

Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Mecânica

**PROJETO DE DISCIPLINA PARA O
APRENDIZADO EM LINUX E
RASPBERRY PI**

Marcel de Sena Dall'Agnol

São Carlos
2014

Marcel de Sena Dall’Agnol

**PROJETO DE DISCIPLINA PARA O
APRENDIZADO EM LINUX E
RASPBERRY PI**

Orientador: Prof. Dr. Daniel Varela Magalhães

Trabalho de conclusão do curso de
Engenharia Mecatrônica apresen-
tado à Escola de Engenharia de
São Carlos como parte dos requisi-
tos necessários à aprovação na dis-
ciplina Trabalho de Graduação.

São Carlos

2014

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

D144p Dall Agnol, Marcel de Sena
 Projeto de disciplina para o aprendizado em Linux e
Raspberry Pi / Marcel de Sena Dall Agnol; orientador
Daniel Varela Magalhães. São Carlos, 2014.

 Monografia (Graduação em Engenharia Mecatrônica) --
Escola de Engenharia de São Carlos da Universidade de
São Paulo, 2014.

 1. Linux. 2. Raspberry Pi. 3. Projeto de
disciplina. 4. GPIO. 5. Python. I. Título.

FOLHA DE AVALIAÇÃO

Candidato(s): Marcel de Sena Dall'Agnol

Título: PROJETO DE DISCIPLINA PARA O APRENDIZADO EM LINUX E RASPBERRY PI.

**Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos da
Universidade de São Paulo
Curso de Engenharia Mecatrônica.**

BANCA EXAMINADORA

Prof. Dr. Rodrigo Nicoletti

Nota atribuída: 9,0 (nove)

Rodrigo Nicoletti
(assinatura)

Prof. Dr. Adriano Almeida Gonçalves Siqueira

Nota atribuída: 9,0 (nove)

A. Siqueira
(assinatura)

Prof. Dr. Daniel Varela Magalhães (orientador)

Nota atribuída: 9,0 (NOVE)

Daniel Varela Magalhães
(assinatura)

Média: 9,0 (NOVE)

Resultado: APROVADO

Data: 28/11/2014

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

☒ SIM ☐ NÃO Visto do orientador: Daniel

Agradecimentos

Aos meus pais, Egláida e Amélio, por me apoiarem em todos os projetos e empreitadas, e pela confiança em minha capacidade e minhas escolhas.

A meus avós, Lincoln e Lourdes, que se fizeram presentes mesmo a distância, e a quem, junto a meus pais, dedico este trabalho.

Aos amigos com quem contei sempre que preciso durante meus anos em São Carlos: Jonas, Henrique, Maycon, Saulo, Rafael e tantos outros com quem compartilhei momentos que me marcaram.

A Amanda, pelos imensos aprendizados, sem a qual não me tornaria a pessoa que sou hoje.

A Aline, pelo companheirismo e em especial pelo suporte e paciência nos últimos meses deste processo.

Resumo

Linux é um sistema operacional com crescente importância no cenário mundial, especialmente se considerados nichos mais técnicos, para além de desktops. As possibilidades oferecidas pelo sistema, juntamente com seu crescimento em popularidade, tornam-o cada vez mais relevante à formação de estudantes em áreas de tecnologia. Esta necessidade é acentuada ainda mais para estudantes de engenharia mecatrônica, para os quais a ubiquidade de dispositivos computacionais e a crescente complexidade de seus sistemas de controle modificam diretamente a conjuntura de trabalho e pesquisa.

Reconhece-se a necessidade, portanto, de uma disciplina que contemple este assunto no curso de graduação em engenharia mecatrônica da EESC-USP. Este trabalho desenvolve um projeto de disciplina que contemple, em nível introdutório, as demandas identificadas. Para a disciplina no contexto do curso de graduação e aproximá-la de aplicações concretas, introduz-se um eixo ao redor do qual conceitos são desenvolvidos: o Raspberry Pi.

Raspberry Pi é um computador pequeno, de baixo custo, baixo consumo e controlado via um sistema operacional Linux. Escolhido como plataforma de desenvolvimento para o curso, o dispositivo oferece todos os recursos necessários para o aprendizado de Linux, além de apresentar características (tanto recursos quanto limitações) semelhantes às que podem ser encontradas em um ambiente profissional, em particular para aplicações embarcadas.

Sendo assim, é construído, tomando Linux e Raspberry Pi como eixos centrais, planos de aula voltados ao professor, correspondentes a 20 aulas, que incluem, além da sequência de conteúdos, exemplos de código, temas para discussão, recursos visuais, entre outros. Os temas abordados nestas aulas incluem desde tópicos básicos (como utilização de comandos básicos em um terminal UNIX) a complexos (como complexidade algorítmica e paradigmas de programação), os últimos disponibilizados como conteúdo adicional.

Palavras-chave: Linux, Raspberry Pi, projeto de disciplina, GPIO, Python

Abstract

Linux is an operating system with growing importance in the global scenario, especially when considering technical niches beyond desktops. The possibilities offered by the platform, along with its growing popularity, make it increasingly relevant for the academic studies of any student majoring in technological areas of knowledge. This need is further accentuated for students of mechatronic engineering, to whom the ubiquity of computing devices and the growing complexity of their control systems are directly addressed, both in industry and research.

Therefore, one identifies the need for an undergraduate course which contemplates this topic in the mechatronic engineering major of EESC-USP. Moving toward this objective, this work develops a course to teach, in an introductory level, these identified demands. However, in order to situate this course in the context of the major in question, and also draw it closer to concrete applications, an axis around which is introduced, around which concepts are to be developed.

This axis is the Raspberry Pi, a small, low cost and low consumption computer, controlled via a Linux operating system. Chosen as the development platform for the course, the device offers all necessary resources to learn Linux, besides revealing characteristics (both resources and limitations) similar to those which may be found in a professional environment, particularly for embedded systems.

Therefore, taking Linux and Raspberry Pi as its central axes, lesson plans built for the professor are laid out, for a corresponding total of 20 classes, which include, besides the sequence of contents, code snippets, topics for discussion, visual resources, among others. Both basic themes (such as basic command operation in a UNIX terminal) as well as complex ones (such as algorithmic complexity and programming paradigms) are touched upon, with the latter made available as additional content.

Keywords: Linux, Raspberry Pi, course project, GPIO, Python

Sumário

Lista de Figuras	9
Lista de Tabelas	11
1 Introdução	12
2 Conceitos iniciais	15
2.1 UNIX	15
2.2 Linux	17
2.3 Cursos anteriores	19
2.4 Software livre	19
2.5 Robot Operating System e Gazebo	20
2.6 Raspberry Pi	21
2.7 Python	22
2.8 Lua	23
2.9 LaTeX	24
2.9.1 Beamer	25
3 Motivação	26
4 Justificativas	34
4.1 Plataforma	34
4.1.1 BeagleBone Black	34
4.1.2 Banana Pi	35
4.1.3 Odroid-U2	35
4.2 Disciplina	37
5 Hipóteses e objetivos	40
5.1 Disponibilidade de tempo	40
5.2 Recursos materiais	41

5.3	Habilidades desenvolvidas	42
6	Resultados	45
6.1	Ementa	45
6.2	Aulas	47
6.2.1	Aula 1: UNIX, Linux e Open Source	47
6.2.2	Aula 2: Operação básica por linha de comando	50
6.2.3	Aula 3: Programas de linha de comando	53
6.2.4	Aulas 4 e 5: Características estruturais de Linux	56
6.2.5	Aula 6: Python	60
6.2.6	Aula 7: Apresentação, montagem e configuração do Raspberry Pi	63
6.2.7	Aula 8: Instalação de programas e configuração de serviços	65
6.2.8	Aula 9: Controle básico de GPIO	67
6.2.9	Aula 10: Controle de GPIO com comunicação pela Internet	69
6.2.10	Aulas 11 a 15: Projetos com Raspberry Pi	70
6.2.11	Aula 16: Uso de Lua no Raspberry Pi	72
6.2.12	Aulas 17 e 18: Noções de engenharia de software e complexidade algorítmica	73
6.2.13	Aula 19: Linguagens de programação e desempenho	78
6.2.14	Aula 20: Paradigmas de programação	83
7	Conclusão	86
	Referências	89
8	Anexos	92
8.1	Anexo 1: Divisão de disciplinas em áreas	92
8.2	Anexo 2: Slides da Aula 1	96
8.3	Anexo 3: Programas da Aula 6	103
8.4	Anexo 4: Programas da Aula 9	104
8.5	Anexo 5: Programas da Aula 10	108

8.6	Anexo 6: Programas da Aula 16	111
8.7	Anexo 7: Programas da Aula 19	115

Lista de Figuras

1	Ken Thompson e Dennis Ritchie	16
2	Terminal UNIX	16
3	Linha do tempo de sistemas UNIX e UNIX-like	17
4	Market share de dispositivos móveis em 2013 [1]	18
5	Vendas de smartphones em 2013 [1]	18
6	Ranking dos 500 supercomputadores com maior poder de processamento do mundo [2]	19
7	Richard Stallman	20
8	Logo do projeto GNU	20
9	Logo do framework ROS com OSRF	21
10	Logo do software de simulação Gazebo	21
11	Vista superior do Raspberry Pi	22
12	Logo do Raspberry Pi	22
13	Guido van Rossum	23
14	Logo da linguagem de programação Python	23
15	Logo da linguagem de programação Lua	23
16	Leslie Lamport	25
17	Logo do processador de texto LaTeX	25
18	Orçamento na empresa Farnell-Newark	29
19	Orçamento na empresa Farnell	29
20	Orçamento na empresa MCM	30
21	Orçamento na empresa Adafruit	30
22	Projetos comerciais com Raspberry Pi	32
23	Vista superior do BeagleBone Black	35
24	Vista isométrica do Banana Pi	35
25	Vista isométrica do Odroid-U2	36

26	Capa da apresentação de slides da aula 1	50
27	Estrutura em camadas de Linux	57
28	Conectores GPIO no Raspberry Pi	68
29	Mapa de numeração dos pinos GPIO no Raspberry Pi	68
30	Tempo de execução de <i>bubble sort</i> , <i>insertion sort</i> e <i>merge sort</i>	76
31	Tempo de execução e função ajustada aos três algoritmos . .	77
32	Tempo de execução de <i>bubble sort</i> em linguagens de progra- mação distintas	80

Lista de Tabelas

1	Distribuição de carga horária em disciplinas do curso de engenharia mecatrônica	28
2	Resumo dos orçamentos para compra de Raspberry Pi e acessórios	31
3	Comparação de hardware de alternativas para plataforma de desenvolvimento	36
4	Comandos iniciais em UNIX	51
5	Comandos básicos UNIX adicionais	53
6	Programas de linha de comando	55
7	System calls em Linux	58
8	Especificações dos modelos de Raspberry Pi	64
9	Comparação de desempenho em programas hipotéticos	74
10	Complexidade algorítmica de alguns algoritmos comuns	78
11	Tempo de execução de <i>bubble sort</i> em linguagens de programação distintas	79
12	Disciplinas classificadas em <i>Ciclo básico</i>	92
13	Disciplinas classificadas em <i>Mecânica</i>	93
14	Disciplinas classificadas em <i>Elétrica/Eletrônica</i>	93
15	Disciplinas classificadas em <i>Computação</i>	94
16	Disciplinas classificadas em <i>Interdisciplinares</i>	94
17	Disciplinas classificadas em <i>Outras</i>	95

1 Introdução

Este trabalho parte da avaliação, do ponto de vista de um aluno prestes a finalizar o curso de engenharia mecatrônica, da ausência de abordagem de um tópico muito importante para um estudante de tal área do conhecimento: Linux.

Linux é um sistema operacional de código aberto (open source), largamente utilizado em aplicações de nível profissional. Juntamente com Windows e MacOS, configura como um dos sistemas operacionais para desktops mais utilizados no mundo. Apesar de ser, por uma larga margem, o menos utilizado dos três nesse nicho, Linux é dominante em outros (dispositivos móveis, supercomputadores, alguns nichos da indústria, como robótica). Existe, em particular, o Robot Operating System (ROS), “um framework flexível para escrever software para robôs. É uma coleção de ferramentas, bibliotecas e convenções com o objetivo de simplificar o processo de criar comportamento robótico complexo e robusto em uma variedade de plataformas robóticas”. ROS é desenvolvido e oferece suporte apenas para Linux, e, com a crescente utilização do pacote de software somada ao igualmente crescente uso de Linux em outras aplicações, torna-se indispensável a familiaridade de um estudante de engenharia mecatrônica com o sistema.

Partindo desta demanda, este trabalho apresenta um projeto de disciplina para a engenharia mecatrônica, tomando como eixo central o aprendizado de Linux. Deste eixo, trabalham-se diversos outros conceitos (software livre, código aberto, operação por linha de comando), linguagens de programação (Python), e plataformas (Raspberry Pi).

Raspberry Pi é um computador aproximadamente do tamanho de um cartão de crédito, com hardware livre, desenvolvido pela Raspberry Pi Foundation no Reino Unido, com o intuito de promover o ensino de computação em escolas. O Raspberry Pi oferece suporte para instalação e uso de um sistema operacional completo, com todos os componentes essenciais de um PC ou laptop de tamanho regular. É disponibilizada oficialmente, pela fundação,

uma versão de Debian Linux otimizada para uso no dispositivo.

Raspberry Pi resume, em um dispositivo, o elo entre a aprendizagem de Linux e habilidades essenciais a um engenheiro mecatrônico. Devido ao seu tamanho e consumo de energia, o dispositivo é apropriado para aplicações embarcadas. Além disso, seu hardware e recursos limitados fornecem desafios que, além de úteis pedagogicamente, correspondem a desafios reais em uma situação profissional.

Sendo assim, forma-se um eixo em torno do qual a disciplina é estruturada: o Raspberry Pi. É ele o dispositivo que serve como plataforma para aplicação, na prática, dos conceitos trabalhados ao longo da disciplina. Para sua operação, é necessária familiaridade com linha de comando em Linux, e o aparelho oferece bibliotecas e recursos em Python para comunicação com componentes eletrônicos. Assim, possibilita o trabalho de uma larga gama de conceitos em nível introdutório.

Este trabalho é organizado na sequência a seguir.

Os *conceitos iniciais* expõem os principais conceitos necessários para a compreensão da disciplina, das razões por que ela é proposta e da produção do material. Realiza-se uma exposição breve sobre Unix, Linux, Raspberry Pi, Python, Lua e LaTeX, além de resgatar trabalhos anteriores, nos quais este se baseia parcialmente.

Nas *justificativas*, argumenta-se em favor das escolhas realizadas neste, desde a fundamentação para o oferecimento da disciplina em si até a preferência pela abordagem de determinados conceitos e plataformas em detrimento de outros.

Em *hipóteses e objetivos*, explicitam-se os pressupostos contidos na formulação do projeto de disciplina, como o nível de conhecimento prévio dos estudantes e disponibilidade de horários e recursos materiais. Além disso, são listados os objetivos propostos pela disciplina, ou seja, quais conceitos são abordados e desenvolvidos, bem como a profundidade de tais abordagens.

Nos *resultados*, mostra-se o material produzido juntamente com este trabalho, incluindo resumos dos conceitos apresentados em cada aula, ementas e material de apoio.

As *conclusões* apresentam um resumo das lições aprendidas na produção

deste trabalho, além de considerações finais e apontamentos para trabalhos futuros.

Finalmente, seguem as *referências* e, nos *anexos*, o material produzido em conjunto com este trabalho.

2 Conceitos iniciais

Esta seção introduz os conceitos a partir dos quais o projeto de disciplina se estrutura. Esses conceitos abrangem três âmbitos do trabalho: a motivação para a necessidade de uma disciplina na engenharia mecatrônica voltada para Linux; os conceitos apresentados na disciplina em si; e a produção de material de apoio.

Primeiramente, apresentam-se Unix e Linux, com uma retomada histórica dos dois sistemas, a conjuntura atual e sua relação com a disciplina. Também constam trabalhos anteriores de cursos introdutórios a Unix e Linux. A seguir, introduz-se o Robot Operating System (ROS), que, apesar de não constar no conteúdo da disciplina, forma parte da motivação para que seja ministrada. Segue uma explicação detalhada sobre o Raspberry Pi, incluindo um breve histórico, exposição de seu hardware e recursos oferecidos; uma introdução à linguagem de programação Python, seguida de uma mais breve sobre a linguagem Lua; e menção ao LaTeX, software utilizado na produção deste trabalho e do material de apoio, juntamente à biblioteca Beamer.

2.1 UNIX

A história do Unix tem raízes em um projeto conjunto entre MIT, AT&T Bell Labs e GE na década de 1960 chamado Multics. Bell Labs se retirou do projeto, mas alguns dos pesquisadores que fizeram parte dele, incluindo Ken Thompson e Dennis Ritchie, passaram a trabalhar em um sistema operacional similar, que veio a ser chamado Unix.

Unix foi desenvolvido principalmente por Dennis Ritchie e Ken Thompson, escrito, inicialmente, em Assembly (especificamente para o computador PDP-11/20, no qual foi desenvolvido). Começou com um processador e um editor de texto.

Em 1972, Unix foi reescrito na linguagem de programação C (também

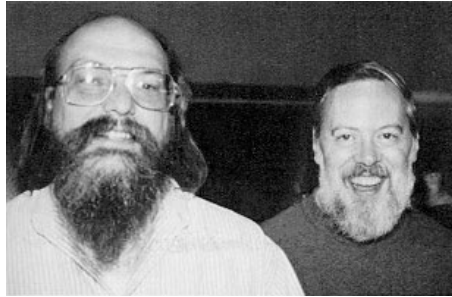


Figura 1: Ken Thompson e Dennis Ritchie

```

Terminal
-rwxr-xr-x 1 sys      52850 Jun  8 1979 hptmunix
drwxrwxr-x 2 bin      320 Sep 22 05:33 lib
drwxrwxr-x 2 root     96 Sep 22 05:46 mdec
-rwxr-xr-x 1 root    50990 Jun  8 1979 rkunix
-rwxr-xr-x 1 root    51982 Jun  8 1979 r12unix
-rwxr-xr-x 1 sys    51790 Jun  8 1979 rphtunix
-rwxr-xr-x 1 sys    51274 Jun  8 1979 rptmunix
drwxrwxr-x 2 root     48 Sep 22 05:50 tmp
drwxrwxr-x12 root    192 Sep 22 05:48 usr
# ls -l /usr
total 11
drwxrwxr-x 3 bin      128 Sep 22 05:45 dict
drwxrwxr-x 2 dmr      32 Sep 22 05:48 dmr
drwxrwxr-x 5 bin     416 Sep 22 05:46 games
drwxrwxr-x 3 sys     496 Sep 22 05:42 include
drwxrwxr-x10 bin     528 Sep 22 05:43 lib
drwxrwxr-x11 bin     176 Sep 22 05:45 man
drwxrwxr-x 3 bin     208 Sep 22 05:46 mdec
drwxrwxr-x 2 bin      80 Sep 22 05:46 pub
drwxrwxr-x 6 root     96 Sep 22 05:45 spool
drwxrwxr-x13 root    208 Sep 22 05:42 src
# ls -l /usr/dmr
total 0
#

```

Figura 2: Terminal UNIX

desenvolvida por Ritchie), acrescentando portabilidade ao sistema, que não mais estaria restrito ao PDP-11/20. No fim da década de 1970 e início de 1980, Unix se difundiu rapidamente. Deste ponto em diante, muitos projetos baseados em Unix se ramificaram, e têm outros nomes (por exemplo, Linux).

Apesar de a história do software livre ter muitos pontos de contato com a do Unix, este era de código fechado. Hoje, a marca Unix é de posse de “The Open Group”, que oferece licenças para uso do nome, sujeitas a certas restrições. Essa é a razão porque há numerosos projetos que, apesar de se assemelharem muito ao Unix, não levam o nome consigo.

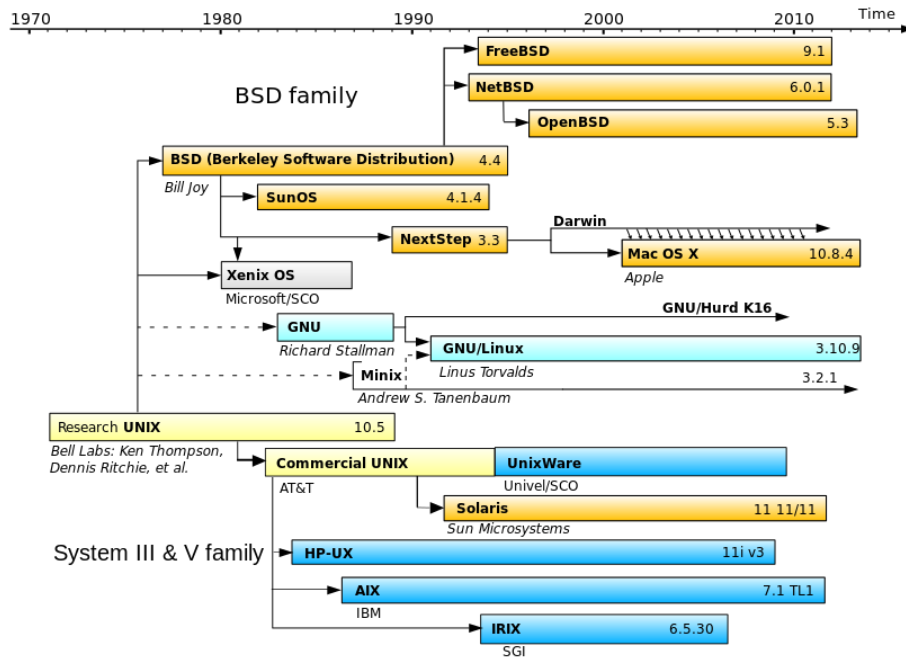


Figura 3: Linha do tempo de sistemas UNIX e UNIX-like

2.2 Linux

Em 1983, Richard Stallman iniciou um projeto (GNU Project) para criar um sistema operacional Unix-like (baseado em Unix, mas sem licenciamento) livre e de código aberto. Entretanto, um dos componentes mais importantes do sistema operacional, o kernel, não se popularizou. Assim, em 1991, Linus Torvalds, na Universidade de Helsinki, começou a desenvolver um novo kernel, que viria a ser chamado Linux. Em agosto de 1991, publicou no Usenet newsgroup "comp.os.minix.":

Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't
be big and professional like gnu) [...].
I'd like any feedback on things people like/dislike in minix,
as my OS resembles it somewhat [...].
[...]
PS. Yes – it's free of any minix code, [...] and it probably
never will support anything other than AT-harddisks, as
that's all I have :-(.
—Linus Torvalds

Após sua criação na década de 1990, o sistema operacional se difundiu rapidamente, e atualmente se posiciona entre os projetos de software mais importantes e bem sucedidos do mundo. Apesar de ter adoção comparativamente muito baixa em comparação com outros sistemas operacionais para desktop (como Windows), domina outras áreas. Linux é utilizado no sistema operacional Android, e, portanto, está presente em boa parte dos dispositivos móveis no mundo.

