UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

RODRIGO ALVES COLONETTI

INSTALAÇÃO E ATUALIZAÇÃO MASSIVA DE APLICAÇÕES PARA SISTEMA OPERACIONAL LINUX EDUCACIONAL ATRAVÉS DA ADAPTAÇÃO DE SOFTWARES OPEN SOURCE

CRICIÚMA

2014

RODRIGO ALVES COLONETTI

INSTALAÇÃO E ATUALIZAÇÃO MASSIVA DE APLICAÇÕES PARA SISTEMA OPERACIONAL LINUX EDUCACIONAL ATRAVÉS DA ADAPTAÇÃO DE SOFTWARES OPEN SOURCE

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. Esp. Sergio Coral

CRICIÚMA

2014

RODRIGO ALVES COLONETTI

INSTALAÇÃO E ATUALIZAÇÃO MASSIVA DE APLICAÇÕES PARA SISTEMA OPERACIONAL LINUX EDUCACIONAL ATRAVÉS DA ADAPTAÇÃO DE SOFTWARES OPEN SOURCE

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Sistema Operacional.

Criciúma, 25 de novembro de 2014.

BANCA EXAMINADORA

Prof. Esp. Sergio Soral - (UNESC) - Orientador

Prof. MSc. Paulo João Martins - (UNESC)

Prof. Esp. Valter Blauth Júnior - (UNESC)

DEDICATÓRIA

Dedico este trabalho às pessoas que sempre me apoiaram ao longo de minha graduação, as meus pais pelo incentivo e a meus irmãos, amigos e professores.

AGRADECIMENTOS

Ao longo do período da graduação, muitas pessoas passaram por minha vida proporcionando conhecimentos, lições e alegrias. Neste momento gostaria de agradecê-las, pois de alguma forma contribuíram para a conclusão desta etapa. Entre estas pessoas, agradeço:

Aos professores do Curso de Ciência da Computação.

Aos amigos que fiz nesta universidade.

Ao professor orientador Sergio Coral que aceitou me auxiliar na realização deste trabalho.

E principalmente, a minha família pelo incentivo e por acreditarem em mim.

Obrigado a todos!

“A criatividade simplesmente consiste em conectar as coisas.”

Steve Jobs

RESUMO

Através do incentivo do governo federal escolas do Brasil são beneficiadas pelo Programa Nacional de Tecnologia Educacional, onde recebe-se um número considerável de computadores para montagem de laboratórios de informática, com o intuito de promover a educação e desenvolvimento dos alunos através do meio tecnológico. Neste contexto destaca-se o sistema GNU/Linux e sua distribuição educacional, utilizada nos computadores que compõem os laboratórios. Pensando nisto foi realizado o presente trabalho de pesquisa voltado para área de sistema operacional, visando o desenvolvimento de uma ferramenta em linguagem de programação *python* e *shell script,* utilizando juntamente à elas chamada de sistema *socket* e suas derivações, tendo como objetivo auxiliar o profissional da educação que lida com a manutenção dos laboratórios de informática diariamente. O trabalho propõe o desenvolvimento de um software que trabalhe sobre o modelo de sistema cliente-servidor, onde através da rede de computadores tem-se uma máquina servidor comunicando-se com as demais máquinas clientes, enviando comandos pré-definidos ou inseridos manualmente pelos responsáveis do laboratório, além de *scripts* diversos, gerados através do gerenciador de pacotes Synaptic presente na distribuição educacional, proporcionando otimização em processos de atualização, correção, instalação e até mesmo remoção de softwares do sistema. Para o desenvolvimento do trabalho foi pesquisado temas como a história do sistema operacional, onde foi visto que sempre houve procura por otimização de processos, pesquisou-se ainda as estruturas e tipos de sistema operacional, como são realizados os processos de obtenção e instalação de softwares em distribuições GNU/Linux, além dos tipos de licenças de software que podem ser utilizadas ao obter ou criar um software livre ou *open source*. Por fim após estudo e desenvolvimento a ferramenta realizou a manutenção de software de forma satisfatória, contando com uma interface simples, podendo melhorar em alguns aspectos, como é o caso do gerenciamento de conexões, mas que consegue auxiliar os profissionais que venham a utilizá-la.

Palavras-chave: Cliente-Servidor; Linux Educacional; Pacotes; Python; Sistema Operacional.

ABSTRACT

By encouraging the federal government schools in Brazil are benefiting from the National Educational Technology Program, which receives a considerable number of computers for computer labs assembly, in order to promote the education and development of students through technological means. In this context highlights the GNU / Linux system and its educational distribution, used in computers that make up the labs. With this in mind we performed the present research focused on operating system area, for the development of a python programming language tool and shell script using them together to call socket system and its derivatives, aiming to assist the professional education that deals with maintenance of computer labs daily. The paper proposes the development of a software that works on the client-server system model, where through the computer network has a server machine communicating with other client machines by sending predefined commands or entered manually by the responsible lab and several scripts, generated through the Synaptic package manager present in the educational distribution, providing optimization upgrade processes, correction, installation and even removing system software. For the development of this work was researched topics such as the history of the operating system, where it was seen that there was always demand for process optimization, still researched the structures and types of operating system, such as taking and software installation are made in GNU / Linux, in addition to types of software licenses that can be used to obtain or create a free software or open source. Finally after studying and developing the tool held software maintenance in a satisfactory manner, with a simple interface and can improve in some aspects, such as the management of connections, but can assist workers who will use it.

Key words: Client-Server; Linux Education; packages; Python; Operating System.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

APT	Advanced Packing Tool
C3SL	Centro de Educação Científica e Software Livre
CETE	Centro de Experimentação em Tecnologia Educacional
CI	Circuitos Integrados
CPU	Unidade Central de Processamento
CP/CMS	Control Program / Conversational Monitor System
CTSS	Compatible Time Sharing System
EPL	Eclipse Public License
FSF	Free Software Fundation
GE	General Eletric
GUI	Graphical User Interface
IBM	International Business Machine
MEC	Ministério da Educação
MIT	Massachusetts Institute of Technology
MPL	Mozilla Public License
OSI	Open Source Initiative
P2P	Peer-Two-Peer
PROINFO	Programa Nacional de Tecnologia Educacional
SaaS	Software as a Service
THE	Technische Hogeschool Eindhoven
TSS	Time Sharing System
VM	Virtual Machine

SUMÁRIO

1 INTRODUÇÃO

Com a criação do Programa Nacional de Tecnologia Educacional (PROINFO), idealizado pelo Governo Federal no ano de 1997, com o intuito de melhorar o uso de ambientes de informática e promover o uso pedagógico da informática utilizando-a como ferramenta de conhecimento através de softwares livres foi desenvolvido o sistema operacional Linux Educacional, uma solução que atende aos propósitos do PROINFO, que favorece o usuário final no que se refere a questões de acessibilidade e também aos responsáveis pelo laboratório no processo de manutenção e atualização de software (PORTAL DO SOFTWARE PÚBLICO, 2014).

Desde a sua criação o sistema vem sendo instalado em várias escolas beneficiadas, atualmente encontra-se na versão 5.0, desde a primeira versão já passou por diversas melhorias, recebeu novos recursos, como a Edubar, barra de ferramentas com conteúdos educacionais oferecidos pelo Ministério da Educação (MEC). Apesar disso um problema que persiste é a dificuldade em realizar a instalação e atualização de aplicativos presentes no sistema, devido a grande quantidade de comandos que devem ser executados computador a computador, considerando um laboratório de informática em uma das escolas, por exemplo, tornando-se um problema ainda maior se esses computadores forem gerenciados por pessoas leigas, que não possuem conhecimentos necessários para realizar tais operações no sistema.

O presente trabalho busca resolver o problema apresentado acima, tornando-se relevante quando buscado conteúdos nas comunidades e trabalhos com os mesmos princípios e não são encontradas muitas alternativas como a proposta para distribuições GNU\Linux, sendo elas de uso doméstico, educacional ou empresarial. Alternativas que trabalhem via ambiente de rede, ou seja, um conjunto de sistemas interligados que se comunicam através de um meio, possibilitando a troca de informações entre eles (CARISSIMI; JUERGEN; GRANVILLE, 2009), que possibilite aos responsáveis pelo parque de máquinas a realização de processos de manutenção de softwares de forma otimizada.

Apresentará também conceitos importantes para quem pretende ou

desenvolve softwares livres, tipos de licenças que podem ser utilizadas e direitos que usuários e desenvolvedores tem ao aplicar tais licenças, além de mostrar o processo evolutivo do que chamamos de sistema operacional.

1.1 OBJETIVO GERAL

Adaptar e desenvolver ferramentas, softwares disponíveis em sistema Linux Educacional para realização de instalação e atualização de software de forma massiva e centralizada via ambiente de rede.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

a) compreender conceitos e funcionamento do sistema operacional Linux e suas distribuições;
b) analisar ferramentas de instalação e atualização automática;
c) compreender softwares livres e *open source*;
d) entender o funcionamento de redes cliente / servidor;
e) entender o funcionamento das chamadas de sistema em sistemas Linux para comunicação via rede;
f) adaptar ferramentas conforme proposto por meio dos conhecimentos adquiridos.

1.3 JUSTIFICATIVA

Com a informatização das escolas e o uso difundido de computadores e sistemas operacionais livres, como é o caso do sistema Linux Educacional, principalmente em escolas públicas beneficiadas por projetos do Governo Federal,

elas passam a possuir um número considerável de computadores e para mantê-los se faz necessário realizar a manutenção de software.

Sendo assim, a possibilidade de ter uma ferramenta que consiga automatizar parte deste processo é de grande ajuda, para pessoas que desempenham essas atividades diariamente. Ferramentas como esta, que fazem o gerenciamento de instalações, são difíceis de encontrar, tanto para sistemas Windows, quanto para sistemas Linux. Normalmente são ferramentas pagas. Entre elas podemos citar o software Ninite para sistemas Windows, de difícil configuração ou ainda as que não têm seu foco voltado para instalações em massa, como é o caso das ferramentas Clonezilla, Redo Backup, Acronis, Ghost, que realizam apenas uma cópia do sistema, gerando uma imagem do mesmo para possível replicação. Mesmo no Linux Educacional existem ferramentas para instalação e atualização de pacotes, mas não em massa e muitas vezes não se sabe o que está sendo instalado, principalmente para usuários com pouco conhecimento. Atualmente na versão 5.0 foi desenvolvida uma ferramenta de atualização automática, mas esta realiza manutenção de software de forma específica, conectando-se automaticamente a repositórios oficiais, não possibilitando ao usuário a escolha de quais aplicações deseja atualizar ou instalar, todas as atualizações são analisadas e disponibilizas pelos mantenedores do projeto.

Tem-se também ferramentas *Network Management System* (NMS) que tem o propósito gerenciar redes em geral e poucas possuem função de instalação ou atualização, mesmo que limitadas no ambiente gerenciado, como exemplo, pode-se citar o OpenNMS.

Ao analisar as informações mencionadas sobre o desenvolvimento da ferramenta, elas podem ser consideradas relevantes, a partir do momento em que se leva em consideração quanto tempo gastam os profissionais que realizam os processos de forma manual, computador a computador, pela falta de alternativas que os auxiliem durante o processo.

Não falta tempo às pessoas, o problema é que há muitas opções para utilização deste tempo (PERSONA, 2007), opções estas que muitas vezes podem ser executadas de forma otimizada, como é o caso do gerenciamento de ambiente de informática, pois não basta gerenciar desktops, servidores, redes, dados e software de forma isolada. Todos esses componentes precisam interagir uns com os

outros, possibilitando conectividade entre serviços e o gerenciamento deve contemplar essas questões (INTEL, 2014). Além disso, por vir a trabalhar sobre um modelo de rede e utilizar chamadas de sistema, que são mecanismos que oferecem uma extremidade para comunicação (COMER, 2006), mecanismos estes que tem suporte para utilização desde 1970, com a criação de sistemas BSD UNIX e sendo desenvolvida em linguagem *python* e *shell script* poderá ser utilização em outras distribuições GNU/Linux e não somente na educacional.

Quando finalizada trará benefícios a qualquer de seus utilizadores, pois será um software livre, ou seja, programa de computador que pode ser utilizado, modificado, copiado, estudado e redistribuído sem nenhum custo ou restrição (MOURA, 2014), ficando disponível em comunidades relacionadas ao sistema operacional. Pretende-se desenvolver uma ferramenta de interface simples, que auxilie os usuários em sua rotina, seja ela de trabalho ou não, proporcionando maior tempo livre, para que possam realizar outras atividades.

1.4 ESTRUTURA DO TRABALHO

O trabalho de pesquisa desenvolvido é composto por sete capítulos. Primeiramente é apresentada a introdução, o objetivo geral e os específicos, além da justificativa sobre o trabalho proposto.

No capítulo dois é abordada a história do sistema operacional, suas diferentes estruturas, monolítica, hierárquica ou em camadas, máquina virtual e cliente / servidor e como cada uma delas funciona. Por fim são apresentados os tipos de sistema, sistemas batch, mono e multiprogramados, de tempo compartilhado e real e sistemas multiprocessados e uma breve descrição de cada um deles.

O tema sistema operacional GNU/Linux é abordado no capítulo três, onde é contada um pouco de sua história, manifesto e projeto GNU, suas distribuições, tendo como foco a distribuição educacional, descrevendo seu funcionamento vantagens e desvantagens em relação ao seu uso, apresenta também o conceito de

repositório e gerenciador de pacotes, mostrando o gerenciador Synaptic, utilizado neste trabalho.

No capítulo quatro são abordados os temas software livre e *open source,* suas diferenças e tipos de licenciamento que cada iniciativa possui e também como cada tipo de licença pode ser aplicada.

Os trabalhos correlatos ficam no capítulo cinco, onde são relacionados os trabalhos que se assemelham ao trabalho desenvolvido.

O capitulo seis tem a finalidade de apresentar as etapas de desenvolvimento da ferramenta proposta, o que foi utilizado em seu desenvolvimento, modelagem, funcionalidades e resultados obtidos. Por fim na conclusão são expostas as considerações finais e trabalhos futuros, tendo como base o que foi desenvolvido.

2 SISTEMA OPERACIONAL

Durante anos desde a invenção dos primeiros computadores e sistemas operacionais, os mesmos já passaram por diversas modificações, que surgiram devido ao avanço tecnológico que acontece de forma acelerada. Parte desta história poderá ser vista no decorrer deste capítulo.

2.1 CONCEITO E HISTÓRIA

O sistema operacional vem evoluindo desde a década de 50, anteriormente na primeira geração de computadores as máquinas eram bastante primitivas e seus programadores implementavam em linguagem de máquina pura. Tudo era feito através de chaves mecânicas, fios ligando painéis e conectores, válvulas sendo substituídas, servindo como chaveadores de corrente, todo esse processo e componentes fazia parte da programação dos computadores na época. Não havia nenhuma linguagem de programação, nem mesmo a linguagem Assembly, “linguagem de montagem que usa abreviaturas parecidas com palavras em inglês para representar operações básicas de um computador, desenvolvidas para acelerar o processo de programação” (DEITEL, 2012, p. 5) era conhecida na época, o que tornava o processo bastante lento.

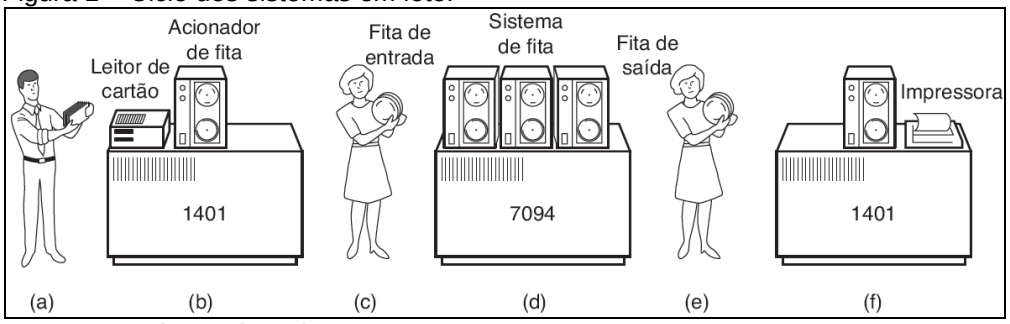
O programa fundamental é o sistema operacional, pois controla os recursos do computador, atuando como base sobre a qual os programas, aplicativos são escritos (TANENBAUM; WOODHULL, 2008).

Já em 1955, na segunda geração, os computadores passam a operar por meio de transistores, substituindo as antigas válvulas, isso os tornava mais confiáveis e pela primeira vez havia um grupo de pessoas, contando com projetistas, construtores, operadores e programadores, além de uma equipe de manutenção nos Laboratórios da Pesquisa da *General Motors* voltados para o desenvolvimento do primeiro conceito de sistema operativo para seus computadores da *International Business Machine* (IBM).

O sistema desenvolvido primeiramente fez com que os computadores executem *jobs*, programas que quando compilados e executados, levam juntamente seus dados de execução (OLIVEIRA; CARISSIMI; TOSCANI, 2010). Estes programas eram escritos em papel na linguagem FORTRAN ou Assembly, já conhecidas neste período, o conteúdo escrito era perfurado em um conjunto de cartões e estes eram interpretados um após o outro pelo computador, realizando a execução de operações, em sua maioria cálculos.

Devido ao alto custo do processo e tempo empregado para desenvolvimento dos códigos, impressão, retirada e inserção de novos programas a equipe desenvolveu o segundo conceito de sistema operacional, o sistema de processamento em lotes. Este sistema foi empregado no computador IBM 7094 e tinha a capacidade de ler vários programas em sequência através de uma fita magnética, gerada anteriormente por outra máquina, IBM 1401, que lia os cartões e reescrevia nas fitas. Assim novos computadores eram construídos, resolvendo cálculos cada vez mais complexos, realizando computação de fato. Nestes computadores após a execução de um lote completo de *jobs* o sistema gerava uma segunda fita magnética, essa fita era lida no IBM 1401 que imprimia os resultados (figura 1), este modelo de sistema foi utilizado por quase uma década e ficou conhecido como IBSYS, sistema operacional da IBM para computadores 7094 (DEITEL, 2012; TANENBAUM, 2010).

Figura 1 – Ciclo dos sistemas em lote.



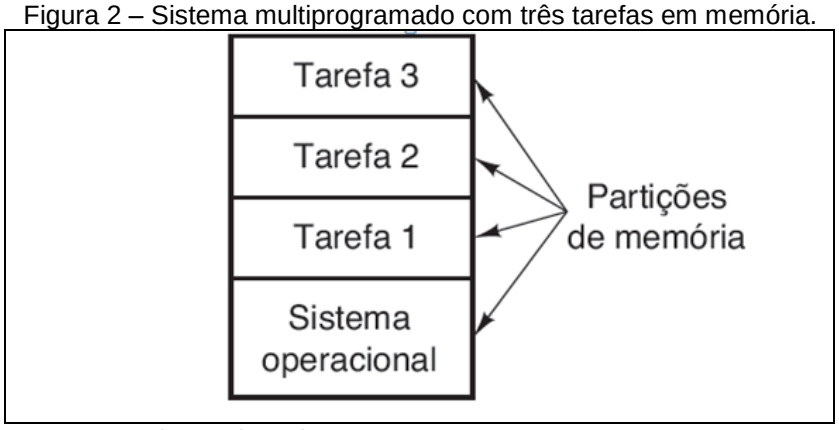
Fonte: Tanenbaum (2010).

Em 1960 a terceira geração dos computadores foi iniciada, desta vez os computadores eram baseados em Circuitos Integrados (CI), que são circuitos eletrônicos formados por miniaturas de diferentes componentes, entre eles transistores, diodos, resistores e capacitores (NEWTECK, 2014), substituindo assim os transistores da geração passada. Nascia assim uma nova série de computadores

IBM, System/360, que tinha como objetivo centralizar as etapas antes feitas por duas ou três máquinas em apenas uma, necessitando assim de um novo sistema, partindo deste propósito a terceira geração de sistema operacional surgiu, o sistema foi apelidado de OS/360.

