UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE SOFTWARE PARA NOTIFICAÇÃO DE RECEBIMENTO DE NOVOS E-MAIL'S, BASEADO NO PROTOCOLO IMAP

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

ALEXANDRE EDUARDO KNORST

BLUMENAU, DEZEMBRO/2000

2000/2-04

PROTÓTIPO DE SOFTWARE PARA NOTIFICAÇÃO DE RECEBIMENTO DE NOVOS E-MAIL'S, BASEADO NO PROTOCOLO IMAP

ALEXANDRE EDUARDO KNORST

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Sérgio Stringari — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Sérgio Stringari

_____

Prof. Francisco Adell Périclas

_____

Prof. Miguel Alexandre Wisintainer

AGRADECIMENTOS

Agradeço a minha família que esteve do meu lado, sempre me motivando e incentivando nos momentos de desânimo e alegria.

Aos meus amigos que prestaram apoio, compreensão e ajuda nos momentos difíceis desta trajetória.

Aos colegas, com os quais convivi durante todos estes anos.

Ao Professor Sérgio Stringari, que na função de orientador, colaborou para com seus conhecimentos, para que este trabalho tivesse toda a excelência necessária.

À todos os professores, pelo conhecimento promovido e o qual me seguirá o resto da minha vida, tanto na vida profissional como na vida pessoal.

À Deus que me deu saúde e forças para sempre continuar no rumo ao objetivo.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE PROCESSOS

LISTA DE SIGLAS E ABREVIATURAS

API	Application Program Interface
ARPA	Advanced Research and Projects Agency
BSD	Berkelry Software Distribution
CR	Carrier Return
FTP	File Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
LF	Line Feed
MTS	Message Transport System
NSF	National Science Foundation
POP	Post Office Protocol
RARP	Reverse Address Resolution Protocol
RFC	Request For Comments
SMTP	Single Mail Transfer Protocol
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
VCL	Visual Component Library
WWW	World Wide Web

RESUMO

Este trabalho apresenta um estudo sobre os principais protocolos de correio eletrônico, enfatizando o protocolo IMAP. Apresenta também considerações em relação a implementação de um protótipo de software para notificação de recebimento de novos *e-mail's,* baseado no protocolo IMAP. Neste trabalho podemos encontrar também algumas considerações em relação a rede TCP/IP e *Sockets*.

ABSTRACT

This article presents a study about the main protocols of e-mail, emphasizing the IMAP protocol. The implementation of an archetype of software for notifying the receiving of new e-mail's, also presents consideration in relation, based in IMAP protocol. In this article we also find some consideration about TCP/IP network and Sockets.

1 INTRODUÇÃO

De acordo com [CYC1997], a Internet pode ser definida como sendo uma grande rede de computadores interligados pelo mundo inteiro, e que utiliza um conjunto de protocolos muito bem definidos para a transferência de informações e dados.

A Internet surgiu por volta de 1969 na agência norte-americana ARPA (*Advanced Research and Projects Agency*), que tinha como objetivo interconectar seus computadores dos quatro departamentos de pesquisa, com isso surgiu a primeira rede de computadores, conhecida com ARPANET. Na década de 1970 foi definido e especificado o protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*) e que ainda hoje é utilizado como protocolo base de rede para a Internet.

No inicio da década 1980 na Universidade da Califórnia de Berkeley, surgiu o primeiro sistema operacional que incorporava o protocolo TCP/IP, conhecido como BSD (*Berkelry Software Distribution*) Unix 4.2, com o qual se possibilitou a integração de várias Universidades através da ARPANET.

Em 1986, surgia oficialmente a Internet, nome que foi criado após a interligação da ARPANET com a NSFNET. A NSFNET era a rede de computadores da NSF (*National Science Foundation*) que interligava os seus centros de pesquisa.

Conforme [CYC1997], a Internet chegou ao Brasil em 1988 por iniciativa da comunidade acadêmica de São Paulo (FAPESP – Fundação de Amparo à Pesquisa do Estado de São Paulo) e do Rio de Janeiro (UFRJ – Universidade Federal do Rio de Janeiro e LNCC – Laboratório Nacional de Computação Científica).

Somente em 1993 a Internet deixou de ser de domínio exclusivo das instituições acadêmicas e militares, e começou a ser explorada comercialmente, surgindo então, os primeiros *backbones* privados como também o fornecimento de diversos serviços ao público em geral.

A partir de então, começarão a surgir diversos serviços, tais como o serviço de WWW (*World Wide Web*), FTP (*File Transfer Protocol*), *e-mail*, e muitos outros, sendo que os dois mais difundidos na atualidade são o WWW e o *e-mail. O* WWW é o serviço que

possibilita a busca de informações e o *e-mail* é um serviço utilizado para o envio e recebimento de mensagens eletrônicas.

O serviço de *e-mail* é composto por diversos protocolos, entre eles: o SMTP (*Single Mail Transfer Protocol*) que é especificado pela RFC (*Request For Comments*) [POS1982], como sendo o protocolo a ser utilizado para a transferência da mensagem de um usuário ***A*** até o servidor do usuário ***B***, onde a mensagem será armazenada.

Quando a mensagem já estiver armazenada no servidor de destino, o cliente dispõe de alguns protocolos para efetuar a leitura destes *e-mail's.* Os mais difundidos são o POP (*Post Office Protocol*) que é um dos mais tradicionais protocolos utilizados e o outro é o IMAP (*Internet Message Access Control*), que, além de oferecer as mesmas características do POP, ele ainda dispõe de algumas características adicionais.

1.1 OBJETIVOS

O objetivo principal do trabalho desenvolvido é a especificação e a implementação de um protótipo de software para notificação de recebimento de novos *e-mail's*, como também o fornecimento de algumas informações adicionais sobre eles, baseado no protocolo IMAP. O período em que será feita cada verificação por novas mensagens será pré-definido pelo próprio usuário.

1.2 JUSTIFICATIVA

Tendo em vista o grande crescimento do número de usuários conectados à Internet, tornando-se assim cada vez mais indispensável a utilização do correio eletrônico e visando a redução de informações a serem trafegadas pela rede, observou-se a necessidade de um mecanismo que possibilitaria ao usuário ser notificado quando do recebimento de novos *e-mails.*

Observou-se também que grande parte dos atuais softwares "cliente de correio eletrônico" que já implementam a função de notificação, tanto através do protocolo IMAP quanto do POP, a implementação deste mecanismo para notificar o usuário do recebimento de

novas mensagens ainda não foi otimizada, ou seja, para identificar as novas mensagens o software busca todas as mensagens contidas no servidor, analisa individualmente cada uma e notifica o usuário.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado em sete capítulos, estando assim distribuídos:

No capítulo 1, encontramos a introdução, os objetivos, a justificativa e organização do trabalho.

No capítulo 2, apresentamos os principais protocolos de *e-mails*: POP e SMTP.

O capítulo 3, está reservado para o protocolo IMAP, que é o principal protocolo abordado neste trabalho.

No capitulo 4, temos um estudo sobre os protocolos de transmissão de dados utilizados no desenvolvimento do protótipo: TCP/IP e *Socket*;

O capítulo 5 faz a apresentação do protótipo desenvolvido, contendo a especificação, detalhes sobre a implementação e a descrição do seu funcionamento.

No capítulo 6, temos a conclusão e sugestões para trabalhos futuros.

E por fim temos o capítulo 7, que apresenta a referência bibliográfica que foi utilizada para a elaboração deste trabalho.

2 PROTOCOLOS DE CORREIO ELETRÔNICO

Existem vários protocolos de correio eletrônico. Os protocolos de correio eletrônico que não são proprietários, estão disponíveis na forma de RFC e podem ser encontrados junto ao site da IETF (*Internet Engineering Task Force*) ([IET2000]).

Os principais e mais utilizados protocolos de correio eletrônico na rede Internet são: SMTP, POP e IMAP.

2.1 SMTP (SIMPLE MAIL TRANSFER PROTOCOL)

O protocolo SMTP é utilizado para transmitir um *e-mail* do usuário de origem ***"A"*** até o usuário de destino ***"B"***, o usuário de destino pode ser um usuário da rede local ou pertencer a uma rede diferente.

Uma das mais importantes características da especificação deste protocolo é a capacidade de transferir *e-mail's* entre diferentes redes e inclusive utilizando protocolo de redes distintas, não necessariamente o protocolo TCP/IP.

O *e-mail* pode ser uma simples mensagem de texto, ou poderá ser uma mensagem contendo um arquivo em anexo, porém, o tratamento de qualquer destas mensagens se dará da mesma forma.

O procedimento de envio de um *e-mail* é realizado da seguinte forma: quando um usuário solicita o envio de *e-mail*, o transmissor-SMTP estabelece uma conexão com o receptor-SMTP; quando esta conexão estiver estabelecida o transmissor-SMTP envia os comandos e aguarda uma reposta válida do receptor-SMTP confirmando o seu recebimento, ou retornando o dado solicitado referente ao comando, ou ainda retornando um código de erro. Conforme o retorno que o transmissor-SMTP receber do receptor-SMTP, ele poderá continuar o processo de envio ou simplesmente abortar o envio.

O protocolo SMTP, conforme consta na sua RFC 821 ([POS1982]), define um conjunto de comandos que devem ser utilizados na implementação de um serviço de SMTP. Os comandos especificados são: *HELO, MAIL, RECIPIENT (RCPT), DATA, SEND, SEND*

OR MAIL (SOML), SEND AND MAIL (SAML), RESET (RSET), VERIFY (VRFY), EXPAND (EXPN), HELP, NOOP, QUIT e *TURN*.

Nem todos os comandos acima são obrigatórios, a implementação mínima exigida para um serviço de SMTP, consiste nos seguintes comandos: *HELO, MAIL, RCPT, DATA, RSET, NOOP* e *QUIT*.

Da mesma forma, foi estabelecida uma ordem na qual estes comandos devem ser utilizados.

Conforme exemplo abaixo, podemos observar uma simulação de envio de uma mensagem do usuário ***"A"*** que utiliza o endereço eletrônico "alexandre@knorst.com.br", até o usuário ***"B"*** e que utiliza o endereço eletrônico "tcc@knorst.com.br", utilizando o protocolo SMTP, e alguns comandos por ele definido:

S[1]: 220 smtp.bnu.terra.com.br; Mon, 25 Sep 2000 00:29:39 –0300
C[2]: HELO knorst.bnu.Terra.com.br
S: 250 smtp.bnu.terra.com.br Hello knorst.bnu.terra.com.br [200.248.247.35]
C: MAIL FROM: alexandre@knorst.com.br
S: 250 alexandre@knorst.com.br.. Sender ok
C: RCPT TO: tcc@knorst.com.br
S: 250 tcc@knorst.com.br.. Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Subject: Teste de envio de e-mail ...
C:
C: Ola ....
C:
C: Isto é uma mensagem de teste ..
C:
C: Alexandre E. knorst
C: .
S: 250 AAA10113 Message accepted for delivery
C: QUIT
S: 221 smtp.bnu.terra.com.br closing connection

[1] Usado para indicar um comando ou dado sendo enviado do Servidor para o Cliente.

[2] Usado para indicar um comando ou dado sendo enviado do Cliente para o Servidor.

A resposta aos comandos SMTP é sempre a ultima linha enviada pelo servidor para o cliente e é formado por uma linha iniciada com um número de três dígitos e seguindo com um texto descritivo, sendo que para interpretação da execução do comando somente é utilizado o número, sendo o texto desprezado.

Sendo que cada número retornado possui um significado, as mais tradicionais respostas podem ser encontrados na tabela 2.1.1.

