SISTEMAS COMPUTACIONAIS
E
SEGURANÇA

PROF. RICARDO SCHWACH
MSC, MBA, PMP, PSM, ITIL, COBIT

Calendário

AGOSTO
15/08 – Início das Aulas
22/08 – Aula 1
29 a 31/08 – **TECHWEEK**

SETEMBRO
05/09 – Aula 2
12/09 – Aula 3
19/09 – Aula 4
26/09 – Aula 5

OUTUBRO
03/10 – Aula 6
10/10 – Aula 7
17/10 – Aula 8
19 e 20/10 – AVALIAÇÃO A1
24/10 – Aula 9
31/10 – Aula 10

NOVEMBRO
07/11 – Aula 11
14/11 – Aula 12
21/11 – Aula 13
28/11 – Aula 14

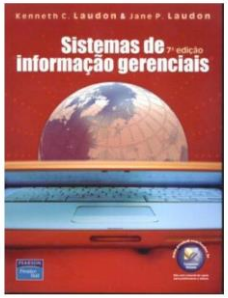
DEZEMBRO
04 a 08/12 – AVALIAÇÃO A3
11 e 12/12 – AVALIAÇÃO A2
19/12 – Término do semestre letivo

AVALIAÇÕES
A1 – Avaliação (30%)
A2 – Avaliação (30%)
A3 – Avaliação (40%)

Estudo de Caso

https://plataforma.bvirtual.com.br/Leitor/Publicacao/375/pdf/0?code=//KyNazKCKntXbLtSZ41HZHfjKxa41TtNkGsyJMHX8PaBEqY65EJsibjk8Bzi8dHz6J2vegBMXFSKaOo04D4sQ==

TECNOLOGIA EM DESTAQUE — O mundo corporativo migra para o código-fonte aberto



1. Quais problemas o Linux e outros softwares de código aberto ajudaram as empresas a enfrentar?
2. Como as ajudaram?
3. Quais problemas e desafios fez aumentar a adoção de softwares de código aberto?
4. O que pode ser feito para enfrentar esses problemas?
5. Para você, qual seria a melhor estratégia para implantar o Linux e outros componentes de código aberto em seu atual estado de evolução?

Simulador Linux

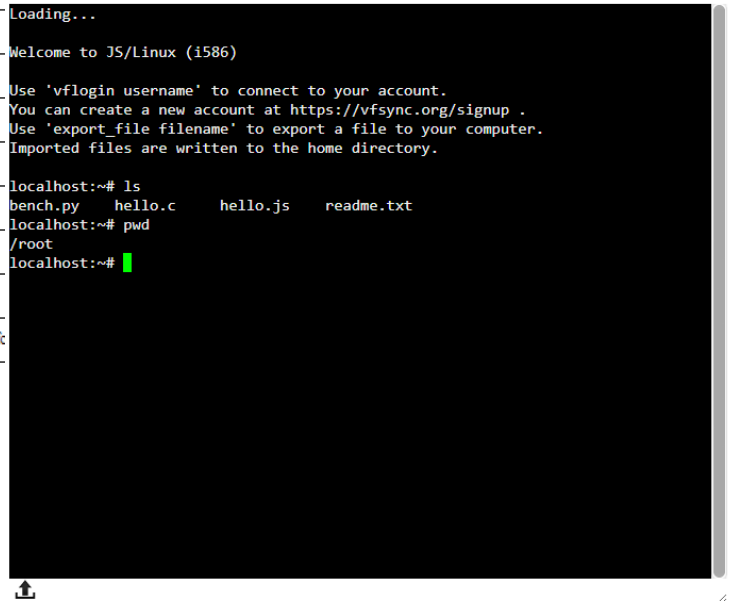
https://bellard.org/jslinux/

JSLinux

Run Linux or other Operating Systems in your browser!