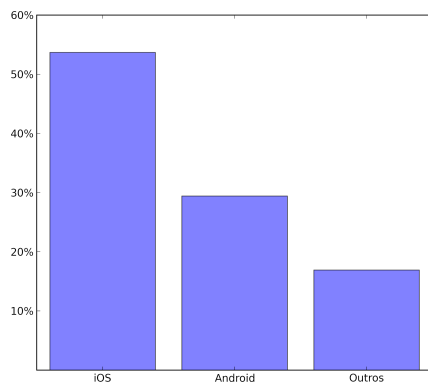


Figura 4: Market share de dispositivos móveis em 2013 [1]

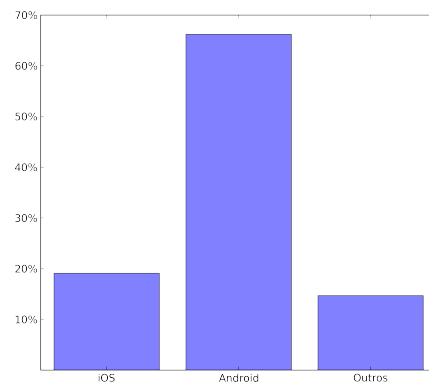


Figura 5: Vendas de smartphones em 2013 [1]

Além disso, é frequentemente utilizado em outros nichos, como servidores e supercomputadores.

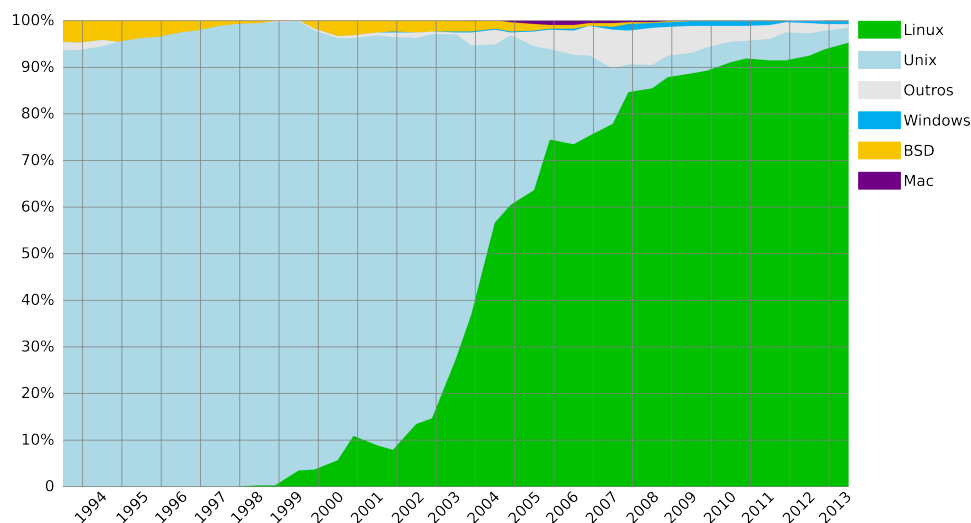


Figura 6: Ranking dos 500 supercomputadores com maior poder de processamento do mundo [2]

2.3 Cursos anteriores

A disciplina aqui desenvolvida toma como base uma série de cursos introdutórios a Unix e Linux, a fim de capturar os elementos mais importantes a serem abordados. Em sua maioria, são cursos universitários de instituições americanas, mas há referências de outros países. São elas:

- CS197U: A Hands-on Introduction to UNIX (University of Massachusetts) [3]
- Introduction to UNIX (Imperial College London) [4]
- Introduction to UNIX (Ohio State University) [5]
- Practical Unix (Stanford OpenClassroom) [6]
- Unix Tools (New York University) [7]

2.4 Software livre

A história do software livre se inicia com o GNU Project, em 1983. Após seu início, Richard Stallman funda a Free Software Foundation (FSF), em

1986, e, em 1989, é lançada a primeira versão da GPL (GNU Public License), a primeira licença para software de código aberto. Hoje em dia, existem várias organizações ligadas ao movimento do software livre, sendo as de maior destaque, além da FSF, a Electronic Frontier Foundation (EFF) e Mozilla Foundation.



Figura 7: Richard Stallman



Figura 8: Logo do projeto GNU

Atualmente, existe uma variedade de licenças e uma multitude de software com código aberto. É importante salientar a distinção entre software livre, onde a questão principal é a liberdade de uso, distribuição e modificação, o que demanda a abertura do código-fonte, e o software de código aberto (open source), cuja única exigência é a abertura do código-fonte. O primeiro é uma questão política, enquanto o segundo vem crescendo aceleradamente como modelo de desenvolvimento de software – tanto para projetos livres quanto comerciais.

Existem, hoje em dia, projetos não só de software livre, mas também hardware livre. Dispositivos de hardware livre têm todas as especificações de sua fabricação publicadas, incluindo, nos eletrônicos, componentes e esquemáticos. O primeiro grande projeto criado neste modelo foi o Arduino.

2.5 Robot Operating System e Gazebo

Robot Operating System (ROS) [8], é um projeto gerenciado pela Open Source Robotics Foundation (OSRF). De acordo com o site oficial, em tradução livre:



Figura 9: Logo do framework ROS com OSRF



Figura 10: Logo do software de simulação Gazebo

um framework flexível para escrever software para robôs. É uma coleção de ferramentas, bibliotecas e convenções com o objetivo de simplificar o processo de criar comportamento robótico complexo e robusto em uma variedade de plataformas robóticas.

ROS contém pacotes de software para auxiliar a criação de software para robôs, provendo operações comuns em suas bibliotecas e uma estrutura modularizada para o acoplamento e desacoplamento de diferentes partes do robô (como sensores), e, assim, gerando robustez no sistema. Todo o código dos pacotes é aberto.

A criação do ROS tem raízes em Stanford, em 2007, quando componentes comuns aos projetos de robótica sendo desenvolvidos lá passaram a ser feitos separadamente, e, com isso, estruturava-se um framework para projetos robóticos. A primeira versão foi lançada em janeiro de 2009, e, desde então, sua adoção vem crescendo rapidamente.

Desenvolvimento e suporte para ROS é oferecido apenas para Linux. O mesmo vale para Gazebo, software de simulação robótica também gerenciado pela OSRF.

Revela-se, portanto, a familiaridade com Linux um pré-requisito para a utilização de ferramentas modernas em robótica.

2.6 Raspberry Pi

Raspberry Pi [9] é um computador aproximadamente do tamanho de um cartão de crédito, com hardware livre, desenvolvido pela Raspberry Pi Foun-

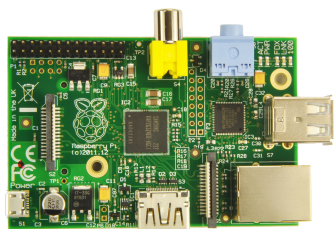


Figura 11: Vista superior do Raspberry Pi

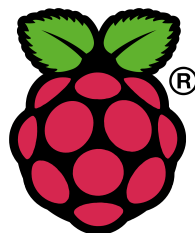


Figura 12: Logo do Raspberry Pi

dation, no Reino Unido, com o intuito de promover o ensino de computação em escolas.

Ao contrário de dispositivos similares, como Arduino, o Raspberry Pi oferece suporte para instalação e uso de um sistema operacional completo, com todos os componentes essenciais de um PC ou laptop de tamanho regular. A Raspberry Pi Foundation recomenda, oficialmente, a instalação de uma versão de Debian Linux otimizada para uso no dispositivo, mas existem outras versões de Linux disponíveis. O computador possui suporte para saída de vídeo em alta qualidade, portas USB, Ethernet e pinos de comunicação digital de uso geral (GPIO). Utiliza um processador de arquitetura ARM, e os modelos B e B+ (que possuem saída Ethernet) custam US\$ 35. O projeto causou repercussão maior que o esperado, extrapolando seu objetivo inicial e criando uma grande e ativa comunidade de entusiastas em seu entorno.

O presente trabalho propõe o uso do Raspberry Pi em alinhamento com seu propósito inicial: como uma ferramenta educacional.

2.7 Python

Python é uma linguagem de programação de alto nível, interpretada dinamicamente, com foco em legibilidade do código. Um dos aspectos mais marcantes visualmente é a ausência de chaves e redução de parênteses para a separação de blocos de código. Em Python, a separação é determinada pela indentação do bloco, promovendo melhor legibilidade.

A linguagem oferece recursos para numerosos paradigmas de programação, incluindo imperativo, orientado a objeto e funcional. Sua implementa-



Figura 13: Guido van Rossum



Figura 14: Logo da linguagem de programação Python

ção foi iniciada em 1989 por Guido van Rossum, então no centro de pesquisa CWI, na Holanda. Python 2.0 foi lançado em outubro de 2000, e Python 3.0, incompatível com as versões anteriores, em dezembro de 2008.

Python é largamente utilizada em projetos de pequena a grande escala. Vem substituindo Java como a linguagem de alto nível preferencial para aplicações Web, e sua adoção como a primeira linguagem apresentada a estudantes de graduação em muitas universidades ao redor do mundo (notavelmente, MIT). A principal motivação para a escolha de Python neste projeto de disciplina é sua escolha como linguagem preferencial no manuseio do Raspberry Pi, pelos próprios criadores do dispositivo. É usada, por exemplo, para controlar os pinos para comunicação externa (GPIO) do Raspberry Pi.

Entretanto, é possível encontrar muitos projetos de renome que utilizam Python no seu desenvolvimento, tais como ROS (que oferece interfaces em C++ e Python), Dropbox, Blender, BitTorrent, OpenStack, reddit e Django.

2.8 Lua



Figura 15: Logo da linguagem de programação Lua

Lua [10] é uma linguagem de programação de scripting semelhante, em muitos aspectos, a Python: tem tipagem dinâmica, incorpora ou possibilita a incorporação de diversos paradigmas de programação (imperativa, funcional, orientação a objetos). Foi desenvolvida por uma equipe da PUC-Rio, para utilização em um projeto da Petrobras. É de código aberto, distribuída sob a licença MIT. É mais leve e menor que Python, tornando-a mais adequada para utilização em sistemas embarcados. A linguagem teve grande adoção em jogos de renome, como World of Warcraft, Angry Birds e Far Cry, em parte por conta de ter se originado como linguagem para configuração. Apesar de a disciplina não incluir o uso da linguagem Lua em nenhuma de suas atividades propostas, é importante situá-la como alternativa ao uso de Python no Raspberry Pi por algumas razões:

1. Lua é uma linguagem mais leve, menor e mais rápida que Python, o que a situa em uma posição mais adequada para sistemas embarcados, como o Raspberry Pi;
2. Existe suporte para controle dos pinos de GPIO pela linguagem, por meio de um módulo de código aberto;
3. Lua é tecnologia nacional e largamente utilizada, o que torna-a mais interessante para a formação de engenheiros brasileiros, tanto por valorizar e promover seu uso quanto pela maior probabilidade de encontrá-la em um contexto profissional no país.

Sendo assim, apresenta-se Lua como uma alternativa ao uso de Python no Raspberry Pi, tópico abordado no conteúdo adicional.

2.9 LaTeX

LaTeX é um sistema de preparação e linguagem de markup de documentos. LaTeX é desenvolvido tendo a linguagem de formatação de texto e tipografia TeX como base, escrito na própria linguagem de macros TeX. É, essencialmente, um processador de texto, cuja proposta é a separação de conteúdo e forma. Assim, o conteúdo e sua separação lógica (em capítulos,

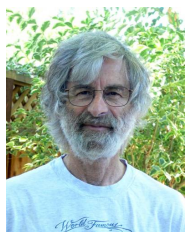


Figura 16: Leslie Lamport



Figura 17: Logo do processador de texto LaTeX

seções, subseções) é informado pelo autor, e LaTeX realiza toda a formatação e diagramação com base em seus algoritmos. Diferencia-se de processadores de texto mais conhecidos como Word e LibreOffice Writer justamente nessa característica: enquanto os dois permitem edição de conteúdo e layout livremente, LaTeX permite ao autor informar somente conteúdo e sua separação lógica, sendo o layout determinado pelo LaTeX.

LaTeX começou a ser desenvolvido em 1980, por Leslie Lamport. É software livre e de código aberto, e sua versão mais recente é LaTeX2e.

LaTeX é frequentemente utilizado na formulação de artigos científicos, principalmente na área de matemática, por fornecer numerosas ferramentas para a edição, apresentação e controle de fórmulas. Por essa mesma razão, também é muito utilizado para a apresentação de fórmulas em páginas na internet, notavelmente na Wikipédia.

2.9.1 Beamer

Beamer é uma classe de documentos para LaTeX direcionada à criação de apresentações de slides. Por oferecer suporte tanto a ferramentas de LaTeX quanto a outras, específicas para slideshows, foi escolhido como instrumento principal para a criação de material de referência para a disciplina.

3 Motivação

Nesta seção, argumenta-se a favor da necessidade de uma disciplina nos moldes da que é aqui proposta, incluindo uma avaliação dos conteúdos abordados no curso de engenharia mecatrônica e as necessidades para que permaneça atualizado e coerente com as habilidades exigidas de um profissional com essa formação. A motivação para a formulação deste projeto de disciplina surgiu de:

1. a crescente necessidade de conhecimento e manuseio de plataformas baseadas em Linux para profissionais nas áreas de ciência e engenharia;
2. uma análise do curso de engenharia mecatrônica, reveladora da necessidade de abordagem de conteúdos relativos à área de computação;
3. uma conjuntura onde é possível obter recursos materiais de alto nível a custo baixo, possibilitando maior aproximação dos conteúdos aprendidos em sala com os desafios que um estudante encontrará tanto em projetos pessoais quanto profissionais.

Projetos open source em geral, mas Linux em específico, obtêm adoção cada vez maior, tanto no contexto de desktops, laptops e dispositivos móveis, quanto em contextos profissionais. Como mostrado anteriormente, mais de 96% dos supercomputadores mais poderosos do mundo usam Linux; ROS e Gazebo, ferramentas difundidas em projetos de robótica, são desenvolvidos especificamente para Linux; e Android representou 79% das vendas de smartphones em 2013. Além disso, 67% dos servidores na internet são Unix ou Unix-like; muitas organizações públicas e empresas de TI de renome adotaram Linux internamente, tais como: Google, SESC, Banco do Brasil, Amazon, IBM, CERN, departamento de agricultura da França e a prefeitura de Munique. Essas são organizações que adotaram Linux para uso interno por seus empregados, sem incluir outras aplicações (como servidores). Um dos maiores motivos para tal adoção é a redução de custo (motivadora da

adoção pelas organizações públicas e grande parte das empresas), mas nota-se, também, que muitas delas adotam o sistema operacional para utilizá-la como plataforma de desenvolvimento profissional de software.

Sendo assim, torna-se evidente que a formação de um profissional cujo trabalho possa envolver computação, robótica ou mesmo a atuação em uma organização governamental, deve incluir o ensino de Linux e as ferramentas a ele associada – ainda que em nível básico. A necessidade torna-se ainda mais latente para estudantes de engenharia mecatrônica, para os quais computação e robótica são partes integrais da formação acadêmica.

Defende-se, portanto, a introdução de uma disciplina que, ao auxiliar no ensino de conteúdo importante para engenheiros mecatrônicos, promova uma formação mais completa. A seguir, argumenta-se em favor de tal disciplina não só pelo conteúdo específico que introduz, mas também porque contribui pela formação em uma área defasada em relação às demais.

A engenharia mecatrônica é comumente dividida em três áreas majoritárias: mecânica, eletrônica e computação. No curso de graduação da EESC-USP, existe uma ênfase maior na área de mecânica, e, dentre as três, a menor é em computação. A fim de ilustração, apresenta-se, na Tabela 1, uma divisão das disciplinas do curso de engenharia mecatrônica em 5 seções: ciclo básico, mecânica, elétrica/eletrônica, computação, mecatrônica, e outros. No *ciclo básico*, colocam-se as disciplinas básicas da formação em engenharia, como as de matemática e física; *mecânica*, *elétrica/eletrônica* e *computação* correspondem às três áreas majoritárias supracitadas; em *interdisciplinares*, incluem-se disciplinas que abordam duas ou mais dentre as três áreas (como as de projeto); em *outros*, incluem-se as que não se enquadram em nenhuma das outras seções. A Tabela 1 apresenta a divisão entre as áreas em número de disciplinas e número de créditos, juntamente com o respectivo percentual. Para os cálculos, considera-se a soma de créditos aula e trabalho. A divisão detalhada das disciplinas em seções encontra-se na Subseção 8.1.

Deve-se ressaltar, também, que esta é uma divisão razoavelmente conservadora. Muitas das disciplinas que têm elementos de computação, porém onde esta não é o foco principal, foram colocadas na seção “interdisciplinares” (como as de sistemas de controle). No entanto, tais disciplinas são menos nu-

	Número de disciplinas	Distribuição de disciplinas (% total)	Número de créditos	Distribuição de créditos (% total)
Ciclo básico	14	20,9	59	23,3
Mecânica	17	25,3	68	26,8
Elétrica/Eletrônica	5	7,5	18	7,1
Computação	5	7,5	12	4,7
Interdisciplinares	18	26,9	66	26
Outros	8	11,9	31	12,2

Tabela 1: Distribuição de carga horária em disciplinas do curso de engenharia mecatrônica

merasas que aquelas, também alocadas na seção interdisciplinares, com um viés mais direcionado à mecânica e/ou eletrônica (como as de automação e robótica).

Da Tabela 1, é possível tirar algumas conclusões. Primeiramente, existe uma distribuição razoavelmente uniforme entre as disciplinas do ciclo básico, de mecânica e interdisciplinares, tanto em número de disciplinas quanto em número de créditos. Entretanto, as áreas de elétrica/eletrônica e computação compõem, comparativamente, uma parcela muito menor do total de disciplinas, com cada uma correspondendo a 7,5% do total. Além disso, a diferença torna-se ainda maior quando analisados em termos de números de créditos: nesse caso, elétrica/eletrônica correspondem a 7,5%, e computação, a apenas 4,7%. Sendo o número de créditos uma aproximação mais apropriada para o tempo dedicado pelos estudantes do que o número de disciplinas, evidencia-se uma necessidade de reavaliar o ensino de computação.

Sendo assim, conclui-se que a área com maior defasagem em relação às demais é a de computação. A disciplina proposta neste trabalho contribuiria, portanto, para um equilíbrio entre as áreas abrangidas pelo curso de engenharia mecatrônica.

Por último, a possibilidade de se estruturar uma disciplina que introduz os estudantes a um dispositivo que, ao mesmo tempo, tem custo muito baixo, e está próximo do que se encontra em aplicações profissionais e comerciais, era praticamente inexistente antes do lançamento do Raspberry Pi.

Apresentam-se, a seguir, estimativas de custo para cada dispositivo acompanhado dos acessórios necessários para seu funcionamento, assim como alguns projetos comerciais realizados com o Raspberry Pi.

Desde seu lançamento, o preço de venda do Raspberry Pi (modelo B ou B+, com saída Ethernet) é \$35. Além do dispositivo em si, é necessário um cartão de memória SD (ou microSD, para o modelo B+) de no mínimo 2GB (que utiliza em lugar de disco rígido) e uma fonte de alimentação de 5V capaz de suprir 1A. Realizou-se o orçamento de todos os componentes em quatro diferentes fornecedores: Farnell-Newark (filial da Farnell no Brasil), Farnell, Adafruit e MCM Electronics.