O OS/360 era composto de milhares de linhas de código em Assembly, sendo executados na própria máquina, acompanhadas por milhares de erros de programação. Mesmo com grande quantidade de erros o sistema foi bem aceito na época, devido possuir recursos importantes se comparado com a segunda geração, como é o caso da multiprogramação, onde vários programas poderiam ser executados simultaneamente em um único processo de trabalho (figura 2) (ALCADE; MORERA; CAMPANERO, 1993). Apesar de ser um bom sistema para realização de cálculos e previsões, os conjuntos de programas passaram a ser executados de forma mais lenta, mesmo executando vários processos seus programadores demoravam mais tempo para realizar depurações complexas, devido a essa partilha de recursos da Unidade Central de Processamento (CPU), fazendo com que cada usuário esperasse pelo outro terminar a execução de seus trabalhos.

Figura 2 – Sistema multiprogramado com três tarefas em memória.



Fonte: Tanenbaum (2010).

Buscando uma alternativa para diminuir o tempo de resposta surgiu o conceito tempo compartilhado ou *time-sharing*, onde cada usuário teria a sua disposição um terminal *online*, interagindo diretamente com o sistema em cada fase do desenvolvimento de suas aplicações e caso necessário poderia modificá-las imediatamente. Devido a esse tipo de interação, estes sistemas também são conhecidos com *online* (MACHADO; MAIA, 2013).

Foram desenvolvidos os sistemas de tempo compartilhado *Compatible*

Time Sharing System (CTSS) pelo *Massachusetts Institute of Technology* (MIT), o *Time Sharing System* (TSS) desenvolvido pela própria IBM, o sistema Multics desenvolvido em conjunto pelo MIT, *General Electric* (GE) e Laboratórios Bell além do *Control Program* / *Conversational Monitor System* (CP/CMS) que evoluiu para o sistema operacional *Virtual Machine* (VM) após aprimoramento da IBM. Estes sistemas mostraram a importância da interatividade em ambientes de desenvolvimento, mesmo sendo desenvolvidos para execução de tarefas básicas, devido à maneira que compartilhavam seus programas, possibilitando ao usuário execução de várias tarefas.

Dentre todos os sistemas da época destacou-se o sistema Multics, o primeiro sistema operacional desenvolvido em uma linguagem de alto nível, a linguagem EPL estruturada segundo modelo PL/1 da IBM. Paralelamente a isso um dos projetistas dos Laboratórios Bell, Ken Thompson que trabalhava no projeto Multics descobriu um computador não utilizado e iniciou um projeto independente de sistema monousuário simplificado do Multics, esse projeto daria início ao desenvolvimento do sistema operacional UNIX, que em breve tornara-se popular entre acadêmicos, trabalhadores do governo e grandes empresas, sistema que revolucionaria mais uma vez, sendo ele desenvolvido em uma nova linguagem de programação, a linguagem de alto nível conhecida como C (DEITEL, 2012; TANENBAUM, 2010).

Nos anos seguintes, mais especificamente na metade da década de 70 os computadores começaram a ser utilizados para comunicação entre departamentos de defesa, dando origem ao padrão de comunicação TCP/IP, que originaria mais tarde o padrão *Ethernet*[1]. Devido a esse desenvolvimento e grande quantidade de informações trafegando pelos meios de comunicação a revolução dos computadores pessoais era iniciada, no final da década de 70 surge um novo sistema no mercado, o Apple II.

No ano de 1980, já na quarta geração de computadores, empresas como IBM e Apple Macintosh investiram pesado em computadores pessoais, outra parte importante deste período foi o desenvolvimento dos *chips* processadores. Neste período a Microsoft comprava também o sistema QDOS de Tim Paterson da *Digital*

[1] Rede comunicação com controle descentralizado, trabalhando em velocidades de 10 Mbps a 10 Gbps (TANENBAUM; WHETERALL, 2010).

Research, que inicialmente servia apenas para testes em processadores e componentes Intel, após sofrer algumas modificações pela equipe da Microsoft este sistema foi apelidado de MS-DOS e começou a ser utilizado em computadores IBM. (DEITEL, 2012; TANENBAUM, 2010).

Com os computadores pessoais ainda em alta, o desenvolvimento de softwares para realização de cálculos, edição de textos e aplicativos gráficos tornava-se comum. Como o objetivo era o ganho de tempo e produtividade, surgiram as *Graphical User Interfaces* (GUI), compostas por janelas, ícones e menus, facilitando a utilização de programas pelos seus usuários. A partir daí os sistemas operacionais só evoluíram, principalmente com surgimento de sistemas operacionais como, Windows baseado em sistema MS-DOS, GNU/Linux e suas distribuições, sistema MAC OS da Apple, esses por sua vez tendo como base sistemas UNIX. Com o passar dos anos os sistemas operacionais vem sofrendo modificações, acompanhando as novas tecnologias, trabalhando sobre estruturas específicas, incorporando novos recursos, proporcionando maior interação e praticidade ao usuário (DEITEL, 2012; TANENBAUM, 2010).

2.2 ESTRUTURA E FUNCIONAMENTO

Ao desenvolver um sistema operacional deve ser analisado para qual finalidade será utilizado e o tipo de processamento que se pretende realizar, se será por bloco, multiprocessamento, tempo repartido, entre outros. É necessário também avaliar as necessidades apresentadas, ou seja, requisitos do utilizador e de software (ALCADE; MORERA; CAMPANERO, 1993).

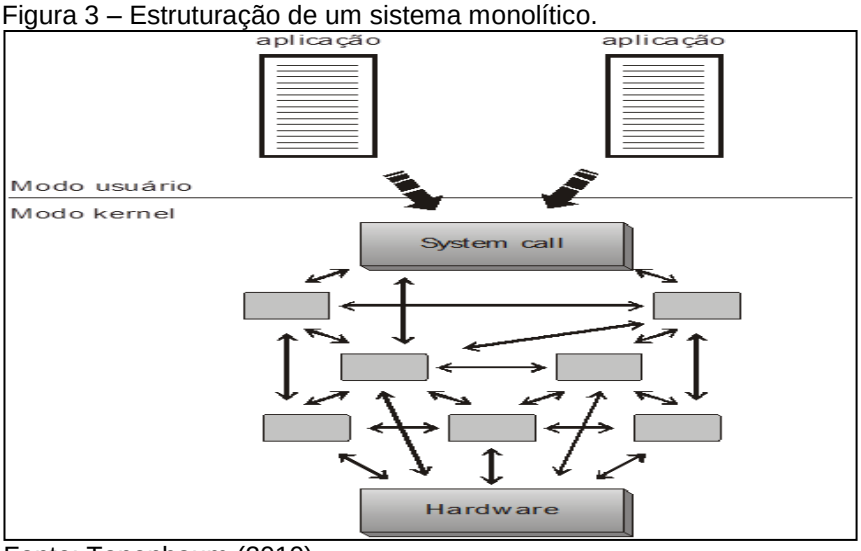
Há quatro estruturas distintas que nos ajudam a atender as necessidades citadas anteriormente, são elas, monolítica, hierárquica ou em camadas, em máquina virtual e cliente / servidor.

2.2.1 Estrutura monolítica

Estrutura utilizada nos primeiros sistemas operacionais, constituída de um programa, sendo este dividido em subprogramas. Sistema monolítico é um modelo onde o sistema operacional é codificado por via de uma coleção de procedimentos, permitindo a qualquer um deles, em qualquer parte do programa acessar outro procedimento (figura 3) (TANENBUAM; WOODHULL, 2008). Como neste tipo de sistema cada procedimento pode acessar um ao outro, há grande carência nos requisitos de proteção e privilégios (ALCADE; MORERA; CAMPANERO, 1993).

Consegue-se observar sua estruturação no momento em que as chamadas de sistema (serviços), são geradas pelo sistema operacional e logo após são requisitadas, via solicitação de dispositivos de entrada e saída, todas as solicitações são organizadas e parametrizadas em registradores ou pilhas, posteriormente é executada uma instrução conhecida como chamada de *kernel*[2], passando o controle do usuário para o sistema operacional, após essa chamada o sistema executa os procedimentos com base nos parâmetros gerados. Finalizando todos os procedimentos o sistema retorna para modo usuário, possibilitando a inserção de novos procedimentos, iniciando mais uma vez o ciclo do sistema (TANENBAUM; WOODHULL, 2008).

Figura 3 – Estruturação de um sistema monolítico.



Fonte: Tanenbaum (2010).

[2] Camada interna que disponibiliza ao software *interface* com os recursos de *hardware*, sendo responsável gerenciar todos os recursos computacionais (ROMERO, 2013).

2.2.2 Estrutura hierárquica ou em camadas

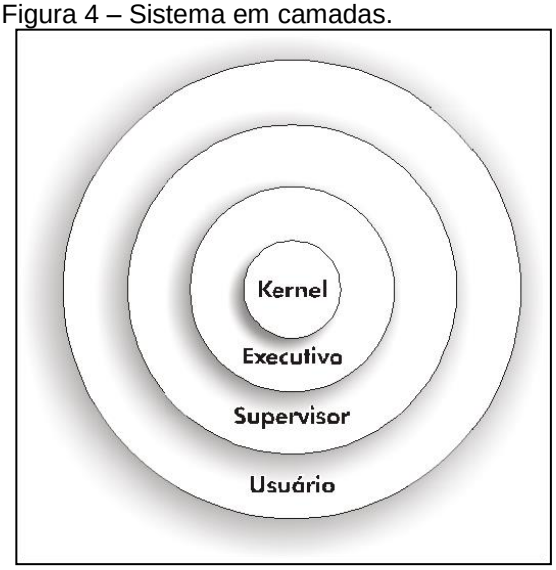
Esta estrutura de sistema operacional composta de seis camadas foi usada pela primeira vez no sistema *Technische Hogeschool Eindhoven* (THE) desenvolvido por Dijkstra na Holanda em 1968, onde cada camada tem uma função específica, todas possuem uma interface clara, permitindo a ligação entre elas (TANENBAUM; WOODHULL, 2008). Essa mudança ocorreu devido ao aperfeiçoamento dos programas, tornando necessário que o sistema operacional tivesse uma maior organização.

As camadas ou níveis são os seguintes:

a) gestão da CPU: permite a alocação do processador e a alternância do mesmo entre processos em caso de interrupções ou temporizadores expirados (multiprogramação);
b) gestão de memória: realiza a alocação de memória para processos na memória principal e secundária, quando ela já está completamente ocupada;
c) comunicação de processos do utilizador: permite a comunicação dos processos com o console do operador, passando as informações executadas à ele;
d) entrada e saída: realiza o controle dos dispositivos de entrada e saída e armazena as informações transferidas através deles;
e) programa usuário: camada onde ficam localizados os programas do usuário;
f) operador: neste nível localizam-se os processos do usuário.

Sistemas como Multics e Open VMS utilizam o conceito de camadas concêntricas (curvas que contém o mesmo centro), onde as camadas internas têm mais privilégios que as externas (figura 4). Este tipo de estruturação em camada é utilizado até hoje em sistemas Windows e derivados de Multics e Unix (TANENBAUM, 2010).

Figura 4 – Sistema em camadas.



Fonte: Tanenbaum (2010).

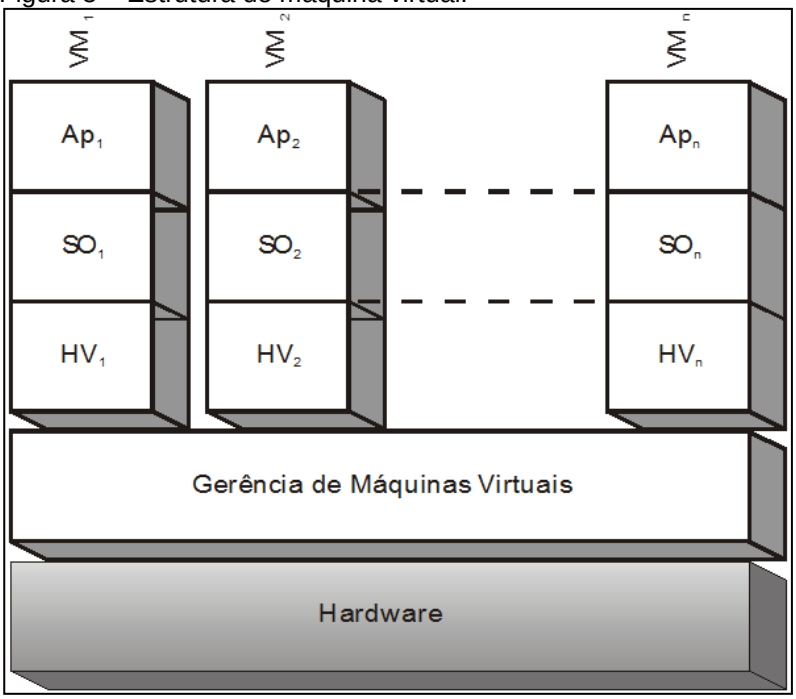
2.2.3 Estrutura em máquina virtual

Este tipo de estrutura originou-se de sistemas CP/CMS que mais tarde foram apelidados de VM/370, tendo como princípio conceitos de multiprogramação e principalmente que este tipo de estrutura pudesse rodar sobre um hardware básico.

> A virtualização de recursos é um conceito relativamente antigo, mas os recentes avanços nessa área permitem usar máquinas virtuais com os mais diversos objetivos, como a segurança, a compatibilidade de aplicações legadas ou a consolidação de servidores (MAZIERO, 2013, p. 3).

Nesta estrutura de sistema cada máquina virtual gerada simula réplicas físicas da máquina real e cada usuário tem a ilusão de que o sistema está totalmente disponível para ele, mas na verdade essas máquinas estão rodando concorrentemente sobre um ambiente abstrato de software (figura 5). Possui também um programa chamado *exokernel*, que trabalha no *kernel* do sistema principal, ficando a cargo de atribuir os recursos às máquinas virtuais, certificando-se de que nenhuma máquina use os recursos alocados à outra, proporcionado assim sistemas com menor sobrecarga (TANENBAUM; WOODHULL, 2008).

Figura 5 – Estrutura de máquina virtual.



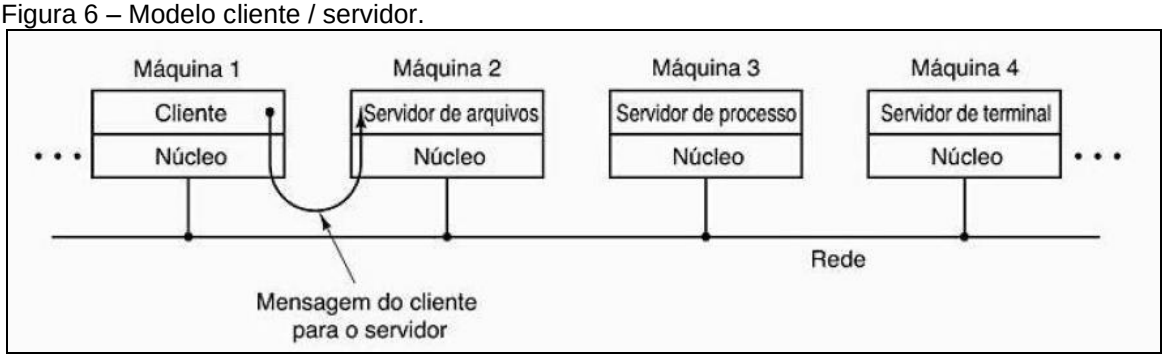
Fonte: Tanenbaum (2010).

2.2.4 Estrutura cliente / servidor

Sistemas operacionais modernos buscam mover seus códigos para camadas mais elevadas, removendo a maior quantidade destes códigos do sistema operacional. É isso que esta estrutura procura, implementar a maior parte das funções do sistema operacional em processos de usuário, para requisição de um serviço, como por exemplo, a leitura de um bloco de texto, o processo de usuário cliente envia uma requisição ao processo servidor, esse então faz o trabalho desejado e envia uma mensagem de resposta (figura 6) (TANENBAUM; WOODHULL, 2008).

Nesse modelo o *kernel* do sistema realiza o gerenciamento da comunicação entre cliente e servidor, em um sistema dividido em três partes, sendo elas serviços de arquivos, processos e memória. Geralmente estes servidores e clientes são interligados por meio de um ambiente de rede, nos servidores são agregadas as funções mais importantes deixando as tarefas básicas para execução nos sistemas clientes.

Figura 6 – Modelo cliente / servidor.



Fonte: Tanenbaum (2010).

2.3 TIPOS DE SISTEMA OPERACIONAL

Além das estruturas o sistema operacional pode ser classificado de acordo com o número de processos de usuário que consegue executar ou ainda pelo número de processadores que possui, sendo eles, sistemas batch, mono e multiprogramados, de tempo compartilhado, de tempo real e por fim os multiprocessados.

2.3.1 Sistemas batch

Os sistemas batch ou em lote são considerados por muitos como os primeiros sistemas multiprogramáveis. Nele programas armazenados em fitas ou discos são submetidos e executados sequencialmente, durante esse processo há pouca ou nenhuma interação com o usuário, uma vez que o processo é iniciado o usuário não tem acesso a qualquer informação (MACHADO; MAIA, 2013).

2.3.2 Sistemas mono e multiprogramados

Em sistemas monoprogramados (monotarefa), há um processador que gerencia a execução de um único programa de um único usuário, devido a isso são sistemas monousuário. Neste tipo de sistema o processador fica grande parte do tempo ocioso enquanto espera por comandos vindos dos dispositivos de entrada e saída.

Já os sistemas multiprogramados (multitarefa), permitem que vários processadores executem uma única tarefa ou ainda que vários programas utilizem recursos concorrentemente do sistema, essa utilização concorrente é chamada de paralelismo lógico ou virtual. Nestes sistemas o tempo da CPU é divido entre os vários usuários e seus vários processos, diminuindo assim o tempo em que o processador permanece ocioso (GUIMARÃES, 2010).

2.3.3 Sistema de tempo compartilhado (*time-sharing*)

São sistemas multiterminais, o processamento é controlado através de um computador central, onde há usuários interagindo com o mesmo por meio de terminais *online*. Este sistema principal executa um *polling*, ou seja, uma varredura nos terminais conectados, após essa varredura, o computador central permite o compartilhamento de tempo entre eles no processador.

Nestes sistemas o usuário tem a impressão de que o sistema está totalmente dedicado a ele, mas o processamento acontece de forma intercalada entre os programas e seus diversos terminais, isso acontece pelo mesmo reservar uma fatia de tempo, *time-slice* para cada programa, com isso pode ser considerado também como multiprogramado (GUIMARÃES, 2010).

2.3.4 Sistema de tempo real (*real-time*)

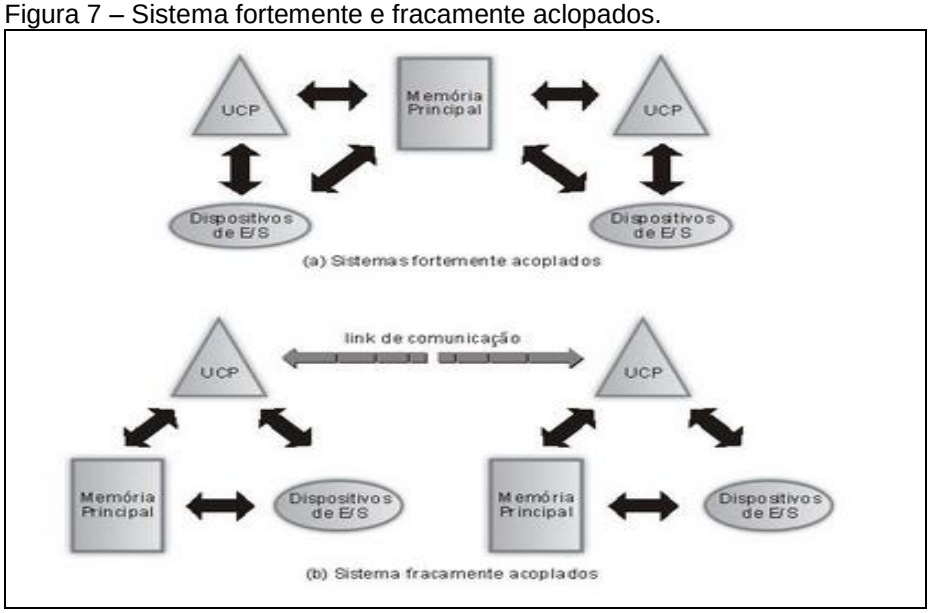
Este tipo de sistema é semelhante ao *time-sharing*, mas exige um tempo de resposta mais eficaz na execução das tarefas. Os processos são executados no tempo necessário e de acordo com o nível de prioridade. Sistemas como este são utilizados em locais onde o tempo é um fator predominante, como é o caso de refinarias de petróleo, controle de tráfego aéreo. Geralmente os processos nele executados são ativados e controlados por sensores. (GUIMARÃES, 2010).