The following emulated systems are available:

CPU	OS	User Interface	VFsync access	Startup Link	TEMU Config	Comment
x86	Alpine Linux 3.12.0	Console	Yes	click here	url	
x86	Alpine Linux 3.12.0	X Window	Yes	click here	url	Right mouse button for the menu.
x86	Buildroot (Linux)	Console	Yes	click here	url	
x86	Buildroot (Linux)	X Window	Yes	click here	url	Right mouse button for the menu.
x86	Windows 2000	Graphical	No	click here	url	Disclaimer.
x86	FreeDOS	VGA Text	No	click here	url	
riscv64	Buildroot (Linux)	Console	Yes	click here	url	
riscv64	Buildroot (Linux)	X Window	Yes	click here	url	Right mouse button for the menu.
riscv64	Fedora 29 (Linux)	Console	Yes	click here	url	Warning: longer boot time.
riscv64	Fedora 29 (Linux)	X Window	Yes	click here	url	Warning: longer boot time. Right mouse button fc



- Mostrar o **CPU** virtual que está sendo simulado pelo JSLinux com `lscpu`

```
localhost:~#
localhost:~#
localhost:~# lscpu
Architecture:          i586
CPU op-mode(s):        32-bit
Byte Order:            Little Endian
Address sizes:         32 bits physical, 32 bits virtual
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
Vendor ID:             AuthenticX86
CPU family:            5
Model:                 4
Model name:            05/04
Stepping:              3
CPU MHz:               100.000
BogoMIPS:              200.00
Virtualization:        AMD-V
Flags:                 fpu tsc msr pae cx8 cmov clflush mmx fxsr sse sse2 rdtscp cpuid svm npt
localhost:~#
```

- Mostrar a **quantidade de memória** do sistema com `free`
  - Usar as opções `free -b` para mostrar em bytes, `free -k` em kiB, `free -m` em MiB

```
localhost:~#
localhost:~# free -b
              total        used        free      shared  buff/cache   available
Mem:      191737856     5021696   185765888        4096      950272   183549952
Swap:             0           0           0
localhost:~# free -k
              total        used        free      shared  buff/cache   available
Mem:         187244        4904      181412           4         928      179248
Swap:             0           0           0
localhost:~# free -m
              total        used        free      shared  buff/cache   available
Mem:            182           4         177           0           0         175
Swap:             0           0           0
localhost:~#
```

- Compilar um programa em C usando o `gcc` e executá-lo

```
localhost:~#
localhost:~# cat > programa.c
#include <stdio.h>

float a, b, c;

int main() {
   a = 3.14;
   b = 5.2;
   c = a * b + 1;
   printf("%f\n", c);
   return 0;
}
localhost:~# gcc programa.c -o programa
localhost:~# ./programa
17.327999
localhost:~#
```

Editores Linux

Vi programa.c

Editar = i
Sair e gravar = <ESC> :wq

OU

Nano programa.c

Isso vai gerar um arquivo executável (opção -o). Usar control-D para terminar o comando `cat`.
Ao executar, o resultado da operação é exibido no terminal.

- Mostrar o conteúdo do arquivo executável gerado (arquivo "programa") em hexadecimal usando **xxd**

```
localhost:~# xxd programa
00000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............
00000010: 0300 0300 0100 0000 6810 0000 3400 0000  ........h...4...
00000020: 7c42 0000 0000 0000 3400 2000 0a00 2800  |B......4. ...(.
00000030: 2100 2000 0600 0000 3400 0000 3400 0000  !. .....4...4...
00000040: 3400 0000 4001 0000 4001 0000 0400 0000  4...@...@.......
00000050: 0400 0000 0300 0000 7401 0000 7401 0000  ........t...t...
00000060: 7401 0000 1700 0000 1700 0000 0400 0000  t...............
00000070: 0100 0000 0100 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 6003 0000 6003 0000 0400 0000  ....`...`.......
00000090: 0010 0000 0100 0000 0010 0000 0010 0000  ................
000000a0: 0010 0000 cd02 0000 cd02 0000 0500 0000  ................
000000b0: 0010 0000 0100 0000 0020 0000 0020 0000  ......... ... ..
000000c0: 0020 0000 b800 0000 b800 0000 0400 0000  . ..............
000000d0: 0010 0000 0100 0000 fc2e 0000 fc3e 0000  .............>..
000000e0: fc3e 0000 0801 0000 3401 0000 0600 0000  .>......4.......
000000f0: 0010 0000 0200 0000 0c2f 0000 0c3f 0000  ........./...?..
00000100: 0c3f 0000 c000 0000 c000 0000 0600 0000  .?..............
00000110: 0400 0000 50e5 7464 0c20 0000 0c20 0000  ....P.td. ... ..
00000120: 0c20 0000 2400 0000 2400 0000 0400 0000  . ..$...$.......
00000130: 0400 0000 51e5 7464 0000 0000 0000 0000  ....Q.td........
00000140: 0000 0000 0000 0000 0000 0000 0600 0000  ................
00000150: 1000 0000 52e5 7464 fc2e 0000 fc3e 0000  ....R.td.....>..
00000160: fc3e 0000 0401 0000 0401 0000 0400 0000  .>..............
```

