
- Identifying processes: $$, $!, $?, *jobs --help* [type jobs?], *ps aux*
- Logging to stdout and stderr
- Custom outputting [use of > "file", >&1, 2>&1 > file, . . .]
- Environment Variables with scoping [*printenv, env*]
- Important Variables: PATH , LD_LIBRARY_PATH , CPATH , MANPATH , PYTHONPATH [use *echo $VAR*]
- Other Info Variables: HOME , USER , GROUPS , SHELL , HISTSIZE
- Scriptable and Encodable.
  Comments begin with #
  HEADING (shebang): #!/bin/bash [e.g. for python use #!/usr/bin/env python]
  After HEADING: # -*- coding: utf-8 -*-
- Conditionals, for & while looping, arithmetic and string operations, . . . even obtain random numbers!
- What is EOF?

the CLI

- Command interpreter: applications and builtin instructions, keywords, . . . [*type, which*]
- Screening and piping processes! [use of |, &, &&]
- Identifying processes: $$, $!, $?, *jobs --help* [type jobs?], *ps aux*
- Logging to stdout and stderr
- Custom outputting [use of > "file", >&1, 2>&1 > file, . . .]
- Environment Variables with scoping [*printenv, env*]
- Important Variables: PATH , LD_LIBRARY_PATH , CPATH , MANPATH , PYTHONPATH [use *echo $VAR*]
- Other Info Variables: HOME , USER , GROUPS , SHELL , HISTSIZE
- Scriptable and Encodable.
  Comments begin with #
  HEADING (shebang): #!/bin/bash [e.g. for python use #!/usr/bin/env python]
  After HEADING: # -*- coding: utf-8 -*-
- Conditionals, for & while looping, arithmetic and string operations, . . . even obtain random numbers!
- What is EOF?
- Addons: autocompletion
- Beautifiers

the CLI

- Command interpreter: applications and builtin instructions, keywords, . . . [*type, which*]
- Screening and piping processes! [use of |, &, &&]
- Identifying processes: $$, $!, $?, *jobs --help* [type jobs?], *ps aux*
- Logging to stdout and stderr
- Custom outputting [use of > "file", >&1, 2>&1 > file, . . .]
- Environment Variables with scoping [*printenv, env*]
- Important Variables: PATH , LD_LIBRARY_PATH , CPATH , MANPATH , PYTHONPATH [use *echo $VAR*]
- Other Info Variables: HOME , USER , GROUPS , SHELL , HISTSIZE
- Scriptable and Encodable.
  Comments begin with #
  HEADING (shebang): #!/bin/bash [e.g. for python use #!/usr/bin/env python]
  After HEADING: # -*- coding: utf-8 -*-
- Conditionals, for & while looping, arithmetic and string operations, . . . even obtain random numbers!
- What is EOF?
- Addons: autocompletion
- Beautifiers

Enter a terminal: *MyBinder*, *Azure Cloud*
Experience the command line

Useful commands

- Help: *man*, *info*, *-h* or *–help*
- Navigation: *ls*, *cd*, *pwd*, , *tree*

Listing & navigating through the filesystem

```
$   ls -a
$   ls -la
$   ls -lha
$   ls -help
$   man ls
$   info ls
$   ls -lha .bashrc
$   cd /
$   ls -lha
$   ls -lha /var/lib
$   cd -
$   cd
$   pwd
$   cd .
$   cd ..
$   pwd
$   tree
$   man tree
$   tree -d -L 2 /etc
```

Enter a terminal: *MyBinder*, *Azure Cloud*

Experience the command line

Useful commands

- Display: *echo*, *less*, *more*, *cat*, editors!
- File/Directory manipulation: *mkdir*, *touch*, *tar*, *zip* & *unzip*
- Testing: *sleep*, *test*, *seq*

Listing & navigating through the filesystem

```
$    cd
$    more .bashrc
$    less .bashrc
$    cat .bashrc
$    mkdir -p monte-carlo/sample1/test1
$    touch monte-carlo/sample1/test1/file1.txt
$    tar cvf mc.tar monte-carlo
$    ls -lha mc.tar
```

Enter a terminal: *MyBinder*, *Azure Cloud*

Experience the command line

Useful commands

- Searching: *locate*, *find*, *grep*
- Computing resources accounting: *top* & *htop*, *ps*, *jobs*
- Storage and devices info: *df*, *du*, *free*, *lsblk*, *lspci*, *lscpu*
- Detailed information on commands: *type* & *which*, *stat*

Environment behavior

- Process screening (&) and Piping (|)
- Identifying processes
- Environment variables, the *printenv* command
- Programming environment (useful commands: *test*, *seq*): if/elif/else, for loops

Running process in the background and retrieving job information

```
$   sleep 100 &
$   echo $!
$   echo $$
$   jobs -l
```

Simple example #1: generating a random number

Source Code 1: Generating multiple random numbers

```
 1    #!/bin/bash
 2    # -*- coding: utf-8 -*-
 3
 4    echo "Printing random numbers with /dev/random"
 5    stat /dev/random
 6    entropy=$(cat /proc/sys/kernel/random/entropy_avail)
 7    echo "How much entropy before calling /dev/random?
       ↪   $entropy"
 8
 9    # Now we create a file where we will store ages of turtles
10    echo "Turtle ages" > test.dat
11    for it in `seq 1000`
12    do
13        num=`od -An -N1 -i /dev/random`
14        if [ $num -gt 150 ]
15        then
16            let num=150+1
17        elif (( $num <= 10 ))
18        then
19            num=$(($num-1))
20        fi
21        echo $num >> test.dat
22    done
23    echo "How much entropy after calling /dev/random? $(cat
       ↪   /proc/sys/kernel/random/entropy_avail)"
```

Notice

1. Notice the output of stat is thrown
2. Notice the scope in line 6
3. Notice the syntax for the for loop and conditionals. There are multiple ways of verifying conditions
4. There are also multiple ways of doing math operations
5. Anything else?

Problem

How many turtles have age 63? 10? How many 21?

Fun Fact

How do i avoid using all CPU's?

```
# Review /proc/cpuinfo
$   taskset -cpu-list 1,2 command args
```