TABELA 2.1.1: Respostas do servidor SMTP

Número	Descrição
211	Status do sistema ou resposta ao comando HELP;
214	Resposta ao comando HELP;
220	Serviço liberado;
221	Serviço finalizando a transmissão;
250	Requisição de email aceita com sucesso;
251	Usuário não local, email sendo encaminhado para <endereço>;
354	Inicia o recebimento da mensagem; terminar com <CRLF>.<CRLF>
421	Serviço não disponível;
450	Requisição de email recusada, caixa postal indisponível ou em uso;
451	Requisição abortada, erro no processamento;
452	Requisicão de email recusada, espaço insuficiente em disco;
500	Erro de sintaxe, comando não reconhecido;
501	Erro de sintaxe, parâmetros inválidos;
502	Comando não implementado;
503	Sequencia incorreta dos comandos;
504	Parâmetro do comando não implementado;
550	Requisicão de email recusada, caixa postal inexistente;
551	Usuário não local, tente enviar para o <endereço>;
552	Requisicão de email recusada, excete o espaço alocado;
553	Requisicão de email recusada, nome de caixa postal não permitido;
554	Transação abortada.

Mais informações e detalhes sobre a especificação e implementação deste protocolo podem ser encontradas junto a RFC 821 ([POS1982]).

2.2 POP (POST OFFICE PROTOCOL)

De acordo com a RFC 1939 ([MYR1996]), o protocolo POP versão 3 é um protocolo que permite que o usuário de forma dinâmica tenha acesso a sua caixa postal que fica

armazenada remotamente no servidor, ou seja, permitindo que ele faça a transferência dos *e-mail's* contidos no servidor para a estação local.

Em virtude de diversos requisitos que são exigidos para que o serviço de *e-mail* funcione adequadamente, fez surgir alguns fatores que impossibilitassem que a mensagem seja entregue diretamente ao computador do usuário, entre eles podemos mencionar: espaço em disco, capacidade de processamento, serviço de SMTP disponível e executando na estação, impossibilidade de estar conectado a rede durante um longo período e outros.

Para resolver estes problemas foram criados os chamados MTS (*Message Transport System*), que tem como função armazenar as mensagens dos usuários em caixas postais junto ao servidor, possibilitando que o usuário possa buscar estas mensagens futuramente.

Quando um usuário desejar verificar se existem novas mensagens em sua caixa postal, ele deverá utilizar um software de correio que implementa o protocolo POP, tal como: *Outlook Express da Microsoft, Eudora da Qualcomm, Netscape Messenger da Netscape*, etc..

O software de correio, para se comunicar com o MTS e transferir os *e-mail's* que estão armazenados no servidor, estabelece uma conexão com a porta[3] 110 via TCP e se utiliza de um conjunto de comandos para processar a transferência dos mesmos.

O servidor responde aos comandos enviados pelo cliente, utilizando duas palavras chaves, sendo elas: "+OK" quando o comando solicitado tiver sido executado com sucesso, sem erro e "-ERR" quando o comando conter erros. Estas palavras chaves obrigatoriamente devem estar em maiúsculo.

Os comandos disponíveis e especificados junto a RFC 1939 ([MYR1996]) são: *APOP, DELE, LIST, NOOP, PASS, QUIT, RETR, RSET, STAT, TOP, UIDL* e *USER.* A descrição em detalhes de cada um destes comandos pode ser encontra na própria RFC.

O processo de recebimento da mensagens passa por 3 (três) estados: *AUTHORIZATION, TRANSACTION* e *UPDATE.*

---

[3] Usado para identificar a aplicação que atende a este serviço junto ao Servidor. Cada serviço oferecido pelo servidor fica aguardando a conexão em uma porta pré-definida, como por exemplo, o SMTP por padrão aguarda conexões na porta 25.

No estado *"AUTHORIZATION"* o usuário faz a sua identificação junto ao servidor, para isso ele utiliza os comandos *USER* seguido com o seu *login* e *PASS* e a senha da conta a ser verificada, ou utilizando o comando *APOP* que é uma forma de autenticação que envia o *login* e a senha em uma linha criptografada com codificação utilizando o algoritmo MD5.

A conexão alcança o estado *"TRANSACTION"*, quando a identificação do usuário obter sucesso. Neste estado o usuário dispõe da relação dos comandos apresentados na tabela 2.2.1.

TABELA 2.2.1: Comandos do protocolo POP no estado TRANSACTION

Comando	Parâmetros	Descrição
STAT	Nenhum	Mostra informações da caixa-postal
LIST	Nº mensagem	Mostra informações da mensagem especifica
RETR	Nº mensagem	Lista a mensagem solicitada
DELE	Nº mensagem	Marca a mensagem para ser removida
NOOP	Nenhum	Verifica se a conexão com o servidor ainda está ativa
RSET	Nenhum	Volta ao estado inicial do *"TRANSACTION"*.

O estado de *"UPDATE"* é alcançado pelo comando *"QUIT"*, que é utilizado para finalizar a conexão, neste estado o servidor remove todas as mensagens marcadas para ser excluídas e libera a caixa postal para que outras conexões tenham acesso a mesma.

Mais informações e detalhes sobre a especificação e implementação deste protocolo podem ser encontradas junto a RFC 1939 ([MYR1996]).

3 IMAP (*INTERNET MESSAGE ACCESS PROTOCOL*)

De acordo com a RFC 2060 ([CRI1996]), o protocolo IMAP - *Version* 4rev1 (IMAP4rev1), permite que o usuário possa acessar e manipular diretamente as suas mensagens armazenadas no servidor. Este protocolo permite manipular pastas remotas chamadas “caixas postais”, como se estas pastas fossem pastas locais.

O protocolo IMAP além de possuir as mesmas características do protocolo POP, ainda permite que o usuário mantenha todas as suas mensagens armazenadas junto ao servidor, podendo ser acessadas a partir de qualquer computador que tenha instalado e habilitado um software que implemente o protocolo IMAP.

Além do mais, o protocolo IMAP4rev1 inclui ainda operações para criar, apagar e renomear caixas postais; verificar por novas mensagens; apagar mensagens definitivamente; limpar e configurar os *FLAG'S*; buscar informações sobre as caixas postais ou mensagens e outras características que serão discutidos no decorrer deste trabalho.

O IMAP4rev1 não trata de que forma a mensagem chega até ao servidor, pois, isto é tarefa do SMTP ([POS1982]).

O texto apresentado na seqüência (Capítulo 3) é referente a RFC 2060 [CRI1996].

3.1 CARACTERÍSTICAS DO IMAP

O protocolo IMAP utiliza um canal de comunicação de dados baseado em TCP. Pela especificação do protocolo, o servidor IMAP por padrão aguarda por conexões na porta 143.

Uma conexão com um servidor IMAP4rev1 consiste em estabelecer uma conexão de rede cliente/servidor, na qual o cliente interage com o servidor e vice-versa através de comandos e respostas. Esta interação se constitui com os comandos que são enviados pelo cliente e o servidor responde com os dados solicitados ou com o resultado do comando enviado pelo cliente.

Toda a interação entre o servidor e o cliente é através de linhas que terminam com os dois caracteres CR (C*arrier Return*) e LF (*Line Feed*) indicando o final da linha ou comando.

Cada comando solicitado pelo cliente começa com um identificador alfa-numérico (ex. A0001, A0002...) chamado de *"rótulo"*. O cliente deve fornecer automaticamente um novo *rótulo* para cada comando enviado ao servidor, este *rótulo* deve ser único durante aquela conexão.

Em alguns comandos o servidor necessita que se forneça informações adicionais ou complementares. Neste caso, o servidor envia uma resposta para o cliente solicitando o restante dos dados pertinentes ao comando, esta solicitação será identificada pelo cliente quando a resposta enviada pelo servidor estiver iniciada pelo caracter "+" (mais).

Quando o servidor detectar um erro no comando ou no dado enviado pelo cliente, ele rejeita o comando com uma resposta de negação, evitando desta forma que o cliente lhe envia qualquer outro comando. Esta resposta de negação será identificada pela palavra chave *"BAD"* (ex.: A0001 BAD Comand Unrecognized).

Quando o servidor enviar os dados do comando solicitado pelo cliente, estes dados podem estar contidos em uma única linha ou em mais linhas. Quando estiver em mais de uma linha para cada linha que pertencer àquele comando será adicionado o caracter "*" (asterisco) no início da linha.

Para cada comando solicitado pelo cliente, o servidor envia, além dos dados, uma resposta indicando se o comando obteve sucesso ou não. As possíveis respostas são: "*OK*" quando o comando obteve sucesso; "*NO*" quando o comando solicitado terminar com erros e "*BAD*" quando o comando enviado pelo cliente não for um comando válido ou a sintaxe do mesmo estiver incorreta.

As respostas dos comandos que o servidor enviar ao cliente serão sempre no formato *"<rótulo> <status do comando> <complemento>"*. Por exemplo:

A0003 OK NOOP completed

Onde:

"A0003" corresponde ao *rótulo*, sendo que é igual ao rótulo do comando correspondente;

"OK" corresponde ao status do comando;

"NOOP completed" ao complemento.

O cliente deve estar preparado para receber qualquer tipo de resposta do servidor.

3.2 ATRIBUTOS DE UMA MENSAGEM

Em adicional ao texto, cada mensagem pode conter atributos relacionadas a ela. Estes atributos podem ser recuperados de forma individual ou junto com o texto da mensagem.

3.2.1 IDENTIFICAÇÃO DAS MENSAGENS

Cada mensagem pode ser acessada através de sua identificação única ou pelo número seqüencial da mensagem.

Em cada mensagem é adicionado um atributo de 32-bit que será utilizado como sendo a identificação única (UID). O algoritmo utilizado para gerar esta identificação garante que não existirá duas mensagens com a mesma identificação na caixa postal, sendo que, para cada mensagem que for adicionada junto a caixa postal será criada uma nova identificação e que será superior a mensagem anterior.

A identificação única é a mesma em qualquer sessão, permitindo desta forma que o cliente possa sincronizar a sua caixa postal armazenada localmente com a pasta remota.

O número seqüencial é um número compreendido entre 1 (um) e o número de mensagens contidas na caixa postal. Para cada mensagem nova que for adicionada a caixa postal, será atribuído um novo número, este número será formado pelo incremento de 1 com o número de mensagens já contidas na caixa postal antes da nova mensagem ser adicionada.

A seqüência dos números das mensagens pode ser alterada durante uma sessão. Por exemplo, quando uma mensagem for excluída da caixa postal, todas as mensagens subsequentes serão decrementadas.

3.2.2 ATRIBUTO DE *FLAG* NAS MENSAGENS

Um conjunto de zero ou mais palavras chaves podem ser associadas com a mensagem, estas palavras chaves são os conhecidos como *"FLAG's"*. Existem dois tipos básicos de *FLAG's*, os *FLAG's* permanentes que contêm informações que não mudam durante uma sessão e outra e os *FLAG's* da sessão que podem conter informações diferentes após uma conexão.

Os *FLAG's* permanentes poderão ser incluídas e alteradas pelo próprio usuário, e são visualizados em qualquer sessão, diferente dos *FLAG's* de sessão que são válidas somente naquela sessão.

Um *FLAG* do sistema é um *FLAG* pré-definido na implementação. Todos os *FLAG* de sistema iniciam com "\". A relação dos *FLAG's* de sistema atualmente definidos podem ser encontrados na tabela 3.2.1:

TABELA 3.2.1: *FLAG's* de sistema

FLAG	Descrição
\Seen	Indicando uma mensagem lida
\Answered	Indicando uma mensagem respondida
\Flagged	Indicando uma mensagem com *FLAG* de Urgente / Atenção especial
\Deleted	Indicando uma mensagem marcada para ser excluída
\Draft	Indicando uma mensagem sendo elaborada.
\Recent	Indicando uma mensagem recebida recentemente.

A *FLAG* "\Recent" somente é visualizada na primeira sessão, as sessões subsequentes não receberão mais esta *FLAG*. Esta *FLAG* não pode ser alterada pelo cliente.

Quando houverem duas conexões simultâneas acessando a mesma caixa postal, neste caso, não tem como se definir quem irá receber a *FLAG* "\Recent" e quem não irá receber esta *FLAG*.