	210183	RASPBERRY-PI - FERRAMENTA DESENV SRC MODEL B 512MB			Local	Intern*	UN	R\$ 189,53
		MODB-512M			1359	0		UN
	23W0457	RPI-PSU-US-MK1			Local	Intern*	UN	R\$ 31,13
		RASPBERRY-PI			9	4885		UN
	00W6772	00W6772			Local	Intern*	UN	R\$ 55,05
		00W6772			0	0		UN

Figura 18: Orçamento na empresa Farnell-Newark

Manufacturer Part Number	Manufacturer / Description	Avail.	Unit Price	Line Price
RASPBERRY-PI/8GB-USD	RASPBERRY-PI MODEL B - 8GB SDCard W/ NOOBS PRE-INSTALLED RASPBERRY-PI-RASPBERRY-PI/8GB-USD-MODEL B - 8GB SDCard W/ NOOBS PRE-INSTALLED	Awaiting Delivery	\$40.00	\$40.00
RPI-PSU-US-MK1	RASPBERRY-PI AC-DC CONV, EXTERNAL PLUG IN, MICRO USB, 1 O/P, 5W, 5V, 1A RASPBERRY-PI-RPI-PSU-US-MK1-AC-DC CONV, EXTERNAL PLUG IN, MICRO USB, 1 O/P, 5W, 5V, 1A	4,910	\$7.95	\$7.95

Figura 19: Orçamento na empresa Farnell

Product Detail	Availability	Price	Total
 83-14421 - 512MB Raspberry Pi Model B Project Board Raspberry Pi : RASPBERRY-MODB-512M	 In Stock	\$35.00	\$35.00
 28-17985 - 5V 2.2A Micro USB Switch Mode Power Adapter Distributed By MCM : MWS158-0502200UC/MIC	 On Backorder (Expected in 27 days)	\$7.75	\$7.75
 83-11628 - 4GB PNY Class 4 Micro SDHC Memory Card with Adapter PNY Technologies : P-SDU4G4-GE	 In Stock	\$8.91	\$8.91

Figura 20: Orçamento na empresa MCM

ITEM	PRICE	QTY	REMOVE	TOTAL
 Raspberry Pi Model B 512MB RAM PID: 998	\$39.95	1		\$39.95
 5V 1A (1000mA) USB port power supply - UL Listed PID: 501	\$5.95	1		\$5.95
 USB cable - A/MicroB - 3ft PID: 592	\$3.95	1		\$3.95
 SD/MicroSD Memory Card (4 GB SDHC) PID: 102	\$7.95	1		\$7.95

Figura 21: Orçamento na empresa Adafruit

Dentre as quatro, apenas a Farnell-Newark fornece os valores em reais. Para os demais, utilizou-se a taxa de conversão conservadora de 3 reais por dólar. Além disso, nenhum deles inclui frete. Os resultados estão resumidos na Tabela 2.

Conclui-se, assim, que, mesmo tomando o maior dos preços encontrados, é possível comprar um computador totalmente funcional por menos de R\$260,00. Este valor não inclui periféricos, como teclados e monitores, mas é possível utilizar aqueles já presentes no laboratório de ensino do prédio da engenharia mecatrônica.

Fornecedor	Preço (US\$)	Preço (R\$)
Farnell-Newark	-	256,31
Farnell	47,95	143,85
MCM Electronics	51,66	173,40
Adafruit	57,80	154,98

Tabela 2: Resumo dos orçamentos para compra de Raspberry Pi e acessórios

Poucos anos atrás, seria impensável obter equipamento do tipo por um custo tão baixo. Desde o lançamento do Raspberry Pi em 2012, tornou-se possível considerar experiências pedagógicas com o auxílio do dispositivo. Com menos de R\$260,00, é possível unir o ensino de Linux com o de Python e sistemas embarcados, incluindo atividades práticas (apesar da possibilidade, o cenário certamente não é ideal. Para a efetiva realização da proposta, seriam necessários mais de um Raspberry Pi).

Além do potencial pedagógico oferecido pelo dispositivo, existem sérias possibilidades comerciais e profissionais tomando-o como base. Seu uso como controlador de sistemas embarcados não se restringe a projetos simples, e escala para além dos requisitos de entusiastas de eletrônica. Para fins de ilustração, apresentam-se, na Figura 22, alguns projetos comerciais baseados no Raspberry Pi:

- Argus [11], um scanner 3D geodésico;
- RainCloud [12], um sistema de irrigação e controle de água;
- Sphinx [13], um dock para tablets para o acesso remoto de desktops;
- YaCyPi [14], um dispositivo de busca online;
- PiCube [15], um serviço de painéis informativos digitais;
- PiWall [16], um pacote de software para o arranjo de vídeo em múltiplos monitores;
- OTTO [17], uma câmera para captura de imagens diretamente para o formato GIF.

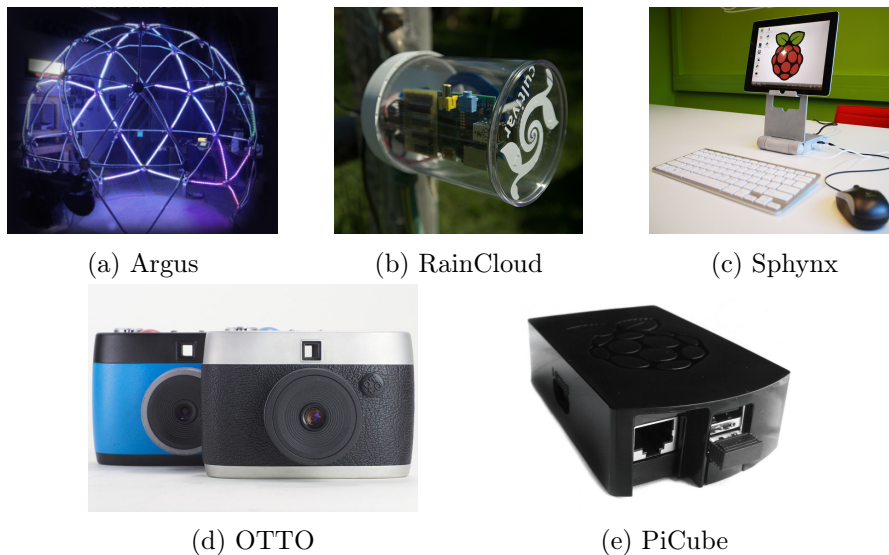


Figura 22: Projetos comerciais com Raspberry Pi

Além disso, a Raspberry Pi Foundation anunciou, em abril de 2014, o Raspberry Pi Compute Module [18]. Este é uma versão compacta do Raspberry Pi direcionada para aplicações comerciais e industriais. Tem formato e dimensões semelhantes a uma placa de memória RAM, com cerca de 7cm por 4cm e pinos para encaixe em um conector DDR2 SODIMM. O lançamento é direcionado especificamente para sistemas embarcados comerciais, acessível somente a projetos com circuitos impressos customizados, capazes de interfacear com uma placa altamente miniaturizada. As vendas serão realizadas principalmente em lotes de 100. Sendo assim, é evidente a demanda e o interesse pela utilização do Raspberry Pi em projetos profissionais. OTTO, um dos projetos acima citados, utiliza o Compute Module.

Sendo assim, conclui-se que um projeto de disciplina com o objetivo de ensinar Linux e Python, utilizando como plataforma o Raspberry Pi, é capaz de atender aos três requisitos expostos no início desta seção: introduz o sistema operacional Linux aos estudantes, ferramenta de crescente importância em ciência e engenharia; atende à demanda de um ensino mais completo e aprofundado na área de computação; e é capaz de fazê-lo a custo baixo e

e oferecer situações próximas ao que possivelmente se encontraria em um ambiente profissional.

4 Justificativas

Esta seção se destina a explorar possíveis alternativas à plataforma utilizada e formato da disciplina, juntamente com as justificativas que levaram às decisões pela estrutura escolhida. As justificativas em relação ao conteúdo abordado foram exploradas na seção anterior.

4.1 Plataforma

As principais razões para a escolha do Raspberry Pi como plataforma para o projeto de disciplina são:

- Utilização de Linux como sistema operacional;
- Disponibilização de linha de comando como principal interface de comunicação entre usuário e plataforma;
- Baixo custo;
- Suporte a uma linguagem de programação de alto nível;
- Facilidade de operação;
- Possibilidade de utilização como sistema embarcado.

Sendo assim, é importante buscar alternativas que cumpram tais requisitos, possivelmente ainda melhor que o Raspberry Pi. Apresentam-se brevemente a seguir algumas plataformas similares ao Raspberry Pi, que poderiam ser usadas em seu lugar.

4.1.1 BeagleBone Black

BeagleBone Black [19] é um computador de hardware livre produzido pela Texas Instruments em conjunto com DigiKey. A última versão produzida, recentemente colocada à venda, BeagleBone Black, utiliza processador de



Figura 23: Vista superior do BeagleBone Black

arquitetura ARM, tem armazenamento incluso de 4GB, possui saída USB e 512MB de memória RAM e custa US\$45.

4.1.2 Banana Pi



Figura 24: Vista isométrica do Banana Pi

Banana Pi [20] é um computador com hardware livre produzido pela chinesa LeMaker. Utiliza microprocessador de arquitetura ARM, custa US\$45, possui saídas USB, HDMI, Gigabit Ethernet e mede 92mm por 60mm. Também suporta conexão SATA para armazenamento de até 2TB. Foi proposto como uma evolução do Raspberry Pi (porém sem relação com a Raspberry Pi Foundation), mantendo tamanho e forma semelhantes, porém com melhorias no hardware.

4.1.3 Odroid-U2

Odroid-U2 [21] é um computador com hardware livre produzido pela sul-coreana Hardkernel Co. Utiliza microprocessador de arquitetura ARM, custa US\$ 89, possui saídas USB, HDMI e Ethernet. Mede 48mm por 52mm. Foi



Figura 25: Vista isométrica do Odroid-U2

proposto principalmente para uso por desenvolvedores para Android, para prototipagem e testes.

A Tabela 3 resume o hardware oferecido por cada uma das plataformas consideradas.

	CPU	Memória RAM	GPU	Preço
Raspberry Pi (Modelos B e B+)	700 MHz ARM1176JZF-S	512 MB	Broadcom VideoCore IV	US\$ 35
BeagleBone Black	1 GHz ARM Sitara AM 335x	256 MB	(Acelerador gráfico incluso na CPU)	US\$ 45
BananaPi	1GHz Dual core ARM Cortex-A7	1 GB	Mali400MP2	US\$ 45
Odroid-U2	Exynos4412 Prime Cortex A9 Quad Core 1.7GHz	2 GB	Mali-400	US\$ 89

Tabela 3: Comparação de hardware de alternativas para plataforma de desenvolvimento

Todos os dispositivos pesquisados satisfazem, no mínimo, os requisitos 1, 2, 4 e 6. O único deles que destoa significativamente dos demais em relação ao custo é o Odroid-U2, praticamente duas vezes mais caro que os demais. Sendo assim, e tendo em vista que o objetivo da disciplina não é prototipagem de Android e não requer poder de processamento muito elevado, descarta-se a opção como plataforma de desenvolvimento para as atividades propostas neste trabalho.

É importante notar que, apesar de o Raspberry Pi ter custo 23% menor

que os demais, o preço base não inclui cartão SD (ou microSD) para armazenamento do sistema operacional. Dentre os orçamentos realizados na seção 3, aquele que inclui cartão (micro)SD com menor preço totaliza US\$40. Assim, a diferença de custo passa a ser de menos de 12%, menos significativo que anteriormente.

Sendo assim, a fim de decidir entre Raspberry Pi, Banana Pi e Beagle-Bone Black, o critério de análise restante é a facilidade de uso. É neste quesito que o Raspberry Pi se destaca: foi o primeiro computador do tipo lançado no mercado, desde a concepção orientado para aplicações em educação, além de receber o suporte de uma fundação dedicada exclusivamente a sua manutenção, desenvolvimento e difusão, a plataforma recebeu atenção muito maior que as demais. Assim, uma grande e vibrante comunidade se desenvolveu em seu entorno. Tutoriais de projetos, atualizações frequentes e soluções de problemas são de fácil acesso na Internet, enquanto os demais, apesar do maior custo-benefício em termos de hardware, tornam-se menos adequados para uma disciplina que não está centrada na plataforma em si ou em desempenho de hardware.

4.2 Disciplina

Em relação à disciplina proposta, podem-se levantar uma série de questões importantes, tais como:

1. Por que apresentar o conteúdo na forma de uma disciplina de graduação?
2. Quais são as alternativas ao formato?
3. Decidido o formato de disciplina de graduação, que tipo de disciplina seria necessária? Qual o número de créditos? Em que contexto seria ela formalmente oficializada?
4. Quanto tempo seria necessário para pô-la em prática?

A primeira questão foi explorada em detalhes na Seção 3. A segunda se relaciona à primeira, porém de forma comparativa. É pertinente explorar

diferentes formas de apresentação do conteúdo que podem, potencialmente, promover um aprendizado mais significativo, ou se adequar melhor ao conteúdo proposto. Uma alternativa é apresentar no modelo de um “Aprendendo à Semear”, evento promovido pelo SEMEAR, grupo de robótica da engenharia mecatrônica da EESC-USP. Este seria um encontro de poucas horas, à noite ou em um fim de semana, com duração total máxima de 8 horas. Outra alternativa é disponibilizar todo o material digitalmente, para que cada estudante possa explorá-lo a seu ritmo.

Das duas, a primeira alternativa é a mais viável. É possível introduzir conceitos básicos de Linux e operação do Raspberry Pi no intervalo de 8h, além de o acesso ao hardware ser facilitado em um grupo grande de pessoas. Entretanto, há dois importantes problemas. Primeiramente, a operação de Linux é requisito fundamental na formação de um engenheiro atualmente. Uma atividade extra, oferecida à noite ou em um fim de semana, não teria alcance suficiente, principalmente em vista das exigências de tempo das atividades da graduação. Além disso, muitos conceitos de interesse não seriam abordados por restrições de tempo.

A segunda alternativa é mais apropriada para promover um aprendizado significativo e duradouro, porém os mesmos problemas referentes à primeira aplicam-se, também, a esta. Somente as restrições de tempo (limitado a 8h no anterior) não seriam problemáticas. Contudo, o acesso à plataforma seria dificultado, e o alcance, também, não seria grande o suficiente. Decidiu-se, portanto, apresentar o conteúdo na forma de disciplina de graduação. Tanto a restrição de tempo quanto o problema de alcance seriam superados, sendo que uma disciplina de um crédito aula fornece, na média, 15 de trabalho em sala, e a obrigatoriedade de seu cumprimento maximizaria o alcance em alunos de engenharia mecatrônica.

Resta, assim, discutir o tipo de disciplina. A conjuntura no curso de engenharia mecatrônica da EESC-USP é apropriada para tal, pois existe uma sequência de disciplinas obrigatórias (*Introdução à Engenharia Mecatrônica* e *Problemas de Engenharia Mecatrônica*) voltada à apresentação de conceitos atuais, cuja abordagem a rigidez do currículo acadêmico torna difícil. As disciplinas, de um crédito cada, provêm 20h para o trabalho de ferramen-

tas e atividades pertinentes à formação de engenheiros mecatrônicos que não são cobertas pelo currículo tradicional (por exemplo, utilização de MATLAB e LabView). Outra vantagem crucial, que diz respeito à terceira e quarta questões listadas no início desta seção, é a flexibilidade promovida por tais disciplinas: não é necessário oficializar o oferecimento de uma ementa específica e aguardar a aprovação pelos órgãos universitários responsáveis. A concretização do projeto de disciplina aqui apresentado poderia, em tese, ocorrer em menos de um semestre de sua finalização.

5 Hipóteses e objetivos

Esta seção se destina a esclarecer os pressupostos nos quais baseia-se o desenvolvimento da disciplina, bem como os objetivos a serem atingidos por um estudante até sua finalização. Pressupõe-se uma disponibilidade de tempo e de recursos materiais, detalhados nas subseções a seguir, para possibilitar a abordagem de uma lista de conteúdos explorada nas últimas subseções.

5.1 Disponibilidade de tempo

É necessário, para a estruturação de uma disciplina de graduação com uma sequência de conteúdos a cobrir, uma disponibilidade mínima de tempo de professor e estudantes em sala de aula (além, possivelmente, de tempo fora de sala para trabalhos e projetos). Esta subseção se destina a esclarecer os requisitos de tempo demandados por este projeto de disciplina, bem como o formato ao qual ele melhor se adaptaria.

Estimam-se necessários, para um desenvolvimento aprofundado dos conteúdos em questão, um total de 10h a 15h de trabalho em sala de aula. O projeto de disciplina não pressupõe horas adicionais para o desenvolvimento de trabalhos e projetos fora de sala de aula, apesar de considerá-las essenciais para o aprendizado efetivo. Em lugar disso, propõe-se um formato flexível o suficiente para a inclusão de um trabalho prático de escolha do professor em conjunto com os estudantes. O projeto aqui desenvolvido se restringe, portanto, a desenvolver, em sala de aula, o ferramental necessário para tal projeto.

Como justificado na Seção 4, o formato mais adequado seria o de uma disciplina de graduação de 1 crédito, uma dentre as de *Introdução à Engenharia Mecatrônica* ou *Problemas de Engenharia Mecatrônica*. Tais disciplinas fornecem uma média de 15h em sala de aula, além de 1 crédito trabalho (correspondente a 30h fora de sala). Portanto, o tempo fornecido neste formato

é largamente suficiente – possibilitando, talvez, a abordagem de conteúdos adicionais. Além disso, oferece tempo para o desenvolvimento de projetos e trabalhos (nos quais, contudo, o projeto não se aprofunda).

Assim, supõe-se disponíveis os recursos de tempo de uma disciplina obrigatória de um crédito do currículo de engenharia mecatrônica, correspondente a uma aula quinzenal de 1h40min, totalizando aproximadamente 15h. Tendo em vista a variação entre semestres, assume-se o tempo suficiente para um total de 10 a 20 aulas.

5.2 Recursos materiais

Além de tempo, são necessários, também, recursos físicos para o aprendizado em sala de aula. Tradicionalmente, os recursos básicos (além do espaço físico) são cadeiras, carteiras, quadro-negro, giz, lápis e/ou canetas, papel para anotações e recursos didáticos como livros. Com o desenvolvimento tecnológico, os recursos oferecidos (e, por vezes, demandados) se expandiram para computadores e projetores, entre outros. Esta subseção se destina a esclarecer os recursos físicos mínimos com os quais o conteúdo será desenvolvido.

Como disciplina de cunho profundamente tecnológico, os recursos necessários para um desenvolvimento desimpedido dos conteúdos excedem aqueles da sala de aula tradicional. Sendo assim, podem ser dispensados a maioria daqueles em lugar de:

1. um computador com Linux e um projetor para o professor, para a exibição de slideshows e execução de comandos;
2. um computador para cada um dos estudantes, todos com Linux ou com acesso remoto a um servidor Linux.

Hoje em dia, tais requisitos são amplamente satisfeitos pela maioria dos laboratórios didáticos de informática oferecidos a estudantes de ensino superior (como é o caso dos estudantes de engenharia mecatrônica da EESC-USP). Além dos recursos acima, necessitam-se Raspberry Pi

para que sejam feitas as atividades específicas para o dispositivo. Portanto, como hipótese simplificadora (que pode não se concretizar no oferecimento da disciplina), tem-se:

3. um Raspberry Pi funcional, com todos os acessórios necessários para uso, para cada estudante.

Entende-se, contudo, que existem dificuldades para a realização de tal. Esta é uma hipótese assumida para concentrar o andamento da disciplina no conteúdo, sem restringi-lo desnecessariamente por uma potencial falta de recursos materiais. Existem diversas soluções que podem aproximar a situação em sala de aula daquela tomada como ideal. Devido ao baixo custo do Raspberry Pi, pode-se ter uma quantidade razoavelmente alta para uma sala de aula; podem-se formar duplas ou grupos pequenos para usar cada dispositivo; pode-se, também, configurar cada Raspberry Pi com mais de uma conta de usuário, possibilitando o uso simultâneo por mais de uma pessoa.

Além disso, os Raspberry Pi podem servir como plataformas Linux em laboratórios de informática com outro sistema operacional. Entende-se que esta é uma dificuldade em diversos laboratórios de ensino dominados por computadores somente com o sistema operacional Windows. Dessa forma, o esforço para obter uma grande quantidade de Raspberry Pi pode servir a demandas múltiplas – possibilitando, inclusive, se instalados permanentemente, alternativas ao uso de Windows em laboratórios didáticos em outras atividades.

Sendo assim, pressupõe-se o acesso a recursos digitais em todas as aulas, incluindo acesso físico a um Raspberry Pi por estudante.

5.3 Habilidades desenvolvidas

Nesta subseção, apresentam-se os objetivos a serem alcançados por meio deste projeto de disciplina. Tais objetivos são listados na forma de uma lista de habilidades. Todo estudante deve, ao término da disciplina, ter desenvolvido algumas habilidades prioritárias. Além disso, devido ao amplo número de horas oferecido a uma disciplina de graduação tal como uma das que a

disciplina utilizaria, apresenta-se, em seguida, uma lista de habilidades adicionais. Tais habilidades não são essenciais, mas, de acordo com o andamento das aulas e de trabalhos e projetos, podem ser abordadas. Deve-se notar, também, que esta é uma disciplina introdutória; sendo assim, nenhuma das habilidades a seguir necessita ser aprofundada para que os objetivos sejam cumpridos. Os quadros a seguir mostram, respectivamente, as habilidades prioritárias e adicionais. Estas podem ser trabalhadas caso haja tempo suficiente para tal.