2.3.5 Sistema multiprocessado

Um sistema multiprocessado possui diversos processadores, sendo estes alocados em um único local ou então espalhados fisicamente por meio de redes. Tem a capacidade de executar várias tarefas, trabalhando com os processadores em paralelo, esse paralelismo pode ser real ou lógico.

Em sistemas como existem duas classificações, sendo elas tratadas como fracamente ou fortemente acoplados (figura 7). Nos sistemas fracamente acoplados cada processador possui sua própria memória e seu sistema operacional. Já nos sistemas fortemente acoplados todos os processadores compartilham de uma mesma memória. Seus processadores ficam localizados em um computador hospedeiro que tem a tarefa de controlar e distribuir os processos que serão executados (GUIMARÃES, 2010).

Figura 7 – Sistema fortemente e fracamente aclopados.



Fonte: Machado; Maia (2013).

3 SISTEMA GNU

No decorrer deste capítulo será apresentada a história anterior e posterior ao surgimento do GNU/Linux, abordará também os conceitos de distribuições Linux, principalmente a distribuição Linux Educacional, repositórios e gerenciadores de pacotes.

3.1 MANIFESTO GNU

O manifesto GNU foi escrito e publicado em 1983 por Richard Stallman, criando o conceito *Free Software* e dando início ao projeto GNU (MORIGI; SANTIN, 2007).

> Este manifesto lançou o alicerce para a formação da identidade do MSL (Movimento do Software Livre) e sintetizou seus princípios fundamentais em sua auto definição, traçando objetivos e anunciando em nome de quem se pronunciava. Afirma a luta do movimento por liberdade a todos os programadores e usuários de softwares, que passaram a reunir-se em grupos na intenção de permitir que programas úteis fossem usados para fomentar uma comunidade de boa vontade, disposta a lutar por liberdade em todo conjunto da sociedade (STALLMAN, 1993).

Em resumo o manifesto GNU começa descrevendo os principais objetivos do projeto GNU, o seu significado, além de seu conteúdo detalhadamente, ou seja, todos os softwares, compiladores, bibliotecas que se pretendia desenvolver. Explica também que o projeto se baseia em UNIX e a importância do mesmo ser concluído, como as pessoas podem contribuir e quais benefícios se têm ao participar.

Neste manifesto são apontadas as formas de distribuição e publicidade dos softwares livres, a necessidade de incentivo monetário, como funciona a filosofia do software livre e ainda a importância de projetos como este para o setor tecnológico (MORIGI; SANTIN, 2007).

3.2 PROJETO GNU

O projeto GNU foi iniciado em 1984 quando seu idealizador Richard Stallman deixou seu emprego na MIT para desenvolver softwares livres, pois acredita que os usuários possuem o direito de modificar programas, ajustá-los conforme suas necessidades e compartilha-los livremente (DIBONA; OCKMAN; STONE, 1999).

O nome GNU foi escolhido por atender a alguns requisitos como, ser um acrônimo recursivo para “GNU Not Unix”, ou seja, “GNU Não é Unix” e também porque segundo seu idealizador era uma expressão divertida de se pronunciar.

Para o desenvolvimento do GNU inicialmente Richard viu necessário o desenvolvimento de um novo compilador multi-linguagem e multi-plataforma, sendo este o primeiro programa do projeto GNU, como não obteve sucesso desenvolvendo baseando-se no compilador *Pastel* desenvolvido pela *Lawrence Livermore Lab*, que lia códigos em Pascal e em C, ele decidi desenvolver um compilador próprio. Foram desenvolvidos dois compiladores o GNU *Emacs*, compilador que já conseguia trabalhar juntamente com UNIX e posteriormente o GCC (DIBONA; OCKMAN; STONE, 1999).

Com a popularização dos compiladores e o desenvolvimento de ferramentas a partir deste, rodando em versões UNIX transportadas e adaptadas muitas vezes pelos seus usuários e desenvolvedores, o projeto começou a ganhar

força e juntamente a isso novos integrantes, surgiu assim a *Free Software Fundation* (FSF), seu objetivo era o desenvolvimento e distribuição de softwares livres. Para que pudesse manter-se a fundação aceitava doações quando fazia a distribuição de *softwares*, binários e manuais (DIBONA; OCKMAN; STONE, 1999).

Na FSF foi desenvolvido dois projetos notáveis, a biblioteca C e a shell (camada externa entre o usuário e núcleo de um sistema operacional), a partir deste momento os desenvolvedores já enxergavam um sistema operacional completo, mais faltava um componente fundamental, o *kernel*, ou seja, um núcleo para se comunicar. Foi então que desenvolveram o *kernel* apelidado de HURD, com base no micronúcleo *Mach*, desenvolvido na *Carnegie Mellon University* e depois na *University of Utah*, que conseguia realizar diversas tarefas que o núcleo do UNIX realizava. Passaram-se vários anos para que o HURD ficasse estável, mesmo assim ele não operava em conjunto aos programas, bibliotecas e shell desenvolvidos no projeto GNU (DIBONA; OCKMAN; STONE, 1999).

Em 1991 Linus Torvalds desenvolve um núcleo compatível com UNIX e o apelida de Linux, por volta de 1992 este núcleo é incorporado ao projeto GNU, onde se encaixa perfeitamente, dando origem a um sistema operacional completo, possibilitando a execução do sistema GNU, originando o sistema operacional GNU/Linux, trabalhando perfeitamente com a biblioteca em C, shell e outros programas do projeto (DIBONA; OCKMAN; STONE, 1999).

3.3 GNU/LINUX

Antes de abordar o tema GNU/Linux e suas distribuições é conveniente saber realmente o que é o Linux. Linux é um *kernel*, ou seja, a parte central do sistema operacional, um sistema complexo responsável processar comandos e realizar a conexão entre software e hardware (HILL et al., 2008).

Foi criado por Linus Torvalds, o motivo do projeto foi seu descontentamento com o desempenho do DOS (Windows). Linus teve como ponto de partida os sistemas Minix que não o atendeu como esperado, devido questões de desempenho, como o sistema Unix tinha um valor elevado ele decidi começar seu

próprio sistema. Uma vantagem foi basear-se em Minix e Unix, além de ter começado o desenvolvimento pelo *kernel* do sistema (MOTA FILHO, 2006).

Desde o início Linus acompanhou e distribuiu seu *kernel* sob uma licença livre, juntamente com outras ferramentas da FSF até que se uniu a elas por definitivo, passando a trabalhar com o sistema de interface gráfica chamada X. Nascia assim o sistema operacional GNU/Linux, totalmente livre seja ele na questão de preço ou definições de liberdade definidas por Stallman (MOTA FILHO, 2006).

Devido aos fatos hoje podemos utilizar as várias distribuições GNU/Linux disponíveis, sendo elas desenvolvidas sobre o trabalho desta colaboração.

3.3.1 Distribuições GNU/Linux

Se existisse apenas um *kernel*, o mesmo não seria de grande utilidade, são necessários programas, *interfaces* e *drivers* para usufruir todo o seu potencial, por consequência disso são necessárias às distribuições GNU/Linux, são elas que proporcionam tais complementos para o *kernel*, formando assim o sistema operacional completo (ARRUDA, 2014).

Em sistemas GNU/Linux e suas distribuições estes complementos são disponibilizados por meio de pacotes, armazenados em repositórios, acessíveis através sistemas gerenciadores de pacotes ou comandos de instalação e atualização.

3.4 REPOSITÓRIO DE PACOTES DE SOFTWARE

Primeiramente precisa-se entender o conceito de pacotes, pacotes são arquivos compactados que contém binários, *scripts*, dados, documentação e outros arquivos necessários para a instalação, atualização ou remoção de um software em sistemas GNU/Linux, esses softwares empacotados são gerenciados pelos gerenciadores de pacotes (FERREIRA, 2006).

Repositórios de pacotes são computadores servidores disponíveis na rede, que tem como princípio disponibilizar pacotes de software para outros computadores clientes (FERREIRA, 2006).

Existem dois tipos de repositórios, são eles:

a) repositório local: configurado em computadores que não tem acesso a rede, qualquer instalação ou atualização necessária será obtida do repositório local.
b) repositório remoto: configurado em servidores de arquivo FTP ou HTTP, disponibilizando pacotes para computadores clientes e à si mesmo.

3.5 GERENCIADOR DE PACOTES

Softwares presentes em sistemas GNU/Linux que auxiliam na instalação, atualização e remoção de pacotes, tem a capacidade também de compilar e empacotar códigos fonte (FERREIRA, 2006).

Existem também gerenciadores de pacote avançados, que realizam a instalação ou atualização de pacotes onde em caso de erro ou dependência de outros pacotes essas dependências são acessadas automaticamente por meio dos repositórios, assim a instalação é realizada sem maiores problemas e seus usuários não precisam buscá-las e instalá-las manualmente (FERREIRA, 2006).

Por padrão na distribuição educacional 5.0 disponibiliza o gerenciador de pacotes Synaptic e este foi utilizado no desenvolvimento prático deste trabalho.

3.5.1 Gerenciador de pacotes Synaptic

O gerenciador Synaptic vem incorporado nativamente a distribuição educacional 5.0, podendo ser instalado em versões anteriores sem maiores problemas. Ele oferece o poder do *apt-get* ao usuário através de uma interface gráfica simples (UBUNTU, 2014).

O comando *apt-get* é derivado da ferramenta *Advanced Packing Tool* (APT) desenvolvida e utilizada primeiramente nas distribuições Linux Debian e Ubuntu, utilizado no terminal Linux pelo usuário *root* este comando e seus parâmetros proporcionam ao usuário a instalação, remoção e atualização de pacotes e suas dependências de forma rápida e prática (DEBIAN, 2014).

Quando o comando é acionado é feita uma consulta no arquivo denominado como *sources.list*, que contém o endereço dos repositórios dos quais serão obtidos os pacotes que o usuário requisitar instalação (DEBIAN, 2014).

Assim como outros comandos o *apt-get* possui algumas derivações, as principais serão listadas abaixo:

a) apt-get update: Usado para atualizar os arquivos de índices dos pacotes a partir de suas fontes. Os índices são obtidos a partir do local especificado em */etc/apt/sources.list,* diretório encontrado em sistema GNU/Linux*.* Uma atualização utilizando este comando deve ser realizada sempre antes de *apt-get* upgrade ou *apt-get dist-upgrade*;

b) apt-get upgrade: Usado para obter as versões mais recentes de todos os pacotes instalados no sistema, obtendo-os das fontes localizadas no */etc/apt/sources.list;*

c) apt-get dist-upgrade: além de realizar a função de atualização, essa opção lida inteligentemente alterando as dependências com novas versões dos pacotes, atualizando assim o sistema para a última versão disponível;

d) apt-get install pacote: instala ou atualiza pacotes de software, onde ao lugar de pacote o usuário deve fornecer o nome do pacote desejado, obtendo ou atualizando o mesmo juntamente com suas dependências;

e) apt-get remove pacote: similar ao comando *apt-get install*, sendo que este ao invés de instalar ou atualizar remove o pacote e suas dependências.

Todas as opções citadas anteriormente estão disponíveis através do gerenciador de pacotes Synaptic, sendo elas apresentadas de forma amigável ao usuário, através da *interface* gráfica.

3.6 LINUX EDUCACIONAL

A distribuição Linux Educacional atualmente é baseada em uma das distribuições mais populares, a distribuição Ubuntu versão 12.04, tendo seu foco direcionado para laboratórios de informática educacional. Anteriormente as versões 4.0 e 3.0 eram baseadas nas distribuições estáveis de Debian e Kubuntu respectivamente.

A primeira versão foi desenvolvida pelo MEC juntamente com o Centro de Experimentação em Tecnologia Educacional (CETE), com o intuito de ajudar professores no preparo de suas aulas e inserir a tecnologia como ferramenta de ensino e aprendizagem, hoje o sistema encontra-se na versão 5.0, desenvolvida pelo Centro de Educação Científica e Software Livre (C3SL) na Universidade Federal do Paraná (LINUX EDUCACIONAL, 2014).

Utilizado desde o ano 2007 no PROINFO, mediante portaria número 522/MEC, de 9 de abril de 1997, regulamentada pelo decreto de número 6.300 em 12 de dezembro de 2007, o sistema Linux Educacional é instalado em escolas da área rural e urbana cadastradas no programa, todas as escolas passam por uma avaliação, se aprovadas ganham o direito de possuir um laboratório de informática, neste laboratório todos os computadores possuem o sistema, com objetivo de promover o uso pedagógico de tecnologias de informática e comunicação, além de difundir o uso de softwares livres (FUNDO NACIONAL DE DESENVOLVIMENTO DA EDUCAÇÃO, 2014).

Atualmente o Brasil possui noventa e oito prestadores de serviços especializados em treinamentos para Linux Educacional, pessoas físicas e jurídicas, destes prestadores três são do estado de Santa Catarina e trabalham como autônomos oferecendo tais serviços (PORTAL DO SOFTWARE PÚBLICO BRASILEIRO, 2014).

3.6.1 Funcionamento

Os parágrafos a seguir foram baseados no site oficial do sistema Linux Educacional, mantido pela Universidade Federal do Paraná.

Devido à versão educacional ser uma distribuição modificada de sistemas GNU/Linux, o mesmo é estruturado em camadas, possuindo gerenciamento de processos, memória, dispositivos de entrada e saída separadamente. Através da *interface* GNOME modificada especialmente para fins educacionais possui grande interatividade (figura 8). Sua versão atual possui uma barra de ferramentas chamada Edubar, aplicação feita em linguagem Java presente desde a versão 3.0 do sistema, proporcionando acesso rápido a conteúdos disponibilizados pelo MEC e governo federal, sendo estes:

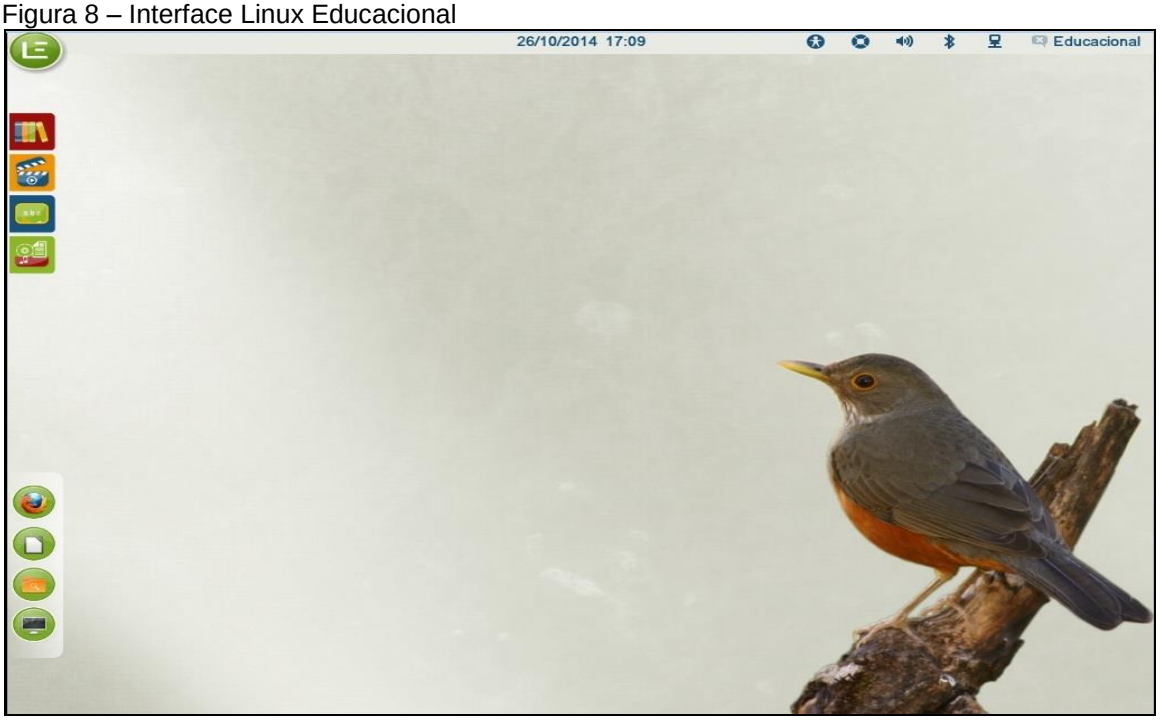
a) portal do domínio público: biblioteca virtual que contém obras de domínio público, acervo de livros, imagens e músicas;
b) portal do professor: visa auxiliar o professor nas atividades de ensino, apresentando notícias da área, oferece cursos de ferramentas para criação de aulas colaborativas e individuais;
c) banco internacional de objetos educacionais: busca online de objetos educacionais disponibilizados em várias línguas, abrange conteúdos de educação infantil até ensino superior;
d) repositório Debian: contém pacotes Debian com conteúdos educacionais, instaláveis para consulta local no Linux Educacional. Nele são compilados conteúdos encontrados no Portal Domínio Publico, Portal do Professor e Banco Internacional de Objetos Educacionais.

Na parte superior o sistema possui o botão *LE*, botão este que permite o gerenciamento de janelas, aplicativos e busca de conteúdos educacionais para professores e alunos. Além da Edubar há uma segunda barra que possui aplicativos como BrOffice, Mozilla Firefox, pasta do usuário e terminal shell.

É oferecida também à versão multiterminal do sistema, desenvolvida e mantida pela empresa *Userful,* neste sistema vários monitores, teclados e mouses podem ser ligados em uma mesma máquina, gerenciados pelo software *Userful*

Multiplier, neste modo todos os recursos e periféricos são divididos entre os terminais.

Figura 8 – Interface Linux Educacional



Fonte: Do autor.

3.6.1.1 Atualização automática

Recentemente foi adicionada ao Linux Educacional uma ferramenta de atualização automática através de pacotes de atualização Debian, as funções destes pacotes vão desde realizar alterações na configuração do sistema operacional até remoção e instalação de softwares.

Para armazenamento são utilizados ao todo três repositórios, cada um tem sua função definida, onde o *le5-unstable* é utilizado para desenvolvedores submeterem e testarem suas modificações. Após realização dos testes iniciais os pacotes são enviados para o *le5-testing* onde são verificadas e validadas pela equipe de testes todas as mudanças propostas e realizadas pelos desenvolvedores. Aprovada a modificação o pacote é enviado ao repositório *le5-stable*, acessíveis pelas máquinas nas escolas equipadas com Linux Educacional 5.0.

Nas escolas o pacote conhecido como *le-autoupgrade* realiza verificações diariamente procurando por mudanças no repositório *le5-stable*, se houver

mudanças elas são aplicadas ao sistema, deixando-o assim sempre atualizado com as versões mais recentes dos pacotes disponibilizados. Seu funcionamento é baseado em *shell script,* executado pelo *cron*[3], que realiza o processo de download e instalação de novos pacotes.

Em horários onde o pacote *le-autoupgrade* está em execução a rede pode ser saturada, tornando-a lenta durante o processo.

3.6.2 Vantagens e desvantagens

Assim como em outras distribuições Linux, a distribuição Linux Educacional possui suas vantagens, a primeira delas é o baixo custo, visto que aplicativos são todos de código aberto, livres, não havendo nenhum tipo de cobrança referente a eles. Outras vantagens apontadas são a confiabilidade dos softwares livres, diversidade dos tipos de licença, proporcionando o uso para variados fins, permitindo ainda a customização das ferramentas, além de ser uma alternativa para usuários que procuram softwares não pagos, evitando assim o uso de softwares piratas (SILVEIRA, 2011).