3.2.3 ATRIBUTO DO HORÁRIO

Data e hora interna da mensagem no servidor. Esta não é a data e hora contida no cabeçalho da mensagem, mas, a data e hora em que a mensagem foi recebida no servidor. Se a mensagem tiver sido recebida via SMTP, a data e hora será a do último servidor.

3.3 TEXTO DA MENSAGEM

Além de poder buscar a mensagem toda, o IMAP4rev1 permite que se busque somente uma parte da mensagem, possibilitando que se recupere separadamente o cabeçalho da mensagem, parte do corpo, um determinado campo e ou ainda a mensagem toda.

3.4 ESTADOS

Uma conexão com o servidor IMAP4ver1 está dividido em quatro estados. Muitos dos comandos somente serão validos em um determinado estado. Quando se tentar utilizar um comando que não pertence ao estado atual, o servidor irá responder ao cliente com o resultado de negação "BAD" ou "NO", dependendo do tipo de erro. Os quatro estados são:

a) Estado "*NON-AUTHENTICATED*"

A conexão entra neste estado logo após a conexão tiver sido estabelecida. Neste estado o cliente deve solicitar a autenticação e efetuar o *login*. A conexão ficará neste estado até cliente ter sido autenticado com sucesso;

b) Estado "*AUTHENTICATED*"

Neste estado o cliente deve selecionar a caixa postal na qual os comandos vão estar relacionados. A conexão entrará neste estado quando a autenticação do cliente obter sucesso, ou quando houver algum erro ao selecionar a caixa postal;

c) Estado "*SELECTED*"

Neste estado, uma caixa postal está selecionada para ser acessada. Este estado é alcançado quando uma caixa postal tiver sido selecionada com sucesso;

d) Estado "*LOGOUT*"

No estado de *Logout,* é quando a conexão estiver sendo finalizada e o servidor irá fechar a conexão. Este estado pode ser alcançado por uma solicitação do cliente ou por uma decisão do servidor.

3.5 FORMATO DOS DADOS

IMAP4rev1 utiliza comandos e respostas na forma de texto. Os dados em IMAP4rev1 podem ser das seguintes formas:

a) ATOM

Um *ATOM* consiste em um ou mais caracteres não especiais;

b) Número

Número consiste em um ou mais caracteres numéricos, e é usado para representar um valor numérico;

c) STRING

Uma *STRING* pode ser de duas formas: literal ou *STRING* entre aspa. Literal é a forma tradicional de uma *STRING*; *STRING* entre aspas, é uma forma alternativa que permite a utilização de caracteres não permitidos pela forma literal;

d) Lista entre parênteses

Estrutura de dados representada como uma lista entre parênteses; é uma seqüência de itens de dados separados por espaços e cada item indica um único parâmetro. Uma lista de parênteses pode estar dentro de uma outra lista de parênteses;

e) NIL

Usado para indicar que o parâmetro não existe ou não possui valor atribuído. Pode ser usado para substituir a representação de uma string vazia "" (duas aspas) ou numa lista de parênteses sem itens () (abre e fecha parênteses).

3.6 CONSIDERAÇÕES OPERACIONAIS

A interpretação do nome das caixas postais depende da implementação. Porém, o nome "*INBOX*" tanto em maiúsculo como em minúsculo é um caso especial de nome, é o nome reservado que é dado à primeira caixa postal que o usuário recebe junto ao servidor.

Em qualquer momento o servidor poderá enviar dados referente a caixa postal, sem que o cliente os tenha solicitado, solicitando ao cliente de forma explícita que sejam atualizadas as informações da caixa postal. Como por exemplo, quando o servidor adicionar uma nova mensagem junto a caixa postal; quando for alterado um *FLAG* da mensagem (quando duas ou mais conexões estiverem utilizando a mesma caixa postal simultaneamente); quando uma mensagem for removida da caixa postal. Este envio destas informações se dará após execução de qualquer comando, exceto ao comando *EXPUNGE*, a ser analisado nas próximas seções.

Se o servidor possuir uma função de controle de inatividade do tipo *autologout timer*, a duração deve ser no mínimo de 30 minutos. Qualquer comando que for enviado pelo cliente irá reiniciar a contagem.

3.7 COMANDOS DO CLIENTE

Os comandos estão organizados pelo estado em que eles têm permissao a serem executados. Alguns comandos podem ser utilizados em mais de um estado, neste caso ele aparecerá no estado mais inferior da conexão.

3.7.1 COMANDOS DE QUALQUER ESTADO

Os comandos que estão disponível em qualquer estado da conexão são: CAPABILITY, NOOP e LOGOUT. Estes comando também podem ser chamados de *comandos universais*.

O comando CAPABILITY requisita ao servidor uma lista de opções suportados pelo servidor. Por exemplo, a opção que começa com "*AUTH=*" identifica o mecanismo de

autenticação a ser utilizado. As outras opções podem ser informações em relação a extensões, revisões ou correções desta especificação.

Exemplo:

```
C: a0003 CAPABILITY
S: * CAPABILITY IMAP4VER1 SORT IDLE SCAN AUTH=LOGIN
S: a0003 OK CAPABILITY completed
```

A função principal do comando NOOP é reiniciar o contador do tempo de inatividade da conexão. Em algumas implementações este comando pode ser utilizado para verificar por novas mensagens ou para verificar o estado da caixa postal selecionada, as informações serão respondidas sem o rótulo.

Exemplo:

```
C: A0009 NOOP
S: * 12 EXPUNGE
S: * 5 EXISTS
S: * 1 RECENT
S: A0009 NOOP Completed
```

O comando LOGOUT informa ao servidor que o cliente deseja finalizar a conexão.

Exemplo:

```
C: A0048 LOGOUT
S: * BYE knorst.bnu.terra.com.br IMAP4rev1 server terminating connection
S: A0048 LOGOUT completed
```

3.7.2 COMANDOS DO ESTADO "*NON-AUTHENTICATED*"

Neste estado o cliente deve utilizar um destes dois comandos LOGIN ou AUTHENTICATE para efetuar a autenticação e entrar no estado de "*AUTHENTICATED*". O comando AUTHENTICATE utiliza um mecanismo de autenticação que pode variar, porém, o comando LOGIN utiliza tradicionalmente o *login* e senha sem criptografia.

O comando AUTHENTICATE é usado para identificar um mecanismo de proteção a ser utilizado, os mecanismos que poderão ser utilizados estão descritos na RFC 1731 ([MYE1982]). Os principais mecanismo disponíveis são: KERBEROS_V4, GSSAPI e SKEY.

Caso o comando de AUTHENTICATE seja executado com sucesso, a partir deste instante toda a interação dos demais comandos será utilizando este mecanismo de proteção.

Exemplo:

```
C: A0054 AUTHENTICATE KERBEROS_V4
S: + AmFYig==
C:  BAcAQU5EUkVXLkNNVS5FRFUAOCAsho84kLN3/IjmrMG+25a4DT+nZI
mJjnTNHJUtxAA+o0KPKfHEcAFs9a3CL5Oebe/ydHJUwYFdWwuQ1MWiy6Ie
sKvjL5rL9WjXUb9MwT9bpObYLGOKi1Qh
S: + or//EoAADZI=
C: DiAF5A4gA+oOIALuBkAAmw==
S: A0054 OK Kerberos V4 authentication successful
```

O comando LOGIN é uma forma de autenticação que não utiliza nenhum tipo de proteção, tanto o *login* como a senha são informados em texto sem criptografia.

Exemplo:

```
C: A0060 LOGIN knorst minhasenha
S: A0060 OK LOGIN Completed
```

3.7.3 COMANDOS DO ESTADO "*AUTHENTICATED*"

Os comandos permitidos neste estado, são comandos que basicamente manipulam as caixas postais. Exceto os comandos SELECT e EXAMINE, que são utilizados para entrar no estado a seguir da conexão, que é o estado SELECTED.

Os comandos são: SELECT, EXAMINE, CREATE, DELETE, RENAME, LIST, STATUS e APPEND.

O comando SELECT é utilizado para selecionar a caixa postal a qual o usuário desejar acessar. Antes do servidor retornar o resultado da execução do comando, o servidor deve enviar para o cliente os dados sem o rótulo. Estes dados basicamente consistem em: os *FLAG'S*; o número de mensagens contidas na caixa postal (*EXISTS*); o número de mensagens nova (*RECENT*); número de identificação único (*UID*); o número da primeira mensagem não lida (*UNSEEN*); entre outros mais.

O cliente somente poderá selecionar uma única caixa postal por vez. Ao selecionar uma outra caixa postal o servidor automaticamente libera a caixa postal selecionada anteriormente.

Quando o cliente tiver permissão para alterar as informações referentes a caixa postal selecionada, o servidor ao enviar o resultado do comando executado, será incluído o prefixo "[*READ-WRITE*]", caso contrário quando for somente para leitura, não tendo permissão de alterar as informações a resposta será com o prefixo "[*READ-ONLY*]".

Exemplo:

```
C: A142 SELECT INBOX
S: * 172 EXISTS
S: * 1 RECENT
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S: A142 OK [READ-WRITE] SELECT completed
```

O comando EXAMINE é similar ao comando SELECT, a única diferença é em relação a permissão de acesso, o comando EXAMINE seleciona a caixa postal em modo somente leitura "[*READ-ONLY*]".

Exemplo:

```
C: A932 EXAMINE inbox
S: * 17 EXISTS
S: * 2 RECENT
S: * OK [UNSEEN 8] Message 8 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS ()] No permanent flags permitted
S: A932 OK [READ-ONLY] EXAMINE completed
```

O comando CREATE é utilizado para criar novas caixas postais. É permitido também criar caixas postais dentro de caixas postais já existentes. Porém, somente é permitido vincular novas caixas postais nas caixas postais que tiverem o *FLAG* "\NoSelect".

Exemplo:

```
C: A004 CREATE principal/auxiliar
```

```
S: A004 OK CREATE completed
```

O comando DELETE serve para remover definitivamente uma caixa postal do servidor. Este comando não permite que se remova as caixas postais que tiverem vinculadas outras caixas postais a ela. Ao remover uma caixa postal ele remove também todas as mensagens nele armazenadas e não podendo mais ser recuperadas.

Exemplo:

```
C: A067 DELETE principal
S: A067 NO Name "principal" has inferior hierarchical names
C: A068 DELETE principal/auxiliar
S: A068 OK DELETE Completed
C: A069 DELETE principal
S: A069 OK DELETE Completed
```

O comando RENAME é utilizado para alterar o nome de uma caixa postal para outro nome. Caso, esta caixa postal tiver vinculada a ela outras caixas postais, automaticamente será alterado o nome destas caixas postais também.

Exemplo:

```
C: A070 LIST "principal/" *
S: * LIST (\NoSelect) "/" principal/
S: * LIST (\NoInferiors) "/" principal/auxiliar
S: A070 OK LIST completed
C: A071 RENAME principal old-principal
S: A071 OK RENAME completed
C: A072 LIST "old-principal/" *
S: * LIST (\NoSelect) "/" old-principal/
S: * LIST (\NoInferiors) "/" old-principal/auxiliar
S: A072 OK LIST completed
```

O comando LIST é utilizado para listar (catalogar) as caixas postais que se encontram disponíveis no servidor. Este comando permite listar todas as caixas postais; utilizar curingas (*) para filtrar a saída; ou listas somente as caixas postais que se encontram disponíveis em uma determinada caixa postal.

Exemplo:

```
C: A080 LIST "" ""
```

```
S: * LIST (\NoSelect) "/" ""
S: * LIST (\NoSelect) "/" principal/
S: * LIST (\NoInferiors) "/" principal/auxiliar
S: A080 OK LIST completed
```

O comando STATUS é utilizado para verificar o status da caixa postal. Este não afeta nenhum *FLAG* das mensagens armazenadas na caixa postal.