- Mostrar o arquivo executável decodificado em instruções de máquina usando `objdump -d`

```
localhost:~#
localhost:~# objdump -d programa

programa:     file format elf32-i386


Disassembly of section .init:

00001000 <_init>:
    1000:       83 ec 0c                sub    $0xc,%esp
    1003:       e8 be 01 00 00          call   11c6 <frame_dummy>
    1008:       e8 8d 02 00 00          call   129a <__do_global_ctors_aux>
    100d:       83 c4 0c                add    $0xc,%esp
    1010:       c3                      ret

Disassembly of section .plt:

00001020 <.plt>:
    1020:       ff b3 04 00 00 00       pushl  0x4(%ebx)
    1026:       ff a3 08 00 00 00       jmp    *0x8(%ebx)
    102c:       00 00                   add    %al,(%eax)
        ...

00001030 <printf@plt>:
    1030:       ff a3 0c 00 00 00       jmp    *0xc(%ebx)
    1036:       68 00 00 00 00          push   $0x0
    103b:       e9 e0 ff ff ff          jmp    1020 <.plt>

00001040 <__libc_start_main@plt>:
    1040:       ff a3 10 00 00 00       jmp    *0x10(%ebx)
    1046:       68 08 00 00 00          push   $0x8
    104b:       e9 d0 ff ff ff          jmp    1020 <.plt>

Disassembly of section .plt.got:
```

- Mostrar no código desassemblado as instruções que fazem os cálculos que aparecem no programa fonte (multiplicação e adição de ponto flutuante). Mostrar que as instruções são executadas em uma sequência de endereços de memória crescentes.

```
    1210:       c3                      ret

00001211 <__x86.get_pc_thunk.cx>:
    1211:       8b 0c 24                mov    (%esp),%ecx
    1214:       c3                      ret

00001215 <main>:
    1215:       8d 4c 24 04             lea    0x4(%esp),%ecx
    1219:       83 e4 f0                and    $0xfffffff0,%esp
    121c:       ff 71 fc                pushl  -0x4(%ecx)
    121f:       55                      push   %ebp
    1220:       89 e5                   mov    %esp,%ebp
    1222:       53                      push   %ebx
    1223:       51                      push   %ecx
    1224:       e8 e4 ff ff ff          call   120d <__x86.get_pc_thunk.ax>
    1229:       05 a3 2d 00 00          add    $0x2da3,%eax
    122e:       8d 90 60 00 00 00       lea    0x60(%eax),%edx
    1234:       d9 80 38 e0 ff ff       flds   -0x1fc8(%eax)
    123a:       d9 1a                   fstps  (%edx)
    123c:       8d 90 58 00 00 00       lea    0x58(%eax),%edx
    1242:       d9 80 3c e0 ff ff       flds   -0x1fc4(%eax)
    1248:       d9 1a                   fstps  (%edx)
    124a:       8d 90 60 00 00 00       lea    0x60(%eax),%edx
    1250:       d9 02                   flds   (%edx)
    1252:       8d 90 58 00 00 00       lea    0x58(%eax),%edx
    1258:       d9 02                   flds   (%edx)
    125a:       de c9                   fmulp  %st,%st(1)
    125c:       d9 e8                   fld1
    125e:       de c1                   faddp  %st,%st(1)
    1260:       8d 90 5c 00 00 00       lea    0x5c(%eax),%edx
    1266:       d9 1a                   fstps  (%edx)
    1268:       8d 90 5c 00 00 00       lea    0x5c(%eax),%edx
    126e:       d9 02                   flds   (%edx)
    1270:       83 ec 04                sub    $0x4,%esp
```