A distribuição possui também suas desvantagens a mais aparente e que dificulta sua adoção em um maior número de escolas é a dificuldade em se trabalhar com o sistema, devido ao despreparo dos professores, principalmente em escolas de locais onde há menos recursos, que não oferecem cursos de especialização. Faltam manuais de utilização para softwares com propósito educacional e não há órgãos que fiscalizam e auxiliam no desenvolvimento dos mesmos, adequando-os as diretrizes curriculares brasileiras (FERREIRA, 2010).

Em sistemas multiterminais dificuldade encontrada pelos usuários é proporcionada devido à divisão de recursos entre os terminais, visto que o software gestor consome parte destes recursos e o restante é compartilhado entre os demais terminais, onde ao realizar determinadas tarefas que exigem maior processamento seus usuários acabam encontrando dificuldades (SOFTWARE LIVRE, 2014).

---

[3] Responsável por gerenciar e executar as tarefas agendadas na lista *crontab* (ROMERO, 2013).

4 SOFTWARE LIVRE E OPEN SOURCE

Quando se fala em software livre e software de código aberto, *open source*, muitos pensam estar se referindo a um mesmo conceito, mas como será visto a seguir há algumas diferenças entre estes termos, principalmente no que se refere à forma de distribuição de softwares.

4.1 SOFTWARE LIVRE

Os parágrafos a seguir baseiam-se no conteúdo de www.gnu.org, site oficial do projeto GNU, patrocinado pela Fundação do Software Livre.

O termo software livre surgiu em 1983, quando o cientista Richard Stallman lançou o projeto GNU, visando proporcionar mais liberdade aos usuários e desenvolvedores de software, logo em seguida em 1985 fundou a FSF, onde se estabeleceram as regras para que um software possa ser considerado livre realmente.

Softwares livres são aqueles que respeitam a liberdade e senso da comunidade dos usuários, estes por sua vez possuem a liberdade de executar, copiar, distribuir, estudar e melhorar o software, sendo assim controlam o programa e o que ele faz por eles.

Em resumo um programa é considerado software livre quando pode ser modificado e redistribuído gratuitamente ou não, modificações feitas para uso privado ou empresarial, não sendo necessário pedir permissão ou pagar a nenhuma pessoa ou organização.

Mesmo sendo livre recomenda-se seguir algumas regras quando for realizar tais modificações ou redistribuições, para que o software possa encaixar-se nas normas estabelecidas pela FSF.

4.2 SOFTWARE OPEN SOURCE

O rótulo *open source* surgiu em 1998 quando personalidades como Todd Anderson, Chris Peterson (*Foresight Institute*), Larry Augustin (*Linux International*), Sam Ockman (*Silicon Valley Linux User's Group*), John *maddog* Hall e Eric Raymond reunidos em uma conferência debateram a insatisfação que tinham diante da postura adotada pelos adeptos ao software livre, estes debates deram início *Open Source Initiative* (OSI).

A OSI é uma corporação sem fins lucrativos que defende os benefícios proporcionados pelos softwares de código aberto em relação às outras categorias de softwares, visando oferecer qualidade, confiabilidade, flexibilidade e menor custo a quem os utilizam (OPEN SOURCE INITIATIVE, 2014).

Software *open source* pode ser considerado todo software que tenha seu código disponível para consulta, mesmo que não se tenha a liberdade de modificá-lo ou distribuir uma versão modificada (CAMPOS, 2009).

Em softwares *open source* não há a preocupação com a moral e ética, como nos softwares livres, não há garantia de liberdade ao utilizador de utilizar partes de um código para gerar outros programas (AFONSO, 2012).

4.3 REGRAS DE UTILIZAÇÃO

Para utilização e redistribuição de softwares livres ou *open source* algumas regras devem ser respeitadas, essas regras são estabelecidas por meio de licenças de software como será visto a seguir.

Os parágrafos a seguir foram baseados nas informações encontradas nos sites www.gnu.org e www.opensource.org.

4.3.1 Software Livre e *General Public License* (GPL)

A licença GPL é uma licença *copyleft*, que ao contrário do *copyright*, garante a liberadade do usuário de compartilhar seus softwares, além de outros tipos de obras. Foi desenvolvida para garantir a liberdade de distribuição de softwares livres, cobrando por isso se desejar, garante também que usuários recebam os códigos-fonte ou possam obtê-los quando quiser, podendo assim alterá-los ou utilizar partes deles em novos programas livres. Essa licença tem como base quatro liberdades fundamentais que são:

a) liberdade para execução de um programa, para qualquer propósito;
b) liberdade para estudar e entender o funcionamento de um programa e adaptá-lo para as suas necessidades. Necessário acesso ao código fonte;
c) liberdade de redistribuir cópias de modo que você possa ajudar ao seu próximo;
d) liberdade de aperfeiçoar o programa e liberar aperfeiçoamentos, de modo que toda a comunidade seja beneficiada.

Ao redistribuir softwares utilizando este tipo de licença, que hoje é utilizada na grande maioria dos softwares livres há certas responsabilidades, como por exemplo, ao distribuir cópias de um programa o destinatário deve ter as mesmas liberdades do usuário que a distribuiu, o usuário deve garantir também que destinatário tenha como obter a cópia do software, além de mostrar os termos da licença GPL juntamente com o software, para que o outro usuário possa conhecer seus direitos.

Os desenvolvedores que utilizam este tipo de licença têm seus direitos protegidos em dois passos, são eles:

a) afirmando seus direitos autorais sobre o software, ou seja, deixando seu nome e versão do mesmo, em local adequado para isso, normalmente nas informações do software;
b) oferecendo a licença GPL que dá a permissão legal para copiar, distribuir e modificar o software.

Este tipo de licença não oferece nenhuma garantia de funcionamento de softwares livres e requer que qualquer versão modificada seja marcada como modificada, de modo que seus problemas não sejam atribuídos erroneamente a autores de versões anteriores. A licença assegura que não se deve utilizar de patentes para tornar o programa não livre.

Termos e condições de uso são vistos detalhadamente no site oficial do projeto GNU[4].

4.3.2 Software Livre e *Leeser General Public Licence* (LGPL)

A licença LGPL é voltada para o uso de bibliotecas de software, desenvolvimento, cópia e modificação, tendo como objetivo permitir que usuários consigam ligar seus softwares livres a bibliotecas não livres ou ainda desenvolver bibliotecas livres que serão utilizadas não apenas em softwares livres, mas também em proprietários. A grande diferença da licença GPL é que quando um programa é ligado a uma biblioteca, os dois passam a trabalhar em conjunto, sendo assim a licença não atende os requisitos necessários, pois preza que tudo ligado ao software, incluindo bibliotecas devem obedecer a seus critérios de liberdade, para as exceções foi criada a licença LGPL.

Por oferecer menos liberdade aos usuários, visto que não restringe a modificação e o uso de bibliotecas apenas em softwares livres, podendo incorporá-las em softwares proprietários, não é a mais indicada pela FSF, mas esta também não restringe seu uso. Termos e condições de uso são vistos detalhadamente no site oficial do projeto GNU[5].

---

[4] http://www.gnu.org/licenses/gpl-3.0.html.

[5] https://www.gnu.org/licenses/lgpl.html.

4.3.3 Software Livre e *Aferro General Public License* (AGPL)

Está licença é uma versão modificada da licença GPL que tem como objetivo assegurar que softwares livres utilizados como serviços, *Software as a Service* (SaaS), rodando em servidores, quando modificados tenham seu código fonte disponibilizado, que os administradores destes servidores disponibilizem todos os códigos e suas modificações em seus servidores, para que todas as comunidades possam usufruir das mesmas, já que a licença GPL não garante tais condições, ou seja, o software pode ser modificado, disponibilizado para o uso de todos, mais seu código fonte e binários não. Termos e condições de uso são vistos detalhadamente no site oficial do projeto GNU[6].

4.3.4 Software *Open Source*

Para distribuição de softwares *open souce* deve-se obedecer alguns critérios assim como os softwares livres, esses critérios são onde se baseiam as muitas licenças disponíveis para esta categoria de software. Os critérios são:

a) redistribuição livre: a licença não deve restringir nenhuma parte de vender ou doar o software como um componente de uma distribuição agregada de software, contendo programas de várias fontes diferentes. A licença não deve exigir um *royalty* ou outra taxa para tal venda;

b) código fonte: o programa deve incluir código fonte, bem como em formato compilado. Quando um produto não é distribuído com o código fonte deve haver um meio bem divulgado de obter o mesmo. O código fonte deliberadamente ofuscado não é permitido. Formas intermediárias como a saída de um pré-processador ou tradutor não são permitidas;

---

[6] http://www.gnu.org/licenses/agpl.html.

c) derivados: a licença deve permitir modificações e trabalhos derivados, permitir-lhes ser distribuído sob os mesmos termos da licença do software original;

d) integridade do código fonte do autor: a licença pode restringir o código fonte de ser distribuído em forma modificada somente se a mesma permitir a distribuição de *patch files* com o código fonte para o propósito de modificar o programa em tempo de compilação. Deve permitir a distribuição de software construído a partir do código fonte modificado e pode exigir que trabalhos derivados tenham um nome ou número de versão diferente do software original;

e) sem discriminação contra pessoas ou grupos: a licença não deve discriminar qualquer pessoa ou grupo de pessoas;

f) sem discriminação contra campos de trabalho: a licença não deve restringir ninguém de fazer uso do programa em um campo específico de atuação, ela não pode restringir o programa de ser usado em uma empresa, ou de ser usado para pesquisa genética;

g) distribuição da licença: o direito associado ao programa deve aplicar-se à todos a quem o programa é redistribuído, sem a necessidade de execução de uma licença adicional por aquelas pessoas;

h) licença não pode ser específica a um produto: Se parte de um programa for extraída e utilizada ou distribuída dentro dos termos da licença do programa, todas as partes para quem o programa é redistribuído devem ter os mesmos direitos que aqueles que são concedidos em conjunto com a distribuição do software original;

i) licença não deve restringir outro software: a licença não deve colocar restrições em outro software que é distribuído juntamente com o software licenciado. Por exemplo, a licença não deve insistir que todos os programas distribuídos na mesma mídia sejam softwares *open source*;

j) licença deve ser tecnologicamente neutra: nenhuma disposição da licença pode ser baseada em qualquer tecnologia individual ou de estilo de *interface*.

Dentre as muitas licenças aprovadas pela OSI está inclusive à licença GPL e outras bastante utilizadas como é o caso da *Apache License, Mozilla Public License* (MPL), *MIT License (MIT)* e *Eclipse Public License* (EPL). As demais licenças são vistas detalhadamente no site da inciativa *open source*[7].

[7] http://opensource.org/licenses.

5 TRABALHOS CORRELATOS

Neste capitulo serão apresentados projetos e trabalhos de pesquisa desenvolvidos, que possuem semelhanças com a ferramenta desenvolvida neste trabalho.

5.1 MENTOR: UM AMBIENTE PARA A DISTRIBUIÇÃO TRANSPARENTE DE SOFTWARE

Projeto apresentado por Maurício Emanuel Dourado Cescato no Instituto de Matemática e do Núcleo de Computação Eletrônica da Universidade do Rio de Janeiro, no ano de 2005 para obtenção do grau de Mestre em Informática.

O Mentor é um sistema de distribuição de softwares desenvolvido em linguagem Java tendo como objetivo criar uma infraestrutura para disponibilização de softwares e suas atualizações envolvendo o usuário o menos possível, pois o autor acredita que este processo pode ser feito automaticamente.

A maneira de distribuição utilizada foi o ambiente de rede, seguindo o modelo de comunicação ponto a ponto, onde todas as máquinas trabalham como cliente e servidor, podendo dividir a carga de execução dos processos.

5.2 CLUMPT: UM SISTEMA PEER-TO-PEER PARA INSTALAÇÃO E MANUTENÇÃO DE SOFTWARE EM AGLOMERADOS DE COMPUTADORES

Projeto apresentado no 24º Simpósio Brasileiro de Rede de Computadores, no II *Workshop* de *Peer-Two-Peer* realizado no ano de 2006, tendo como autores, Diego Kreutz, Marcelo Neves, Elton Mathias, Tiago Scheid, Andrea Charão, Rafael Righi.

Clumpt é um sistema para gerenciamento de instalação e manutenção de softwares em *clusters*, sua arquitetura é baseada em um modelo P2P híbrido, obtendo e armazenando dados sobre o estado dos *peers* em uma base de dados externa, onde cada *peer* é um nó de um cluster gerenciado. Fornece ainda a possibilidade de um *peer* manter seu próprio repositório local, podendo atuar como cliente ou servidor se necessário, para uma fonte para obtenção de pacotes no cluster.

Foi desenvolvido com o objetivo de minimizar as dificuldades no processo de atualização de softwares em *clusters*, permitindo a automatização do processo, visto que outras ferramentas propõem apenas a replicação por completo dos sistemas operacionais ou instalações parciais de softwares, quando já há sistemas instalados. Nele o agrupamento dos *peers* é configurado manualmente pelos administradores do sistema, através de um arquivo de configuração chamado de *clumptconfig*, neste arquivo também são configuradas as regras para envio e recebimento de atualizações entre os nós.

Referente ao seu funcionamento, após a inicialização o sistema realiza uma busca na base de dados, obtém a listagens dos nós e a estrutura de grupos que modelam a rede, por fim faz a análise do histórico das últimas atualizações realizadas. Possuindo tais informações se conecta a rede, quando conectado verifica a existência de instalações pendentes nos *peers* vizinhos, garantindo que todas as máquinas atinjam a homogeneidade de software em relação ao restante do grupo.

5.3 TONTECH – PATCH MANAGEMENT

Projeto publicado na Revista de Ciências Exatas e Tecnologia, volume III no ano de 2008, tendo como idealizadores do projeto Luis Henrique Medeiros da Rocha, Rodrigo Xavier de Souza, Rogers Vicco Paredes, Wellington Campari e Jeanne Dobgenski.

O TonTech – Patch Management é uma alternativa para gerenciamento e controle de atualizações voltadas a segurança em sistema operacionais Linux.

Este mecanismo busca minimizar o tempo gasto para realização de atualizações em um parque de máquinas, mais especificamente servidores, caso processo fosse realizado manualmente. Voltado a questões de segurança a ferramenta visa reduzir vulnerabilidades de segurança dos ambientes de forma eficaz, através do processo de atualização.

Em sua estrutura há um servidor que coleta informações dos pacotes de softwares atuais disponíveis nos repositórios oficiais, podendo compará-los com os pacotes já instalados em outros computadores monitorados pela ferramenta, essas informações são armazenadas em cinco tabelas no banco de dados.

A ferramenta TonTech possui interface web, onde há a possibilidade de definir parâmetros para as atualizações, cadastrar e gerenciar as máquinas, podendo analisar os pacotes desatualizados, erros, agendar inventários e instalações.

5.4 SISTEMA GERENCIADOR DE SCRIPTS

Trabalho apresentado por Bruno Romano Müller para obtenção do grau de Tecnólogo em Sistemas para Internet pela Universidade Tecnológica Federal do Paraná no ano de 2011.

Este sistema tem como objetivo reduzir os problemas encontrados por administradores de rede em sistemas GNU/Linux, no que se refere à coleta de dados, geração de eventos e configuração de uma rede de computadores. O sistema trabalha sobre o modelo cliente / servidor, contando com um computador central, onde se encontram as funcionalidades e *scripts*, através destes os clientes tem a disposição todos os dados necessários para execução de tarefas e coleta de informações.

Após execução dos *scripts* todos os dados são tratados pelas máquinas clientes, respeitando algumas regras para que sejam importados em um banco de dados. Toda comunicação entre cliente e servidor é feita através de um módulo próprio de troca de mensagens desenvolvido exclusivamente para o sistema, visto

que para o cliente são enviados os códigos fonte do *script* escolhido e o retorno do cliente também é enviado via formato de texto.

Todo o sistema é gerenciado via *interface web*, desenvolvida em Java que se comunica com o banco de dados SQLite, onde ficam armazenadas as informações enviadas pelos computadores clientes.

Através desta interface há a possibilitada de gerenciar e realizar a inserção de novos usuários e grupos de clientes, assim como cadastro de novos *scripts* e agendamento de datas e horários para execução dos mesmos. Há ainda a possibilidade da consulta dos dados e tarefas executadas, através da geração de relatórios.

6 INSTALAÇÃO E ATUALIZAÇÃO MASSIVA DE APLICAÇÕES

Este capítulo apresenta o desenvolvimento do trabalho final que visa realizar a instalação e atualização de softwares para sistema Linux Educacional via ambiente de rede, para o desenvolvimento da ferramenta são utilizados softwares de código aberto.

O objetivo da ferramenta desenvolvida é possibilitar que através de uma estação definida como servidor, se consiga gerar *script*s de atualização e que estes sejam enviados via ambiente de rede, sendo lidos na sequência pelas máquinas clientes conectadas. O *script* a princípio é gerado através do software de gerenciamento de pacotes Synaptic, disponível na distribuição Educacional do sistema GNU/Linux, software este que se conecta a repositórios oficiais e não oficiais, realizando o *download* dos pacotes de arquivos necessários para instalação ou atualização de software. Posteriormente estes *scripts* são selecionados no servidor através de uma interface de gerenciamento e o mesmo os enviará para máquinas clientes conectadas pela rede local, para o envio é necessário o uso de chamadas de sistema, onde se optou pelo uso de *socket*, que fará a ligação do servidor e seus clientes.

Após conclusão deste trabalho pretende-se distribuir a ferramenta em comunidades relacionadas, para que assim outros usuários possam contribuir em seu desenvolvimento.

6.1 METODOLOGIA

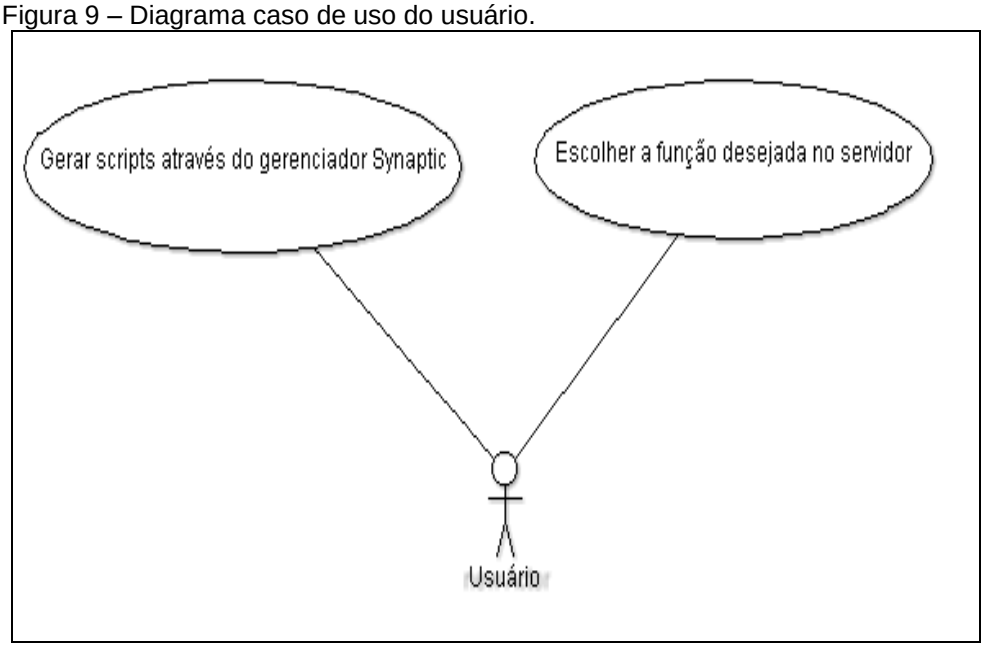
Para realização dos objetivos propostos foi necessário o cumprimento de algumas etapas metodológicas, onde primeiramente foi realizado o levantamento bibliográfico, buscando aprofundar o conhecimento sobre sistema operacional, seu processo evolutivo e suas diferentes estruturas. Posteriormente foi estudado o sistema GNU/Linux e sua distribuição educacional, foi pesquisado também os tipos

de licenças de softwares e suas diferenças, para usá-las de forma correta no decorrer do desenvolvimento, devido à utilização de softwares livres e *open source*.