Este comando não é muito eficaz, para obter todas as informações referente a caixa postal selecionada, o servidor é obrigado abrir toda a caixa postal e coletar estas informações, a caixa postal será aberta em "[*READ-ONLY*]", garantindo que nenhum *FLAG* será alterado. Os principais itens de consulta que poderão ser utilizados são: *MESSAGES*, que indica o número de mensagens contidas na caixa postal; *RECENT*, o número de mensagens novas; *UIDNEXT*, o próximo *UID* que será utilizado para grava uma nova mensagem; *UIDVALIDITY*, o *UID* da própria caixa postal e o *UNSEEN*, que indica o número de mensagens ainda não lidas.

Exemplo:

```
C: A083 STATUS principal (UIDNEXT MESSAGES RECENT)
S: * STATUS principal (RECENT 3 MESSAGES 23 UIDNEXT 8979)
S: A083 OK STATUS completed
```

O comando APPEND permite que sejam adicionadas mensagens a uma caixa postal determinada. Ao utilizar o comando APPEND deve se levar em consideração que o formato e os campos (To:, From:, Date:, Subject:, etc) que pertencem ao cabeçalho da mensagem, devem estar de acordo com o padrão definido na RFC 822 ([CRO1982]).

O comando APPEND somente é permitido para adicionar mensagens nas caixas postais armazenadas naquele servidor, pois, este comando não possui nenhum mecanismo de transporte, como o SMTP.

Exemplo:

```
C: A090 APPEND principal (\Seen) {310}
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
C: From: Fred Foobar <alexandre@knorst.com.br
C: Subject: Teste
C: To: tcc@knorst.com.br
C: Message-Id: <B27397-0100000@knorst.com.br>
```

```
C: MIME-Version: 1.0
C: Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
C:
C: Ola ..   este é mais uma mensagem de teste.
C:
C: .
S: A090 OK APPEND completed
```

3.7.4 COMANDOS DO ESTADO "*SELECTED*"

Todos os comando que podem ser utilizados estado "*AUTHENTICATED*", também se encontram disponíveis neste estado, mais os comandos CHECK, CLOSE, EXPUNGE, SEARCH, FETCH, STORE, COPY e UID que são comandos permitidos somente neste estado.

O comando CHECK solicita ao servidor que seja feita uma atualização da caixa postal selecionada. Em algumas implementações este comando é similar ao comando NOOP.

Exemplo:

```
C: A090 CHECK
S: A090 OK CHECK completed
```

O comando CLOSE remove todas as mensagens que estiverem com a *FLAG* \Deleted configurada e retorna ao estado de "*AUTENTICATED*", possibilitando que se selecione outra caixa postal.

Exemplo:

```
C: A093 CLOSE
S: A093 OK CLOSE completed
```

O comando EXPUNGE simplesmente remove todas as mensagens da caixa postal selecionada que estiverem com a *FLAG* \Deleted ativa.

Exemplo:

```
C: A095 EXPUNGE
S: * 3 EXPUNGE
S: * 7 EXPUNGE
S: A095 OK EXPUNGE completed
```

O comando SEARCH efetua busca por mensagens que se enquadram com os argumentos de pesquisa especificados. O argumento de pesquisa é especificado utilizando algumas palavras chaves, as quais serão apresentados na tabela 3.7.1.

Um argumento de pesquisa pode ser composto por vários argumentos de pesquisa, o resultado será a intersecção de todos os argumentos, ou seja, ao executarmos o comando SEARCH utilizando o argumento de pesquisa como sendo “FROM "KNORST" SUBJECT "TCC"”, o servidor irá retornar a relação de todas as mensagens que foram enviadas pelo usuário “KNORST” e que no campo do assunto tiver a palavra “TCC”.

Os argumentos que poderão ser utilizados para montar uma consulta são:

TABELA 3.7.1: Argumentos do comando SEARCH

Argumento	Parâmetro	Descrição
ALL		Todas as mensagens desta caixa postal.
ANSWERED		Mensagens que estiverem com a *FLAG* "\Answered".
BCC	<string>	Mensagens que contiverem a <string> especificada no campo BCC.
BEFORE	<data>	Mensagens que estiverem com a <data> anterior a data especificada.
BODY	<string>	Mensagens que tiverem a <string> especificada no corpo da mensagem.
CC	<string>	Mensagens que contiverem a <string> especificada no campo CC.
DELETED		Mensagens que estiverem com a *FLAG* "\Deleted".
DRAFT		Mensagens que estiverem com a *FLAG* "\Draft".
FLAGGED		Mensagens que estiverem com a *FLAG* "\Flagged".
FROM	<string>	Mensagens que contiverem a <string> especificada no campo "FROM".
HEADER	<campo> <string>	Mensagens que tiverem o <campo> especificado no cabeçalho e com a informação <string> selecionada.
KEYWORD	<*FLAG*>	Mensagens com a *FLAG* especificada.
LARGER	<tamanho>	Mensagens maiores que o <tamanho> especificado.
NEW		Mensagens que tiverem a *FLAG* "\Recent", mas não tiverem a *FLAG* "\Seen”.
NOT	<argumento de pesquisa>	Negação do argumento de busca.
OLD		Mensagens que não tiverem a *FLAG* "\Recent".
ON	<data>	Mensagens recebidas na <data> especificada.
OR	<argumento1> <argumento2>	Mensagens que se adequarem a qualquer de umas das duas consultas.
RECENT		Mensagens que tiverem a *FLAG* "\Recent".

SEEN		Mensagens que tiverem a *FLAG* "\Seen".
SENTBEFORE	<data>	Mensagens enviadas antes da <data> especificada.
SENTON	<data>	Mensagens enviadas na <data> especificada.
SENTSINCE	<data>	Mensagens enviadas após a <data> especificada.
SINCE	<data>	Mensagens que foram enviadas na <data> especificada, ou após esta <data>.
SMALLER	<tamanho>	Mensagens que forem menor ou igual ao <tamanho> especificado.
SUBJECT	<string>	Mensagens que tiverem no campo Subejt a <string> especificada.
TEXT	<string>	Mensagens que tiverem a <string> especificada em seu corpo.
TO	<string>	Mensagens que foram enviadas para a <string> especificada.
UID	<número>	Mensagem com a identificador único igual ao <número> especificado.
UNANSWERED		Mensagens que não estiverem com a *FLAG* "\Answered".
UNDELETED		Mensagens que não estiverem com a *FLAG* "\Deleted".
UNDRAFT		Mensagens que não estiverem com a *FLAG* "\Draft".
UNFLAGGED		Mensagens que não estiverem com a *FLAG* "\Flagged".
UNKEYWORD	<*FLAG*>	Mensagens que não conter a *FLAG* especificada.
UNSEEN		Mensagens que não tiverem a *FLAG* "\Seen".

A relação completa de todos os argumentos de pesquisa, como também a descrição em detalhes destes argumentos, poderá ser encontrada na especificação da RFC 2060 ([CRI1996]).

Exemplo:

```
C: A193 SEARCH RECENT FROM “knorst” SUBJECT “Tcc”
S: SEARCH 2 35 64 125
S: A193 OK SEARCH completed
```

O comando FETCH é utilizado para recuperar dados associados a uma mensagem na caixa postal.

Os parâmetros que podem ser utilizados para recuperar estes dados são: ALL, BODY, BODY[], BODY.PEEK[], BODYSTRUCTURE, ENVELOPE, FAST, FLAGS, FULL, INTERNALDATE, RFC822, RFC822.HEADER, RFC822.SIZE, RFC822.TEXT e UID.

O parâmetro BODY[] é utilizado para recuperar informações mais especificas relacionadas ao cabeçalho da mensagem, tais como: o campo "TO:", "FROM:", "SUBJECT:" e outros. Este parâmetro implicitamente atribui a *FLAG* "\Seen" a mensagem, indicando que a mensagem já foi vista ou acessada, que é justamente a diferença entre este parâmetro e o parâmetro BODY.PEEK[].

As opções que pode ser utilizados em conjunto com os comandos BODY[] e BODY.PEEK[] para recuperar estes dados mais específicos são: HEADER, HEADER.FIELDS, HEADER.FIELDS.NOT e TEXT.

Exemplo:

```
C: A200 FETCH 1 (FLAGS BODY[HEADER.FIELDS (FROM)])
S: * 1 FETCH (FLAGS (\Seen) BODY[HEADER.FIELDS ("FROM")] {42}
S: From: Stefan Haase <stefan@furb.rct-sc.br>
S: A200 OK FETCH completed
```

O comando STORE é usado para manipular os *FLAG'S* de uma ou mais mensagens. Após a execução do comando STORE o servidor responde ao cliente o status atual das FLAGS após a alteração.

A relação dos parâmetros que podem ser utilizados para alterar os *FLAG'S*, podem ser encontradas na tabela 3.7.2.

TABELA 3.7.2: Relação dos parâmetros para manipular as *FLAG'S*

Parâmetro	Descrição
FLAGS	Substitui os atuais *FLAG'S* da mensagens pelas *FLAG'S* fornecidas.
FLAGS.SILENT	Mesma função que o parâmetro *FLAG*, porem, o servidor não responde ao cliente o novo valor.
+FLAGS	Adiciona a *FLAG* fornecida junto com as *FLAG'S* já existentes na mensagem.
+FLAGS.SILENT	Mesma função que o parâmetro +*FLAG*, porem, o servidor não responde ao cliente o novo valor.
-FLAGS	Remove a *FLAG* fornecida da mensagem.
-FLAGS.SILENT	Mesma função que o parâmetro -*FLAG*, porem, o servidor não responde ao cliente o novo valor.

Exemplo:

```
C: A233 STORE 4:6 +FLAGS (\Seen)
```

```
S: * 4 FETCH FLAGS (\Seen)
S: * 5 FETCH FLAGS (\Seen \Deleted)
S: * 6 FETCH FLAGS (\Seen \Flagged \Draft)
S: A233 OK STORE completed
```

O comando COPY é utilizado para copiar uma ou mais mensagens de uma determinada caixa postal para outra. O comando COPY preservar as informações dos *FLAG'S* e a data interna da mensagem.

Exemplo:

```
C: A263 COPY 5:7 principal
S: A263 OK COPY completed
```

O comando UID é usado quando se deseja executar um comando especificado junto as mensagens que possui um determinado intervalo de identificadores únicos (UID´s).

O comando UID pode ser usado em conjunto com um dos comandos, COPY, FETCH, STORE ou SEARCH e seus parâmetros já vistos anteriormente.

Além de especificar o intervalo dos UID´s, alguns comandos permitem que seja especificado o intervalo do número seqüencial das mensagens, neste caso, o servidor irá executar o comando somente quando as duas condições estiverem satisfeitas.

Exemplo:

```
C: A267 UID SEARCH 1:20 ANSWERED
S: * SEARCH 10 11 12
S: A267 UID SEARCH completed
C: A268 UID FETCH 10:12 (FLAGS UID)
S: * 5 FETCH (UID 10 FLAGS (\Answered))
S: * 6 FETCH (UID 11 FLAGS (\Answered))
S: * 7 FETCH (UID 12 FLAGS (\Answered))
S: A268 OK UID FETCH completed
```

Além dos comandos acima mencionados e abordados, temos comandos que são conhecido como comandos experimentais ou expansões, são os comandos que iniciam com a letra "X". Estes comandos não são especificados pela RFC 2060, porém, a sua descrição da funcionalidade e utilização podem ser encontrada junto a IESG (*Internet Engineering Steering Group*) ([IES2000]).

Exemplo:

```
C: a441 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=KERBEROS_V4 XPIG-LATIN
S: a441 OK CAPABILITY completed
C: A442 XPIG-LATIN
S: * XPIG-LATIN ow-nay eaking-spay ig-pay atin-lay
S: A442 OK XPIG-LATIN ompleted-cay
```

3.8 RESPOSTAS DO SERVIDOR

Existem três tipos de respostas do servidor: resposta de estado, dados e requisição de continuação de um comando.

O cliente deve estar preparado para aceitar uma reposta do servidor a qualquer momento.

Respostas de estado podem conter ou não o rótulo. Quando a resposta estiver com o rótulo, isso indica que é o resultado do comando solicitado pelo cliente.

Algumas respostas de estado e todos os dados enviados pelo servidor são sem rótulo. Uma resposta sem rótulo é indicada iniciando com o caracter “*”. Na requisição de continuação de um comando a resposta inicia com o caracter “+”.