Identificar diferentes sistemas operacionais e reconhecer sua presença em diversos sistemas computacionais.
Compreender, em linhas gerais, o histórico do sistema operacional Linux e sua situação atual.
Compreender as diferenças entre software de código aberto e software de código fechado e identificar a presença de cada um em situações diferentes.
Operar com os comandos básicos de UNIX em linha de comando, navegando pela árvore do sistema de arquivos, copiando, criando e removendo arquivos e pastas.
Compreender, em linhas gerais, a estrutura de um sistema Linux: kernel e camadas exteriores, sistema de arquivos, usuários e grupos.
Operar programas comumente oferecidos em sistemas operacionais Linux para instalar programas, editar arquivos de texto e acessar sistemas remotamente.
Compreender diferenças entre uma linguagem de programação de alto nível e uma de baixo nível, bem como noções de paradigmas de programação.
Escrever programas simples em Python.
Configurar um Raspberry Pi para operação por acesso remoto, incluindo montagem dos acessórios e instalação do sistema operacional.
Utilizar o Raspberry Pi para resolução de situações-problema que envolvam a utilização do dispositivo permanentemente em operação.
Integrar conhecimentos de Linux e Python para controlar pinos de GPIO no Raspberry Pi.
Configurar o Raspberry Pi para comunicação com sensores e/ou dispositivos externos via pinos de GPIO.
Utilizar o Raspberry Pi para resolução de situações-problema que envolvam a comunicação do dispositivo com sensores, dispositivos externos e/ou internet.
Identificar situações onde o uso do Raspberry Pi é uma solução viável.

Escrever programas simples em Lua.
Reconhecer diferentes linguagens de programação e escrever programas simples em algumas.
Reconhecer linguagens de alto e baixo nível, suas vantagens e desvantagens e recomendá-las para projetos adequados a cada uma.
Utilizar métricas de desempenho de software para avaliar a qualidade e eficiência de programas, bem como avaliar códigos segundo tais métricas.
Compreender a função dos paradigmas de programação e relacioná-los com medidas de qualidade e desempenho de software.

6 Resultados

Esta seção se destina à apresentação do material produzido juntamente com este trabalho. A maior parte do material é destinada ao professor, servindo como guia detalhado dos conteúdos a serem abordados em cada aula. Entretanto, também foi desenvolvido algum material de apoio aos estudantes, que, apesar de limitado, tem a função de nortear e padronizar o desenvolvimento de material adicional pelo professor.

O material produzido se divide nas seguintes partes:

- Ementa da disciplina
- Programa do conteúdo de cada aula, destinado ao professor
- Apresentação de slides da primeira aula
- Material de apoio para os estudantes, incluindo exemplos de código e imagens

O material é apresentado nas seguintes subseções, iniciando-se pela ementa do curso, com um resumo do conteúdo total abrangido pela disciplina (separado em prioritário e adicional); em seguida, expõe-se o programa dos conteúdos divididos por aula, para um total de 20. Escolheu-se um total de 20 aulas devido ao número de semanas letivas em um semestre; entretanto, este número é uma cota superior para o número de aulas em um semestre, que costuma ser inferior. Por isso, o conteúdo prioritário é incluído na primeira parte do curso (aulas 1 a 10), e o conteúdo adicional na segunda (aulas 11 a 20). Além disso, o material de apoio aos estudantes é explorado em cada uma das aulas. Portanto, é incluído no programa de cada aula para a qual foi desenvolvido. Todo o material de apoio também é disponibilizado em anexo.

6.1 Ementa

A ementa da disciplina está descrita no quadro abaixo.

Prioritária:

História dos sistemas UNIX e Linux; distribuição de uso de sistemas operacionais e conjuntura atual de Linux no mundo.

Conceitos de software livre e de código aberto; conceito de hardware livre; projetos que adotam tais modelos de desenvolvimento;

Operação por linha de comando em Linux; comandos básicos, com e sem opções; bash.

Características estruturais de sistemas operacionais baseados em Linux; conceito de kernel, processos, estrutura de diretórios, usuários e grupos.

Programas de linha de comando em Linux; gerenciadores de programas, editores de texto e acesso remoto.

Linguagem de programação Python; história, características e funcionalidades; uso em projetos profissionais.

Raspberry Pi; história, modelos e especificações, sistemas operacionais; montagem, setup e uso remoto; configuração de programas e serviços; controle de pinos GPIO com Python para comunicação externa.

Adicional:

Projetos com Raspberry Pi; leitura de sensores e comunicação com dispositivos digitais externos; utilização embarcada.

Noções de engenharia de software; complexidade algorítmica; métricas de desempenho; eficiência de CPU e memória; legibilidade, modularidade e reuso de código.

Paradigmas de programação e suas aplicações; imperativo, funcional, orientado a objeto.

Classificação em alto e baixo nível e uso de linguagens de programação; noções de C/C++, Python, Octave, Java e Lua; análise comparativa de diferentes linguagens.

Uso de Lua em integração com Raspberry Pi; noções de programação na linguagem, controle e comunicação de GPIO com Lua.

6.2 Aulas

6.2.1 Aula 1: UNIX, Linux e Open Source

A primeira aula tem a função de situar os estudantes no curso. Serve, essencialmente, o mesmo propósito que um dos principais deste trabalho: argumentar em favor da necessidade da disciplina, tomando como eixo a conjuntura atual de Linux no mundo. O Raspberry Pi é citado ao final, porém discutido apenas tangencialmente. O aprofundamento é feito nas últimas aulas do conteúdo prioritário, após a introdução dos principais conceitos utilizados na disciplina.

Inicia-se a aula por uma breve exploração da história de UNIX, apresentação dos criadores Ben Thompson e Dennis Ritchie, desenvolvimento da plataforma e pontos de contato com a linguagem de programação C. Lançado no ano de 1970, foi escrito inicialmente em Assembly, mas reescrito em 1973 em C (linguagem criada para tal tarefa). Difundiu-se nas décadas de 70 e 80, e hoje é marca registrada de posse de The Open Group. Por essa razão, muitos sistemas operacionais que se baseiam em UNIX (tal como Linux) são denominados UNIX-like.

A seguir, introduz-se um histórico do Linux. Apresenta-se o criador Linus Torvalds, bem como as motivações pelas quais o kernel foi criado, pela necessidade de Linus de utilizar funções de seu computador com processador 386. Relacionando com a iniciativa promovida pelo projeto GNU, GNU Hurd, pode-se esclarecer tanto o contexto do desenvolvimento inicial do kernel Linux quanto o significado do termo “kernel”. GNU Hurd é o kernel desenvolvido pelo projeto GNU, que, na época, não atendia às demandas de Linus.

Então, mostra-se um resumo do desenvolvimento de sistemas UNIX e UNIX-like desde a criação do primeiro até os dias atuais. Nesta parte, vale frisar que a maior parte dos sistemas operacionais historicamente conhecidos, exceto Windows, derivam de UNIX: Linux, Solaris, BSD, MacOS, entre outros. Também é importante notar a estrutura em árvore, na qual grupos de desenvolvedores utilizam trabalho prévio para aprimorar as plataformas. Tal estrutura é possibilitada, principalmente, pelo modelo de desenvolvimento

aberto adotado por eles.

Na sequência, com o intuito de argumentar em favor da importância de Linux no mercado de sistemas operacionais, discutem-se os market shares da plataforma em diferentes nichos: desktops, dispositivos móveis e super-computadores. Neste ponto, também é uma opção incluir servidores. Concluindo em uma tendência de crescimento de Linux nos nichos, bem como seu domínio em todos exceto desktops, pretende-se mostrar a necessidade de conhecimento da plataforma para profissionais de tecnologia. Aqui, também são listadas algumas distribuições de Linux, a fim de familiarizar os estudantes com as mais populares e mostrar a variedade de sistemas operacionais que utilizam Linux como kernel. Alguns dos principais são Ubuntu, Linux Mint, Arch Linux, Fedora, CentOS, Manjaro e Debian.

Também explora-se, em maiores detalhes, a estrutura de sistemas operacionais em camadas, com o intuito de permitir, ao mesmo tempo, flexibilidade e segurança. O kernel é o nível (ou camada) mais próximo do hardware, o que demanda dele as maiores garantias de estabilidade e segurança. Nota-se, também, que esta estrutura é adotada não só por Linux, mas também pelos demais sistemas operacionais (incluindo Windows).

A última parte da aula se concentra nos conceitos de software livre e código aberto. Inicia-se por um histórico do software livre (que possui muitos pontos de contato com os históricos anteriores, notavelmente com o projeto GNU). Posterior ao início do desenvolvimento de UNIX, começou com a criação do projeto GNU por Richard Stallman, na década de 80, e resultou em um modelo de desenvolvimento de software com enorme e crescente adoção. É importante diferenciar os termos “free software” (software livre) de “open source” (código aberto). O primeiro estende o princípio de liberdade para além do código, colocando-o como questão de princípio moral. O segundo se limita somente ao modelo de desenvolvimento, podendo o software ser comercial ou até mesmo de uso restrito.

Por último, listam-se diversos projetos de renome que se classificam como software de código aberto e/ou livre. Dentre os principais, têm-se:

- *Android*, sistema operacional móvel de maior adoção no mundo;

- *Apache* e *NGINX*, servidores web utilizados na maioria dos servidores no mundo;
- *Blender*, software de modelagem 3D utilizado profissionalmente por estúdios de animação e desenvolvedores de jogos;
- *Chromium*, precursor do navegador Google Chrome;
- *GIMP*, software de edição de imagens;
- *Octave*, implementação livre de MATLAB;
- *VLC*, player de áudio e vídeo;
- *edX*, plataforma educacional fundada pela parceria entre Harvard e MIT;
- *LibreOffice*, implementação livre de pacote Office;
- *Ubuntu*, sistema operacional Linux de maior adoção no mundo;
- *VirtualBox*, software de virtualização;
- *MySQL*, software de gerenciamento de banco de dados.

Por fim, como exemplos do conceito de software livre adotado para além de projetos de software, introduzem-se Arduino e Raspberry Pi, dispositivos de hardware livre.

Esta aula também inclui, como material de apoio, uma apresentação de slides com o conteúdo nela desenvolvido, na Subseção 8.2. A apresentação tem o propósito de fornecer um padrão para criação de slides para as aulas seguintes, além de servir como material para consulta dos estudantes. A capa, com o padrão gráfico adotado, é mostrada na 26.

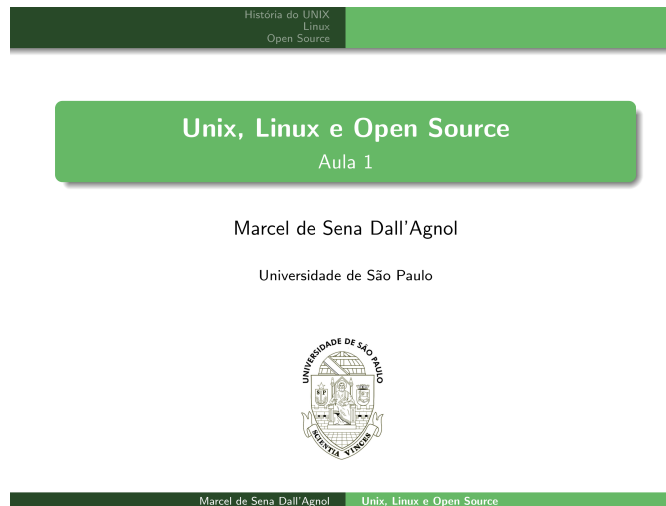


Figura 26: Capa da apresentação de slides da aula 1

6.2.2 Aula 2: Operação básica por linha de comando

As aulas seguintes são destinadas a familiarizar os estudantes com a interface de linha de comando, mostrando não só suas raízes históricas, mas também suas vantagens em relação a outras interfaces para determinadas tarefas. Esta, no entanto, permanece em um nível superficial, somente situando a linha de comando como uma possível maneira de operar um computador.

Esta aula tem muitos assuntos em comum com as aulas seguintes (por exemplo, permissões de usuários e estrutura de árvore de sistema de arquivos). Entretanto, o seu intuito não é o de aprofundar em tais conceitos, mas meramente situar a linha de comando como uma interface de interação humano-computador, tal qual interfaces gráficas com as quais os estudantes devem estar melhor acostumados. Sendo assim, o foco reside nos comandos simples, mas comandos mais complexos também são utilizados.

Em um primeiro momento, é importante explicitar o papel de cada elemento da interface de linha de comando (prompt, comandos recentes, etc). Uma vez familiarizados com a forma de introduzir input (pode-se pedir aos estudantes que tentem executar qualquer comando e posteriormente observarem as mensagens de erro), pode-se introduzir a estrutura geral de um

comando em Linux:

`comando -opção1 ... -opçãon argumento1 ... argumenton`

Não é necessário trabalhar com opções de comandos neste momento, mas é importante apresentar a forma geral de um comando de forma completa. A seguir, pode-se iniciar a manipulação de arquivos e navegação pelo sistema de arquivos. Na Tabela 4, lista-se uma sequência de comandos básicos, juntamente com o nome que o originou (quando relevante ao nome do comando) e uma breve descrição de cada um.

Comando	Nome	Descrição
cd	Change directory	Muda o diretório atual.
ls	List directory contents	Lista o conteúdo de um diretório.
pwd	Print working directory	Imprime o diretório atual em tela.
cp	Copy	Copia um arquivo.
mv	Move	Move um arquivo.
rm	Remove	Remove um arquivo.
mkdir	Make directory	Cria um diretório.
rmdir	Remove directory	Remove um diretório.
touch	-	Cria um arquivo.

Tabela 4: Comandos iniciais em UNIX

Uma vez introduzidos os comandos, pode-se pedir aos estudantes que os explorem, e lançar desafios, tais como:

- Criar, copiar e mover arquivos para diretórios determinados;
- Desenhar um esquema do conteúdo de uma pasta e pedir aos estudantes que torne-a como tal;
- Descobrir o conteúdo de um diretório após uma sequência de comandos.

Como exemplo de uso dos comandos acima, pode-se executar a sequência abaixo:

```
$ mkdir Pasta
$ cd Pasta
$ pwd
$ touch Arquivo1
$ ls
$ cp Arquivo1 ../Arquivo2
$ mv Arquivo2 ..
$ rm Arquivo1
$ cd ..
$ ls
$ rmdir Pasta
$ ls
$ rm Arquivo2
```

Neste ponto, é interessante mostrar a funcionalidade de completar diretórios com a tecla **Tab**; a repetição de comandos com as teclas direcionais para cima e para baixo, além de **!** e **!!**; e o uso de wildcards (*****) para executar comandos em um número de arquivos que seguem um padrão. Além disso, podem-se apresentar os comandos acima com opções.

Por fim, introduzem-se mais alguns comandos e a funcionalidade de “pi-ping”, que permite direcionar o output de um comando ao input de outro, além da de salvar o output de um comando em um arquivo. Na Tabela 5, listam-se os comandos adicionais a serem trabalhados, juntamente com uma breve descrição.

A sequência de comandos no quadro em seguida exemplifica seu uso. É importante salientar, para os estudantes, o que está sendo feito em cada um deles.

```
$ mkdir Pasta
$ cd Pasta
$ echo 'Eco!'
```

Comando	Descrição
<code>rm -r</code>	Remove uma pasta e seus conteúdos, recursivamente.
<code>ls -l</code>	Imprime o conteúdo de um diretório com detalhes.
<code>echo</code>	Imprime uma string em tela.
<code>cat</code>	Concatena arquivos e imprime o resultado em tela.
<code>wget</code>	Baixa um arquivo de uma URL especificada.
<code>man</code>	Mostra uma descrição detalhada de um comando e suas opções.
<code>grep</code>	Imprime linhas de um texto que contêm um padrão especificado.

Tabela 5: Comandos básicos UNIX adicionais

```
$ echo 'Eco!' > Arquivo1
$ echo 'Alo!' > Arquivo2
$ cat Arquivo1 Arquivo2 > Arquivo3
$ cat Arquivo3
$ grep E Arquivo3
$ wget http://www.google.com/index.html
$ man wget
$ mkdir Pasta2
$ cp Arquivo* Pasta2/
$ ls -l
$ cd Pasta2
$ ls -l
$ cd ../..
$ rm -r Pasta
```

6.2.3 Aula 3: Programas de linha de comando

Esta aula toma como base o ferramental desenvolvido na última para realizar tarefas mais complexas, servindo, assim, dois propósitos: finalizar o aprendizado das ferramentas necessárias para operação do Raspberry Pi no futuro e mostrar casos de uso nos quais a linha de comando é uma interface

mais viável que interfaces gráficas. Esta aula já assume que o estudante tenha acesso a um ambiente similar ao Raspberry Pi (um sistema com Debian Linux) ou ao Raspberry Pi em si.

Primeiramente, é necessário pontuar as diferenças entre os recursos de instalação e gerenciamento de software em Linux em comparação com outros sistemas operacionais (principalmente Windows). A forma de distribuição de software em Windows é, tradicionalmente, por meio de arquivos de instalação disponibilizados pelos autores dos programas em seus próprios sites ou por meio de CDs ou DVDs. Isso significa que não existia uma fonte central de onde os usuários pudessem obter os programas que desejavam. Tal cenário mudou com o Windows 8, que trouxe a Windows Store (seguindo um modelo criado para dispositivos móveis) para a distribuição de software. Apesar disso, a maneira tradicional de distribuição e instalação de software permanece muito utilizada, pois é o único meio de obter programas ainda indisponíveis na Windows Store.

Os mecanismos de distribuição de software em Linux, entretanto, são centrados em repositórios. Tais repositórios são estruturas que permitem, ao mesmo tempo, a centralização que faltava às iterações de Windows anteriores ao Windows 8, e a descentralização que permite independência em relação a um ambiente como a Windows Store. Repositórios podem ser criados por qualquer pessoa ou grupo, que se torna responsável por garantir que os programas disponibilizados sejam compatíveis com as plataformas que atendem. Os repositórios também são uma maneira muito mais simples de garantir que o software de um sistema permaneça atualizado, pois, em lugar de uma solução para atualizações criada por cada desenvolvedor de software, os repositórios são responsáveis por atualizar os programas que disponibilizam. Uma vez atualizados nos repositórios, as atualizações são automaticamente distribuídas a todos que os utilizam. Tal estrutura permite, além disso, repositórios com versões de software que priorizam estabilidade ou atualizações recentes (respectivamente, estáveis e instáveis).

Pode-se citar, portanto, alguns programas de gerenciamento de software específicos de diferentes sistemas Linux: `apt-get` para Debian; `yum` para Fedora; `pacman` para Arch Linux, entre outros. Contudo, como Debian é a

distro de escolha para o Raspberry Pi, instalações e atualizações de software serão realizadas com o programa **apt-get**. Os programas utilizados nesta aula são listados na Tabela 6, juntamente com uma breve descrição de cada um.

Programa	Descrição
apt-get	Gerencia repositórios e programas; realiza instalações, atualizações e remoções de software.
nano	Editor de texto em linha de comando.
vi	Editor de texto em linha de comando.
htop	Visualizador de processos e monitor de recursos.
ssh e sshd	Cliente e servidor para operação remota de máquinas.
tightvncserver	Servidor VNC, para operação remota com interface gráfica.

Tabela 6: Programas de linha de comando

Inicia-se utilizando o programa **apt-get** para atualizar os programas já presentes no sistema, e, a seguir, para instalar os demais. Alguns deles já devem estar disponíveis e não necessitam ser instalados, mas pode-se adicioná-los como argumentos para ilustrar a forma de instalação. A sequência de comandos a seguir realiza tal tarefa.