Dando continuidade ao projeto foram pesquisadas alternativas de software que realizam atualização e instalação automática conectando-se diretamente aos repositórios oficiais de softwares disponibilizados para o sistema Linux e que posteriormente proporcionasse ao usuário a geração de um *script* de atualização, o que seria de fundamental importância para o funcionamento do projeto desenvolvido. Com o software definido foi escolhida a linguagem de programação utilizada e a chamada de sistema usada para que a ligação dos computadores em rede local pudesse ser realizada.

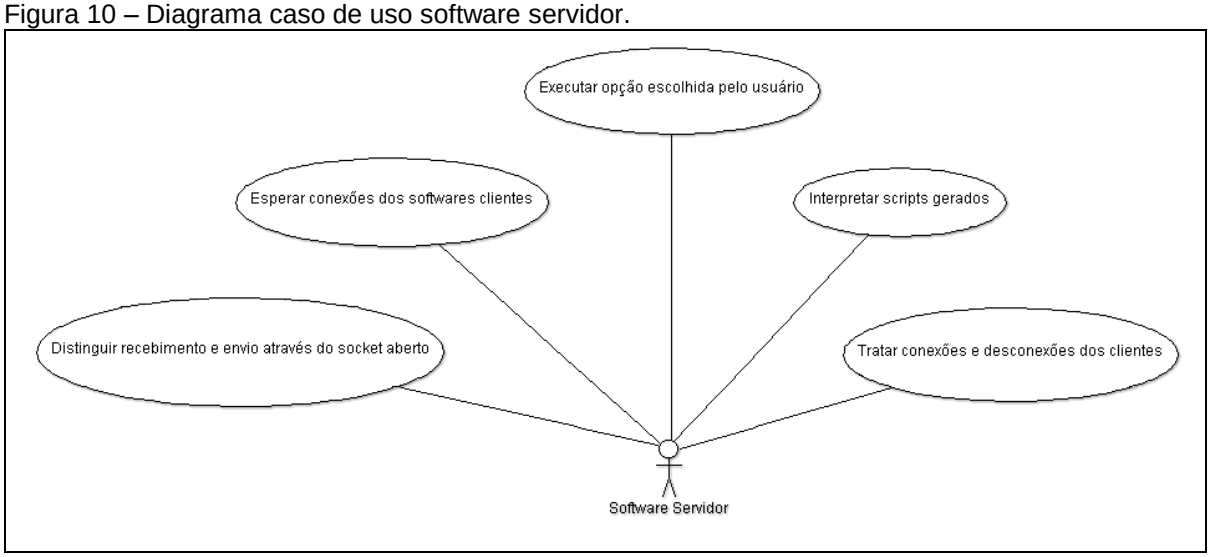
Após tais levantamentos e definições foi dado início ao desenvolvimento da ferramenta, modelando primeiramente seus processos, os quais serão apresentados a seguir nas figuras 9, 10, 11, 12 e 13.

Figura 9 – Diagrama caso de uso do usuário.



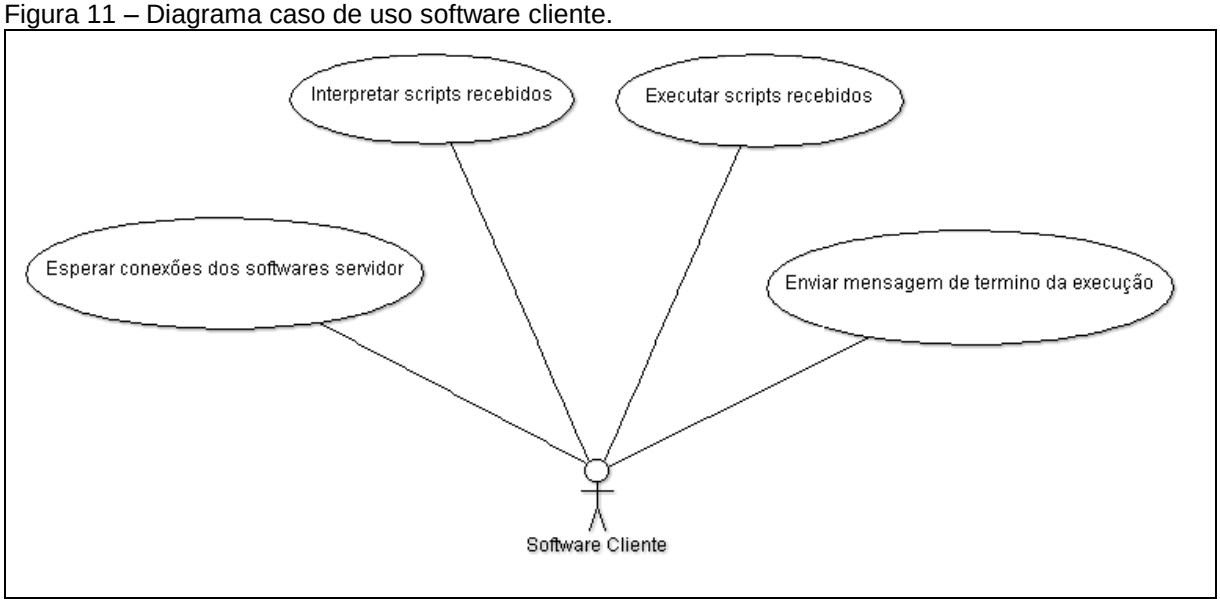
Fonte: Do autor.

Figura 10 – Diagrama caso de uso software servidor.



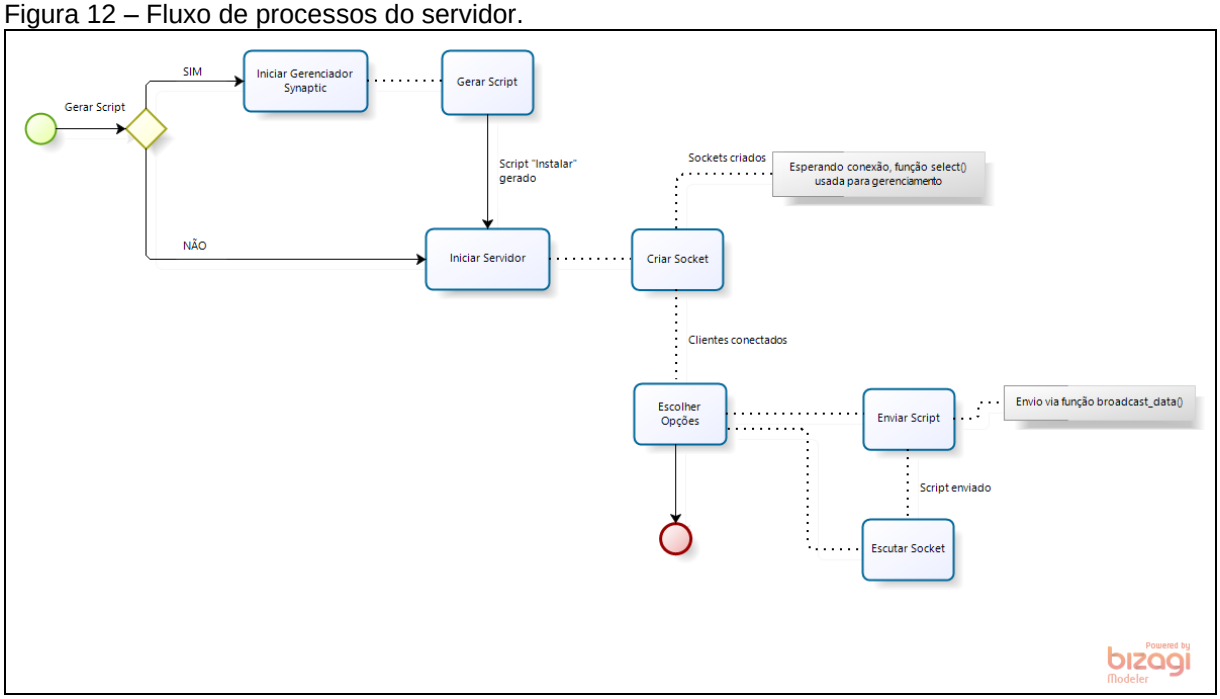
Fonte: Do autor.

Figura 11 – Diagrama caso de uso software cliente.



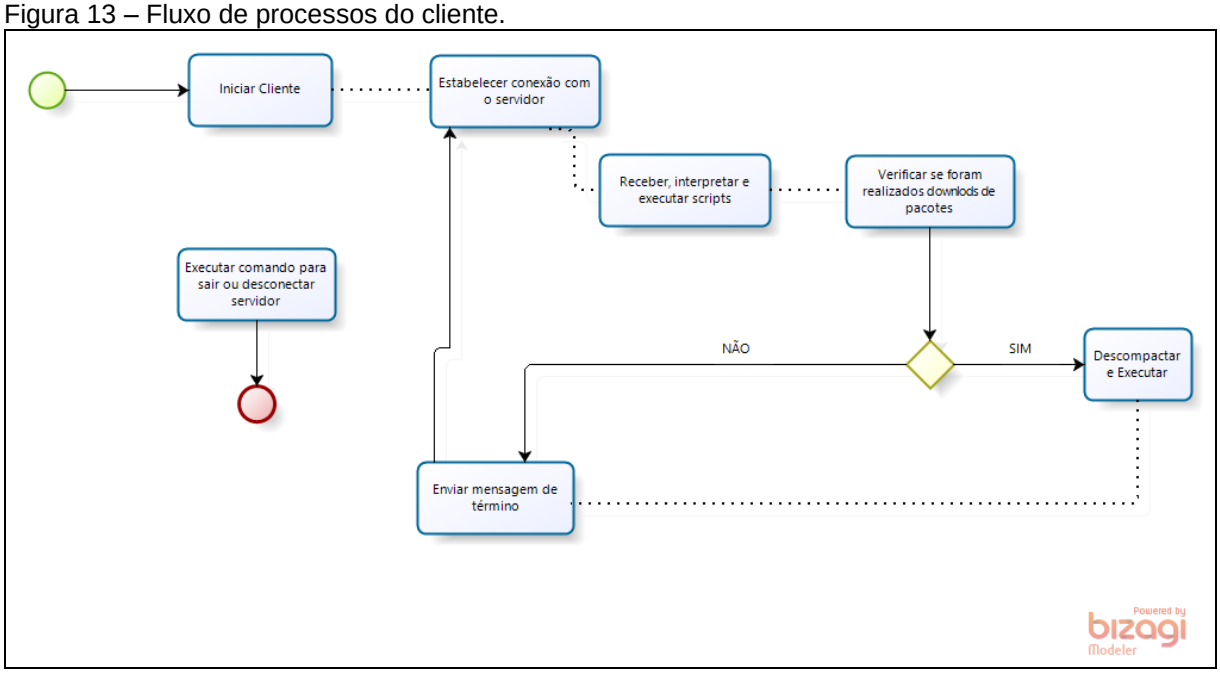
Fonte: Do autor.

Figura 12 – Fluxo de processos do servidor.



Fonte: Do autor.

Figura 13 – Fluxo de processos do cliente.



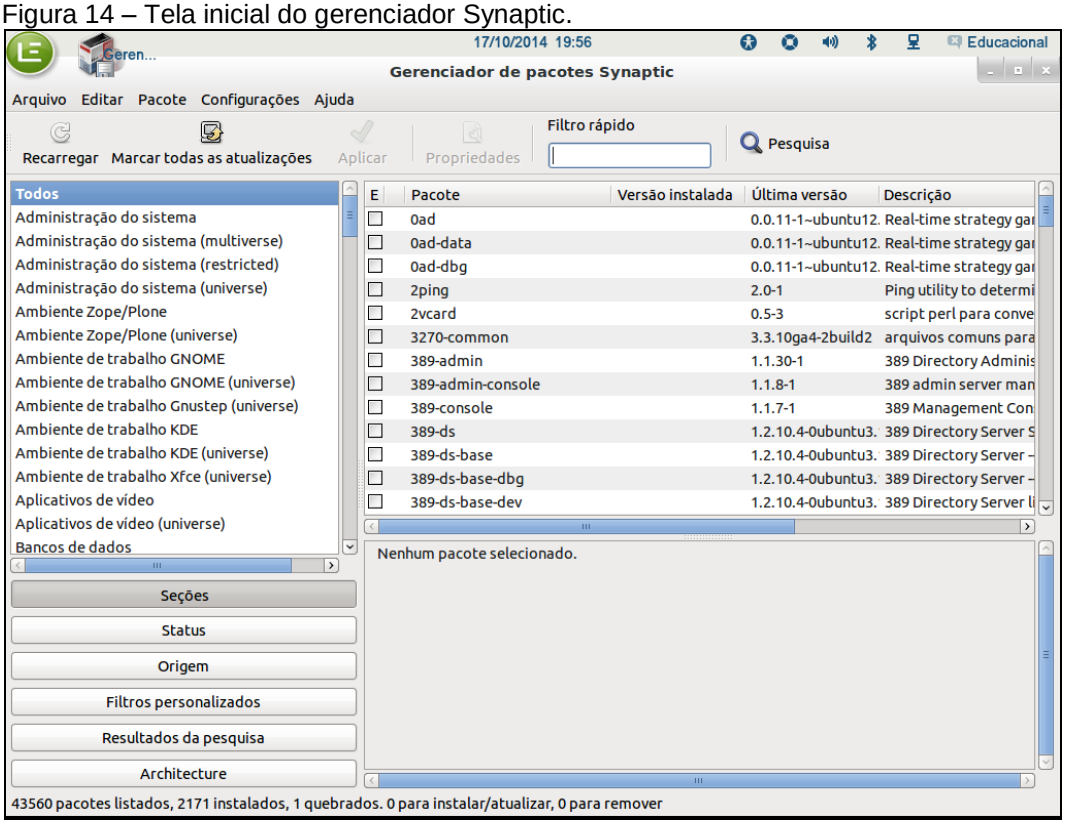
Fonte: Do autor.

6.1.1 Geração de scripts pelo Synaptic

Para geração dos *scripts* é necessário acessar o Synaptic através do sistema operativo, esse acesso ao gerenciador é disponibilizado de duas maneiras, uma delas é como usuário *root* e outra como usuário padrão, através do usuário *root* todas as funções do gerenciador estão disponíveis, utilizando o usuário padrão não será permitida qualquer alteração no sistema, mas há a possibilidade de escolha e geração dos *scripts* de atualização.

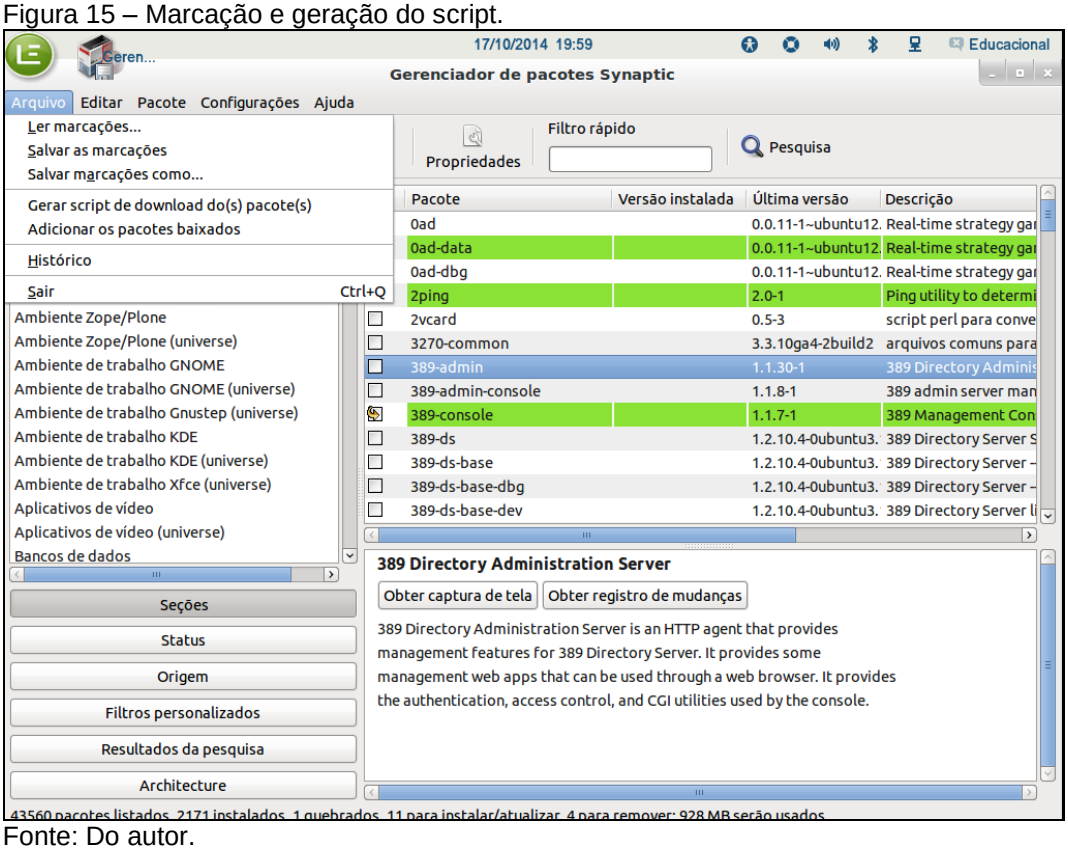
Para geração de um *script* de instalação ou atualização, estando com o gerenciador aberto há a possiblidade de escolha de quais pacotes deseja instalar, caso os pacotes selecionados dependam de algum outro pacote o gerenciador emite uma mensagem informando da necessidade, deixando o usuário decidir se instala ou não tais dependências, permite ainda a pesquisa de pacotes por categorias. Oferece também a opção de marcar todas as atualizações, a qual selecionará automaticamente todas as atualizações pendentes do sistema, após ter selecionado os pacotes basta ir à barra de menus na guia arquivo e realizar a geração do arquivo *shell script,* que será enviado pelo servidor para instalação de pacotes nas demais máquinas clientes (figura 14 e 15).

Figura 14 – Tela inicial do gerenciador Synaptic.



Fonte: Do autor.

Figura 15 – Marcação e geração do script.



Fonte: Do autor.

Realizado tais processos basta escolher o local de destino e o nome que deseja dar ao arquivo de *script*. Para que haja um melhor funcionamento foi definido o nome do arquivo como “Instalar” sendo este gerado na pasta onde se encontra o executável da ferramenta desenvolvida, assim o *script* já está pronto para ser enviado para os outros computadores através da rede local.

6.1.2 Software servidor

Na implementação do software servidor primeiramente foi declarado o tipo de codificação utilizada, neste caso foi escolhida a *utf-8*, por reunir o melhor das codificações ASCII e Unicode. Em seguida foi realizado os *imports,* permitindo a importação e uso de módulos durante o desenvolvimento do código, dentre eles o módulo *socket* (figura 16).

Figura 16 – Importação dos módulos necessários.

```
import socket
import select
import sys
import commands
```

Fonte: Do autor.

Dando prosseguimento ao código é declarada a variável *intf* que tem o valor *eth0* atribuído à ela, referindo-se a *interface* de rede do computador, essa variável é utilizada em uma nova função, a *commands.getoutput* (figura 17), que obtém apenas o endereço IPV4 da *interface* de rede definida anteriormente, sua saída é atribuída a uma nova variável *intf_ip*. O uso dessa função foi necessária devido ao utilizar outras funções o retorno era sempre o endereço IP conhecido como *localhost*, ou seja, 127.0.0.1. Para seu uso foi necessária a importação do módulo *commands*, que permite comunicação direta com a shell Linux, possibilitando obter informações e configurações do sistema.

Figura 17 – Função para retorno do endereço IP.

```
intf = "eth0"
intf_ip = commands.getoutput("ip address show dev " + intf).split()
intf_ip = intf_ip[intf_ip.index("inet") + 1].split("/")[0]
```

Fonte: Do autor.

Dando continuidade são declaradas as variáveis que armazenarão a lista de conexões legíveis, tamanho de *buffer e* porta, definida para comunicação com os programas clientes. Logo após é criado o *socket*, onde são passados dois parâmetros, indicando que o mesmo trabalhará em um domínio de *Internet*, através do protocolo TCP/IP. Abaixo deste é definido que em casos onde o programa servidor tenha sua conexão encerrada inesperadamente seu *socket* deverá ser fechado imediatamente, esta regra foi definida para que ao iniciar novamente o servidor a mensagem "*Address already in use"*, ou seja, que o endereço do servidor já está em uso, não seja apresentada. Temos ainda a passagem do endereço de IP e porta onde o *socket* estará ligado e a quantidade de conexões que poderão ficar em espera durante o atendimento de outras conexões, na maioria dos sistemas é estabelecida cinco conexões em espera (figura 18).