3.8.1 RESPOSTAS DO SERVIDOR – RESPOSTA DE ESTADO

As respostas de estado são: OK, NO, BAD, PREAUTH e BYE. OK, NO e BAD são respostas que podem ou não conter o rótulo. Já as respostas PREAUTH e BYE são sempre sem rótulo.

As respostas de estado podem conter código de respostas opcionais. Os códigos de respostas estão entre colchetes (" ["]") na forma de *ATOM's* e os argumentos estão separados por espaços.

Os códigos de respostas atualmente definidos podem ser encontrados junto a tabela 3.8.1.

TABELA 3.8.1: Códigos de Respostas

Código	Descrição

ALERT	Utilizado para chamar a atenção do usuário.
NEWNAME	Resposta utilizada quando uma caixa postal muda de nome ou é renomeada.
PARSE	Um erro encontra ao processar as informações da mensagem.
PERMANENTFLAGS	A lista de *FLAG'S* que o usuário poderá utilizar pelo comando STORE.
READ-ONLY	A caixa postal está selecionada para somente leitura.
READ-WRITE	A caixa postal está selecionada para leitura e gravação.
TRYCREATE	Uma resposta de erro indicando que o comando APPEND ou COPY, foram executados e a caixa postal de destino não existia.
UIDVALIDITY	Indica o UID da caixa postal.
UNSEEN	Indica a primeira mensagem não lida.

3.8.1.1 RESPOSTA "OK"

Quando a resposta OK estiver com o rótulo, isso indica que o comando que o cliente havia solicitado foi executado com sucesso. Quando a resposta estiver sem o rótulo, a resposta indica somente uma mensagem de informação.

Exemplo:

```
S: * OK knorst.bnu.terra.com.br IMAP4rev1 v12.264 server ready
C: A000 LOGIN knorst secret_password
S: A000 OK LOGIN completed
```

3.8.1.2 RESPOSTA "NO"

Quando esta resposta NO estiver com o rótulo, isso indica que o comando enviado pelo cliente não foi executado com sucesso. Quando estiver sem o rótulo, isso indica somente uma mensagem de aviso, porém, o comando pode ter sido executado com sucesso.

Exemplo:

```
C: A004 SELECT principal
S: A004 NO SELECT failed: Can't open principal: not a selectable mailbox
```

3.8.1.3 RESPOSTA "BAD"

A resposta BAD indica uma mensagem de erro. Quando a resposta estiver com um rótulo, ela indica um erro no comando, podendo este erro ter sido causado pela falta de parâmetros, ou na forma incorreta na qual eles foram especificados. Quando estiver sem o rótulo o servidor não conseguiu identificar o comando que possa ter causado este erro, ou ainda poderá ser um erro interno do servidor.

Exemplo:

```
C: A014 LIST
S: A014 BAD Missing required argument to LIST
C: A015 EXPUNGE
S: * BAD Disk crash, attempting salvage to a new disk!
S: * OK Salvage successful, no data lost
S: A015 OK EXPUNGE completed
```

3.8.1.4 RESPOSTA "PREAUTH"

Esta resposta sempre é sem rótulo e indica que a conexão já está autenticada por um outro meio externo e não é necessário executar o comando LOGIN.

Exemplo:

```
C: <estabelece conexão com o servidor>
S: * PREAUTH IMAP4Rev1 server logged in as knorst
```

3.8.1.5 RESPOSTA "BYE"

Esta resposta sempre é sem rótulo e indica que o servidor irá fechar a conexão. A conexão pode ser fechada por um dos quatro motivos abaixo:

a) O cliente solicitou fechar a conexão pelo comando LOGOUT;

b) Com um *SHUTDOWN* no servidor.

c) Através da inatividade da conexão (*autologout*);

d) O servidor recebe a conexão e por algum motivo não está habilitado a atender este pedido de conexão.

Exemplo:

```
C: A0048 LOGOUT
S: * BYE knorst.bnu.terra.com.br IMAP4rev1 server terminating connection
S: A0048 LOGOUT completed
```

3.8.2 RESPOSTAS DOS SERVIDOR – SERVIDOR E CAIXA POSTAL

Estas respostas são sempre sem rótulo. Esta resposta indica o estado do servidor ou da caixa postal, que são transmitidos do servidor para o cliente. Muitas destas respostas são resultados provenientes de um comando que o cliente solicitou.

3.8.2.1 RESPOSTA "CAPABILITY"

A resposta CAPABILITY é resultado do comando CAPABILITY. A resposta é composta com a lista de aptidões que o servidor suportar.

A aptidão que começar com "*AUTH=*" indica o tipo de autenticação que se encontra disponível para a conexão. As aptidões que iniciarem com "X" são extensões ao protocolo IMAP e sua documentação de descrição podem ser encontradas junto a IANA (*Internet Assigned Numbres Authority*).

Exemplo:

```
C: a0003 CAPABILITY
S: * CAPABILITY IMAP4VER1 SORT AUTH=LOGIN XPIG-LATIN
S: a0003 OK CAPABILITY completed
```

3.8.2.2 RESPOSTA "LIST"

A resposta LIST é o resultado do comando LIST. Na resposta que o servidor envia ao cliente além da palavra-chave LIST, o servidor ainda envia algumas informações adicionais, tais como: lista de atributos; caracter delimitador e o nome da caixa postal.

Os atributos que o comando LIST pode retornar, podem se encontrados na tabela 3.3.

TABELA 3.8.2: Atributos do comando LIST

Atributo	Descrição
\NoInferiors	Indica que não é possível ter caixas postais, na hierarquia inferior a esta pasta.
\NoSelect	Indica que não é possível selecionar estas pasta.
\Marked	Indica que existem mensagens que foram adicionadas a esta caixa postal desde a última vez que a caixa postal havia sido selecionada.
\UnMarked	Indica que não foram adicionadas mensagens desde a última vez que esta caixa postal havia sido selecionada.

Exemplo:

```
C: A080 LIST "" ""
S: * LIST (\NoSelect) "/" ""
S: * LIST (\NoSelect) "/" principal/
S: * LIST (\NoInferiors) "/" principal/auxiliar
S: A080 OK LIST completed
```

3.8.2.3 RESPOSTA "STATUS"

A resposta STATUS é o resultado do comando STATUS. Nesta resposta o servidor envia ao cliente o nome da caixa postal e as informações da caixa postal.

Exemplo:

```
C: A083 STATUS principal (UIDNEXT MESSAGES RECENT)
S: * STATUS principal (RECENT 3 MESSAGES 23 UIDNEXT 8979)
S: A083 OK STATUS completed
```

3.8.2.4 RESPOSTA "SEARCH"

A resposta SEARCH é o resultado do comando "SEARCH" ou do comando "UID SEARCH". A resposta será composta com a palavra-chave "SEARCH" seguida com o UID ou os UIDs que satisfazem o critério de pesquisa.

Exemplo:

```
C: A193 SEARCH RECENT FROM “knorst” SUBJECT “Tcc”
```

```
S: SEARCH 2 35 64 125
S: A193 OK SEARCH completed
```

3.8.2.5 RESPOSTA "*FLAG*"

A resposta "*FLAG*" poderá ser o resultado do comando "SELECT" ou do comando "EXAMINE". O servidor irá retornar a resposta em uma lista entre parênteses contendo as *FLAG's* da caixa postal selecionada.

Exemplo:

```
C: A142 SELECT INBOX
S: ....
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: ....
S: A142 OK [READ-WRITE] SELECT completed
```

3.8.3 RESPOSTAS DOS SERVIDOR – TAMANHO DA CAIXA POSTAL

Estas respostas são sempre sem o rótulo. Estas informações são enviadas do servidor para o cliente. Após o caracter "*" se encontra um número que representa o número de mensagem contidas na caixa postal.

3.8.3.1 RESPOSTA "EXISTS"

A resposta "EXISTS" é o resultado dos comandos "EXAMINE" ou "SELECT". Esta resposta indica o número de mensagens armazenadas junto ao servidor.

Exemplo:

```
C: A142 SELECT INBOX
S: ....
S: * 17 EXISTS
S: ....
S: A142 OK [READ-WRITE] SELECT completed
```

3.8.3.2 RESPOSTA "RECENT"

Da mesma forma como a resposta "EXISTS" esta resposta é o resultado dos comandos "SELECT" ou "EXAMINE". Após o caracter "*" se encontra um número que representa o número de mensagem que estiverem com a *FLAG* \Recent.

Exemplo:

```
C: A142 SELECT INBOX
S: ....
S: * 7 RECENT
S: ....
S: A142 OK [READ-WRITE] SELECT completed
```

3.8.4 RESPOSTAS DOS SERVIDOR – ESTADO DAS MENSAGENS

Estas respostas são sempre sem rótulo. Após o caracter "*" aparece o número seqüencial da mensagem que está sendo alterada.

3.8.4.1 RESPOSTA "EXPUNGE"

A resposta "EXPUNGE" é o resultado do comando "EXPUNGE", que tem como função remover mensagens da caixa postal que estiverem com o *FLAG* \Deleted ativo. Para cada mensagem removida com sucesso o servidor irá enviar ao cliente uma linha que inicia com o caracter "*" e em seguida aparece o número seqüencial da mensagem que foi afetada.

Após remover uma mensagem da caixa postal, automaticamente o servidor irá decrementar em 1 todos os número seqüencial das mensagens subsequentes a mensagem removida.

Exemplo:

```
C: A095 EXPUNGE
S: * 3 EXPUNGE
S: * 7 EXPUNGE
S: * 7 EXPUNGE
S: A095 OK EXPUNGE completed
```

3.8.4.2 RESPOSTA "FETCH"

A respostas FETCH é o resultado dos comandos FETCH e STORE. Esta resposta retorna informações da mensagem que foi afetada por um destes dois comandos.

Após a execução de um dos comandos acima, o servidor irá enviar uma resposta para cada mensagem envolvida no comando. A resposta será no seguinte formado: toda resposta inicia com o caracter "*", em seguida vem o número seqüencial da mensagem que está envolvida no comando, como terceiro item da resposta está a palavra "FETCH", a partir do quarto item vem os dados, a disposição destes dados depende dos parâmetros que foram fornecidos ao comamdo.

Exemplo:

```
C: A200 FETCH 1 (FLAGS BODY[HEADER.FIELDS (FROM)])
S: * 1 FETCH (FLAGS (\Seen) BODY[HEADER.FIELDS ("FROM")] {42}
S: From: Stefan Haase <stefan@furb.rct-sc.br>
S: A200 OK FETCH completed
```

4 TECNOLOGIA UTILIZADAS PARA O DESENVOLVIMENTO

A seguir se encontram as tecnologias que influenciaram indiretamente neste trabalho, porém, o seu estudo foi essencial para a adequada implementação do protótipo descrito neste trabalho.

Estas tecnologias servem basicamente de suporte para a conexão entre o cliente e o servidor.

4.1 TCP/IP

De acordo com [MAR1994b], o protocolo de rede TCP/IP pode ser encontrado em grande parte dos sistemas operacionais, desde os servidores de grande porte até os computadores pessoais. Este protocolo é suportado pela maioria dos sistemas operacionais e é suportado por quase todas as variações do UNIX.

A rede TCP/IP é composta pela união dos principais protocolos: TCP e IP, formando assim o TCP/IP. O IP é um serviço de rede não orientado a conexão (datagrama não confiável) e o TCP é considerado como sendo um serviço de transporte orientado à conexão. O conjunto do TCP/IP desta forma oferece um serviço de alta confiabilidade.

Além destes dois protocolos, existe disponível o protocolo UDP (*User Datagram Protocol*), protocolo projetado e especificado para ser utilizado em rede de alta qualidade. Este protocolo pode ser considerado como sendo uma versão simplificada do TCP. Este protocolo não oferece nenhum tipo de controle em relação a ordem dos pacotes, como também não garante a entrega efetiva do dado.