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install nano vi htop ssh tightvncserver
```

É interessante pontuar, aqui, a diferença entre instalar programas via interfaces gráficas e linha de comando. A segunda permite automatização, e, portanto, um processo muito mais rápido, principalmente quando são instalados múltiplos programas. Além disso, é mais simples seguir (e fornecer) sequências determinadas de operações; basta copiar e colar os comandos em sequência, em contraste com a procura de itens em menus gráficos.

Além disso, surgiu, pela primeira vez, a necessidade de realizar operações como superuser, pois a instalação de programas é tarefa que exige permissões

indisponíveis a usuários regulares. Isso pode ser problematizado e discutido neste momento, porém será abordado com maiores detalhes na Aula 4.

As últimas atividades desta aula são a execução e a exploração dos programas instalados. Se disponíveis, é interessante acessar o sistema com os programas de operação remota (**ssh** para linha de comando e **tightvncserver** para interface gráfica). Além disso, pode-se utilizar **nano** e/ou **vi** para edição de arquivos de texto, cuja manipulação com os comandos apresentados na Aula 2 (**echo**, **touch** e **cat**) é limitada. **htop** é um programa útil não só para monitorar uso de memória e CPU, mas também para introduzir o conceito de processo.

6.2.4 Aulas 4 e 5: Características estruturais de Linux

Até esta aula, surgiram algumas situações que sugerem a necessidade de entender as características estruturais de um sistema Linux com algum detalhamento. A navegação no sistema de arquivos está intimamente relacionada com sua estrutura em árvore; a instalação de programas requer permissões que um usuário não possui. Sendo assim, nesta aula, são explorados alguns conceitos de sistemas operacionais: kernel, usuários, grupos e permissões e estrutura de diretórios. É importante salientar que os conceitos não se restringem a Linux, e se aplicam também a outros sistemas operacionais. Entretanto, a exploração é realizada em referência a um sistema Linux. Devido à quantidade de conteúdo, recomenda-se ministrar o assunto em duas aulas consecutivas.

Pode-se iniciar com uma discussão sobre as operações que devem ser permitidas a um usuário do computador. Pode-se debater a viabilidade de permitir a um usuário qualquer de um sistema (que, em muitos casos, sequer possui todos os dados nele armazenados ou o hardware em que é executado) as seguintes operações:

- Alterar a frequência de operação do processador;
- Gravar dados em uma região arbitrária do disco rígido;
- Modificar o conteúdo de arquivos do sistema operacional;

- Desligar ou reiniciar a máquina.

Os estudantes provavelmente chegarão à conclusão que um usuário deve ser limitado em suas possibilidades, pois, caso contrário, erros ou comandos maliciosos podem ter consequências graves. Neste momento, portanto, introduz-se a solução utilizada pelos sistemas operacionais modernos: o kernel.

Os sistemas operacionais Linux são divididos em duas camadas principais, cada uma com permissões distintas de operações. A camada externa, que tem permissões restritas, é a camada de usuário; a interna, responsável pela comunicação direta com hardware, é chamada kernel. Tal estrutura permite maior segurança na execução de programas, e restringe a quantidade de código que deve ser mantida o mais estável e sem bugs possível.

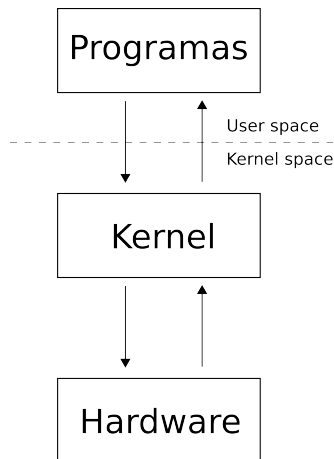


Figura 27: Estrutura em camadas de Linux

Em sequência, apresenta-se o conceito de processo: uma instância de programa em execução. Não é necessário entrar em detalhes em relação a conceitos como distribuição de tempo de uso do processador ou threads. A introdução deste conceito se destina a mostrar que um programa, que é comumente compreendido como um conjunto estático de dados no disco rígido, tem também um aspecto dinâmico. E, além disso, é na instância dinâmica que surge a necessidade de avaliar se uma operação deve ser executada ou

não. Após mostrar que tal avaliação é feita pelo kernel (que deve, portanto estar constantemente em execução), permanece a necessidade dos processos de executar operações que podem ser perigosas. O “pedido de permissão” ao kernel para que essas operações sejam executadas é feito por meio de system calls. A Tabela 7 resume algumas system calls utilizadas em Linux, que podem ser mostradas para ilustrar os tipos de situação nas quais o kernel é acionado:

System call	Descrição
open()	Abre um arquivo.
write()	Escreve bytes de um buffer em um arquivo.
fork()	Cria um novo processo como cópia do que chamou fork()
kill()	Finaliza um processo.

Tabela 7: System calls em Linux

A seguir, introduz-se a estrutura do sistema de arquivos. Em Linux, essa estrutura toma o formato de árvore, onde cada arquivo ou diretório gera um novo ramo, partindo do diretório que o contém, e as folhas são diretórios vazios ou arquivos. Tal estrutura requer uma raiz (um diretório que contém todos os demais). A raiz (root, em inglês) é simbolizada por '/’.

Neste ponto, é interessante esclarecer as diferentes formas pelas quais um usuário pode navegar pela árvore. Caminhos absolutos devem sempre iniciar com '/', pois todo diretório tem um caminho cuja origem é a raiz; caminhos relativos são aqueles que foram utilizados nos exemplos anteriores, onde a não precedência por '/' indica que parte-se do diretório atual. Além disso, existem atalhos para a pasta home do usuário (~), a pasta atual (.) e a pasta em que a atual está contida (..). Assumindo que a pasta home do usuário é '/home/pi' (como é o caso na distro que é utilizada no Raspberry Pi, que será adotada deste ponto em diante), os dois comandos abaixo são equivalentes:

```
$ cd /home/pi
$ cd ~
```


Finalizando a introdução ao sistema de arquivos, deve-se também citar os arquivos e diretórios ocultos (devem ser precedidos por `'.'`) e características que tornam o sistema orientado a arquivos. Neste tipo de sistema, todo canal de leitura ou escrita de dados é abstraído para um arquivo; por exemplo, todos os dispositivos externos (mouse, teclado, pen drive, leitores de CD/DVD, entre outros) são acessados por meio de arquivos no diretório `'/dev'`.

Por último, é necessário apresentar aos estudantes os conceitos de permissões de acesso, leitura e escrita de arquivos. Em Linux, a permissão para manipulação de arquivos é dividida em três tipos, para três tipos de inclusão de usuários em grupos (usuários podem estar inclusos em um ou mais grupos). As subdivisões de permissões são:

- *read* (r), permissão para leitura de arquivo;
- *write* (w), permissão para escrita em arquivo;
- *execute* (x), permissão para executar o arquivo, interpretando-o como código de máquina.

As subdivisões de inclusões em grupos são:

- *user* (u), o usuário que possui o arquivo;
- *group* (g), os grupos nos quais o usuário que possui o arquivo está contido;
- *all* (a), todos os usuários.

Além disso, existe um usuário especial (chamado root, analogamente ao sistema de arquivos) que possui permissões de todos os tipos em qualquer arquivo ou diretório. Este usuário é utilizado para manipular arquivos e diretórios contidos no diretório raiz que não são possuídos por outros usuários. Este é o caso, por exemplo, de programas disponíveis a todos os usuários, instalados por meio do comando `apt-get`; daí a necessidade de executá-lo

temporariamente como root, papel a que se prestam os comandos `sudo` e `su`).

Como atividade final, pode-se indicar aos estudantes que mudem inclusões de grupos, posse de arquivos e diretórios, bem como permissões, com os comandos `chown`, `chgrp` e `chmod`.

6.2.5 Aula 6: Python

Esta aula se destina a desenvolver algumas habilidades básicas com a linguagem de programação Python. As principais motivações são o aprendizado de uma linguagem de alto nível de uso bastante difundido e a disponibilidade de controle dos pinos de GPIO do Raspberry Pi por meio da linguagem. Assume-se, nesta aula, que os estudantes têm experiência com outras linguagens de programação, e, portanto, não é necessário entrar em detalhes sobre uso de variáveis, fluxo de controle e outros conceitos de programação com os quais os estudantes já devem ter trabalhado. Escolheu-se a linguagem de programação C como base comparativa com Python, pois é a linguagem ensinada nos primeiros anos de graduação aos estudantes de engenharia mecatrônica na EESC-USP.

Primeiramente, é importante salientar as principais diferenças de sintaxe entre C e Python. Para isso, pode-se partir de um programa escrito nas duas linguagens. Utilizando o algoritmo *bubble sort*, podem-se traçar algumas diferenças visíveis entre as linguagens. A fim de suscitar perguntas e observações, podem-se utilizar os programas `bubblesort.c` e `bubblesort.py`, na Subseção 8.3.

Juntamente com os estudantes, algumas das diferenças que podem ser levantadas são:

- O programa em Python requer menos linhas que o mesmo em C;
- Não são utilizadas chaves em Python;
- Não há declaração de variáveis em Python;
- Indentação em Python é mais rígida;

- Não se utiliza ponto e vírgula no fim das linhas em Python.

Tais características são tomadas como ponto de partida para explicar as maiores diferenças entre Python e C. Primeiramente, nota-se que a linguagem não utiliza chaves para determinar escopos. Isto ocorre devido ao foco em legibilidade de Python. A linguagem é sensível a espaços brancos, agrupando linhas de texto com a mesma indentação no mesmo bloco, o que torna o uso de chaves desnecessário (mas aumenta a exigência ao programador para manter o padrão de indentação que a linguagem exige). Além disso, Python é uma linguagem interpretada dinamicamente. Isto significa que não há declarações nem tipos explícitos de variáveis; toda atribuição meramente liga um objeto a um nome, que pode ser utilizado mais de uma vez para objetos de tipos distintos; a avaliação da validade de uma operação em uma variável é deferida para o momento da execução (em contraste com C, que realiza avaliações no momento da compilação). Isto também implica que o arquivo de texto não é compilado, mas interpretado linha por linha em sequência, no momento da execução.

Existem importantes ferramentas na linguagem Python denominadas list comprehensions. Estas são formas de navegar as estruturas de dados que podem acessíveis como listas de forma mais clara e compacta, sem a necessidade de manipular índices. O snippet de código abaixo ilustra a utilização de list comprehensions.

```
a = [1, 2, 3, 4, 5]
b = [2, 4, 6, 8, 10]
for num in a:
    print(num * num)
print([num * num for num in b])
print([na + nb for na in a for nb in b])
```

Por fim, podem-se discutir as diferenças entre uma linguagem de alto nível (como Python) e uma de baixo nível (como C), para além de sintaxe. Um dos pontos mais importantes levantados anteriormente é o tamanho do

código. Uma linguagem de alto nível possui recursos que permitem ao programador se concentrar no problema em questão, oferecendo um conjunto de ferramentas que torna tanto o tempo de desenvolvimento quanto o tamanho do programa menores. Ao abstrair características que linguagens de baixo nível deixam explícitas (como alocação de memória e propriedades de hardware), o desenvolvimento é facilitado, mas o programa incorre em redução de eficiência.

É importante pontuar a origem dos termos alto/baixo nível, que não têm o sentido usual. Linguagens de alto nível estão “distantes” do hardware, abstraindo e gerenciando os componentes para permitir uma interação do programador mais próxima da lógica do programa e das estruturas de dados nele contidas. Linguagens de baixo nível estão mais “próximas” do hardware, permitindo ao programador um controle mais fino do que ocorre com hardware, aumentando a complexidade do programa mas possibilitando aumento de eficiência.

Ao fim da aula, apresentam-se alguns projetos de renome que utilizam Python em seu desenvolvimento. Alguns deles são:

- *ROS*, sistema operacional para robôs visto na primeira aula;
- *Dropbox*, programa popular para sincronização de pastas;
- *Blender*, software profissional de modelagem 3D;
- *BitTorrent*, software para compartilhamento de dados P2P;
- *OpenStack*, sistema operacional distribuído desenvolvido por um consórcio com empresas incluindo Red Hat, IBM, Cisco, HP e VMWare;
- *reddit*, website agregador de links extremamente popular;
- *Django*, framework para desenvolvimento web;
- *Raspberry Pi*.

6.2.6 Aula 7: Apresentação, montagem e configuração do Raspberry Pi

As aulas anteriores, apesar de tangenciarem o Raspberry Pi em vários pontos, não o colocavam como prioridade. Na verdade, para nenhum dos assuntos anteriores foi necessária a disponibilização de um dispositivo aos estudantes. Uma máquina com Debian e acesso a internet seria suficiente para tal.

Nesta aula, inicia-se a utilização do Raspberry Pi em si; portanto, é a partir dela que se torna indispensável o acesso ao dispositivo. Inicialmente, realiza-se uma exploração do hardware, de suas especificações e características físicas (e, para tanto, é importante o acesso físico ao aparelho). Então parte-se para a instalação e configuração do dispositivo, para que esteja preparado para uso nas aulas seguintes.

No início da aula, os dispositivos são entregues aos estudantes. Então, realiza-se uma explicação mais detalhada do histórico do Raspberry Pi, bem como de suas especificações técnicas.

Raspberry Pi é um computador desenvolvido por um grupo de professores da universidade de Cambridge com o propósito específico de fomentar o ensino de computação nas escolas e universidades. Para tanto, o objetivo era criar um computador com baixo custo, compacto e manipulável, pois acreditavam que a crescente complexidade e encapsulamento dos computadores vinha tornando a exploração autônoma e curiosa cada vez mais difícil. Vendo os efeitos disso nos estudantes que chegavam à universidade, começaram esforços em 2006 para criar o que veio a ser o Raspberry Pi, lançado em 2012.

O Raspberry Pi tem 2 modelos disponíveis para venda: Model A, com custo de US\$25, e Model B, com custo de US\$35. Recentemente, foi lançada uma versão atualizada do segundo, chamada Model B+, mantendo-se o preço. As principais especificações dos modelos estão resumidas na Tabela 8.

Uma vez que os estudantes manipularam a placa, podem-se discutir os acessórios necessários para que ela funcione de fato. Como a placa provê apenas os componentes essenciais ao funcionamento de um computador, são

Modelo	Preço	CPU	Memória RAM	Conectores
A	US\$25	ARM11 700MHz	256MB	HDMI; RCA; 3,5mm TRS; USB 2.0 (1); GPIO (26); SD.
B	US\$35		512MB	Ethernet; HDMI; RCA; 3,5mm TRS; USB 2.0 (2); GPIO (26); SD.
B+			512MB	Ethernet; HDMI; RCA; 3,5mm TRS; USB 2.0 (4); GPIO (40); MicroSD.

Tabela 8: Especificações dos modelos de Raspberry Pi

necessários alguns componentes adicionais: fonte de alimentação, armazenamento permanente, teclado e monitor. Conforme orçado na Seção 3, a fonte de alimentação e o armazenamento permanente (cartão SD ou microSD) estarão disponíveis. Pode-se, então, utilizar os mouses e teclados USB e monitores HDMI do laboratório de ensino. Como este setup é temporário (os dispositivos serão controlados remotamente), é simples retirá-los e colocá-los nos lugares de origem após a configuração. Os estudantes podem ligar a fonte de alimentação e verificar que o LED de alimentação acende. Posteriormente a isso, é necessário formatar o cartão SD (ou microSD) com o sistema operacional. Tendo disponível a imagem do Raspbian (disponível para download em [22]), basta seguir o passo-a-passo na documentação oficial do Raspberry Pi (disponível em [23]). Existem guias para os três sistemas operacionais mais populares (Windows, Mac e Linux).

A formatação do cartão SD pressupõe que os computadores dos laboratórios de ensino possuam entrada para esse tipo de cartão de memória. Entretanto, não é comum que computadores de laboratórios de ensino o tenham. Como alternativas, pode-se adquirir um adaptador para portas USB, ou mesmo fornecer os cartões SD já com o sistema pré-instalado (apesar de ser preferível que os estudantes realizem a formatação por conta própria). Ao realizar a formatação, é interessante notar que o sistema de arquivos localiza o cartão SD em um diretório dentro de `/dev`.

Após a formatação, deve-se conectar o cartão SD ao respectivo slot no

Raspberry Pi, além de teclado e monitor. Conectando-se a fonte de alimentação, podem-se observar os LEDs na placa acenderem e o processo de boot mostrado no monitor. Entrando com login e senha padrão (login `pi` e senha `raspberrypi`), explora-se, então, o ambiente Linux com os comandos aprendidos nas aulas anteriores, e, por fim, configurar o dispositivo com o comando `raspi-config`. É interessante, neste momento, sugerir aos estudantes que experimentem interagir também por meio da interface gráfica oferecida pelo dispositivo, que pode ser habilitada executando o comando `startx`. Devido às limitações de hardware, o processo deve ser lento; isto serve como ilustração adicional de que o uso da linha de comando é justificado em muitos casos, mesmo com o hardware avançado que temos hoje em dia. Conferindo que o servidor SSH está ativo (essencial para o login remoto nas aulas seguintes), pode-se então desligar o aparelho com o comando `shutdown -h now`. O professor deve, finalmente, recolher os aparelhos e conectá-los à rede para operação na aula seguinte.

6.2.7 Aula 8: Instalação de programas e configuração de serviços

Esta aula tem o intuito de mostrar, de forma prática, as potencialidades do Raspberry Pi como servidor. Devido ao baixo consumo de energia, é viável mantê-lo ligado continuamente, realizando tarefas para as quais comumente um computador de mesa ou laptop não é adequado. Esta aula ainda não utiliza o dispositivo para aplicações em eletrônica; estas são abordadas na aula seguinte. Ao longo desta aula, os estudantes devem instalar e configurar uma série de programas que possibilitam ao Raspberry Pi realizar funções de um servidor web a sincronização de arquivos. Não é necessário que realize todas as funções simultaneamente; entretanto, é interessante que pelo menos duas estejam em operação. As aplicações e o passo-a-passo para a configuração de cada uma foram retirados de tutoriais disponíveis na Internet. Aqueles mais interessantes e didáticos são:

- Servidor web com Apache e MySQL [24];
- Servidor de arquivos com Samba [25];

- Sincronização de pastas entre dispositivos com OwnCloud [26];
- Download e upload de arquivos por BitTorrent [27];
- Relay para a rede Tor (The Onion Router) [28];

Deve-se frisar que nem todos os tutoriais acima são apropriados para o setup escolhido nos Raspberry Pi utilizados no curso, e, portanto, devem ser realizadas algumas modificações para que funcionem de acordo com o esperado. Além disso, a maior parte das verificações de que os serviços estão funcionando depende de acessar o Raspberry Pi por meio de outro computador, algumas vezes com software específico.

Devido à sobreposição de muitos dos programas que devem ser instalados para que todos os serviços sejam executados, uma lista unificada é disposta a seguir. Executando-se este comando, os programas requeridos por todas as funções serão instalados:

```
apt-get install apache2 php5 php5-json php5-gd php5-sqlite
curl libcurl3 libcurl4-openssl-dev php5-curl php5-gd
php5-cgi php-pear php5-dev build-essential libpcre3-dev
php5 libapache2-mod-php5 php-apc mysql-server mysql-client
php5-mysql owncloud samba noip transmission-daemon tor
```

É necessário ressaltar, entretanto, que outros programas podem ser necessários de acordo com a estrutura da rede em que o Raspberry Pi é conectado. Por exemplo, caso o IP externo seja dinâmico, para acessá-lo por fora da rede é necessário instalar um daemon para DNS dinâmico, tal como **noip**. Se os IPs da rede interna são distribuídos por DHCP, pode também ser necessário investigar a rede para descobrir o IP designado ao Raspberry Pi; tal operação pode ser realizada por um software como **nmap**.

É possível, também, baixar versões dos programas mais atualizadas diretamente dos sites dos desenvolvedores (o que pode ser útil, em particular, para o OwnCloud). Nesse caso, tais pacotes de software devem ser removidos do comando acima.

Uma vez instalados os pacotes, basta configurar os serviços. Muitos deles requerem edição de arquivos de configuração e execução de comandos adicionais. Após configurados, algumas atividades podem ser propostas para verificar seu funcionamento:

- Criar um arquivo HTML simples com o nome `index.html` na pasta `/var/www`, e acessar o IP do Raspberry Pi por meio de um navegador. O arquivo criado deve ser reproduzido no navegador.
- Transferir arquivos de/para o Raspberry Pi por meio do protocolo Samba; isso pode ser realizado com software como Filezilla, ou mesmo identificando o servidor de arquivos no Windows Explorer.
- Sincronizar uma pasta via OwnCloud e verificar que os conteúdos são copiados para uma pasta respectiva no Raspberry Pi (necessita instalação do cliente OwnCloud no computador sincronizado).
- Acessar o programa de gerenciamento de torrents Transmission via navegador e adicionar o torrent da versão mais atual de Raspbian (a qual foi utilizada para formatar o cartão SD). Os estudantes poderão ver o progresso do download, e, se possível, a quantidade de dados enviada após alguns dias.
- Utilizar o programa **nethogs** para monitorar a quantidade de tráfego sendo roteada pela rede Tor.

6.2.8 Aula 9: Controle básico de GPIO

Nesta aula, inicia-se o trabalho de controle dos pinos digitais de GPIO (General Purpose Input and Output) com o Raspberry Pi. Esta é a última das ferramentas necessárias para que o estudante possa, de fato, realizar um projeto que envolva eletrônica. Como os pinos de GPIO são utilizados para a comunicação com componentes externos, o leque de possibilidades para projetos é ampliado imensamente.

Os pinos de GPIO do Raspberry Pi Model B são dispostos ao longo de um dos lados da placa, próximos à saída analógica de vídeo. São 26 pinos

Para os testes realizados nesta aula, utilizam-se, como material adicional, um LED, um resistor e jumpers para conectar portas de GPIO. Nos testes feitos durante o desenvolvimento do material para esta aula, foram utilizados um LED vermelho de 25mA e um resistor de 640Ω.

69

A seguir, experimenta-se com o controle digital dos pinos em Python. O Raspberry Pi inclui, em seu pacote de Python, uma biblioteca para controle dos pinos denominada `RPi.GPIO`. Ela fornece uma interface simples para leitura e controle dos pinos. Uma vez conectados os terminais do LED e resistor em um pino de GPIO e um aterrado, deve ser estimulada a exploração dos recursos de controle por parte dos estudantes. Além disso, incluem-se, em anexo, três programas como referência: `blink.py`, `on-off.py` e `morse.py` (todos os programas desta aula encontram-se na Subseção 8.4). O primeiro apenas alterna entre os níveis alto e baixo do pino, fazendo com que o LED pisque; o segundo recebe comandos do teclado para ligar ou desligar o LED; e o terceiro lê caracteres do teclado e os traduz em código Morse. Sugere-se que os estudantes fiquem livres para criar com esta base. Apesar de o feedback promovido pelo LED ser muito simples, o ambiente flexível oferecido por Python possibilita criações interessantes.

Assim, uma vez explorada o controle (escrita) de pinos, deve-se explorar, também, a leitura. Para tanto, seria interessante conectar chaves aos terminais, mas não é absolutamente necessário. Pode-se conectar e desconectar os pinos manualmente caso não haja chaves disponíveis. Novamente, encoraja-se que os estudantes sejam estimulados a explorar. Entretanto, o maior propósito da leitura é a comunicação com dispositivos externos; sem eles, as possibilidades são menos interessantes do que o controle dos pinos. Apesar disso, pode-se trabalhar com funções básicas, como no arquivo de referência `connected.py`. Neste programa, verifica-se se um dos pinos está conectado ou desconectado ao terra. Um exercício interessante seria reescrevê-lo de forma que a verificação seja em relação ao nível alto de tensão (o que requer trocar a configuração de resistor de pull-down para pull-up).

6.2.9 Aula 10: Controle de GPIO com comunicação pela Internet

Esta aula procura expandir o trabalho iniciado na aula anterior. Uma das grandes vantagens no uso do Raspberry Pi é a possibilidade de trabalhar em dois níveis normalmente incompatíveis: a eletrônica de baixo nível e a Internet. Sendo assim, trabalha-se a integração do feedback fornecido pelo

LED com as possibilidades de interação fornecidas pela Internet.

Ainda mais do que na aula anterior, é importante que os estudantes estejam livres para explorar. As únicas ferramentas que ainda não possuem são as de comunicação com a Internet via Python; entretanto, neste momento, devem ser capazes de procurar por bibliotecas e suas respectivas funções por meio da documentação oficial de Python. Pode-se referir os estudantes à biblioteca `urllib2`.

Como referência, incluem-se 3 programas na Subseção 8.5: `ping.py`, `morse_status.py` e `search_string.py`. O primeiro verifica se há conexão com a Internet, por meio de pings a um site conhecido. Caso haja, o LED acende. O segundo acessa uma URL e retorna o status da resposta HTTP em código morse, e o terceiro procura por uma string dada dentro da resposta HTTP a uma URL, acendendo o LED caso a encontre.

6.2.10 Aulas 11 a 15: Projetos com Raspberry Pi

Portando os conhecimentos necessários para utilizar o Raspberry Pi tanto como servidor quanto como controlador para experimentos em eletrônica, é, neste momento, possível aos estudantes realizar projetos de maior complexidade. A ordem escolhida entre este bloco de 5 aulas e o restante do conteúdo adicional colocou os projetos em primeiro lugar, pois dá continuidade ao trabalho prático realizado nas últimas aulas do primeiro bloco. Entretanto, esta ordem pode ser alterada de acordo com as necessidades da disciplina, podendo, inclusive, alocar algumas das aulas seguintes entre o primeiro bloco e este. Apesar disso, é importante manter este bloco em aulas consecutivas.

A razão para a divisão do conteúdo adicional em dois blocos com 5 aulas cada advém da variabilidade do número de aulas em diferentes semestres. Estima-se um total de 15 a 20 semanas de aula em um semestre, e, portanto, a divisão em dois blocos permite que a disciplina se adapte a semestres com diferentes números de semanas letivas (pode ser ministrado nenhum bloco adicional em semestres com muito poucas aulas, um bloco em um semestre com poucas ou os dois em um semestre com muitas).

Estas 5 aulas se destinam à realização de projetos em grupo. Na primeira,

os estudantes devem se separar em grupos, idealmente, de 2 a 4 pessoas (número, entretanto, sujeito à disponibilidade de Raspberry Pi para uso). Para que os estudantes possam trabalhar nos projetos, é necessário que tenham disponíveis os Raspberry Pi e acessórios consigo para trabalho fora de sala de aula. A viabilidade disso deve ser estudada pelo professor e discutida com os estudantes.

Os projetos devem, idealmente, surgir a partir de desejos e propostas dos estudantes. Apesar disso, pode-se também propor alguns dos projetos abaixo, caso seja relevante ou um grupo não apresente uma proposta:

- Monitor de temperatura e/ou umidade. Pode-se registrar valores lidos de sensores por GPIO por dias ou semanas, traçar gráficos, reconhecer padrões, etc.
- Sistema de segurança com câmera. Pode-se conectar uma câmera desenvolvida especificamente para o Raspberry Pi ou transmitir os dados de uma câmera comum via rede. O vídeo pode ser armazenado por um dado período de tempo e então removido. É interessante, também, a possibilidade de acessar o vídeo remotamente em tempo real, via VNC.
- Robô móvel. Pode-se utilizar comunicação via internet para controlar direção e velocidade, ou controle via rádio, com receptor ligado aos pinos GPIO.
- Controlador de display LED. Utilizando os pinos de GPIO, é possível controlar um display de LEDs, utilizando-o como mostrador para diversos tipos de dados: hora, temperatura, velocidade de upload/download, número de acessos a um site, entre outros.
- Interpretador de comandos de voz. Ligando um microfone aos pinos de GPIO e utilizando software de reconhecimento de voz, é possível controlar o Raspberry Pi via comandos de voz.
- Cluster para processamento distribuído. Utilizando mais de um Raspberry Pi em conjunto com software como OpenMPI, pode-se criar um cluster para computação distribuída.

Além dos projetos acima, existe uma enorme gama de projetos criados pela comunidade em torno do Raspberry Pi. Desde babás eletrônicas a impressoras 3D, existem inúmeros recursos disponíveis na Internet para o desenvolvimento de projetos.

6.2.11 Aula 16: Uso de Lua no Raspberry Pi

Esta aula se concentra na construção de uma alternativa no controle dos pinos de GPIO. Existe, além de Python, a possibilidade de utilizar a linguagem de programação Lua para tal tarefa. Os motivos para esta escolha são variados, e foram discutidos em maior detalhe na Seção 1. Entretanto, pode-se justificar o uso de Lua para possibilitar projetos onde o desempenho é mais crítico, além de ser tecnologia nacional.

Esta aula pode ser oferecida de forma complementar à Aula 9 ou em seu lugar, de acordo com a disponibilidade de tempo e a viabilidade de aprender uma nova linguagem de programação, no lugar de Python. Deve-se lembrar, entretanto, que, caso a Aula 9 seja substituída por esta, a Aula 10 também deverá ser readaptada para o uso de Lua. O formato da Aula 9 é mantido quase integralmente, a não ser pela substituição de Python por Lua nos arquivos de referência. Entretanto, como a instalação padrão do sistema operacional Raspbian não contém a linguagem, ela e o módulo para GPIO (obtido em [29]) podem ser instalados executando-se os comandos a seguir.