Figura 18 – Definições do *socket.*

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((intf_ip, PORTA))
server_socket.listen(5)
```

Fonte: Do autor.

Dando continuidade ao código o *socket* criado e adicionado à lista de conexões legíveis, juntamente com o as opções que serão escolhidas através do menu pelo usuário (figura 19), na sequência é disponibilizado ao usuário o menu de opções para que o mesmo possa escolher que tipo de *script* irá enviar.

Figura 19 – Criação da lista de conexões.

```
LISTA_CONEXAO.append(server_socket)
LISTA_CONEXAO.append(sys.stdin)
```

Fonte: Do autor.

Prosseguindo há uma condição *while*, nesta é utilizada a função *select*, passando os parâmetros *read_sockets*, *write_sockets* e *error_sockets*, sendo eles

três *arrays* independentes, proporcionando assim um controle dos *sockets* prontos para leitura, escrita ou que apresentaram algum erro, ainda no *while* temos um *for* que verifica se socket está dentro do conjunto de lidos, se verificado e nele contiver o endereço do servidor, o mesmo aceita novas conexões de clientes, criando novos *sockets* e endereços para cada um, adicionando-os posteriormente na lista de conexões legíveis, caso essa condição não seja aceita é verificado se há solicitação pendente a enviar para o *socket* lido ou ainda se o ele tem algo a receber do cliente, caso tenha, o conteúdo é mostrado na tela da aplicação do servidor. Se nenhuma dessas condições é atendida o socket é fechado e removido da lista de conexões legíveis (figura 20).

Figura 20 – Controle de envio e recebimento de dados dos clientes.

```
while True:
    read_sockets, write_sockets, error_sockets = select.select(LISTA_CONEXAO, [], [])

    for sock in read_sockets:
        if sock == server_socket:

            nsock, addr = server_socket.accept()
            LISTA_CONEXAO.append(nsock)
            print "\n"
            print "Cliente (%s, %s) conectado " % addr
            Opcao()

        elif sock == sys.stdin:
            opcao = sys.stdin.readline()
            Tratamento(int(opcao))
        else:
            try:
                data = sock.recv(BUFFER)

                if data:
                    print "" + data
                    Opcao()
            except:
                    sock.close()
                    LISTA_CONEXAO.remove(sock)
                    continue
```

Fonte: Do autor.

Ainda no servidor podemos ver na figura apresentada anteriormente que há uma chamada da função Opcao, que obtém a opção escolhida pelo usuário e a envia para a função Tratamento, que realiza a execução da tarefa escolhida, chamando caso necessário à função Envio (figura 21). Finalizando o ciclo de

trabalho do servidor há a chamada da função *Broadcast_data,* que recebe como parâmetro o *socket* e o *script* selecionado, enviando o mesmo para todos os clientes conectados, tratando de que o servidor não receba tais *scripts* (figura 22).

Figura 22 – Função de tratamento de opções e função de envio.

```
def Tratamento(opcao):
    global SCRIPT

    if opcao == 1:
        os.system('chmod +x Executar')
        os.system('./Executar')
        Opcao()
    elif opcao == 2:
        SCRIPT = "Instalar"
        Envio(SCRIPT)
    elif opcao == 3:
        SCRIPT = "Update"
        Envio(SCRIPT)
    elif opcao == 4:
        SCRIPT = "Upgrade"
        Envio(SCRIPT)
    elif opcao == 5:
        SCRIPT = raw_input("Digite o comando a enviar: " )
        print "Enviando....."
        cmd = "#!/bin/sh"+"\n"+str(SCRIPT)
        broadcast_data(sock, cmd)
    elif opcao == 6:
        SCRIPT = "Corrigir"
        Envio(SCRIPT)
    elif opcao == 7:
        SCRIPT = "Desligar"
        Envio(SCRIPT)
    elif opcao == 8:
        SCRIPT = "Reiniciar"
        Envio(SCRIPT)
    elif opcao == 9:
        os.system('gedit Uso')
        Opcao()
    elif opcao == 10:
        sys.exit()
    else:
        print "Opção Incorreta"
        Opcao()

def Envio(SCRIPT):
        print "Enviando....."
        arquivo = open(SCRIPT)
        SCRIPTarq = arquivo.read()
        Broadcast_data(sock, SCRIPTarq)
```

Fonte: Do autor.

Figura 23 – Função Broadcast_data.

```
def Broadcast_data(sock, script):

    for socket in LISTA_CONEXAO:
        if socket != server_socket and socket != sock and socket != sys.stdin:
            try:
                socket.send(script)
            except:
                socket.close()
                LISTA_CONEXAO.remove(socket)
```

Fonte: Do autor.

6.1.3 Software cliente

Na implementação do software cliente assim como no servidor foi utilizado o tipo de codificação *utf-8*, em seguida os *imports*, no cliente é importado um módulo que não estava presente no desenvolvimento do software servidor, o módulo *glob,* que possibilita a análise do conteúdo de pastas e arquivos.

Primeiramente é solicitado que o usuário entre com a senha do usuário *root*, assim todos os processos de instalação podem ser realizados sem maiores problemas, na sequência é chamada a função para criação do *socket* assim como no servidor, informando os parâmetros que indicam que trabalhará em um domínio de *Internet* e através do protocolo TCP/IP. No cliente também foi validado para que em caso de encerramento de conexão de forma inesperada, o cliente consiga se reconectar e não seja apresentada a mensagem de endereço já em uso.

Após esse processo o software cliente está pronto para conectar-se ao servidor, chamando a função *connect,* passando o endereço de IP do servidor e a porta de conexão (figura 24).

Figura 24 – Criando conexão com o servidor.

```
cliente_socket.connect((ipserver, int(porta)))
```

Fonte: Do autor.

Para obtenção desses parâmetros o software realiza uma consulta nas linhas do arquivo no arquivo *Config* (figuras 25 e 26), que guarda os endereços para conexão, tais endereços são lidos e armazenados nas variáveis passadas como

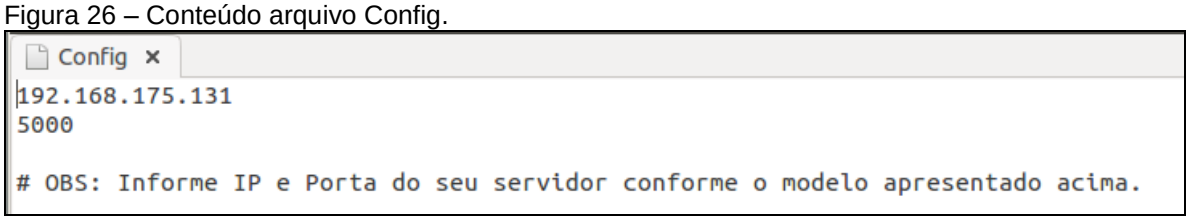
parâmetro. Este arquivo pode ser manipulado pelo usuário, adequando-o ao endereço do servidor na rede, podendo ser encontrado na pasta que contém o executável do software cliente.

Figura 25 – Leitura do arquivo Config.

```
arquivo = open('Config', 'r')
ipserver = arquivo.readline()
porta = arquivo.readline()
arquivo.close()
```

Fonte: Do autor.

Figura 26 – Conteúdo arquivo Config.



Fonte: Do autor.

Em seguida temos a criação de uma lista de que contém o endereço do *socket* criado e sua resposta ao servidor. Assim como no software servidor temos a função *select*, tendo o controle do *socket* em modo de leitura, escrita ou com erros.

Posteriormente é feita uma verificação através da condicional *for* (figura 27), onde é verificado se o *socket* faz parte da lista em modo leitura, se esta condição for respeitada é verificado ainda se o ele é o definido como cliente, em caso de resposta positiva, o cliente lê os dados enviados pelo servidor e os armazena em um arquivo de chamado ScriptRecebido. Depois de criado o arquivo de *script* é executado, fazendo uma verificação se durante a execução foi efetuado download de pacotes de softwares, essa verificação é feita através função *glob*, havendo pacotes de software eles são extraídos e executados, após o termino da execução são excluídos automaticamente. Ao final do processo é chamada a função Resposta (figura 28), que envia ao cliente o *status* do processo. Se processo anterior não for realizado significa que o servidor não está mais conectado, enviando dados, sendo assim o software cliente identifica que o servidor está desconectado e encerra a comunicação.

Figura 27 – Analise realizada pelo software cliente.

```
for sock in read_sockets:

    if sock == cliente_socket:
        dados = sock.recv(4096)
        arquivo = open('ScriptRecebido', 'wb')
        arquivo.writelines(dados)
        arquivo.close()
        os.system('chmod +x ScriptRecebido')
        os.system('./ScriptRecebido')
        if glob.glob('*.deb'):
            os.system("sudo dpkg -iGE *.deb -y")
            os.system("sudo rm *deb")
            Resposta()
        else:
            Resposta()