4.1.1 ARQUITETURA TCP/IP

Conforme a RFC [IET1989], a arquitetura do protocolo TCP/IP está dividida em 4 (quatro) camadas distintas, onde cada uma executa um conjunto bem definido de funções de comunicação. O protocolo esta dividido conforme a figura 4.1.

FIGURA 4.1.1: Arquitetura TCP/IP

4.1.1.1 CAMADA "INTERFACE DE REDE"

É a camada mais inferior da arquitetura TCP/IP, sua função é fazer com que as informações sejam transmitidas de um computador para o outro em uma mesma mídia de acesso compartilhado (Ex.: Rede Local) ou uma ligação ponto-a-ponto (Ex.: modem). A preocupação destes protocolos é permitir o uso do meio físico que conecta os computadores na rede e fazer com que os bytes enviados por um computador cheguem ao outro computador.

A conexão entre os computadores pode ser feitas através dos mais diversos tipos de tecnologia disponíveis, sendo os mais tradicionais: *Ethernet*, PPP (acesso discado), FDDI, X.25, ATM, SLIP e muitos outros.

4.1.1.2 CAMADA "REDE"

De acordo com [MAR1994a], esta camada é responsável por transferir os dados do computador de origem até o computador destino. O computador de origem e o computador de destino podem estar na mesma rede física, ou em redes diferentes. Quando estiverem em redes diferentes é necessário a utilização de roteadores.

A função principal de um roteador é identificar qual o melhor caminho partindo do computador de origem ***A*** até o computador de destino ***B***. Se o endereço de destino não pertencer a mesma rede do endereço de origem, o roteador deve identificar quem será o

próximo *hops*. Para isso, utiliza uma tabela de roteamento, na qual estão armazenadas as informações de todas as redes que estão ligadas diretamente a ele, e uma entrada conhecida como *default* ou padrão que será utilizada no caso em que o endereço de destino não pertence a nenhuma das rede ligadas diretamente a ele. O endereço do novo *hops* de destino deve ser um computador ou roteador ligado fisicamente ao roteador de origem.

Cada computador ligado a Internet possui uma identificação única em toda a rede Internet, esta identificação é chamada de "endereço IP". O endereço IP é um numero de 32 bits, normalmente escrito como quatro octetos e está dividido em duas partes: a parte inicial identifica a rede e a segunda parte identifica o computador (host).

Conforme [MAR1994b], os primeiros bits de um endereço IP identificam a classe a qual este endereço pertence, atualmente existem quatro classes diferentes. As mais tradicionais são somente duas:

a) Classe B: utiliza 14 bits para identificar a rede e 16 bits para os *hosts*, com 16384 redes distintas e 64 mil *hosts*, disponibilizado somente para grandes corporações com um número grande de *hosts* ligados a Internet;

b) Classe C: utiliza 21 bits para identificar a rede e 8 bits para os *hosts*, formando assim mais de 2 milhões de redes distintas com 256 *hosts*, sendo a mais tradicional encontrada nas redes atuais;

Nesta camada não há nenhuma preocupação com relação à ordem em que os dados chegarão, eventuais erros e outros problemas, passando assim esta função às camadas superiores.

Nesta camada, o protocolo mais importante é o IP (*Internet Protocol*). Além do protocolo IP, estão presentes mais três protocolos de controle. O ICMP (*Internet Control Message Protocol*) utilizado para reportar situações de erro e teste. O ARP (*Address Resolution Protocol*) responsável pelo mapeamento da codificação que identifica um *host* nesta camada (endereço IP). O RARP (*Reverse Address Resolution Protocol*), ao contrário do anterior, permite que um *host* sabendo seu endereço de rede, pergunte a um outro qual é o seu endereço IP.

4.1.1.3 CAMADA "TRANSPORTE"

Esta camada permite que os *hosts* de origem e destino mantenham uma conversação (conexão). Esta camada possui dois protocolos o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*).

De acordo com [MAR1994a], o TCP é um protocolo orientado a conexão e confiável. Ele é orientado a conexão pois para que os dados sejam transmitidos é necessário que haja uma conexão prévia, seguida de uma transmissão de dados e depois uma liberação da conexão. É considerado confiável por garantir que os dados transmitidos sejam livres de erros de transmissão, inclusive garantindo a ordem de chegada dos pacotes. Este protocolo fragmenta o fluxo de dados recebidos e remonta no destino.

O UDP é um protocolo sem conexão e não confiável. É utilizado em sistemas de supervisão, onde a velocidade de entrega é mais importante do que sua precisão, por exemplo em transmissão de voz e vídeo. Uma aplicação que esteja utilizando este protocolo fica responsável por implementar as retransmissões, filtragem e outros controle inerentes a um serviço confiável. Ao trabalhar com este protocolo ganha-se velocidade e perde-se confiabilidade.

4.1.1.4 CAMADA "APLICAÇÃO"

Nesta camada está presente os protocolos de alto nível, ou seja aqueles que interagem diretamente com as aplicações dos usuários da rede TCP/IP. O protocolo TELNET permite que se abra um terminal virtual de um *host* para outro. O FTP e o TFTP são utilizados para a transferência de arquivos entre *hosts*, sendo que o primeiro utiliza o TCP e o segundo o UDP. O SMTP é utilizado para troca de mensagens (Correio Eletrônico) entre usuários de *hosts* distintos. O DNS é o serviço de nome que associa um nome a um endereço IP de um *host* para facilitar a sua memorização. O SNMP define um conjunto de regras para gerenciamento de *hosts* remotos. Da mesma forma, nesta camada também se encontram os protocolos IMAP e POP3, como sendo dois protocolos de correio eletrônico utilizados para ler as mensagens que estão armazenadas no servidor.

O protótipo desenvolvido pertence a esta camada da arquitetura TCP/IP, como também todo e qualquer software que fará conexão entre dois ou mais computadores utilizando TCP ou UDP.

4.2 SOCKET

De acordo com [DIG2000], *Socket* é um mecanismo de comunicação originariamente implementado na versão BSD do sistema operacional UNIX. *Sockets* são usados como terminadores para enviar e receber dados entre computadores.

A interface *socket* é uma API (*Application Program Interface*) que foi desenvolvida inicialmente para as variações do sistema operacional BSD. A Microsoft disponibilizou uma API para seus sistemas operacionais, conhecida como *WinSock*.

Esta API implementa as chamadas das funções que deverão ser utilizada para possibilitar o acesso a camada de transporte e promover a comunicação entre os computadores envolvidos. A tabela 4.2.1 apresenta a relação das mais tradicionais chamadas de funções encontradas na API.

TABELA 4.2.1: Funções da API *Socket*

Função	Descrição
Accept	Uma conexão nova foi recebida oriunda de um cliente e associada a um novo *socket* imediatamente criado.
Bind	Assinala o endereço IP local ao *socket*
Close	Finaliza o uso do *socket*
Connect	Assinala o endereço IP remoto ao *socket*
Gethostbyaddr	Pega dados de um host (nome, IP, tipo, ...) a partir de seu endereço IP
Gethostbyname	Pega dados de um host (nome, IP, tipo, ...) a partir de seu nome
Gethostid	Pega o endereço IP local
Gethostname	Pega o nome do computador local
Getpeername	Pega o endereço IP e a porta do processo remoto
Getsockname	Pega o nome local para o *socket* especificado
Getsockopt	Pega as opções associadas com o *socket* especificado.
Listen	Utilizado no servidor e sua função é aguardar por conexões
Read	Usado para ler os dados de um *socket*.
Recv	Similar ao "*read*", porém, com um argumento adicional, utilizado como *FLAG*.
Recvfrom	Pega os dados de um determinado host. Esta função só tem sentido no UDP, quando mensagens de diferentes hosts remotos chegaram no seu

	socket, mas só queremos as mensagens de um determinado host.
Send	Similar ao "*write*", porém, com um argumento adicional, utilizado como *FLAG*.
Sendto	Envia dados para um host específico. Só tem sentido no UDP, quando o server quer enviar mensagem para um determinado cliente
Setsockopt	Usado para alterar as opções associadas a um *socket* específico
Shutdown	Fecha a conexão local do *socket*. Para o *socket* estar totalmente fechado é necessário que o outro lado também execute um shutdown.
Socket	Cria uma estrutura para controlar uma comunicação, de acordo com o tipo solicitado – TCP ou UDP. Esta função retorna um valor inteiro como sendo um descritor. Este descritor será utilizado para referenciar este *socket* nas funções subquentes
Write	Usada para enviar dados sobre a conexão *socket*

Antes de iniciar o estabelecimento da conexão, são executadas as primitivas "*socket*", que cria um ponto terminal de comunicação de acordo com o protocolo de comunicação selecionado, podendo ser TCP ou UDP. O "*bind*" que registra o endereço da aplicação (número da porta). Para estabelecer a conexão (com o protocolo TCP), a aplicação servidora executa a primitiva "*listen*" enquanto que a cliente executa "*connect*". A aplicação servidora usa o "*accept*" para receber e estabelecer a conexão. Já o UDP, como não é orientado à conexão, logo após o "*socket*" e o "*bind*", utiliza as primitivas "*sendto*" e "*recvfrom*". As funções "*write*", "*send*", "*read*" e "*recv*", são usadas para enviar e receber dados pela conexão estabelecida. Por final é usado o comando "*close*" para fechar a conexão.

5 DESENVOLVIMENTO DO PROTÓTIPO

O protótipo foi desenvolvido para ser utilizado junto ao sistema operacional Windows 95/98. Optou-se por este sistema operacional para o desenvolvimento do protótipo, por ser o sistema operacional mais utilizado no mercado.

O protótipo desenvolvido somente irá verificar por novas mensagens e notificar o usuário ao recebimento de novas mensagens. O protótipo não possui nenhum mecanismo para tratar as mensagems, ou manipular as caixas postais remotas.

Este capítulo apresenta em detalhes todos os passos do desenvolvimento do protótipo e está divido em três partes:

a) Especificação do protótipo: apresenta toda a fluxogramação do processo de verificação por novas mensagens;

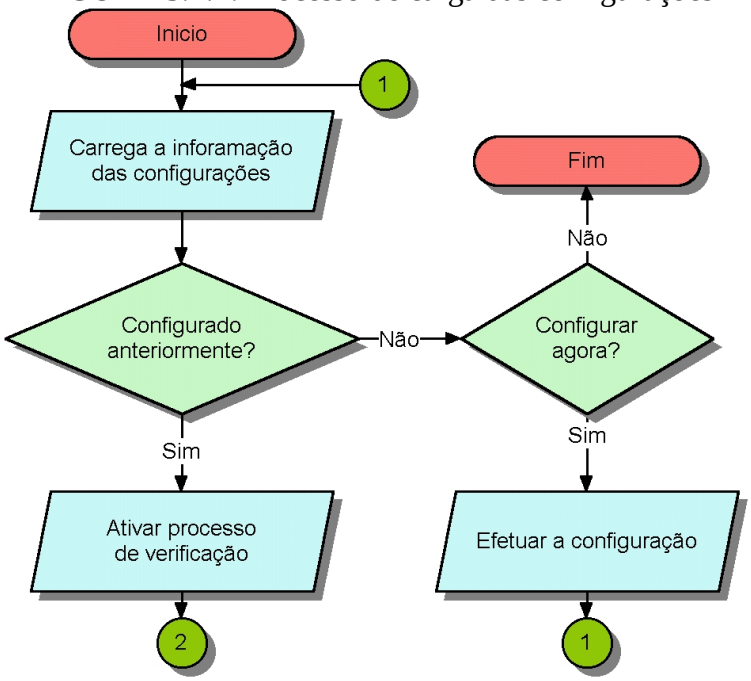
b) Implementação: nesta parte do capítulo podemos encontrar detalhes sobre a implementação do protótipo desenvolvido, como também parte do código fonte implementado;

c) Funcionamento: nesta parte do capítulo podemos encontrar instruções sobre o funcionamento do protótipo, como também detalhes de cada interface (janela) apresenta ao usuário.