```
$ sudo apt-get install lua5.1
$ sudo apt-get install liblua5.1-0-dev
$ sudo apt-get install luarocks
$ sudo apt-get install openssl
$ sudo apt-get install libssl-dev
$ sudo luarocks install luarocks
$ sudo luarocks install luasec
$ sudo luarocks install rpi-gpio
```

Sendo assim, o professor deve se referir à Aula 9 para a estrutura da aula

e sequência de atividades. Na Subseção 8.6, encontram-se as versões em Lua dos programas de referência utilizados para aquela aula: `link.py`, `on-off.py` e `morse.py` para o controle de pinos e `connected.py` para a leitura.

Caso a aula seja dada de forma complementar à Aula 9, é provável que refazer os mesmos programas daquela demandem pouco tempo. Nesse caso, podem-se reproduzir os programas da Aula 10, ou mesmo programar algoritmos conhecidos (como *bubble sort*) para que os estudantes obtenham maior familiaridade com a linguagem.

6.2.12 Aulas 17 e 18: Noções de engenharia de software e complexidade algorítmica

Esta aula explora, de maneira superficial devido a limitações de tempo, algumas noções de engenharia de software e complexidade algorítmica. Inicia-se por uma discussão sobre as características que tornam programas eficientes. Abordam-se algumas métricas de desempenho usuais, bem como práticas de programação que possibilitam (ou facilitam) ganhos de desempenho. Finaliza-se a aula com uma exploração de conceitos de complexidade algorítmica, incluindo atividades práticas e notação assintótica.

Pode-se iniciar a aula por meio de uma pergunta norteadora: “O que é um programa eficiente?”. A partir da pergunta, registram-se as impressões prévias dos estudantes. Comumente, a maior parte das respostas centra-se em torno de duas características: velocidade e memória. A primeira delas é a mais usual (um programa bom é aquele que executa rapidamente). A partir das respostas, é possível debater maiores detalhes sobre os tipos de velocidade e memória que são utilizados nas ponderações. Existem velocidades de inicialização e execução; memória pode referir-se a RAM ou a disco rígido. Na Tabela 9, comparam-se os seguintes programas hipotéticos, que realizam exatamente a mesma tarefa, em relação a sua eficiência:

Deve-se estimular os estudantes a oferecer justificativas para suas respostas. É interessante, após algum amadurecimento do debate, no qual se espera que sejam encontradas diferentes opiniões sobre a ordenação de programas em relação a sua eficiência, listar tipos de dispositivos computacionais com

Programa	Tempo de inicialização	Tempo de execução	Uso de memória RAM	Uso de memória de disco
1	10s	45s	2MB	10MB
2	1s	80s	2MB	10MB
3	30s	15s	2MB	10MB
4	20s	35s	500KB	10MB
5	1s	15s	10MB	500MB

Tabela 9: Comparação de desempenho em programas hipotéticos

suas respectivas limitações de hardware. Um servidor ou desktop, em geral, tem amplo espaço de armazenamento em disco rígido e memória RAM, mas o desktop pode ter requerimentos de tempo de inicialização mais estritos devido à interação com o usuário; o servidor pode ter requerimentos em relação a tempo de execução e memória RAM, pois costuma ter muitos processos simultaneamente em execução; um dispositivo embarcado (como pode ser o caso do Raspberry Pi) pode ter requisitos estritos em relação a uso de memória em disco, mas ser maleável em relação a tempo de inicialização e execução.

Uma vez discutidos os programas e as plataformas, conclui-se que a eficiência de um programa está, em geral, relacionada a uma ou poucas métricas de desempenho; assim, cada programa seria mais adequado a um dos tipos de dispositivo mencionados. Um bom exercício seria designar um dos programas a cada tipo de dispositivo. Também é interessante citar a questão de eficiência energética, importante para dispositivos móveis que dependem de bateria. Além dela, outras características também são importantes métricas de desempenho, que, apesar de cruciais para desenvolvedores de software, não são óbvias. Algumas métricas de desempenho adicionais são listadas abaixo (elas serão abordadas em mais profundidade na aula sobre paradigmas de programação):

- Consumo de energia;
- Número de bugs;
- Facilidade de modificação do código;

- Legibilidade do código;
- Facilidade de localização de bugs;
- Tamanho do código;
- Tempo utilizado para o desenvolvimento do programa.

A partir deste ponto, pode-se desviar o eixo da discussão para o contexto de uma pessoa ou grupo de desenvolvedores de software. Métricas de desempenho, a partir desta perspectiva, devem incorporar: o processo de desenvolvimento e manutenção do software; a comunicação entre desenvolvedores sobre o código; a facilidade de transmissão da responsabilidade de manutenção; a possibilidade de modificar e/ou ampliar os recursos oferecidos; o processo de procura e correção de bugs; entre outros.

Após a discussão sobre diferentes métricas de desempenho, o professor pode se restringir somente àquelas que são mais comumente tomadas como principais: velocidade de execução e uso de memória RAM. Para os fins desta aula, é útil, inclusive, focar-se somente em velocidade de execução.

Para ilustrar a questão de eficiência em tempo de execução, utilizam-se alguns programas para ordenação de vetores de números em Python, cujo código foi retirado de [30].

O programa gera, automaticamente, gráficos de tempo de execução em função do tamanho dos vetores, juntamente com uma função ajustada aos valores reais. Na Figura 30, o algoritmo *bubble sort* corresponde à linha azul, *insertion sort* à linha vermelha e *merge sort* à linha preta. Além disso, a Figura 31 mostra cada um dos algoritmos com o valor real (em azul) e a curva correspondente à função de sua complexidade ajustada aos valores reais (em preto). É interessante notar o erro no ajuste de *merge sort*, que, para uma estimativa condizente com as propriedades teóricas, necessitaria de vetores muito maiores.

Neste momento, pode-se iniciar a abordagem ao aspecto teórico, matemático, de complexidade algorítmica. É interessante estimular os estudantes a modificar o código para medir, por exemplo, o tempo gasto por uma função que percorre o vetor e realiza uma operação com cada elemento (por exemplo,

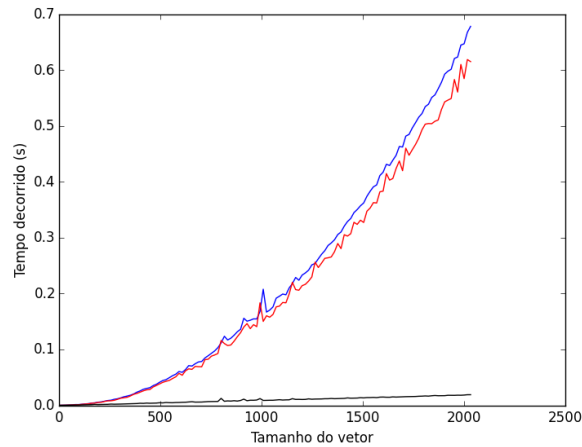


Figura 30: Tempo de execução de *bubble sort*, *insertion sort* e *merge sort*

dobrar ou elevar ao quadrado). Os estudantes, avaliando o resultado, podem propor hipóteses sobre o formato da curva resultante. Então, introduzem-se os últimos conceitos para que seja possível a compreensão plena do conceito de complexidade. É necessário retomar alguns conceitos de cálculo (em particular, limites e suas técnicas de resolução). Pode-se pedir aos estudantes que deem exemplos de funções comumente vistas, e, então, calcular os limites de suas razões. Os principais exemplos são mostrados abaixo.

$$\lim_{n \rightarrow \infty} \frac{n}{n^2} = 0$$

$$\lim_{n \rightarrow \infty} \frac{\log n}{n} = 0$$

$$\lim_{n \rightarrow \infty} \frac{n}{e^n} = 0$$

$$\lim_{n \rightarrow \infty} \frac{e^n}{n!} = 0$$

A partir deles, pode-se aprofundar em alguns exemplos específicos, mostrando que os resultados são independentes de constantes multiplicativas. Somar ou subtrair diversas delas também são exercícios interessantes. Conclui-se, assim, que, para um valor suficientemente grande, as funções dos denominadores sempre superam as dos numeradores. O propósito desta sequência

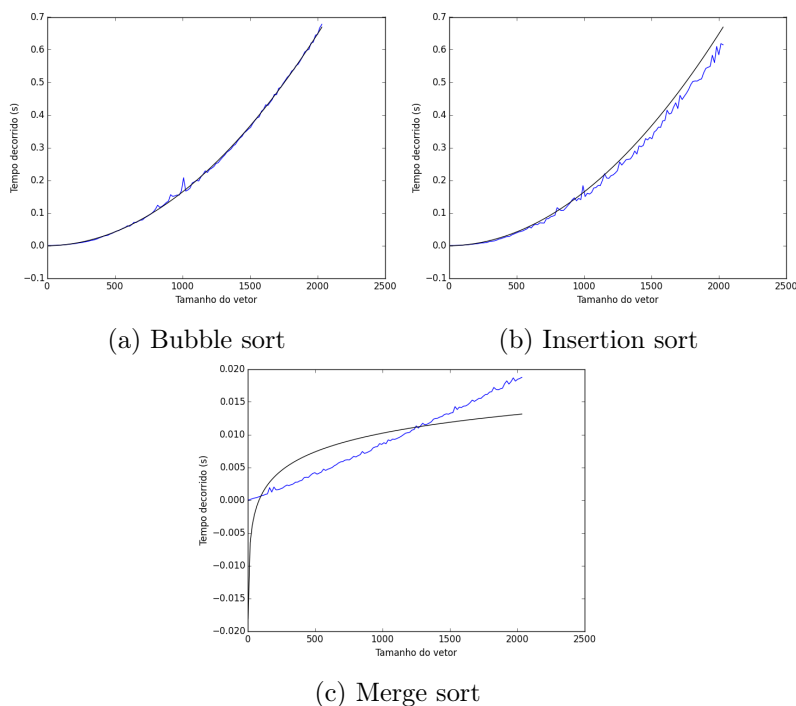


Figura 31: Tempo de execução e função ajustada aos três algoritmos

é trabalhar, com os estudantes, a forma de pensar no comportamento de funções com entradas muito grandes, e ignorar os resultados que não dizem respeito ao termo dominante; para essa análise, basta reconhecer se o limite de uma função é 0, outra constante, ou infinito. O importante é reconhecer o comportamento da função em termos gerais, não em detalhes.

Segue-se, então, com uma discussão sobre a razão por raciocinar com números grandes; o tempo de execução de programas em inputs pequenos é desprezível, e a taxa de crescimento raramente é linear. Assim, faz-se necessário raciocinar de forma diferente do que em analogia com vários programas executados sequencialmente.

Finalmente, com as ferramentas matemáticas e a intuição do problema em mãos, pode-se trabalhar o conceito de complexidade com a notação *big-oh*. Não é necessário aprofundar no desenvolvimento matemático, mas apenas relacionar com os limites citados acima. Da mesma forma, são ignorados

constantes e termos de menor ordem. A complexidade dos algoritmos vistos em aula são resumidas na Tabela 10:

Dobrar ou elevar ao quadrado os elementos do vetor	Bubble sort	Insertion sort	Merge sort
$O(n)$	$O(n^2)$	$O(n^2)$	$O(n \log n)$

Tabela 10: Complexidade algorítmica de alguns algoritmos comuns

Pode-se escrever um ou mais em pseudocódigo, no quadro, e percorrê-lo passo a passo para chegar aos resultados acima (exceto *merge sort*, que requer ferramentas mais avançadas para lidar com recursão). É interessante, também, discutir a razão de *insertion sort* ser mais rápido que *bubble sort*, sendo que, com termos dominantes de mesma ordem, a diferença é dada pelas constantes ou termos de menor ordem.

6.2.13 Aula 19: Linguagens de programação e desempenho

Esta aula segue o mesmo assunto norteador da anterior, porém, por um eixo diferente. O tema continua sendo desempenho, mas, em lugar de uma abordagem teórica, voltada à análise do comportamento geral de um algoritmo, esta parte para uma mais prática. A análise comparativa realizada nesta aula mantém fixo o algoritmo, mas adota a linguagem de programação como variável. Sendo assim, pode-se adquirir uma noção das eficiências das diferentes linguagens de programação disponíveis ao programador, e avaliar os contextos em que é preferível utilizar uma em vez das demais.

Deve-se ressaltar que as análises realizadas nesta aula servem somente para fornecer alguns aspectos da eficiência dos algoritmos. Não se destina a detalhar situações específicas onde uma linguagem pode ser mais eficiente que outra, mas os papéis possam se inverter. Assume-se que, se um algoritmo é mais rápido em uma linguagem em relação a outra, o padrão deve se manter para demais algoritmos e em outros contextos.

Para orientar a análise comparativa, utiliza-se o algoritmo bubble sort para ordenação de vetores de floats no intervalo $[0, 1]$, cujo tamanho varia

entre 100 e 100000 (os vetores estão em arquivos com nome **vecX**, onde **X** é seu tamanho). O algoritmo é implementado em 4 diferentes linguagens de programação: C, Python, Java e Octave, e os respectivos códigos estão na Subseção 8.7. Executam-se os programas em vetores de tamanhos variados, cujos conteúdos estão em arquivos em anexo, juntamente com os códigos dos 4 programas. A Tabela 11 resume os resultados obtidos.

Arquivo	Tempo de execução (ms)			
	C	Java	Python	Octave
vec100	0	0	9	116
vec1000	1	31	326	5817
vec2000	6	5	1173	23091
vec3000	13	11	2587	52168
vec4000	24	23	4817	91881
vec5000	38	31	7130	145170
vec6000	55	49	10848	207010
vec7000	73	72	19222	283970
vec8000	95	96	25465	481460
vec9000	119	128	32861	704570
vec10000	146	166	42792	880840
vec20000	591	835	156255	2974300
vec30000	1317	1935	335606	-
vec40000	2343	3542	639013	-
vec50000	3681	5589	-	-
vec60000	5186	8134	-	-
vec70000	7057	11160	-	-
vec80000	9190	17400	-	-
vec90000	11761	20322	-	-
vec100000	14616	26781	-	-

Tabela 11: Tempo de execução de *bubble sort* em linguagens de programação distintas

Devido ao tempo demandado pelas linguagens mais lentas nos arquivos maiores, o valor não foi registrado. A progressão de linguagens de baixo para linguagens de alto nível é clara: o desempenho é maior naquelas, e a diferença aumenta de acordo com os tamanhos dos vetores. Os resultados são apresentados visualmente na Figura 32.

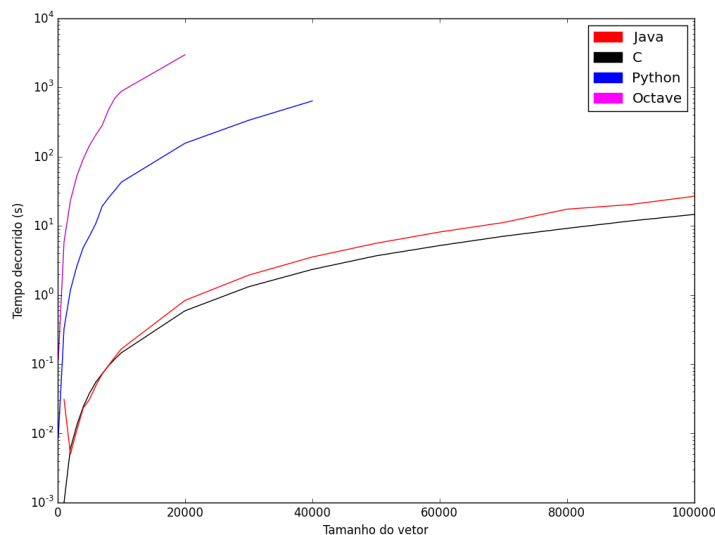


Figura 32: Tempo de execução de *bubble sort* em linguagens de programação distintas

Os resultados, resumidos nos gráficos acima, permitem uma série de observações a respeito da eficiência do algoritmo nas 4 linguagens. É facilmente notável a perda de eficiência incorrida na utilização das linguagens Python e Octave, acentuada nesta. Além disso, é interessante verificar a diferença entre C e Java, cuja tendência relativa de crescimento é perceptível somente em vetores muito grandes.

Munidos de dados empíricos sobre o assunto, pode-se iniciar uma discussão sobre as razões por que as linguagens demandam intervalos de tempo diferentes para realizar uma mesma tarefa. É possível que surjam explicações muito diferentes ou até contraditórias, e debatê-las é um exercício importante para que se compreendam as razões da existência de inúmeras linguagens de programação. O professor pode, então, introduzir o conceito de abstração em linguagens de programação. Pode-se defini-la, de forma grosseira, como um recurso que permite ao programador escrever um programa com menos detalhes, e a linguagem “preencherá os vazios”. É importante trazer

alguns exemplos concretos, como tipagem dinâmica (que permite eliminar declarações de variáveis) e coleta de lixo (que permite eliminar liberações de memória explícitas). Tais abstrações consomem recursos computacionais, o que implica em redução de eficiência.

A discussão pode seguir com comparações de custo, por um lado, de recursos computacionais (demandados em maior medida por programas menos eficientes) e custos de desenvolvimento e manutenção (demandados em maior medida por programas de desenvolvimento complexo). A reflexão parte do custo de horas de trabalho de um programador: pode-se escrever um programa em menos tempo, porém, um que consumirá mais recursos computacionais em execução, ou escrever um programa em mais tempo, com a possibilidade de executar com maior eficiência.

Então, apresentam-se os conceitos de linguagem compilada e interpretada. Linguagens compiladas reduzem grande parte da complexidade de seus programas no momento da compilação, quando traduzem o programa para linguagem de máquina e o otimizam. Assim, o código nunca é executado diretamente, mas somente após tal tradução. Em contraste, linguagens interpretadas executam o código diretamente, linha por linha, em sequência. Isto permite maior flexibilidade de desenvolvimento, mas incorre em redução de eficiência. Das linguagens vistas nesta aula, uma é compilada (C), duas são interpretadas (Python e Octave) e uma é um misto (Java). Pode-se discutir como se aplicariam ambos os modelos de execução em uma mesma linguagem, bem como as motivações para tal e as características de uma linguagem que adote esta abordagem. No caso de Java, por exemplo, portabilidade foi um fator crucial para a decisão de estruturar a linguagem de tal maneira.

A seguir, retoma-se a classificação de linguagens em alto e baixo nível. Neste momento, é possível avaliar linguagens de maneira muito mais detalhada, com posse de dados empíricos e teóricos. Pode-se procurar, em conjunto com os estudantes, por uma definição sucinta de linguagens de alto/baixo nível. Relacionar a “distância ao hardware” dos recursos manipulados na linguagem de programação e o nível de abstração desses recursos é um exercício interessante, pois induz maneiras diferentes de classificar as

linguagens de programação. Essas maneiras, entretanto, resultam em classificações muito similares.

Por último, integram-se as linguagens abordadas nesta aula. Neste momento, os estudantes devem ter noções coerentes das relações entre diferentes linguagens de programação. Além disso, reconhecem as características que podem tornar uma ou outra mais adequada para determinados projetos. Basta, portanto, mostrar que grande parte dos projetos de software mais reconhecidos não utilizam apenas uma linguagem de programação. Como grandes projetos são compostos por blocos, componentes que se comunicam, a escolha da linguagem pode ser, em princípio, feita independentemente para cada um. Sendo assim, uma linguagem de baixo nível pode ser adotada para componentes cujo desempenho é crítico (como processamento de vídeo), e uma de alto nível pode ser adotada para componentes com alta complexidade lógica, cuja minimização de tempo de execução, contudo, não é tão crucial (como interpretação de input do usuário). Tal estrutura pode ser reconhecida inclusive em linguagens de programação, que são, em si, projetos de grande porte. Por exemplo, existe uma implementação de Python em C (CPython), e a linguagem permite, também, a incorporação de rotinas em C.

Finaliza-se com um exercício sobre alguns tipos de projetos de software e as linguagens de programação adequadas para seu desenvolvimento. É importante que os estudantes separem os projetos em componentes, porém, não é necessário que o façam corretamente. O crucial é que sejam capazes de argumentar pela escolha de uma linguagem em detrimento das demais para cada componente de cada projeto. Alguns tipos de projetos de software que podem ser discutidos são:

- Sistema operacional
- Website
- Driver
- Aplicativo móvel

- Interface gráfica
- Compilador

6.2.14 Aula 20: Paradigmas de programação

Esta aula tem o intuito de expor os padrões de raciocínio pelos quais se acostuma a programar, mostrando seu caráter convencional, além de alternativas ao mesmo. Em cursos de linguagens de programação, é comum uma incursão em alguma linguagem de programação não imperativa com este propósito em mente. É importante, para estudantes que se proponham a entender como são estruturadas as ferramentas que utilizam, remover características não essenciais (mas tomadas como essenciais) para programar mais conscientemente.

Para ilustrar diferentes maneiras de programar, produz maior efeito mostrá-las antes de analisar a maneira usual. Para tanto, podem-se mostrar exemplos de programas que fujam ao padrão esperado. Abaixo, dispõem-se dois programas simples, um em Ocaml (linguagem funcional), obtido em [31] e um em Prolog (linguagem declarativa), obtido em [32].

OCaml	Prolog
<pre>let rec at k = function [] ->None h :: t ->if k = 1 then Some h else at (k-1) t;;</pre>	<pre>likes(mary,food). likes(mary,wine). likes(john,wine). likes(john,mary). A seguir, perguntam-se: ?- likes(mary,food). yes. ?- likes(john,wine). yes. ?- likes(john,food). no.</pre>

Neste momento, discute-se o que cada programa é destinado a fazer, e aquilo que se percebe diferente dos programas vistos até agora. É interessante focar naquilo que se percebe como estranho, e, a partir daí, gerar um debate

sobre o que pareceria estranho, por exemplo, nas linguagens C ou Java para alguém que só conheça OCaml.

Com uma percepção do que configura alguns diferentes paradigmas de programação, pode-se já apresentar os principais e descrevê-los brevemente:

- **Funcional:** paradigma que tem relação próxima com a área da matemática conhecida como cálculo lambda. O paradigma foca a imutabilidade (impossibilidade de mudança de valores de variáveis) e o tratamento de funções como elementos básicos, manipuláveis, da linguagem. O tratamento de funções desta forma proporciona um enorme potencial de reuso de código, por meio de funções de ordem superior (funções que agem sobre funções). OCaml é um exemplo de linguagem que incorpora o paradigma.
- **Declarativa:** paradigma que permite ao programador apenas declarar as propriedades de um programa. A linguagem, então, manipula as propriedades para, possivelmente, deduzir outras. Prolog é um exemplo de linguagem que incorpora o paradigma. Linguagens para configuração (como originou-se Lua) costumam utilizar este paradigma.
- **Imperativa:** paradigma que utiliza sentenças para alterar o programa de um estado para o seguinte. É o paradigma mais conhecido nas linguagens de programação populares (como C e Java). Caracteriza-se pela declaração e mutação de variáveis ao longo da execução do programa.
- **Orientada a objeto:** paradigma que modulariza seções de código em partes comunicáveis e relacionadas entre si (objetos). Proporciona grande potencial de reuso de código por meio de estruturas de herança entre objetos, que retêm as características dos objetos dos quais herdaram com poucas linhas de código. É comumente utilizada em desenvolvimento de software em grandes equipes, cujos membros podem trabalhar em uma seção específica com maior independência das demais. C++ e Java são as linguagens mais populares que incorporam este paradigma.

Como o objetivo desta aula não é aprofundar em nenhum dos paradigmas, mas sim fazer um apanhado geral de formas diferentes de se programar, existem diversas formas para se prosseguir. Pode-se, por exemplo, examinar alguns programas funcionais para entender melhor a forma como funções são definidas e manipuladas; outra alternativa é introduzir alguns exemplos de código de C++ ou Java, juntamente com diagramas representativos das relações entre objetos. Entretanto, a alternativa que se segue neste roteiro é uma discussão sobre situações-problema onde se avalia a utilidade de incorporação dos paradigmas para suas resoluções. Devido ao tempo limitado de uma aula para o assunto, é provável que os estudantes não tenham sequer as noções necessárias para discutir as situações-problema sob a perspectiva de paradigmas de programação. Assim, pode-se considerar estender esta aula em duas, mas acredita-se que, para o tempo de uma aula, a reflexão é mais válida que as demais alternativas. Assim, discutem-se os paradigmas adequados a:

- Programas para processamento distribuído, destinados a altos volumes de dados. Pode-se apresentar o algoritmo Map-Reduce, juntamente com exemplos de código de uma linguagem funcional que o implemente.
- Programas para geração de terreno em jogos;
- Programas com perspectiva de número alto de atualizações futuras;
- Programas com altos requisitos de eficiência;
- Programas de alta complexidade lógica;
- Programas com altos custos de atualização.

Ao final, podem-se apresentar os recursos para programação funcional em Python. Alguns exemplos são mostrados no quadro abaixo:

```
list(map(lambda x: x * x, [1, 2, 3, 4]))
list(zip([1, 2, 3, 4], [2, 4, 6, 8]))
list(filter(lambda x: x % 2 == 0, [1, 2, 3, 4, 5, 6]))
```

7 Conclusão

Este trabalho partiu de uma necessidade reconhecida do curso de engenharia mecatrônica na EESC-USP: o uso de Linux em um ou mais cursos de graduação. Tal necessidade justifica-se por uma série de fatores, que, resumidamente, aglutinam-se em dois: a relevância de Linux no contexto profissional de um engenheiro mecatrônico e a falta de cursos que desenvolvessem habilidades em computação nesta graduação.

Identificada a necessidade, partiu-se a uma busca por maneiras viáveis de introdução do conteúdo. Analisadas diversas possibilidades, optou-se por uma disciplina de graduação obrigatória de um crédito como modelo, porque o número de horas demandado e a garantia de participação garantiriam a eficácia. Além disso, existem disciplinas flexíveis que se encaixam nesses requisitos, com possibilidade de utilização do material produzido neste trabalho em breve.

Neste contexto, é possível estender o desenvolvimento de habilidades em Linux também para aplicações práticas de interesse de um engenheiro mecatrônico. Com isso, na conjuntura atual, o Raspberry Pi oferece uma plataforma de desenvolvimento que utiliza Linux para projetos embarcados, com desafios que um estudante deve encontrar em sua atuação profissional. Sendo assim, o Raspberry Pi foi escolhido como plataforma.

A seguir, detalham-se os conteúdos abordados, levando em consideração o modelo de apresentação escolhido. Para adaptá-los ao modelo, que fornece um número de horas maior do que o necessário para o trabalho de todos os conceitos essenciais, dividiu-se o conteúdo em prioritário e adicional. O conteúdo adicional serve ao propósito de discutir temas em computação, que, apesar de importantes, não influenciam diretamente o eixo principal da disciplina. Além disso, com a organização em módulos menores autocontidos, também se propicia maior flexibilidade em relação ao número exato de aulas em um semestre, que pode variar significativamente.

Finalmente, dispõem-se em sequência os planos de aula, que delineam

minuciosamente os conteúdos, conceitos, perguntas, exercícios e reflexões sugeridos para o desenvolvimento das habilidades que os estudantes devem adquirir ao fim do curso. Estas envolvem desde o uso de comandos básicos em Linux a projetos de eletrônica com o Raspberry Pi, tangenciando, inclusive, conceitos avançados de linguagens de programação e engenharia de software, caso o conteúdo adicional seja trabalhado.

Acredita-se que este trabalho fornece uma fundação sólida para o professor que deseje introduzir Linux com aplicações práticas e condizentes com os tipos de desafios que os estudantes devem enfrentar. A progressão dos conceitos segue uma sequência lógica, com exemplos, gráficos e tópicos para discussão, além de oferecer flexibilidade com aulas autocontidas de acordo com a disponibilidade de tempo e com o interesse em tópicos adicionais.

Entretanto, há pleno espaço para melhoras ao material, principalmente naquele direcionado ao estudante. Os planos de aula fornecem referências ao professor, mas não são adequadas aos estudantes. Há apenas uma apresentação de slides na qual o professor pode se basear para desenvolver as seguintes. Além de slides, também há espaço para tutoriais, vídeos, exemplos de programas, gráficos, entre outros.

Outra extensão deste projeto de disciplina, que pode ser adaptada sem grandes diferenças, é aquela direcionada a outro público, além de estudantes de engenharia mecatrônica. O curso pressupõe certas habilidades básicas (como, por exemplo, noções de programação), mas uma readaptação do mesmo pode situá-lo em diferentes contextos. Uma das possibilidades que, inclusive, foi considerada dentre os modelos de apresentação do conteúdo, é a de reduzi-lo a um curso de curta duração (cerca de 8h), que pode ser ministrado em um fim de semana. Um trabalho futuro que se pretende seguir, apesar de demandar uma reestruturação profunda deste projeto, é a atuação com crianças e adolescentes.

Também é interessante considerar a possibilidade de oferecer o conteúdo completo (proritário e adicional) em maiores detalhes, o que requer mais de um crédito. Tópicos como complexidade algorítmica e paradigmas de programação (os quais, individualmente, certamente demandam mais de um crédito para uma abordagem mais aprofundada) poderiam ser trabalhados

com mais tempo, além de existir a possibilidade de introduzir uma ou mais linguagens de programação (tal como Lua) para exercitar os conceitos referentes a computação.

Para que se mantenha atualizado, este projeto de disciplina deve ser adaptado a dispositivos mais modernos que superem o Raspberry Pi no nicho em que ocupa. Uma alternativa que desponta é o BeagleBone Black, mas, com a popularidade do Raspberry Pi, não seria surpreendente o surgimento de dispositivos nele inspirados.

Referências

- [1] NetMarketShare. <http://www.netmarketshare.com/>. Data de acesso: 22/11/2014.
- [2] TOP500 Supercomputer Sites. <http://www.top500.org/>. Data de acesso: 22/11/2014.
- [3] CS197U: A Hands-on Introduction to UNIX. <http://people.cs.umass.edu/~jddevaughn/csc197u/>. Data de acesso: 22/11/2014.
- [4] Introduction to UNIX. www.doc.ic.ac.uk/~wjk/UnixIntro. Data de acesso: 22/11/2014.
- [5] Introduction to UNIX. http://sci.informatik.uni-kl.de/rechnerzugang/unix/unix_book.pdf. Data de acesso: 22/11/2014.
- [6] Practical UNIX. <http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=PracticalUnix>. Data de acesso: 22/11/2014.
- [7] UNIX Tools. www.cs.nyu.edu/~mohri/unix08/. Data de acesso: 22/11/2014.
- [8] ROS. <http://www.ros.org/about-ros/>. Data de acesso: 22/11/2014.
- [9] Raspberry Pi. <http://www.raspberrypi.org/>. Data de acesso: 22/11/2014.
- [10] Lua. <http://www.lua.org/portugues.html>. Data de acesso: 22/11/2014.
- [11] Argus. <http://arg.us.com/>. Data de acesso: 22/11/2014.
- [12] RainCloud. <http://ecultivar.com/rain-cloud-product-project/>. Data de acesso: 22/11/2014.

- [13] Sphinx. <http://dock2office.com/sphinx/>. Data de acesso: 22/11/2014.
- [14] YaCyPi. <http://www.yacypi.org/>. Data de acesso: 22/11/2014.
- [15] PiCube. <http://www.firstviewds.com/picube/>. Data de acesso: 22/11/2014.
- [16] PiWall. <http://www.piwall.co.uk/>. Data de acesso: 22/11/2014.
- [17] OTTO. <http://www.nexttthing.co/otto/>. Data de acesso: 22/11/2014.
- [18] Raspberry Pi Compute Module. <http://www.raspberrypi.org/raspberry-pi-compute-module-new-product/>. Data de acesso: 22/11/2014.
- [19] BeagleBone Black. <http://www.beagleboard.org/black>. Data de acesso: 22/11/2014.
- [20] Banana Pi. <http://www.bananapi.org/p/product.html>. Data de acesso: 22/11/2014.
- [21] Odroid-U2. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G135341370451. Data de acesso: 22/11/2014.
- [22] Download de sistemas operacionais para Raspberry Pi. www.raspberrypi.org/downloads/. Data de acesso: 22/11/2014.
- [23] Documentação oficial para instalação. <http://www.raspberrypi.org/documentation/installation/installing-images/>. Data de acesso: 22/11/2014.
- [24] Tutorial: How to Set Up a Raspberry Pi Web Server. <https://www.jeremymorgan.com/tutorials/raspberry-pi/how-to-raspberry-pi-web-server/>. Data de acesso: 22/11/2014.

- [25] Build a file server with the Raspberry Pi. <http://www.linuxuser.co.uk/tutorials/build-a-file-server-with-the-raspberry-pi>. Data de acesso: 22/11/2014.
- [26] Raspberry Pi Owncloud (dropbox clone). <http://www.instructables.com/id/Raspberry-Pi-Owncloud-dropbox-clone/>. Data de acesso: 22/11/2014.
- [27] how to install xbmc + transmission on raspberry pi. <http://eraldmariano.com/how-to-install-xbmc-transmission-on-raspberry-pi/>. Data de acesso: 22/11/2014.
- [28] Raspberry Pi Tor relay. <http://www.instructables.com/id/Raspberry-Pi-Tor-relay/>. Data de acesso: 22/11/2014.
- [29] rpi-gpio. <https://github.com/Tieske/rpi-gpio/>. Data de acesso: 22/11/2014.
- [30] Basic Sorting Algorithms Implemented in Python. <http://danishmujeeb.com/blog/2014/01/basic-sorting-algorithms-implemented-in-python>. Data de acesso: 22/11/2014.
- [31] 99 Problems (solved) in OCaml. <http://ocaml.org/learn/tutorials/99problems.html>. Data de acesso: 22/11/2014.
- [32] Some simple Prolog Examples. <http://www.cs.toronto.edu/~sheila/384/w11/simple-prolog-examples.html>. Data de acesso: 22/11/2014.

8 Anexos

8.1 Anexo 1: Divisão de disciplinas em áreas

Código	Nome	Créditos aula	Créditos trabalho
FCM0101	Física I	6	0
FFI0180	Laboratório de Física Geral I	2	0
SMA0300	Geometria Analítica	4	0
SMA0301	Cálculo I	6	0
SQM0405	Química Geral e Experimental	4	1
FCM0102	Física II	6	0
FFI0181	Laboratório de Física Geral II	2	0
SMA0304	Álgebra Linear	4	0
SMA0332	Cálculo II	6	0
SMA0333	Cálculo III	4	0
SME0301	Métodos Numéricos para Engenharia I	3	0
SME0340	Equações Diferenciais Ordinárias	4	0
SME0302	Métodos Numéricos para Engenharia II	3	0
SME0320	Estatística I	4	0

Tabela 12: Disciplinas classificadas em *Ciclo básico*

Código	Nome	Créditos aula	Créditos trabalho
SEM0564	Desenho Técnico Mecânico I	4	0
SEM0500	Estática Aplicada às Máquinas	4	0
SEM0565	Desenho Técnico Mecânico II	2	1
SEM0388	Princípios de Metrologia Industrial	4	1
SEM0501	Dinâmica Aplicada às Máquinas	4	0
SET0183	Mecânica dos Sólidos I	4	0
SMM0193	Engenharia e Ciência dos Materiais I	4	0
SEM0403	Fundamentos da Mecânica dos Fluidos	4	0
SET0184	Mecânica dos Sólidos II	4	0
SMM0194	Engenharia e Ciência dos Materiais II	4	0
SEM0104	Mecanismos	4	0
SEM0233	Termodinâmica I	4	0
SEM0241	Elementos de Máquinas I	4	1
SEM0172	Dinâmica das Máquinas	4	0
SEM0534	Processos de Fabricação Mecânica	3	0
SEM0550	Transferência de Calor e Massa	4	1
SEM0397	Modelagem e Simulação de Sistemas Térmicos	3	0

Tabela 13: Disciplinas classificadas em *Mecânica*

Código	Nome	Créditos aula	Créditos trabalho
SEL0403	Elettricidade I	4	0
SEL0203	Princípios de Eletrônica	4	0
SEL0405	Introdução aos Sistemas Digitais I	4	0
SEL0336	Aplicação de Microprocessadores I	2	0
SEL0404	Elettricidade II	4	0

Tabela 14: Disciplinas classificadas em *Elétrica/Eletrônica*

Código	Nome	Créditos aula	Créditos trabalho
SCC0120	Introdução à Ciência da Computação	2	2
SSC0300	Linguagens de Programação e Aplicações	2	0
SEL0415	Introdução à Organização de Computadores	2	0
SEM0543	Organização de Dados em Computadores	2	0
SEM0546	Engenharia de Software	2	0

Tabela 15: Disciplinas classificadas em *Computação*

Código	Nome	Créditos aula	Créditos trabalho
SEM0528	Introdução à Engenharia Mecatrônica	1	1
SEM0529	Problemas de Engenharia Mecatrônica I	1	1
SEM0530	Problemas de Engenharia Mecatrônica II	1	1
SEM0531	Problemas de Engenharia Mecatrônica III	1	1
SEM0532	Problemas de Engenharia Mecatrônica IV	1	1
SEM0537	Problemas de Engenharia Mecatrônica V	1	1
SEM0533	Modelagem e Simulação de Sistemas Dinâmicos I	4	0
SEM0535	Modelagem e Simulação de Sistemas Dinâmicos II	4	0
SEM0536	Sistemas de Controle I	5	1
SEM0538	Sistemas de Controle II	4	0
SEM0539	Instrumentação e Sistemas de Medidas	4	1
SEM0540	Elementos de Automação	4	0
SEM0541	Projeto de Sistemas Mecatrônicos I	4	1
SEM0317	Dinâmica e Controle de Sistemas Robóticos I	4	0
SEM0320	Interfaces Eletromecânicas	4	0
SEM0542	Projeto de Sistemas Mecatrônicos II	4	1
SEM0544	Desenvolvimento de Produtos Mecatrônicos	4	1
SEM0545	Sistemas Microeletromecânicos	4	0

Tabela 16: Disciplinas classificadas em *Interdisciplinares*

Código	Nome	Créditos aula	Créditos trabalho
IAU0126	Humanidades e Ciências Sociais	2	0
SHS0623	Gestão Ambiental para Engenheiros	2	0
SEP0527	Gestão e Organização	2	0
SEP0587	Princípios de Economia	2	0
SEM0398	Estágio Supervisionado	2	6
SEM0399	Trabalho de Conclusão de Curso I	2	4
SEM0404	Trabalho de Conclusão de Curso II	2	4
SEP0171	Gerenciamento de Projetos	3	0

Tabela 17: Disciplinas classificadas em *Outras*

8.2 Anexo 2: Slides da Aula 1

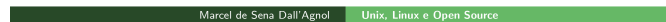


Unix, Linux e Open Source

Aula 1

Marcel de Sena Dall'Agnol

Universidade de São Paulo



História do UNIX

Desenvolvido na AT&T Bell Labs por Dennis Ritchie e Ken Thompson

Lançado em 1970

Escrito em Assembly, em 1973 **reescrito em C**

Difusão no fim da década de 70 e começo de 80

UNIX é marca registrada:

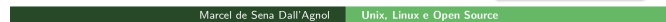
- UNIX: Mac OS X
- UNIX-like: **Linux**



Dennis Ritchie



Ken Thompson



História

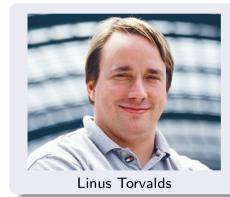
Criado em 1991 por Linus Torvalds, estudante na Universidade de Helsinki

Baseado em MINIX, o qual era baseado em UNIX

GNU Project: sistema operacional livre

GNU Hurd: **Kernel** incompleto e inacessível

Linus programou especificamente para o hardware que tinha, pois queria utilizar funções de seu PC com processador 386.



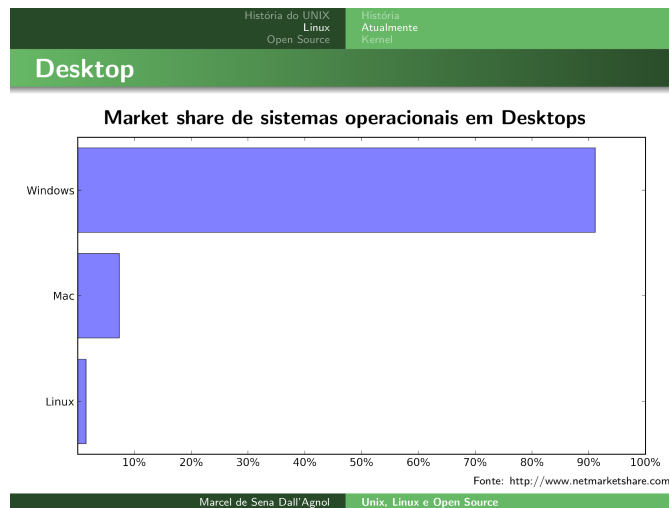
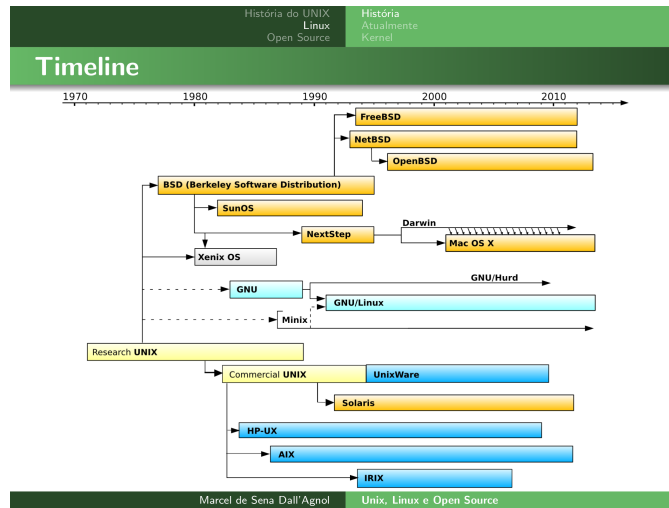
Linus Torvalds

História

Em agosto de 1991, no Usenet newsgroup "comp.os.minix.":

*Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) [...]. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat [...].
[...]
PS. Yes - it's free of any minix code, [...] and it probably never will support anything other than AT-harddisks, as that's all I have :-(.
—Linus Torvalds*

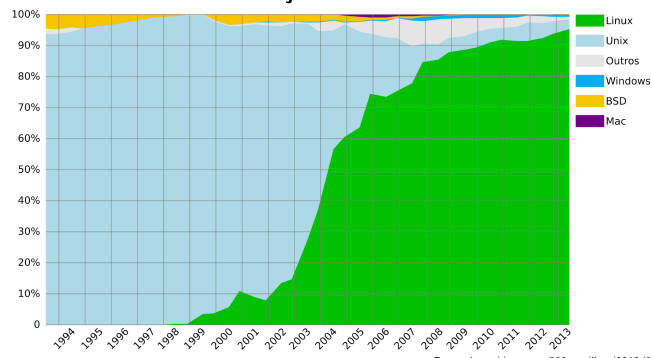




Distribuições Linux (comumente chamadas **distros**) são sistemas operacionais que utilizam o kernel Linux e outros pacotes de software. Estes pacotes incluem bibliotecas e ferramentas do GNU Project, outros software livres, e podem conter também código proprietário.

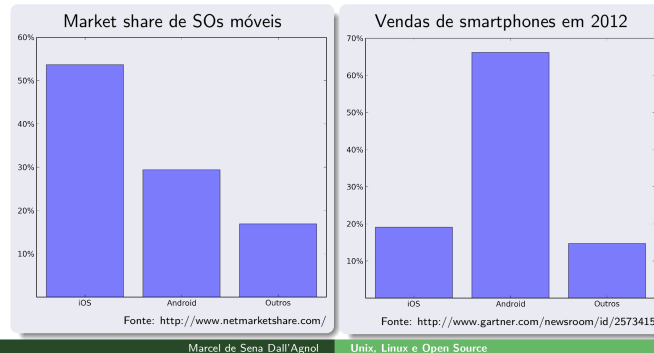


Projeto TOP500



Android

Android é um sistema operacional móvel baseado no kernel Linux, voltado principalmente para dispositivos com tela touchscreen.



Marcel de Sena Dall'Agnol

Unix, Linux e Open Source

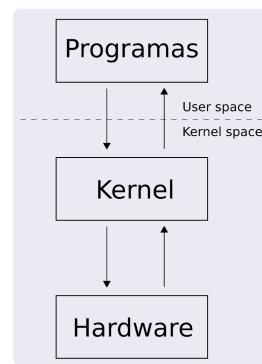
Kernel

Kernel (núcleo) é a parte do sistema operacional responsável pela comunicação com o hardware.

Estável, seguro e sem bugs

Níveis de privilégio e **permissões**

Windows também tem (Windows NT kernel)



Marcel de Sena Dall'Agnol

Unix, Linux e Open Source

História

1983: Richard Stallman inicia o **GNU Project**.

1986: funda a Free Software Foundation.

1989: a primeira versão da **GNU General Public License** (GPL) é lançada.

1998: o código fonte do Netscape Communicator é aberto, no qual o navegador **Firefox** é baseado.

2003: funda-se a Mozilla Foundation, originada do Netscape.



Richard Stallman

Software Livre vs. Open Source

Free Software:

Questão de liberdade, não de preço (*Free as in freedom*).
Definição de Richard Stallman, adotada pela Free Software Foundation.

Open Source Software:

Termo definido pela Open Source Initiative. Criado para dissociar a conotação ideológica do anterior e remover ambiguidade.
"Programa de marketing para free software".

FOSS/FLOSS:

Free (Libre) Open Source Software. Termo criado para abarcar ambos.

A grid of 12 logos for various open-source projects, arranged in three rows and four columns. The logos are: Row 1: Android (green robot), Firefox (orange and red flame), Next.js (orange hand), Nginx (blue circle), GIMP (grey cat), CPython (blue circle), VLC (orange cone), edX (pink and blue text), LibreOffice (grey document icon), Pylons (purple penguin), Django (orange circle), and MySQL (blue cube). Row 2: MySQL (blue cube), Django (orange circle), Pylons (purple penguin), LibreOffice (grey document icon), edX (pink and blue text), VLC (orange cone), CPython (blue circle), GIMP (grey cat), Next.js (orange hand), Firefox (orange and red flame), Android (green robot), and Nginx (blue circle).

Marcel de Sena Dall'Agnol | Unix, Linux e Open Source

Esquemáticos open source: **Arduino** e **Raspberry Pi**.

Open source blueprints for civilization

Impressão 3D: Seção "Phisibles" no PirateBay

Open Hardware Repository e CERN Open Hardware License

The image displays two open-source hardware schematics. The top schematic is for the Arduino Uno Rev3, showing the ATmega328P microcontroller, USB interface, and various peripheral components. The bottom schematic is for the Raspberry Pi modelo B, showing the Broadcom BCM2835 SoC, memory, and other components. Both are presented as detailed circuit diagrams with component labels and connection lines.

Marcel de Sena Dall'Agnol | Unix, Linux e Open Source

8.3 Anexo 3: Programas da Aula 6