        if not dados:
            print '\n Desconectado do Servidor'
            sys.exit()
```

Fonte: Do autor.

Figura 28 – Função Resposta.

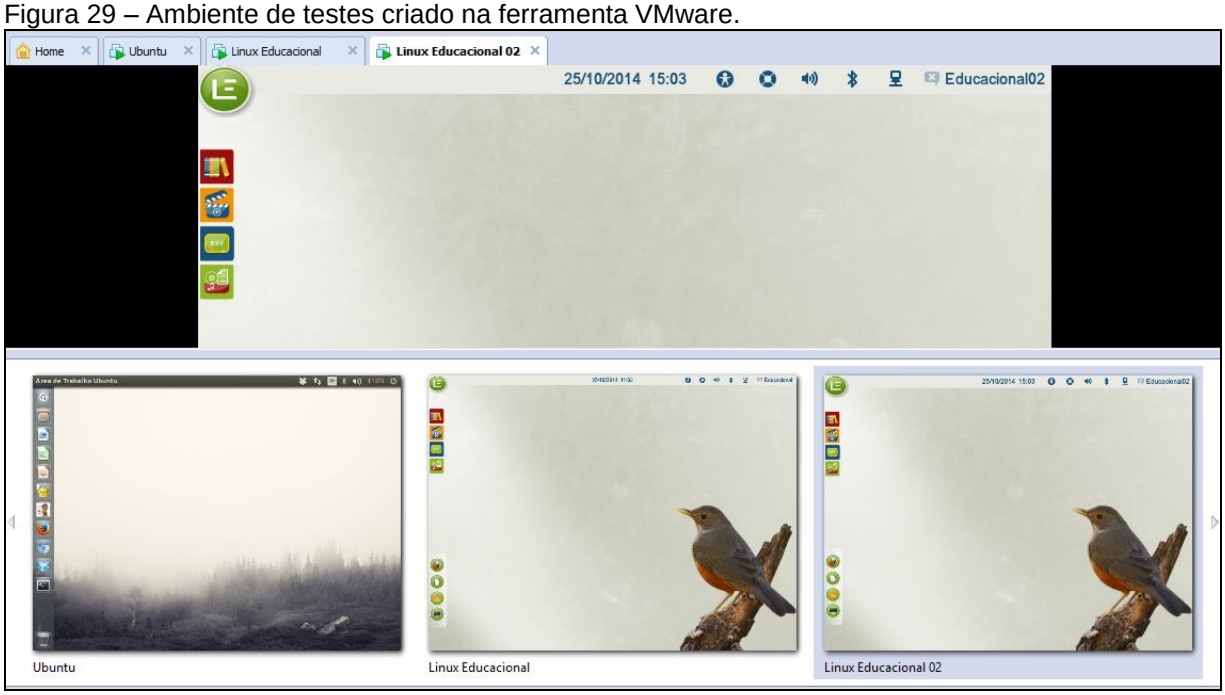
```
def Resposta():
    maq = socket.gethostname()
    status = "Processo em "+str(intf_ip)+ " >>> " + str(maq) + " foi finalizado"
    cliente_socket.send(status)
```

Fonte: Do autor.

6.3 RESULTADOS OBTIDOS

Concluída a etapa de implementação foram realizados os testes através de uma pequena rede formada de três máquinas virtuais, de onde se pode executar um grande número de aplicações cliente. Para criação deste ambiente foi utilizado um computador com sistema operacional Windows 8.1, processador Core i5 de 2.30Ghz e 4Gb de memória RAM. Para criação das máquinas virtuais foi utilizada a ferramenta VMware Workstation 10, onde foram criadas máquinas com sistema GNU/Linux, em uma delas foi instalada a distribuição Ubuntu 14.04 LTS, nas outras foi instalada a distribuição Linux Educacional 5.0 (figura 29).

Figura 29 – Ambiente de testes criado na ferramenta VMware.



Fonte: Do autor.

Nos testes realizados constatou-se que a ferramenta desenvolvida cumpre com o proposto, através de sua interface simples envia comandos e *scripts* solicitados da máquina definida como servidor às máquinas clientes conectadas, o que é um diferencial já que nas ferramentas de atualização analisadas no decorrer do trabalho teórico, o usuário muitas vezes não sabe ou não controla quais aplicações e pacotes estão sendo instalados.

Foi obtido sucesso também na alternativa de controle do *socket,* onde se utilizou o módulo e função *select* juntamente com listas de conexão, sendo que mesmo quando o servidor está em processo de envio de *scripts* ou recebimento de respostas vindas do cliente, permite que novas conexões sejam realizadas, ao contrário do que acontecia quando foram testadas alternativas utilizando *Threads* e *Fork,* onde todas as conexões deviam ser estabelecidas para que depois o *socket* fosse liberado para realização do processo de envio de dados.

Além de funções de instalação, foram desenvolvidos *scripts* com funções utilizadas cotidianamente pelos usuários como desligar, reiniciar as máquinas clientes, com o intuito de auxiliá-los ainda mais, foi dado ao usuário também a possibilidade de escrever seu próprio comando, proporcionando assim a realização de variadas tarefas.

Foram encontradas dificuldades na implementação de uma *interface* gráfica mais aprimorada, devido ao pouco conhecimento das bibliotecas e ferramentas disponíveis para integração com a linguagem Python, mas apesar disso o software funciona em diversas distribuições Linux, por utilizar recursos compatíveis com as mesmas.

7 CONCLUSÃO

A distribuição educacional do Linux vem sendo utilizada desde 1997 em escolas de todo o Brasil, proporcionando aos alunos conhecimento. Durante todos esses anos o sistema ainda não possui ferramentas de fácil compreensão que atendam uma rede de computadores de forma massiva, onde o responsável do laboratório pudesse escolher quais ações deseja realizar e essas ações fossem replicadas às máquinas conectadas a sua rede local, pensando nisto foi proposto o desenvolvimento deste trabalho, buscando auxiliar os profissionais, agilizando sua rotina de trabalho.

Logo no início do desenvolvimento deste trabalho uma grande dificuldade foi encontrar softwares semelhantes ao proposto para possível adaptação ou que estivesse em processo de desenvolvimento por outros acadêmicos. Com isso partiu-se para a ideia do desenvolvimento, onde durante a escolha da linguagem de programação houve a oportunidade de utilizar a linguagem Python, o que acabou sendo de grande valia, devido a sua integração com o sistema GNU/Linux, sem a necessidade de instalação de complementos para que funcionasse corretamente. Devido ao uso de seus módulos a comunicação com a shell Linux tornou-se mais simplificada, ajudando muito no desenvolvimento. Outra dificuldade encontrada foi a de gerenciar a entrada e saída de dados no *socket,* para que ele não ficasse esperando um processo ser finalizado para iniciar o outro, a qual foi contornada de forma satisfatória com a utilização do módulo *select* em conjunto com a chamada de sistema *socket*.

Através da utilização de sistema GNU/Linux e *scripts* foi proporcionado um grande ganho de conhecimento em relação ao funcionamento do sistema operacional e as maneiras que o mesmo tem de se comunicar com suas aplicações.

Apesar de outras dificuldades encontradas, as quais muitas surgiram pela falta de conhecimento até então das ferramentas utilizadas, o desenvolvimento pode ser concluído e testado de forma positiva, proporcionando o envio de comandos e *scripts* da máquina definida como servidor para todos os clientes conectados a ela através da rede, proporcionando a esses clientes instalação, atualização de pacotes

de software e envio de comandos diversos de forma centralizada, além de comodidade e agilidade ao trabalho de quem realiza tais procedimentos.

Considerando os conhecimentos adquiridos diante do desenvolvimento do trabalho, tanto do sistema GNU/Linux e suas distribuições, quanto das ferramentas utilizadas para desenvolvimento prático e principalmente a dificuldade que os profissionais têm em encontrar ferramentas do gênero, para trabalhos futuros, a fim de dar continuidade ao projeto pretende-se:

a) adicionar novas funções e *scripts* ao servidor, proporcionando o envio de comandos mais avançados, sendo estes executados no cliente de forma segura;
b) adicionar relatórios de *log*, contendo as principais informações dos processos realizados;
c) Implementar a interface gráfica, visando maior usabilidade ao usuário do servidor.
d) melhorar as funções de gerenciamento de dados e comunicação, permitindo que o usuário do servidor consiga escolher se deseja enviar o comando para todos os clientes conectados ou apenas para parte deles;
e) melhorar as mensagens de *feedback* apresentadas ao término dos processos executados no servidor e cliente.

REFERÊNCIAS

AFONSO, A. **Apresentação da Sessão sobre Software Libre e Comunidades no Mestrado de Open Source Software**. Disponível em: <http://pt.scribd.com/doc/86824528/apresentacao-da-sessao-sobre-software-libre-e-comunidades-no-mestrado-de-open-source-software>. Acesso em: 13 jun. 2014.

ALCADE, E.; MORERA, J.; CAMPERO, J. A. P. **Introdução aos Sistemas Operativos:** MS/DOS, UNIX, OS/2, MVS, VMS, OS/400. Lisboa: McGraw-Hill, 1993. 271 p.

APGAUA, R. **O Linux e a Perspectiva da Dádiva**. Disponível em: <http://www.scielo.br/pdf/ha/v10n21/20626.pdf>. Acesso em: 27 abr. 2014.

ARRUDA, F. **O que são Distros Linux e qual devo usar?**. Disponível em: <http://canaltech.com.br/tutorial/linux/O-que-sao-distros-Linux-e-qual-devo-instalar/>. Acesso em: 20 jun. 2014.

CAMPOS, C. A. **Open Source é...**. Disponível em: http://www.linuxmag.com.br/images/uploads/pdf_aberto/LM_53_14_15_01_col_augusto.pdf>. Acesso em: 13 jun. 2014

CARISSIMI, S. A.; ROCHOL, J.; GRANDVILLE, Z. L. **Redes de Computadores**. Porto Alegre: Bookman, 2009. 391 p.

CASTALDELI, R. **Classificação de Sistemas Operacionais quanto a:** Processamento, Tarefas, Usuário e Interface. Disponível em: <http://rcastaldelli.blogspot.com.br/2010/12/classificacao-de-sistemas-operacionais.html>. Acesso em: 05 abr. 2014.

CESCATO, D. E. M. **Mentor:** Um ambiente para a Distribuição Transparente de Software. Disponível em: http://teses.ufrj.br/NCE_M/MauricioEmanuelDouradoCescato.pdf>. Acesso em: 19 jun. 2014.

COMER, D. **Interligação de Redes com TCP / IP:** Princípios, protocolos e arquitetura. 5. ed. Rio de Janeiro: Elsevier, 2006. 435 p.

DANESH, A. **Dominando o Linux:** A Bíblia. São Paulo: Makron Books, 2000. 574 p.

DEBIAN. **Como usar o APT**. Disponível em: <https://www.debian.org/doc/manuals/apt-howto/ch-apt-get.pt-br.html>. Acesso em: 18 out. 2014.

DEITEL, H. M.; DEITEL, P. J. **Sistemas Operacionais**. 3 .ed. São Paulo: Pearson Prentice Hall, 2012. 760 p.

DELL. **Soluções para Gerenciamento de Sistemas**. Disponível em: <http://www.dell.com/learn/br/pt/brbsdt1/smb-by-tech-systems-mgmt>. Acesso em: 14 jun. 2014.

DIBONA, C.; OCKMAN, S.; STONE, M. **Open Sources:** Voices from the Open Source Revolution. 1. ed. Sebastopol: O'Reilly Media, 1999. 284 p.

FERREIRA, E. R. **Gerenciamento de Pacotes de Software no Linux**. São Paulo: Novatec, 2006. 288 p.

FERREIRA, G. J. **Vantagens e Dificuldades na Relação Software Livre x Educação**. Disponível em: <http://www.periodicos.letras.ufmg.br/index.php/ueadsl/article/viewFile/2641/2595>. Acesso em: 19 jun. 2014.

FREIRE, E. **Desenvolvimento de Sistemas Cliente/Servidor**. Disponível em: <http://pt.slideshare.net/eneck/desenvolvimento-de-sistemas-clienteservidor-estrutura-de-sistemas-cliente-servidor>. Acesso em: 07 abr. 2014.

FUNDO NACIONAL DE DESENVOLVIMENTO DA EDUCAÇÃO. **ProInfo.** Disponível em: http://www.fnde.gov.br/programas/programa-nacional-de-tecnologia-educacional-proinfo/proinfo-perguntas-frequentes>. Acesso em: 18 jun. 2014.

GUIMARÃES, P. M. **Introdução aos Sistemas Operacionais**. Disponível em: http://marcelodepaiva.com.br/iso/notasdeaula.pdf>. Acesso em: 10 abr. 2014.

GNU.ORG. **A Definição de Software Livre**. Disponível em: <http://www.gnu.org/philosophy/free-sw.pt-br.html>. Acesso em: 02 mai. 2014.

GOLDMAN, A. **Sistemas de Multiprogramação**. Disponível em: <http://www.ime.usp.br/~gold/cursos/2002/mac2301/ep2/ep2/node3.html>. Acesso em: 05 abr. 2014.

HILL, M. B. et al. **O Livro Oficial do Ubuntu**. 2. ed. Porto Alegre: Bookman, 2008. 448 p.

INTEL. **Gerenciamento de TI**. Disponível em: <http://www.nextgenerationcenter.com/scriptServices/courseToPdf.ashx?courseId=2ca72ffd-5e52-4786-8982-a707f1ddf9d5>. Acesso em: 16 jun. 2014

KREUTZ, D. et al. **Clumpt**: Um Sistema Peer-to-Peer para Instalação e Manutenção de Software em Aglomerados de Computadores. Disponível em: http://marceloneves.org/papers/wp2p2006-clumpt.pdf>. Acesso em: 20 jun. 2014.

LINUX EDUCACIONAL. **Linux Educacional 5.0**. Disponível em: http://linuxeducacional.c3sl.ufpr.br/index.html>. Acesso em: 18 jun. 2014.

MACHADO, B. F.; MAIA, L. P. **Arquitetura de Sistemas Operacionais**. 5. ed. Rio de Janeiro: LTC, 2013. 265 p.

MARCOS, L. **Máquinas Virtuais e Emuladoras:** Conceitos, Técnicas e Aplicações. Disponível em: <http://www.mlaureano.org/aulas_material/so/livro_vm_laureano.pdf>. Acesso em: 07 abr. 2014.

MARTINEZ, M. T. **Linux:** Software Livre. Disponível em: < http://www.uabjales.com.br/joomla2/images/stories/Fotos_Artigos/Linux%20Polo%20UAB%20-%20Minicurso.pdf>. Acesso em: 25 mai. 2014.

MATIAS, R. **Sistemas Distribuídos – Aplicação simples Java RMI**. Disponível em: <http://www.vivaolinux.com.br/dica/Sistemas-Distribuidos-Aplicacao-simples-Java-RMI>. Acesso em: 15 jun. 2014.

MAZIERO, A. C. **Sistemas Operacionais:** Conceitos e Mecanismos. Disponível em: <dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/so:so-cap09.pdf>. Acesso em: 07 abr. 2014.

MICROSOFT. **Gerenciamento de Atualizações.** Disponível em: <http://technet.microsoft.com/pt-br/updatemanagement/bb245736.aspx>. Acesso em: 14 jun. 2014.

MORIGI, V.; SANTIN, M. D. **Reflexões sobre os Valores do Movimento Software Livre na Criação de Novos Movimentos Informacionais**. Disponível em: <http://www.uel.br/revistas/uel/index.php/informacao/article/view/1746/1495>. Acesso em: 25 jun. 2014.

MOTA FILHO, E. J. **Descobrindo o Linux:** Entenda o sistema operacional GNU/Linux. São Paulo: Novatec, 2006.

MOURA, E. **Conceito de Software Livre (Open Source)**. Disponível em: <http://www.bysoft.com.br/conceito-do-software-livre-open-source.html>. Acesso em: 16 jun. 2014.

NEWTECK. **Circuitos Integrados**. Disponível em: <http://www.newteck-ci.com.br/circuitos-integrados.php>. Acesso em: 20 jun. 2014.

NORTON, P.; GRIFFITH, A. **Guia Completo do Linux**. 3. ed. São Paulo: Berkeley Brasil, 2000. 597 p.

OLIVEIRA R.; CARISSIMI, A.; TOSCANI, S. **Sistemas Operacionais**. 4. ed. Porto Alegre: Bookman, 2010. 375 p.

OPEN SOURCE INITIATIVE. **About the Open Source Initiative**. Disponível em: <http://opensource.org/about>. Acesso em: 14 jun. 2014.

PERSONA, M. **Gestão do Tempo:** Qual é o principal motivo para da falta de tempo das pessoas de hoje?. Disponível em: <http://mariopersona.com.br/entrevista_revista-fecontesp_gestao_tempo.html>. Acesso em: 13 jun. 2014.

PORTAL DO SOFTWARE PÚBLICO BRASILEIRO. **Linux Educacional.** Disponível em: <http://www.softwarepublico.gov.br/pt/ver-comunidade?community_id=11809207>. Acesso em: 13 jun. 2014.

QUINTÃO, R. **Fundamentos de Sistemas Operacionais.** Disponível em: <http://slideplayer.com.br/slide/45898/>. Acesso em: 08 jun. 2014.

RASKIN, F. S. **Arquitetura Cliente – Servidor**. Disponível em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=577>. Acesso em: 07 abr. 2014.

REVISTA DE CIÊNCIAS EXATAS E TECNOLOGIA. **Uma Proposta de Gerenciamento e Controle de Atualizações de Segurança de Sistemas Operacionais**. São Paulo: Anhanguera Educacional. v. 3, n. 3, dez. 2008.

ROMERO, D. **Começando com Linux:** Comandos, serviços e administração. 1. ed. São Paulo: Casa do Código, 2013. 131 p.

SILVEIRA, F. P. **Software Livre e Educação:** Vantagens e Desvantagens das Novas Tecnologias. Disponível em: <http://www.periodicos.letras.ufmg.br/index.php/ueadsl/article/view/2871/2830>. Acesso em: 19 jun. 2014.

STALLMAN, M. R. **The GNU Manifesto**. Massachusetts: Free Software Fundation, 1993. Disponível em: <https://facwiki.cs.byu.edu/OSSResearch/images/7/70/Stallman_TheGNUManifesto.pdf>. Acesso em: 15 mai. 2014.

SOFTWARE PÚBLICO. **Cadastro de Prestadores de Serviço**. Disponível em: <http://www.softwarepublico.gov.br/mpv/organizations/>. Acesso em: 18 jun. 2014.

TANENBAUM, A.; WETHERALL, J. D. **Computer Networks**. 5 ed. São Paulo: Pearson Prentice Hall, 2011. 600 p.

TANENBAUM, A.; WOODHULL, A. **Sistemas Operacionais:** Projeto e Implementação. 3. ed. Porto Alegre: Bookman, 2008. 992 p.

TANENBAUM, A. **Sistemas Operacionais Modernos**. 3 .ed. São Paulo: Pearson Prentice Hall, 2010. 672 p.

UBUNTU DOCUMENTATION. **O que é Synaptic?**. Disponível em: <https://help.ubuntu.com/community/ComoSynaptic>. Acesso em: 15 out. 2014.

APÊNDICE A - ARTIGO

Instalação e Atualização Massiva de Aplicações em Sistema Operacional Linux Educacional

Rodrigo A. Colonetti[1], Sergio Coral[2]

[1]Acadêmico do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) - Caixa Postal 3.167 – 88806-000 – Criciúma – SC – Brasil

[2]Professor do Curso de Ciência da Computação – Unidade Acadêmica de Ciências, Engenharias e Tecnologias – Universidade do Extremo Sul Catarinense (UNESC) - Caixa Postal 3.167 – 88806-000 – Criciúma – SC – Brasil

rodrigocolonetti@gmail.com, sergiocoral@unesc.net

Abstract. *This article aims to demonstrate a tool developed with resources and free software for GNU/Linux operating system, specifically for its educational distribution, typically used in public schools in Brazil, attended by the National Educational Technology Program (PROINFO). This tool created as free software to assist its members in the administration of computer labs, performing software maintenance (installation, update and removal) centrally via network environment, through the structure client / server.*

Resumo. *Este artigo tem como meta demonstrar uma ferramenta desenvolvida com recursos e softwares livres para sistema operacional GNU/Linux, mais especificamente para sua distribuição educacional, normalmente utilizada em escolas públicas do Brasil, atendidas pelo Programa Nacional de Tecnologia Educacional (PROINFO). Ferramenta essa criada como software livre para auxiliar seus usuários na administração dos laboratórios de informática, realizando manutenção de software (instalação, atualização e remoção) de forma centralizada via ambiente de rede, através da estrutura cliente / servidor.*

1. Introdução

Com a criação do Programa Nacional de Tecnologia Educacional (PROINFO) em 1997, escolas públicas brasileiras vem sendo beneficiadas e passam a possuir um número considerável de computadores, equipados com sistema operacional Linux Educacional, sendo necessário o uso de ferramentas que auxiliem os profissionais na manutenção das máquinas presentes nos laboratórios [PORTAL DO SOFTWARE PÚBLICO, 2014].

Desde então ferramentas vem sendo desenvolvidas, mas em sua grande maioria voltadas para outros sistemas operacionais e de difícil compreensão, ou ainda que não realizam a tarefa da forma deseja. Podem-se citar exemplos como Clonezilla, Redo Backup, Acronis e Ghost que realizam uma cópia do sistema por inteiro, possibilitando apenas a replicação em outras máquinas. Mesmo o sistema Linux Educacional possui ferramenta de

atualização automática, mas essa não é transparente ao usuário, pois todas as atualizações são controladas, revisadas e liberadas pelos mantenedores do sistema [LINUX EDUCACIONAL 2014].

Devido a isso foi realizado levantamento bibliográfico para posteriormente desenvolver uma ferramenta voltada à área de sistema operacional, que tem como objetivo auxiliar os profissionais que lidam diariamente com a distribuição educacional e necessitam realizar processos de instalação, atualização, remoção de softwares, entre outros.

Com isso viu-se que haveria a possibilidade de desenvolver uma ferramenta que trabalhasse no modelo de rede cliente / servidor, realizando um processo baseado em troca de mensagens, sendo que através destas mensagens são enviados comandos, *scripts* e outras máquinas poderiam interpretá-las e executá-las de forma correta.

2. Manutenção de Software em sistema Linux Educacional

O Linux Educacional é uma distribuição do sistema GNU/Linux, desenvolvida inicialmente pelo Ministério da Educação (MEC) em parceria com o Centro de Experimentação em Tecnologia Educacional (CETE) baseada inicialmente em outras distribuições populares, Debian e Kubuntu. Atualmente em sua versão 5.0 baseia-se na versão 12.04 da distribuição Ubuntu, sendo mantida pelo Centro de Educação Científica e Software Livre, na Universidade Federal do Paraná com o apoio do MEC [LINUX EDUCACIONAL, 2014].

Desde 1997 a distribuição educacional já passou por diversas melhorias e vem sendo utilizada nas escolas públicas brasileiras beneficiadas, com o intuito de auxiliar professores no preparo de suas aulas e utilizar a tecnologia como ferramenta de ensino e aprendizagem.

Mesmo com melhorias uma grande dificuldade encontrada pelos profissionais que utilizam a distribuição, o que acaba dificultando a aceitação da mesma, é a falta de conhecimento por parte dos deles, pois grande parte tem computador mais utiliza sistemas privados, proprietários e ao se deparar com algo diferente não conseguem realizar processos de instalação de softwares, por exemplo, tarefa que acaba consumindo muito tempo de tais profissionais quando feita em grande quantidade [BATISTA, 2013].

Analisadas tais informações torna-se viável o desenvolvimento de uma ferramenta utilizando recursos e softwares livres para auxiliar esses profissionais que não possuem o domínio adequado das funções do sistema operacional, a qual possa diminuir o tempo gasto nos processos de instalação, atualização e remoção de softwares, além de execução de outras tarefas, executando-as em todas as máquinas da rede local, de forma homogênea.

3. Ferramenta para Instalação e Atualização Massiva de Aplicações

Este capítulo apresenta o desenvolvimento da aplicação proposta como objetivo no trabalho realizado. A ferramenta possui uma aplicação cliente e outra servidor que se comunicam através da rede de computadores local. Após estabelecer comunicação o usuário pode enviar *scripts* de comandos gerados via software e escolher outros pré-definidos na aplicação servidor, além de possibilitar ao usuário que envie seus próprios comandos, como se estivesse utilizando o terminal *shell* do GNU/Linux diretamente nas máquinas clientes.

3.1 Metodologia

Para atingir os objetivos propostos neste projeto foi necessário o cumprimento de algumas etapas metodológicas, onde se realizou o levantamento bibliográfico, buscando conhecer de

forma detalhada a história e estrutura de sistemas operacionais, voltando à pesquisa mais para sistema GNU/Linux e sua distribuição educacional. Foi visto também as diferenças dos termos *softwares* livres e *open source* por utilizá-los no decorrer do projeto, além da escolha da linguagem de programação utilizada, onde se optou pela linguagem *Python* e *Shell Script* e ainda a maneira interligar a máquinas cliente e servidor, a qual se utilizou a chamada de sistema *socket* para realizar tal serviço.

Foram modeladas também as tarefas básicas de cada aplicação e usuário que podem ser vistas nas figuras 1, 2 e 3.

Figura 1. Diagrama caso de uso do usuário

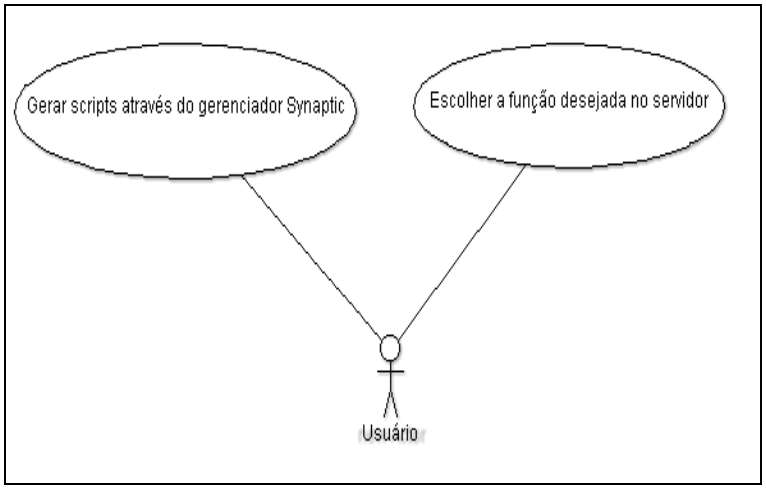


Figura 2. Diagrama caso de uso aplicação servidor

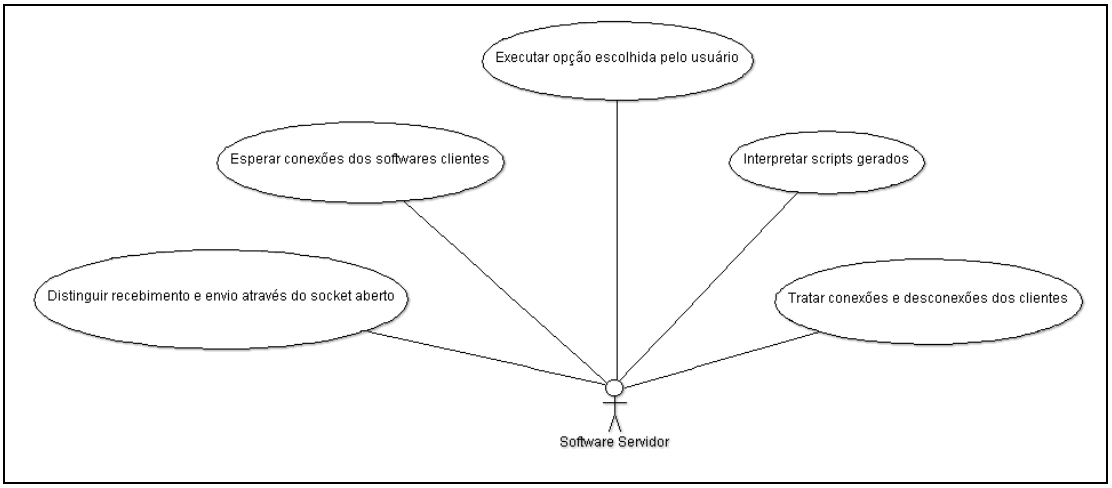
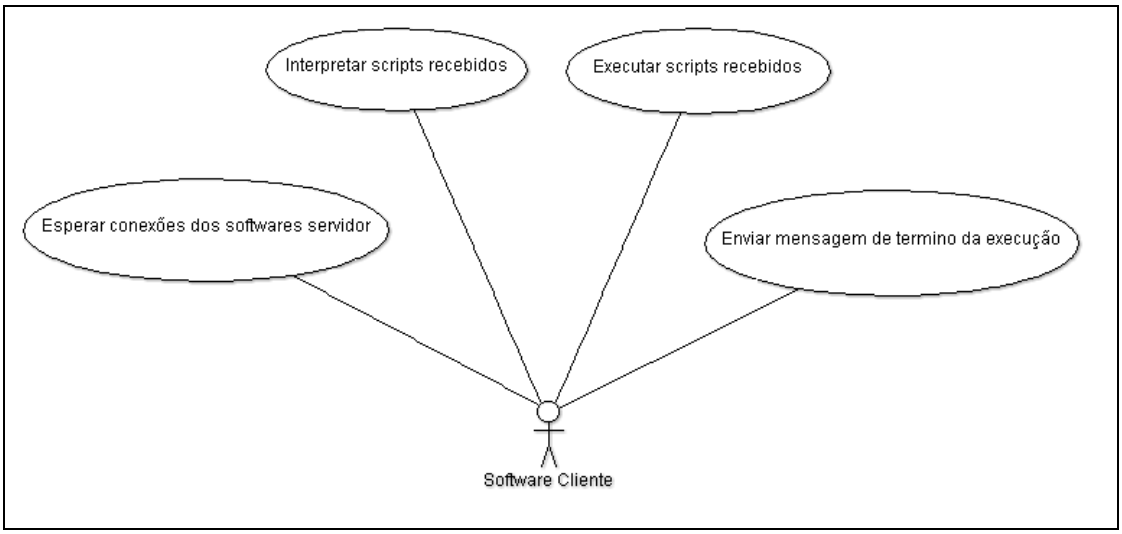


Figura 3. Diagrama caso de uso aplicação cliente



3.2. Adaptação e Desenvolvimento

Inicialmente para o desenvolvimento da aplicação como um dos objetivos é utilizar outras ferramentas para oferecer suporte à mesma, foram buscadas alternativas que auxiliassem o usuário de forma simples a gerar tarefas de instalação, atualização e remoção de software, onde optou-se pelo uso do gerenciador de pacotes Synaptic.

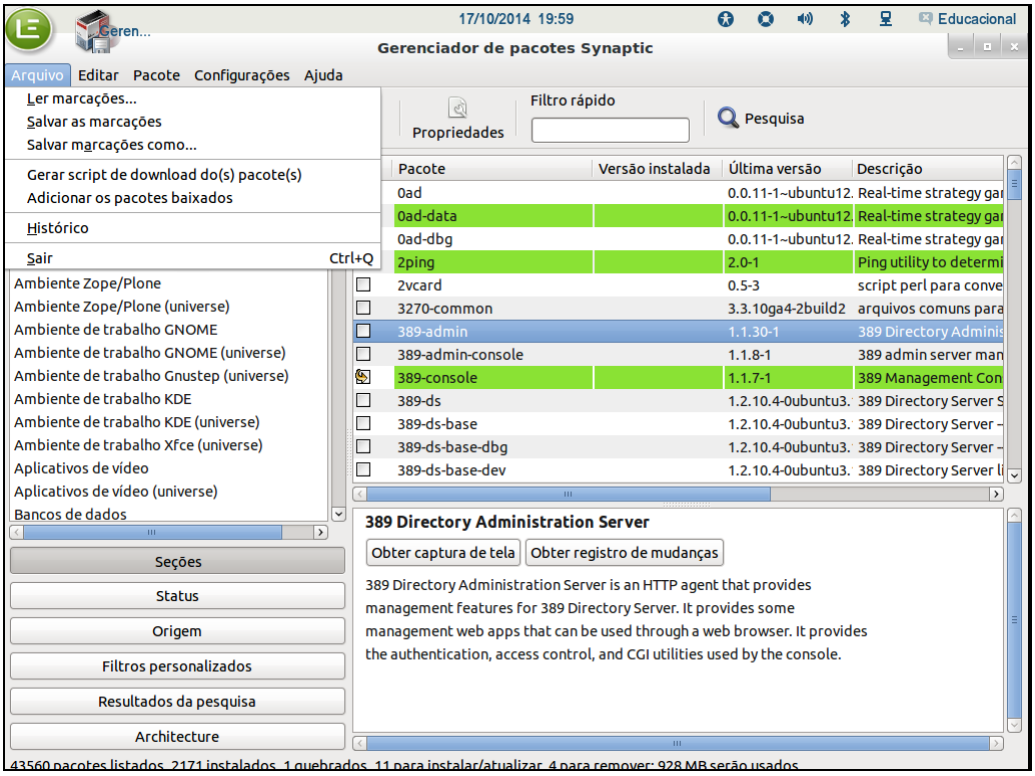
O gerenciador Synaptic vem instalado nativamente na versão 5.0 do sistema Linux Educacional e oferece através de uma *interface* gráfica simples comandos provenientes da ferramenta *Advanced Packing Tool* (APT), normalmente utilizada via comando em terminal *shell* das distribuições GNU/Linux [UBUNTU, 2014]. Através dela é possível gerar *scripts* de atualização e instalação de pacotes de software, nestes *scripts* são armazenados os endereços dos repositórios onde cada pacote de software se encontra e através do comando *wget –c* reconhecido pelo interpretador de comandos do Linux, o *bash*, sendo este por sua vez chamado pelo comando *#!/bin/sh*, o sistema consegue realizar o *download* dos pacotes selecionandos (figura 4) [DEBIAN, 2014].

Figura 1. Exemplo de *script* gerado pelo gerenciador Synaptic