5.1 ESPECIFICAÇÃO DO PROTÓTIPO

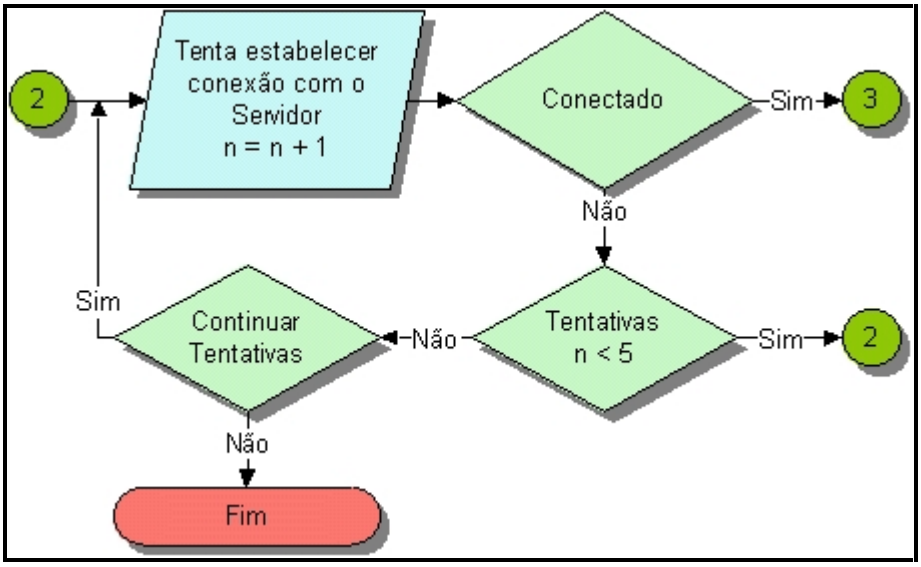
Para diagramar o protótipo foi utilizado a técnica denominada de fluxogramação, conforme figuras apresentadas a seguir.

FIGURA 5.1.1: Processo de carga das configurações



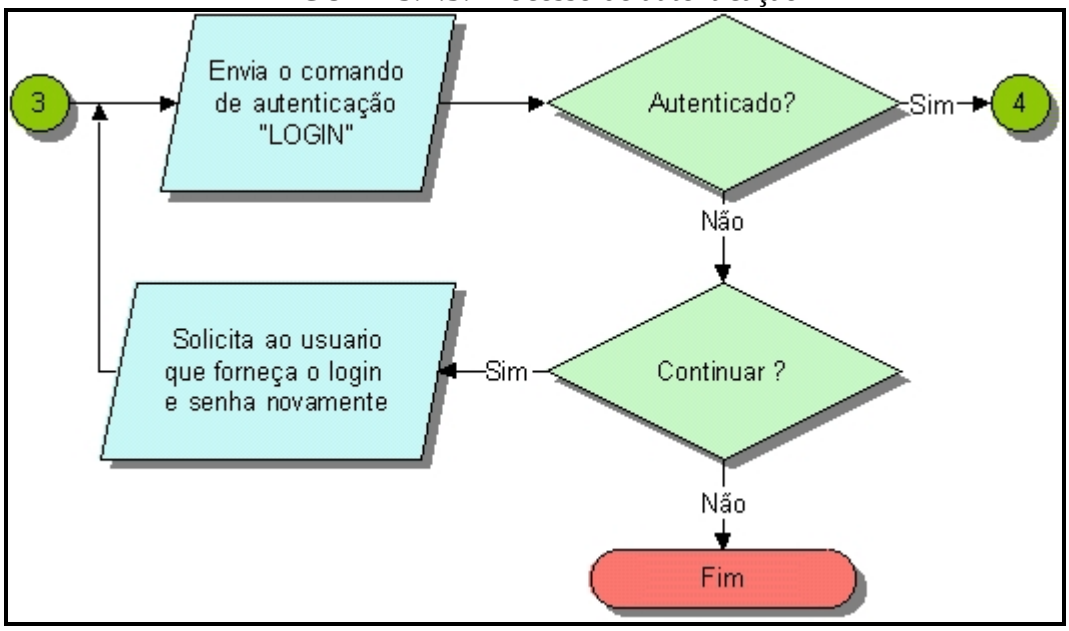
A figura 5.1.1, mostra o processo de inicialização das configurações do protótipo. Inicialmente, é realiazada uma verificação para identificar se o protótipo já havia sido configurado anteriormente ou não, caso negativo, o protótipo irá permitir ao usuário optar em efetuar a configuração ou sair do protótipo, porém, se o protótipo já havia sido configurado anteriormente, neste caso, será feita a inicialização das configurações e será ativado o processo de verificação.

FIGURA 5.1.2: Estabelecer a conexão com o servidor



A figura 5.1.2, mostra o processo da tentativa de conexão com o servidor. O protótipo tenta estabelecer a conexão com o servidor através de uma conexão TCP na porta 143. Caso esta conexão não for estabelecida com sucesso em 60 segundos o protótipo irá tentar estabelecer a conexão com o servidor novamente, porém, se o protótipo efetuar 5 (cinco) tentativas de conexão sem sucesso, questionar o usuário se ele deseja continuar as tentativas, ou simplesmente abortar o processo de verificação e sair do protótipo.

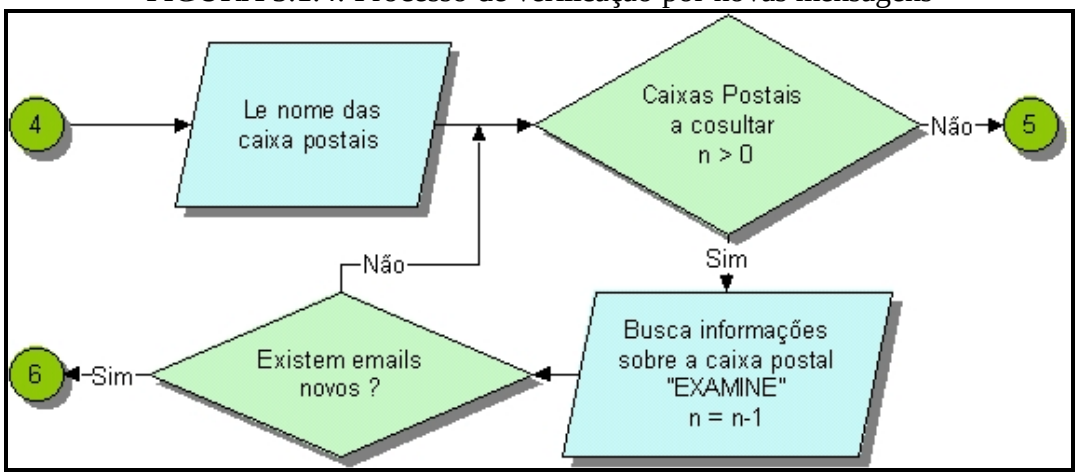
FIGURA 5.1.3: Processo de autenticação



A figura 5.1.3, mostra o processo de autenticação. Uma vez estabelecida a conexão com o servidor, o protótipo inicia o processo de autenticação (identificação) do usuário junto ao servidor, a autenticação se dará através do envio do *login* e da senha do usuário utilizando o comando "LOGIN <*login*> <senha>", se o processo de autenticação não obtiver sucesso e o servidor retornar erro, o protótipo irá solicitar ao usuário se ele deseja continuar o processo de verificação, caso ele optar em continuar o processo, neste caso, o protótipo irá solicitar que seja fornecido novamente o login e a senha, caso contrário, o processo de verificação será desativado.

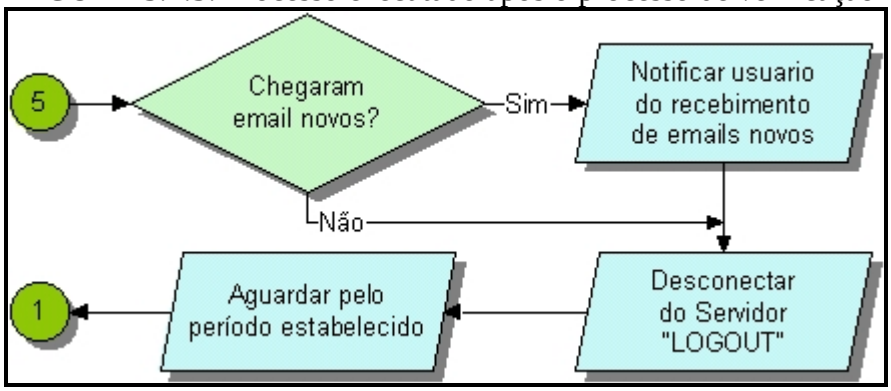
O protótipo desenvolvido, implementa somente a autenticação pelo comando "LOGIN", não utilizando nenhum tipo de criptografia.

FIGURA 5.1.4: Processo de verificação por novas mensagens



A figura 5.1.4, mostra o processo de verificação por novas mensagens junto ao servidor. Inicialmente, o protótipo irá identificar a relação de caixas postais a serem verificadas junto ao servidor, a informação de quais caixas postais devem ser verificadas, fica armazenada na configuração do protótipo. Para cada caixa postal listada o protótipo deverá efetuar a verificação por novas mensagens, primeiramente o protótipo envia o comando "EXAMINE", este comando seleciona a caixa postal em modo “READ-ONLY” e já retorna a relação de quantas mensagens existem na caixa postal selecionada, como também a quantidade de mensagens novas. Este processo será executado para todas as caixas postais listadas.

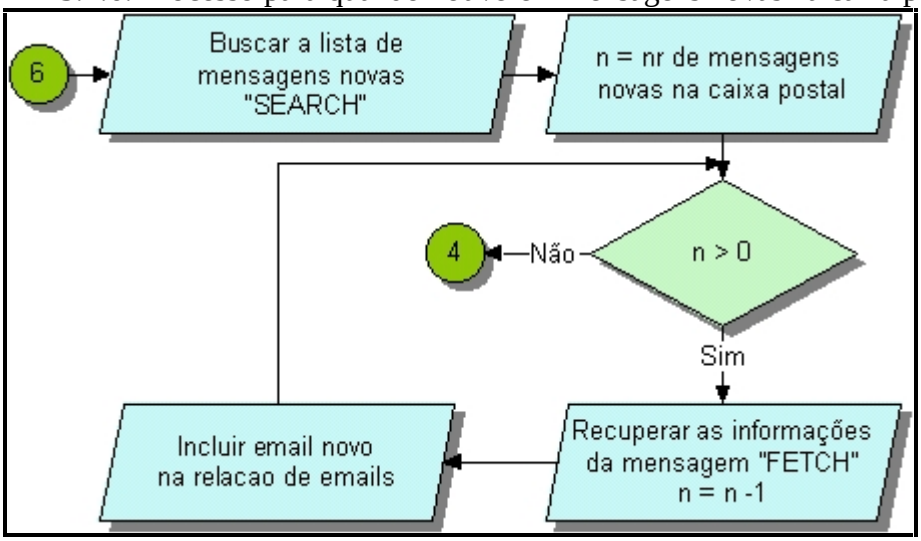
FIGURA 5.1.5: Processo executado após o processo de verificação



A figura 5.1.5, mostra o processo que será executado logo após a verificação por novas mensagens, neste processo o protótipo irá identificar se chegaram novas mensagens

para o usuário ou não, se houver chegado novas mensagens o protótipo irá notificar o usuário exibindo a relação dos *e-mail´s* novos contidos no servidor. Caso não houver mensagens novas o protótipo simplesmente irá fechar a conexão com o servidor através do comando "LOGOUT" e aguardar o período pré-estabelecido, transcorrido este período o protótipo irá executar o processo de verificação novamente iniciando no processo do ponto 1 apresentado na figura 4.1.

FIGURA 5.1.6: Processo para quando houverem mensagens novas na caixa postal.



A figura 5.1.6 mostra o processo que será executado quando o protótipo identificar que há novas mensagens na caixa postal.