```
                                bubblesort.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[])
{
    int n, i, j, temp;
    int *v;

    n = argc - 1;
    v = (int*) malloc(n * sizeof(int));
    for(i = 0; i < n; i++)
        v[i] = atoi(argv[i + 1]);

    for(i = 0; i < n - 1; i++)
        for(j = i + 1; j < n; j++)
            if(v[i] > v[j])
            {
                temp = v[i];
                v[i] = v[j];
                v[j] = temp;
            }

    printf("[");
    for(i = 0; i < n - 1; i++)
        printf("%d ", v[i]);
    printf("%d\n", v[i]);
}
```

bubblesort.py

```
import sys

n = len(sys.argv) - 1;
v = [int(a) for a in sys.argv[1:]]

for i in range(n - 1):
    for j in range(i + 1, n):
        if v[i] > v[j]:
            temp = v[i];
            v[i] = v[j];
            v[j] = temp;
print(v)
```

8.4 Anexo 4: Programas da Aula 9

blink.py

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, False)

state = True
for i in range(10):
    GPIO.output(pin, state)
    time.sleep(1)
```

```
    state = not state
GPIO.cleanup()
```

connected.py

```
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
pin = 3
GPIO.setup(pin, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

if(GPIO.input(pin) == 0):
    print('Pino conectado.')
else:
    print('Pino desconectado.')
GPIO.cleanup()
```

morse.py

```
import RPi.GPIO as GPIO
import sys
import time

GPIO.setmode(GPIO.BOARD)
pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, False)

short_beep = 0.5
long_beep = 3 * short_beep
```

```

def morse(character):
    GPIO.output(pin, True)
    morse_code = {'a': 'sl',
                  'b': 'lsss',
                  'c': 'lsls',
                  'd': 'lss',
                  'e': 's',
                  'f': 'ssls',
                  'g': 'lls',
                  'h': 'ssss',
                  'i': 'ss',
                  'j': 'slll',
                  'k': 'lsl',
                  'l': 'slss',
                  'm': 'll',
                  'n': 'ls',
                  'o': 'lll',
                  'p': 'slls',
                  'q': 'llsl',
                  'r': 'sls',
                  's': 'sss',
                  't': 'l',
                  'u': 'ssl',
                  'v': 'sssl',
                  'w': 'sll',
                  'x': 'lssl',
                  'y': 'lsll',
                  'z': 'llss'}

    if character in morse_code:
        for beep in morse_code[character]:
            GPIO.output(pin, True)

```



```

        if(beep == 's'):
            time.sleep(short_beep)
        if(beep == 'l'):
            time.sleep(long_beep)
        GPIO.output(pin, False)
        time.sleep(short_beep)

print 'Digite uma letra do alfabeto: '
character = sys.stdin.read(1).lower()
morse(character)
GPIO.cleanup()

```

on-off.py

```

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BOARD)
pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, False)

for i in range(10):
    inp = raw_input('Digite 1 para ligar ou 0 para desligar: ')
    if(inp == '1'):
        GPIO.output(pin, True)
    if(inp == '0'):
        GPIO.output(pin, False)
GPIO.cleanup()

```

```

                                morse_status.py

import RPi.GPIO as GPIO
import urllib2
import time

GPIO.setmode(GPIO.BOARD)
pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, False)

short_beep = 0.3
long_beep = 3 * short_beep
interval = 6 * short_beep

def morse(number):
    GPIO.output(pin, False)
    morse_code = {'0': '11111',
                  '1': 's1111',
                  '2': 'ss111',
                  '3': 'sss11',
                  '4': 'ssssl',
                  '5': 'sssss',
                  '6': 'lssss',
                  '7': 'llsss',
                  '8': 'lllss',
                  '9': 'lllls'}

    if number in morse_code:
        for beep in morse_code[number]:
            GPIO.output(pin, True)

```

```

        if(beep == 's'):
            time.sleep(short_beep)
        if(beep == 'l'):
            time.sleep(long_beep)
        GPIO.output(pin, False)
        time.sleep(short_beep)

site = raw_input('Digite o endereco que deseja acessar: ')
try:
    code = str(urllib2.urlopen(site).getcode())
    for number in code:
        morse(number)
        time.sleep(interval)
except urllib2.HTTPError as error:
    code = str(error.code)
    for number in code:
        morse(number)
        time.sleep(interval)
except:
    GPIO.output(pin, False)
GPIO.cleanup()

```

ping.py

```

import RPi.GPIO as GPIO
import urllib2
import time

GPIO.setmode(GPIO.BOARD)
pin = 3
GPIO.setup(pin, GPIO.OUT)

```

```

GPIO.output(pin, False)

site = raw_input('Digite o endereco que deseja acessar: ')
try:
    urllib2.urlopen(site, timeout = 5)
    GPIO.output(pin, True)
except:
    GPIO.output(pin, False)
time.sleep(3)
GPIO.cleanup()

```

search_string.py

```

import RPi.GPIO as GPIO
import urllib2
import time

GPIO.setmode(GPIO.BOARD)
pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, False)

site = raw_input('Digite o endereco em que deseja procurar: ')
search = raw_input('Digite a string a ser procurada: ')
try:
    content = urllib2.urlopen(site).read()
    if content.find(search) != -1:
        GPIO.output(pin, True)
except:
    GPIO.output(pin, False)
time.sleep(3)

```

```
GPIO.cleanup()
```

8.6 Anexo 6: Programas da Aula 16

blink.lua

```
local GPIO = require "GPIO"

function sleep(n)
    os.execute("sleep " .. tonumber(n))
end

GPIO.setmode(GPIO.BOARD)
local pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, GPIO.LOW)

local state = true
for i = 1,10 do
    GPIO.output(pin, state)
    sleep(1)
    state = not state
end
GPIO.cleanup()
```

connected.lua

```
local GPIO = require "GPIO"

GPIO.setmode(GPIO.BOARD)
```

```

local pin = 3
GPIO.setup(pin, GPIO.IN)

if(GPIO.input(pin)) then
    print('Pino desconectado.')
else
    print('Pino conectado.')
end
GPIO.cleanup()

```

on-off.lua

```

local GPIO=require "GPIO"
local io = require "io"

GPIO.setmode(GPIO.BOARD)
local pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, GPIO.LOW)

local inp
for i = 1,10 do
    io.write('Digite 1 para ligar ou 0 para desligar: ')
    inp = io.read()
    if(inp == '1') then
        GPIO.output(pin, GPIO.HIGH)
    end
    if(inp == '0') then
        GPIO.output(pin, GPIO.LOW)
    end
end
end

```

```
GPIO.cleanup()
```

morse.lua

```
local GPIO=require "GPIO"
local io = require "io"

function sleep(n)
    os.execute("sleep " .. tonumber(n))
end

GPIO.setmode(GPIO.BOARD)
local pin = 3
GPIO.setup(pin, GPIO.OUT)
GPIO.output(pin, GPIO.LOW)

local short_beep = 0.5
local long_beep = 3 * short_beep

function morse(character)
    GPIO.output(pin, GPIO.HIGH)
    local morse_code = {a = 'sl',
                        b = 'lsss',
                        c = 'lsls',
                        d = 'lss',
                        e = 's',
                        f = 'ssls',
                        g = 'lls',
                        h = 'ssss',
                        i = 'ss',
                        j = 'slll',
```

```

        k = 'ls1',
        l = 'slss',
        m = 'll',
        n = 'ls',
        o = 'lll',
        p = 'slls',
        q = 'llsl',
        r = 'sls',
        s = 'sss',
        t = 'l',
        u = 'ssl',
        v = 'sssl',
        w = 'sll',
        x = 'lssl',
        y = 'lsll',
        z = 'llss'}
if morse_code[character] then
  for i = 1, morse_code[character]:len() do
    GPIO.output(pin, GPIO.HIGH)
    if(morse_code[character]:sub(i,i) == 's') then
      sleep(short_beep)
    end
    if(morse_code[character]:sub(i,i) == 'l') then
      sleep(long_beep)
    end
    GPIO.output(pin, GPIO.LOW)
    sleep(short_beep)
  end
end
end
end

```



```

io.write('Digite uma letra do alfabeto: ')
character = io.read(1)
morse(character)
GPIO.cleanup()

```

8.7 Anexo 7: Programas da Aula 19

bubblesort.c (C)

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int main(int argc, char* argv[])
{
    int f, i, j, size;
    long int times[20], start, end;
    float vector[100000], temp;
    char* filenames[21] = {"vec100", "vec1000", "vec2000",
                           "vec3000", "vec4000", "vec5000",
                           "vec6000", "vec7000", "vec8000",
                           "vec9000", "vec10000", "vec20000",
                           "vec30000", "vec40000", "vec50000",
                           "vec60000", "vec70000", "vec80000",
                           "vec90000", "vec100000"};

    char buffer[100];
    FILE* file;
    printf("Runtimes: \n");
    for(f = 0; f < 20; f++)

```

```

{
    file = fopen(filenamees[f], "r");
    for(size = 0; !feof(file); size++)
        atoi(fgets(buffer, 100, file));
    fclose(file);
    start = clock();
    for(i = 0; i < size - 1; i++)
        for(j = i + 1; j < size; j++)
            if(vector[i] > vector[j])
            {
                temp = vector[i];
                vector[i] = vector[j];
                vector[j] = temp;
            }
    end = clock();
    times[f] = (1000 * (end - start)) / CLOCKS_PER_SEC;
    printf("%d\n", times[f]);
}
}

```

bubblesort.java (Java)

```

import java.io.*;
import java.util.Date;

public class bubblesort
{
    public static void main(String []args) throws
                                FileNotFoundException,
                                IOException
    {

```

```

int f, i, j, size;
float temp;
long start, end;
long[] times = new long[20];
float[] vector = new float[100000];
String[] filenames = {"vec100", "vec1000", "vec2000",
                      "vec3000", "vec4000", "vec5000",
                      "vec6000", "vec7000", "vec8000",
                      "vec9000", "vec10000", "vec20000",
                      "vec30000", "vec40000", "vec50000",
                      "vec60000", "vec70000", "vec80000",
                      "vec90000", "vec100000"};

String buffer;
BufferedReader br;

System.out.println("Runtimes: ");
for(f = 0; f < 20; f++)
{
    br = new BufferedReader(new InputStreamReader(new
                                                FileInputStream(filenames[f])));
    for(size = 0; (buffer = br.readLine()) != null; size++)
        vector[size] = Float.parseFloat(buffer);
    br.close();
    start = (new Date()).getTime();
    for(i = 0; i < size - 1; i++)
        for(j = i + 1; j < size; j++)
            if(vector[i] > vector[j])
            {
                temp = vector[i];
                vector[i] = vector[j];
                vector[j] = temp;
            }

```

```

        }
        end = (new Date()).getTime();
        times[f] = end - start;
        System.out.println(times[f]);
    }
}
}

```

bubblesort.py (Python)

```

import time

end = time.clock()
files = ["vec100", "vec1000", "vec2000", "vec3000",
        "vec4000", "vec5000", "vec6000", "vec7000",
        "vec8000", "vec9000", "vec10000", "vec20000",
        "vec30000", "vec40000", "vec50000", "vec60000",
        "vec70000", "vec80000", "vec90000", "vec100000"]
times = []
print('Runtimes: ')
for filename in files:
    vector = [float(line) for line in
              open(filename).readlines()]
    n = len(vector)
    start = time.clock() * 1000
    for i in range(n - 1):
        for j in range(i + 1, n):
            if vector[i] > vector[j]:
                temp = vector[i];
                vector[i] = vector[j];
                vector[j] = temp;

```

```

end = time.clock() * 1000
times.append(end - start)
print(end - start)

```

bubblesort.m (Octave)

```

filenames = {"vec100", "vec1000", "vec2000", "vec3000",
             "vec4000", "vec5000", "vec6000", "vec7000",
             "vec8000", "vec9000", "vec10000", "vec20000",
             "vec30000", "vec40000", "vec50000", "vec60000",
             "vec70000", "vec80000", "vec90000", "vec100000"};
runtimes = zeros(1,20);

disp('Runtimes: ');
for f = 1:length(filenames)
    vector = load(filenames{f});
    t_start = time();
    for i = 1:length(vector) - 1
        for j = i + 1:length(vector)
            if(vector(i) > vector(j))
                temp = vector(i);
                vector(i) = vector(j);
                vector(j) = temp;
            end
        end
    end
    t_end = time();
    runtimes(f) = 1000 * (t_end - t_start);
    disp(runtimes(f));
end

```