```
#!/bin/sh
wget -c http://br.archive.ubuntu.com/ubuntu/pool/main/libr/libreoffice/libreoffice-core_3.5.7-0ubuntu6.1_i386.deb
wget -c http://br.archive.ubuntu.com/ubuntu/pool/universe/h/hal/libhal1_0.5.14-8_i386.deb
wget -c http://archive.canonical.com/pool/partner/a/adobeair/adobeair_2.6.0.19170-0lucid1_i386.deb
wget -c http://br.archive.ubuntu.com/ubuntu/pool/main/p/python3.2/python3.2-minimal_3.2.3-0ubuntu3.6_i386.deb
wget -c http://br.archive.ubuntu.com/ubuntu/pool/main/p/python3.2/python3.2_3.2.3-0ubuntu3.6_i386.deb
wget -c http://br.archive.ubuntu.com/ubuntu/pool/main/p/python3.2/libpython3.2_3.2.3-0ubuntu3.6_i386.deb
wget -c http://br.archive.ubuntu.com/ubuntu/pool/main/libr/libreoffice/libreoffice-emailmerge_3.5.7-0ubuntu6.1_all.deb
wget -c http://br.archive.ubuntu.com/ubuntu/pool/main/p/python3-defaults/python3-minimal_3.2.3-0ubuntu1.2_i386.deb
wget -c http://br.archive.ubuntu.com/ubuntu/pool/main/p/python3-defaults/python3_3.2.3-0ubuntu1.2_i386.deb
```

Para geração destes *scripts* basta o usuário localizar e executar o Synaptic, o mesmo possui um *interface* que não apresenta grandes desafíos ao usuário, todo pacote que o usuário deseja obter pode ser pesquisado e localizado, qualquer depedência de software necessário será selecionada automaticamente, após marcados os pacotes escolhidos através do menu, na guia arquivo tem-se a possibilidade da geração de *scripts* de *downloads* dos pacotes (figura 5).

Figura 5. Seleção de pacotes e geração de script



3.2.1. Aplicação Servidor

Durante o desenvolvimento da aplicação que é executada na máquina servidor alguns módulos compatíveis com a linguagem *Python* tiveram que ser importados através do comando *import*, fornecendo complementos essenciais para seu funcionamento. Os módulos importados foram:

a) ***socket*:** possibilita a comunicação entre as máquinas presentes em um ambiente de rede, normalmente utilizada em aplicações cliente / servidor, comunicando-se através dos protocolos de comunicação UDP e TCP, sendo que este foi utilizado na implementação, por garantir a entrega completa das informações transmitidas entre as máquinas;

b) ***select:*** juntamente com a chamada de sistema *socket* gerência a comunicação do servidor e máquinas presentes na rede, fornecendo uma lista de *sockets* prontos para leitura, escrita e que causam algum tipo de exceção, possibilitando que mesmo quando o servidor esteja enviando determinada tarefa novas conexões de clientes sejam realizadas;

c) ***sys:*** permite a comunicação com o interpretador de comandos presentes nas distribuições GNU/Linux;

d) ***commands:*** permite executar e capturar saídas de comandos inseridos via terminal *shell*, a importação deste módulo possibilita através da função *commands.getoutput* a captura do endereço de IP da máquina onde a aplicação está em execução, guardando-a em uma variável, quando outras funções retornavam apenas o endereço *localhost*, ou seja, 127.0.0.1.

Além dos módulos há outras funções importantes declaradas na implementação do servidor, que serão apresentadas a seguir:

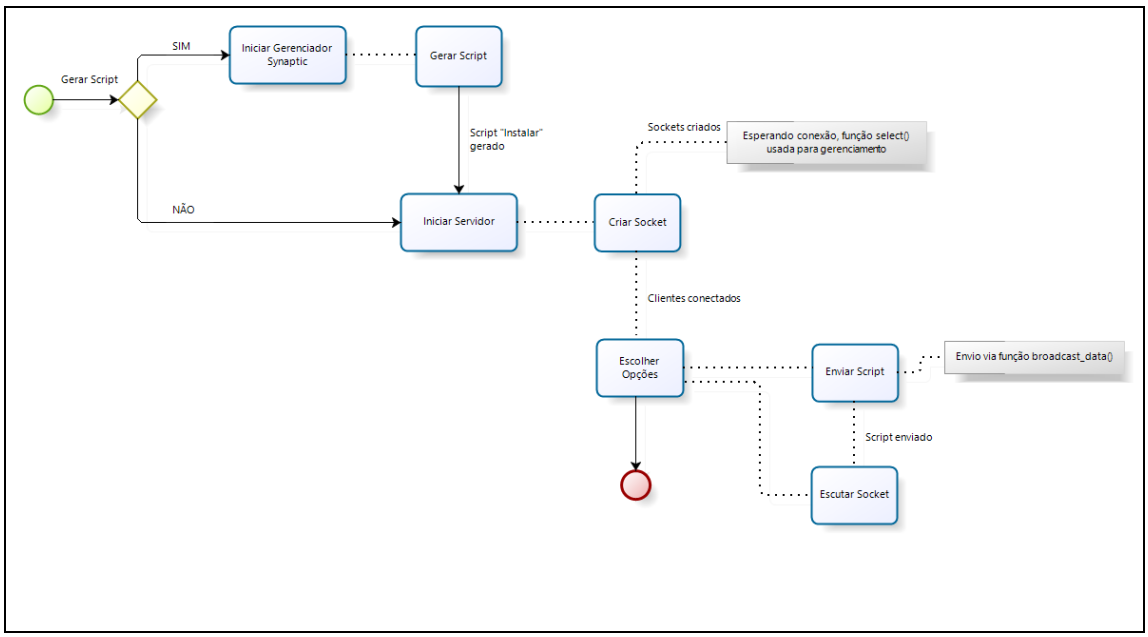
a) **opcao:** aciona o menu de opções para a escolha da tarefa que o usuário administrador do laboratório deseja realizar nas máquinas clientes;

c) **tratamento:** realiza a montagem e devidos tratamentos nos *scripts* para que possam ser interpretados pelas máquinas clientes;

d) **broadcast_data:** envia o *script* de comando escolhido do computador servidor para as máquinas clientes conectadas, onde é tratado para que todas as máquinas recebam tal comando, exceto a máquina que o enviou.

Utilizando destes recursos podemos completar o clico de trabalho da aplicação (figura 6).

Figura 6. Ciclo de execução aplicação presente no servidor



3.2.2. Aplicação Cliente

Assim como na aplicação presente no servidor é necessário o uso do comando *import*, são utilizados os mesmos módulos do servidor além de dois novos módulos o *glob* e *os*. O módulo *glob* possibilita a verificação de arquivos, extensões presentes em diretórios do sistema operacional, enquanto o *os* realiza a integração com GNU/Linux, possibilitando execução de comandos como se fosse executados diretamente no terminal *shell*.

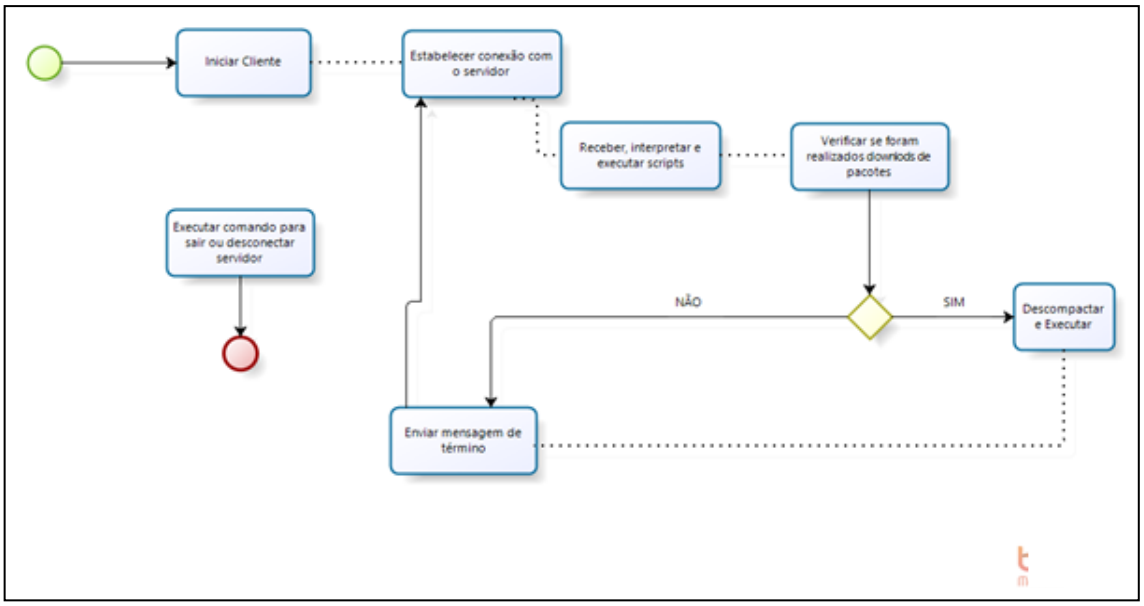
Inicialmente a aplicação solicita que o usuário forneça a senha do usuário *root*, para que sejam executados comandos que precisam de maiores privilégios. Na sequência é verificado o conteúdo do arquivo *Config*, que contém as informações de IP e porta de conexão com o servidor, e devem ser configurados adequando-os a cada ambiente de rede. Após abertura da conexão e comunicação devidamente estabelecida todo conteúdo, *script* recebido é lido e tratado, posteriormente o *script* é executado utilizando comandos próprios do sistema GNU/Linux, via função *os.system*, derivada do módulo *os* importado no início.

Após execução dos *scripts* através de função do módulo *glob* é verificado se foram realizados *downloads* de pacotes padrão Debian, obtidos quando se utiliza o gerenciador de pacotes Synaptic para gerar os arquivos de atualização. Quando a verificação for positiva os pacotes são descompactados, executados e excluídos ao termino da execução, chamando na sequência função Resposta, que envia o *feedback* da execução ao servidor, caso negativa é

chamada apenas a função Resposta, retornando em seguida a escutar o *socket* criado, para o recebimento de novas tarefas, sendo que a comunicação é desfeita apenas se a aplicação for encerrada no cliente ou servidor.

Abaixo na figura 7 podemos observar o fluxo de execução do aplicativo cliente.

Figura 7. Ciclo de execução aplicação presente no cliente



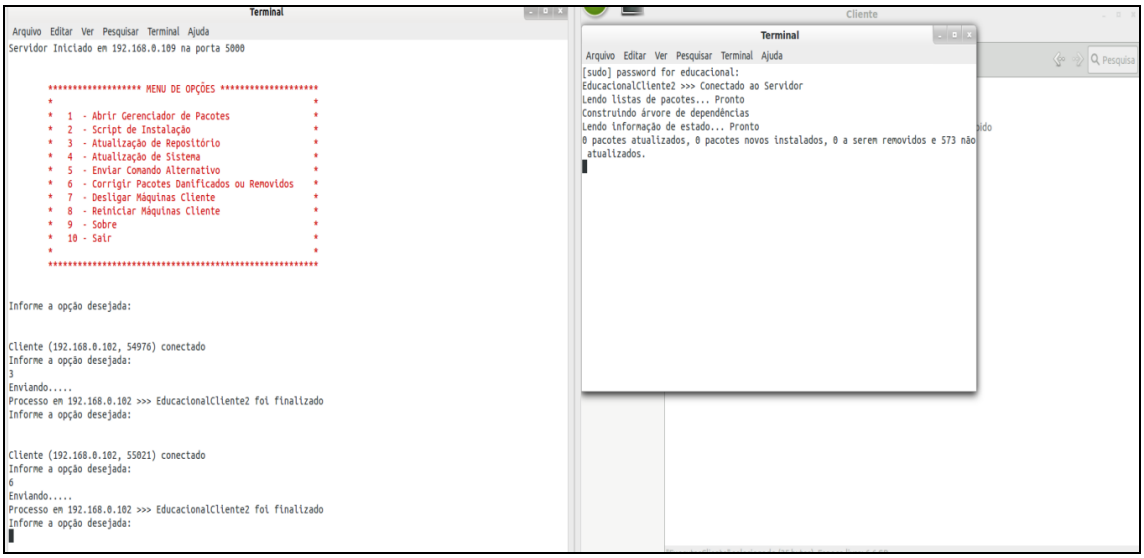
4. Resultados Obtidos

Foi criada uma ferramenta utilizando apenas recursos e softwares livres, que possibilita de maneira rápida a conexão entre máquinas presentes em uma rede de computadores. Durante os testes criou-se uma rede com três máquinas de diferentes configurações, máquinas virtuais, criadas a partir do software VMWare Workstation 10, onde uma máquina possuí a distribuição Ubuntu 14.04 LTS, processador de 1.60 Ghz, 1 GB de memória RAM e 20 GB de HD, que foi utilizada como servidor, já as outras máquinas possuem a distribuição educacional 5.0, utilizando as mesmas configurações do servidor exceto pela memória, onde uma possuí 512 MB de memória RAM e a outra apenas 256 MB.

Apesar de não terem sido executados testes em larga escala pode-se comprovar que a utilização do método *select* juntamente com a chamada de sistema *socket* consegue gerenciar as conexões realizadas entre servidor e cliente de forma satisfatória, visto que não houve nenhum erro durante a execução da aplicação, o que ocorreu quando o método de gerenciamento foi implementado utilizando *Threads* e *Fork.*

Todas as opções implementadas e enviadas do servidor para o cliente foram executadas conforme o programado, como exemplo pode ser visto a figura 8, onde primeiramente foi enviada a opção número três, a qual foi executada e em seguida enviada a opção número 6 para correção de pacotes danificados ou removidos, a qual retornou que não havia nada a ser corrigido.

Figura 8. Funcionamento da ferramenta



Além das funções para realização da manutenção de softwares foram desenvolvidos alguns *scripts* com funções simples, que para as pessoas com pouco conhecimento auxiliam ainda mais, funções como reiniciar e desligar as máquinas conectadas e abrir o gerenciador de pacotes Synaptic através da ferramenta desenvolvida.

Comprovou-se ainda que a ferramenta pode ser executada em várias distribuições GNU/Linux, por ser desenvolvida utilizando recursos compatíveis com grande parte delas.

5. **Conclusão**

Logo no início do desenvolvimento da fundamentação, quando buscado ferramentas que trabalhassem conforme maneira proposta para possível adaptação, que proporcionassem agilidade na execução de tarefas e essa pudessem ser utilizadas sem grandes dificuldades, não foram encontrados muitos resultados. Poucas ferramentas possibilitam que o usuário escolha quais atualizações realizar, necessitam de configuração de vários parâmetros, servidores *web* entre outros, para realização de todas as suas funções.

Devido às dificuldades partiu-se para o desenvolvimento de uma ferramenta, qual foi desenvolvida utilizando linguagem de programação *Python* e *Shell Script*, devido ao pouco conhecimento das linguagens até então ocorrem alguns problemas, que foram resolvidos e ao final serviram para que se adquirisse um bom nível de conhecimento das mesmas, além de todo ganho proporcionado durante o levantamento bibliográfico, onde se viu detalhadamente as estruturas e tipos de sistema operacional e também como são divididas as categorias de software em sistema GNU/Linux, podendo um software ser livres ou *open source*, sendo que cada categoria possui seus diversos tipos de licenciamento, contendo regras para desenvolvimento, distribuição e redistribuição.

Apesar das dificuldades o que foi proposto como objetivo foi ser realizado, a ferramenta desenvolvida funcionou de forma satisfatória e pode auxiliar os profissionais que virem a utilizá-la e que ainda pode melhorar em vários quesitos, melhorias que podem ser implementados em trabalhos futuros, realizados até mesmo por outros acadêmicos ou pesquisadores da área, visto que foi desenvolvida como um software livre.

Por fim é recomendado para trabalhos futuros:

f) adicionar novas funções e *scripts* ao servidor, proporcionando o envio de comandos mais avançados, sendo estes executados no cliente de forma segura;
g) adicionar relatórios de *log*, contendo as principais informações dos processos realizados;
h) Implementar a interface gráfica, visando maior usabilidade ao usuário do servidor.
i) melhorar as funções de gerenciamento de dados e comunicação, permitindo que o usuário do servidor consiga escolher se deseja enviar o comando para todos os clientes conectados ou apenas para parte deles;
j) melhorar as mensagens de *feedback* apresentadas ao término dos processos executados no servidor e cliente.

6. Referências

Batista, C. F. S. (2012) "O Linux Educacional e as Novas Práticas Pedagógicas", http://www.hardware.com.br/artigos/linux.educacional/, Dezembro

Debian. (2014) "Como usar o APT", https://www.debian.org/doc/manuals/apt-howto/ch-apt-get.pt-br.html, Dezembro.

Linux Educacional. (2014) "Linux Educacional 5.0", http://linuxeducacional.c3sl.ufpr.br/index.html, Dezembro.

Portal do Software Público. (2014) "Linux Educacional", http://www.softwarepublico.gov.br/pt/ver-comunidade?community_id=11809207, Dezembro.

Ubuntu Documentation. (2014) "O que é Synaptic?", https://help.ubuntu.com/community/ComoSynaptic, Dezembro.