WORKING IN UNIX ENVIRONMENTS: THE SHELL

Juan Pablo Mallarino

jp.mallarino50@uniandes.edu.co https://github.com/jpmallarino/FISI2028-202120

August 11, 2021

Recomendaciones para Clases virtuales

Concentración

Atención es muy importante

Comodidad

Estar muy cómodos NO , lo necesario.

¿Quiénes somos?

A pesar de la distancia, todavía somos personas. No existe manera adecuada de decirlo, pero POR FAVOR PRENDAN LA CÁMARA

Ambientación

Iluminación y sonido adecuados. Evitar el celular y las distracciones innecesarias

Disfrutar

Aprender puede ser frustrante, más si estamos solos. Pero la verdad no estamos solos. Si es necesario, interrumpan y volvemos a comenzar.

the big question

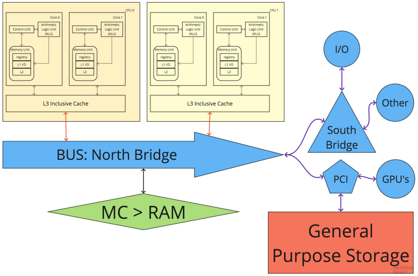

Figure: Von Neumann Architechture

Computation

What is computation?

the big question

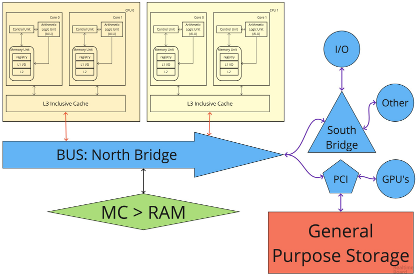

Figure: Von Neumann Architechture

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

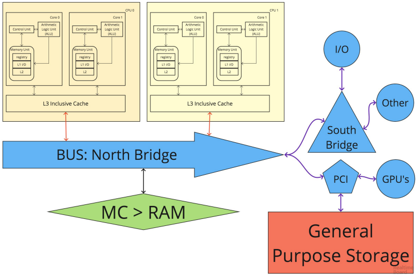

Figure: Von Neumann Architechture

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

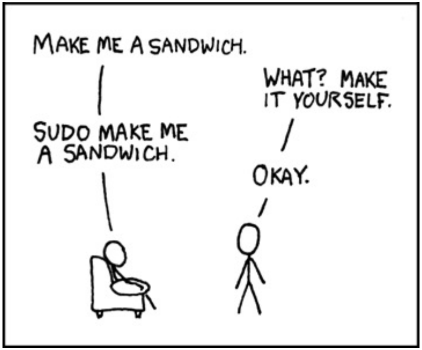

Figure: Not my jokes

Here comes UNIX

the big question

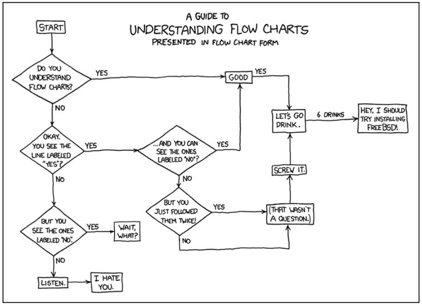
A GUIDE TO
UNDERSTANDING FLOW CHARTS
PRESENTED IN FLOW CHART FORM
START
DO YOU UNDERSTAND FLOW CHARTS?
YES
GOOD
YES
NO
LET'S GO DRINK.
6 DRINKS
HEY, I SHOULD TRY INSTALLING FREEBSD!
OKAY. YOU SEE THE LINE LABELED "YES"?
YES
...AND YOU CAN SEE THE ONES LABELED "NO"?
YES
NO
SCREW IT.
NO
BUT YOU JUST FOLLOWED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

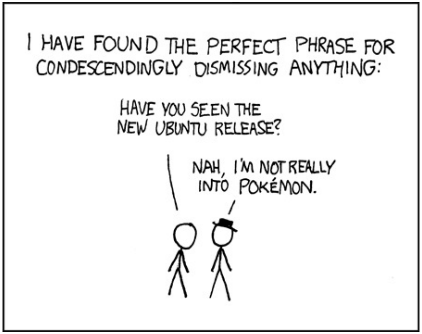
I HAVE FOUND THE PERFECT PHRASE FOR CONDESCENDINGLY DISMISSING ANYTHING:
HAVE YOU SEEN THE NEW UBUNTU RELEASE?
NAH, I'M NOT REALLY INTO POKÉMON.

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

Enter a terminal: *MyBinder*

Experience the command line

Useful commands

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

Enter a terminal: *MyBinder*

Experience the command line

Useful commands

- Display: *echo*, *less*, *more*, *cat*, *head*, *tail*, editors (Nano, Vim, Emacs)!
- File/Directory manipulation: *mkdir*, *tar*, *zip* & *unzip*, *cp*, *scp*, *mv*, *rm*
- Dummy or symbolic files/directories: *touch*, *mktemp*, *ln*
- Testing: *sleep*, *test*

Enter a terminal: *MyBinder*

Experience the command line

Useful commands

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

Enter a terminal: *MyBinder*

Experience the command line

Useful commands

- For loops: *seq*, *{#start..#finish}*
- String manipulation and replacement: *wc*, *cut*, *sed*, *awk*
- Advanced finding: *grep*
- Numerical calculations: *awk*
- Filtering, ordering: *sort*, *uniq*
- Comparing: *diff*, *md5*
- Patching: *patch*

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

I caní remember my code!

1. Design Patterns
2. Refactoring

https://refactoring.guru/

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?

Environment behavior

- Special instructions: \ ; & && | || > (>&1 >&2 1> 2>) <
- Identifying processes: ($$)
- Environment variables, the *printenv* command
- Programming environment (useful commands: *test*, *seq*): if/elif/else, for loops
- Only for scripting! $@, $#, $<any number>
- Status of a process? $?

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

Fun Fact

How do i avoid using all CPU's?

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

Instructions

1. Clone the github repo
2. Run the script `app1.sh`

Exercise 1

Run the script `app1.sh` with proper arguments

Exercise 2

Run the script `app1.sh` with proper arguments and store app info into a log file

Exercise 3

Create a text file with inputs for the application and run `./app1.sh < inputs.txt`

Exercise 4

Create a tree structure for multiple experiments

Exercise 5

Create a script that can run x number of experiments (as an input to the script) simultaneously in each of the folders. Homework for tuesday 28th of January 2020 at 12:30pm on SICUA

Questions on `app1.sh`?

Help another teammate!