Exercícios
Linux

1. Criar um Shell Script

Vi oi.sh

```
#!/bin/bash
echo "Oi, tudo bem?"

~
~
```

Digitar os comandos ":wq" → Enter

Executar os comandos abaixo:

```
localhost:/professor# chmod a+x Oi.sh
localhost:/professor# ls
Oi.sh
localhost:/professor# ls -l
total 4
-rwxr-xr-x    1 root     root            34 Aug 24 22:28 Oi.sh
localhost:/professor# ./Oi.sh
Oi, tudo bem?
```

Editores Linux

Vi programa.c

Editar = i
Sair e gravar = <ESC> :wq

OU

Nano programa.c

2. Criar o Shell Script abaixo:

Primeiro shell script

```
#!/bin/bash

echo "Hello, World!"
```

```
TEXTO="Hello, World!"

echo $TEXTO
```

```
#!/bin/bash
data=$(date +"%y/%m/%d-%H%M")
echo $1 "$data"-$2
```

Exemplo de loop e condição

```
#!/bin/bash

COUNT=0

for i in `ls $1`

do                done

COUNT=$((         if [ "$COUNT" -eq 0 ]; then

                  echo "Nenhum arquivo encontrado"

                  elif [ "$COUNT" -eq 1 ]; then

                  echo "Apenas 1 arquivo foi encontrado"

                  else


                 echo "Foram encontrados $COUNT arquivos"

                 fi
```

Exercícios

Exercícios

1. Determine o número que se obtém ao se escrever o número 3 no sistema de numeração de base 2.

A) (111)

B) (101)

C) (10)

D) (01)

E) (11)

Temos que 3 = 1.2 + 1.1

Logo, 3 na base 2 é 11

Exercícios

2. Considere A = (11000) e B = (10001) , números escritos no sistema de numeração de base 2. Escreva-os no sistema de numeração de base 10 e determine o valor de A- B.

A) -7

B) 41

C) -17

D) 0

E) 7

Passando para a base 10:

11000 = 1.16 + 1.8 + 0.4 + 0.2 + 0.1 = 24

10001 = 1.16 + 0.8 + 0.4 + 0.2 + 1.1 = 17

24 – 17 = 7

Exercícios

3. Converter $110111_{(2)}$ para a base decimal.

$$110111_{(2)} = 1x2^0 + 1x2^1 + 1x2^2 + 0x2^3 + 1x2^4 + 1x2^5 = 55_{(10)}$$

Exercícios

4. Converter $1101100_{(2)}$ para a base hexadecimal.

Como 16 é múltiplo de 2 (2^4), basta separar o número dado de 4 em 4 algarismos da direita para a esquerda e convertê-los da base 2 para a base 10 para se encontrar o resultado na base 16.

$1101100_{(2)} = 110|1100 = 6C_{(16)}$

Exercícios

5. Converter $198_{(10)}$ para a base binária.

198/2 = 99, resto 0
99/2 = 49, resto 1
49/2 = 24, resto 1
24/2 = 12, resto 0
12/2 = 6, resto 0
6/2 = 3, resto 0
3/2 = 1, resto 1
$198_{(10)} = 11000110_{(2)}$

Exercícios

6. Converter $889_{(10)}$ para a base hexadecimal.

Exercícios

7. Converter $ABB_{(16)}$ para a base binária.

Exercícios

8. Converter $FEA_{(16)}$ para a base decimal.

Exercícios

9. Converter o número binário 10011011 para decimal:

Exercícios

10. Conversão do Binário 100101100 para Hexadecimal :

Obrigado

PROF. RICARDO SCHWACH
MSC, MBA, PMP, PSM, ITIL, COBIT