Se houverem mensagens novas na caixa postal o protótipo enviará o comando "SEARCH" para o servidor com o argumento "RECENT", este comando irá retornar a relação das mensagens que estiverem com a *FLAG* \Recent e que representam as mensagens novas contidas no servidor. A partir da relação das mensagens novas o protótipo irá coletar as informações individualmente de cada mensagem contida na lista. Esta informação é obtida através do comando "FETCH" que será enviado ao servidor com os argumentos dos campos necessários para completar a lista de informações que será apresentada ao usuário.

Quando não houverem mais mensagens novas na caixa postal o protótipo irá para o processo 4 apresentado na figura 5.1.4, neste processo o protótipo irá verificar se há mais caixas postais a serem verificadas.

5.2 IMPLEMENTAÇÃO DO PROTÓTIPO

Para a implementação do protótipo utilizou-se do ambiente de desenvolvimento Delphi, pela sua facilidade na criação da interface para o usuário. Sua programação é baseada em Object Pascal, utilizando a programação orientada a objetos.

Na instalação básica deste ambiente de desenvolvimento, automaticamente são instalados os mais tradicionais VCL (*Visual Component Library*), tais como o componente TEdit, TStringGrid, TLabel e outros mais, além destes componentes, o Delphi ainda possui bibliotecas de funções, tais como a biblioteca SysUtils, Math, WinSock, e outras mais.

Para o desenvolvimento do protótipo utilizou-se da biblioteca WinSock nativa que acompanha a instalação do Delphi 4.0.

O processo 5.2.1 mostra o processo de estabelecimento da conexão TCP/IP com o servidor.

Neste processo também é feita a inicialização das variáveis, porém, caso o protótipo não houver sido configurado anteriormente, o processo irá questionar o usuário se deseja ou não fazer a configuração, caso opte em efetuar a configuração o protótipo irá abrir a janela de configuração do servidor, caso contrário irá abortar o processo de conexão.

PROCESSO 5.2.1: Estabelecimento da conexão

```
Function TFPrincipal.Conecta :Integer;
var
  Inicializado :Boolean;
  SockAddrIn   :TSockAddrIn;
begin
  result := 99999;

  // Veirifica se ja estah conectado ou nao ..
  if Conectado then
    begin
      MessageDlg('Atualmente já existe uma conexão aberta, fecha a conexão
primeiro', mtError, [mbOK], 0);
      exit;
    end;

  Repeat
    // Carrega as informacoes de inicializacao ..
    Inicializado := Iniciliza_Configuracao;
    if Not Inicializado then
      begin
        if   MessageDlg('A   configuração   do   servidor   ainda   não   foi
```

```
realizada.'+#10+#13+
                      'Desejas Efetuar a configuração agora ??',
                      mtWarning, [mbYes, mbNo], 0) = mrYes then
          begin
            FServidor.ShowModal;
          end
        else
          break;
      end;
  until Inicializado;

  if Inicializado then
    begin
      Result :=  0;

      FormDEBUG.DEBUG('C: <abrindo conexão>');

      //  Inicializa o WinSock
      if WSAStartup($0101, WSAData) <> 0 then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          Exit;
        end;

      // Cria o Socket
     FDSocket := socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
      if FDsocket = INVALID_SOCKET then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          Exit;
        end;

      // Seta o tamanho do BUFFER de Send e Recv
      SetSockOpt(FDSocket, SOL_SOCKET, SO_RCVBUF, PChar(IntToStr(MAXBUFF)),
Length(IntToStr(MAXBUFF)));
      SetSockOpt(FDSocket, SOL_SOCKET, SO_SNDBUF, PChar(IntToStr(MAXBUFF)),
Length(IntToStr(MAXBUFF)));

      // Faz a inicializacao do Socket
      if not InicializaSockAddr(Servidor, Porta, SockAddrIn) then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          FDSocket := INVALID_SOCKET;
          Exit;
        end;

      // Abre a conexao com o Servidor
      if Connect(FDSocket, SockAddrIn, Sizeof(SockAddrIn)) <> 0 then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          FDSocket := INVALID_SOCKET;
          Exit;
        end
      else
        RecebeText;
```

```
    end;
  if Result <> 0 then
    FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
end;
```

No processo 5.2.2, temos a função responsável pelo envio de qualquer comando para o servidor, no qual será necessário informar o comando e os parâmetros, o retorno da função será o número 0 (zero) quando o comando houver sido enviado com sucesso, ou senão, o número do respectivo erro.

PROCESSO 5.2.2: Envio de comandos ao servidor

```
Function         TFPrincipal.Envia_Comando(QualComando,         Parametros:
string):Integer;
Var
  Aux_comando  :String;
Begin
  Comando := QualComando;
  Aux_comando := NovoRotulo+' '+QualComando+' '+Parametros;
  FormDEBUG.DEBUG('C: '+Aux_comando);
  Aux_comando := Aux_comando+#10+#13;
  Send(FDSocket, Pchar(Aux_Comando)^, Length(Aux_comando), 0);
  Result := WSAGetLastError;
  If Result <> 0 then
    FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
end;
```

No processo 5.2.3, temos a função que é responsável para gerar um novo rótulo para cada comando a ser enviado ao servidor, esta função garante também que um mesmo rótulo não seja utilizado mais de uma vez.

PROCESSO 5.2.3: Gera novo rótulo

```
Function TFPrincipal.NovoRotulo: String;
var
  Aux :string;
begin
  Aux := IntToStr(NSeq);
  while Length(Aux) < 4 do Aux := '0'+Aux;
  Inc(NSeq);
  Rotulo := 'aek'+Aux;
  result := Rotulo;
end;
```

A função implementada pelo processo 5.2.4, trata da leitura dos dados enviados do servidor para o cliente. Esta função somente lê os dados não faz nenhum tipo de interpretação da informação.

PROCESSO 5.2.4: Leitura dos dados enviados pelo Servidor.

```
Function TFPrincipal.RecebeText :String;
Var
  Len    :Integer;
begin
  SetLength(Result, MAXBUFF);
  Len := Recv(FDSocket, Pointer(Result)^, MAXBUFF, 0);
  if Len <> SOCKET_ERROR then
    SetLength(Result, Len)
  else
    Result := ERRO_INTERNO;
  FormDEBUG.DEBUG('S: '+Result);
end;
```

A função do processo 5.2.5, identifica que tipo de retorno representa o conteúdo passado para a função. Da mesma forma como o processo 5.2.4, esta função não faz nenhum tipo de interpretação da informação, somente retorna o tipo de retorno, podendo ser "VAZIO", "SEM_ROTULO", "OK", "NO", "BAD" e "DESCONHECIDA".

PROCESSO 5.2.5: Tipo de resposta

```
Function TFPrincipal.TipodeResposta(Resposta :String): TResposta;
var
  Posicao  :Integer;
begin
  if Length(Resposta) <= 0 then
    begin
      Result := VAZIO;
      Exit;
    end;

  if Resposta[1]= '*' then
    Result := SEM_ROTULO
  else
    begin
      Posicao := Pos(' ', Resposta);
      Delete(Resposta, 1, Posicao);
      Posicao := Pos(' ', Resposta);
      Delete(Resposta, Posicao, Length(Resposta));
      if Uppercase(Resposta) = 'OK' then
        Result := OK
      else if Uppercase(Resposta) = 'NO' then
        Result := NO
      else if Uppercase(Resposta) = 'BAD' then
        Result := BAD
      else
        Result := DESCONHECIDA;
    end;
end;
```

O processo 5.2.6, mostra o processo de login junto ao servidor. Este processo consiste no envio do comando "LOGIN" e seus parâmetros ao servidor. O retorno será "TRUE" caso se o login e a senha conferirem, caso contrário, será solicitado ao usuário que seja fornecido o login e a senha novamente.

PROCESSO 5.2.6: Processo de LOGIN

```
Function TFPrincipal.Efetua_LOGIN :Boolean;
var
  Resultado   :String;
  Logado      :Boolean;
begin
  Logado := False;
  if Conectado then
    begin
      repeat
        // Tenta efetuar o LOGIN junto ao Servidor
        if Envia_Comando('LOGIN', Login+' '+Senha) = 0 then
          begin
            Resultado := RecebeText;
            // Verifica se o LOGIN
            if  TipodeResposta(Resultado) = OK then
              Logado := True
            else
              begin
                if   MessageDlg('Sua   senha   e   login   nao   conferem
!!'+#10#13+'Desejar informar eles agora ??',mtConfirmation, [mbYes, mbNO],
0) = mrYes then
                  begin
                    Login   :=   InputBox('Autenticação',   'Informe   o   seu
Login', '');
                    Senha   :=   InputBox('Autenticação',   'Informe   a   sua
Senha', '');
                  end
                else
                  break;
              end;
          end
      until Logado;
    end;
  Result := Logado;
end;
```

No processo 5.2.7, temos o processo utilizado para selecionar a caixa postal em qual deverá ser feita a verificação por novas mensagens, para isso é utilizado o comando "EXAMINE" que seleciona a caixa postal em modo "READ-ONLY", desta forma, não alterando as FLAGS. Além do mais, esta função retorna através das variáveis "Exists" e "Recent", o número de mensagens contidas na caixa posta e a quantidade de mensagens novas, respectivamente.

PROCESSO 5.2.7: Processo EXAMINE

```
Function TFPrincipal.Examine(CaixaPostal :String; var Exists, Recent
:Integer):Boolean;
var
  Aux,
  Str_INT,
  Linha      :string;
  PrimCRLF   :Integer;
begin
  Result := False;
  Exists := 0;
  Recent := 0;

  // Verifica se está conectado com o servidor
  if not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao EXAMINE não pode
ser executada.');
      Exit;
    end;

  if Length(CaixaPostal) = 0 then
    Exit;

  // Seleciona a caixa postal em modo READ-ONLY
  if Envia_Comando('EXAMINE', CaixaPostal) = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        While Length(Aux) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
              SEM_ROTULO : begin
                             Delete(Linha, 1, 2); // Tira o '* '
                             if Pos('EXIST', Uppercase(Linha)) <> 0 then
                               begin
                                 Str_INT := Copy(Linha, 1, Pos(' ', Linha)
-1);
                                 Exists := StrToInt(Str_INT);
                               end
                             else
                               if Pos('RECENT', Uppercase(Linha)) <> 0 then
                                 begin
                                   Str_INT  :=  Copy(Linha,  1,  Pos('  ',
Linha) -1);
                                   Recent := StrToInt(Str_INT);
                                 end;
                           end; // SEM_ROTULO
              OK : Result := True;
            end; // CASE
          end;
    end;
end;
```

No processo 5.2.8, temos o processo utilizado para identificar quais as mensagens que não foram lidas, para isso utiliza-se o comando "SEARCH" passando como parâmetro o argumento "RECENT". Este processo através da variável "Emails", retorna a relação de *email´s* que não foram lidos ainda.

PROCESSO 5.2.8: Processo SEARCH

```
Function TFPrincipal.Search(var Emails :String):Boolean;
Var
  Linha,
  Aux    :String;
  PrimCRLF :Integer;
begin

  Result := False;
  Emails := '';

  if Not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao SEARCH não pode
ser executada.');
      Exit;
    end;

  // Envia ao servidor o comando Search, que retorna o
  // numero de mensagens novas no Servidor
  if Envia_Comando('SEARCH', 'RECENT') = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        while Length(AUX) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
              SEM_ROTULO : begin
                             if Pos('SEARCH', Linha) <> 0 then
                               begin
                                 Delete(Linha, 1, 9);
                                 Emails := Linha;
                               end;
                           end;
              OK : Result := True;
            end;
          end;
    end;
end;
```

No processo 5.2.9, temos o comando FETCH, que é utilizado para coletar as informações sobre os *email's* novos armazenados no servidor. Este comando retorna os dados de "UID", "De", "Data", "Assunto" e "Tamanho" da mensagem.

PROCESSO 5.2.9: Processo FETCH

```
Function TFPrincipal.Fetch(Email :String; var UID, De, Data, Assunto,
Tamanho :String):Boolean;
Var
  Aux,
  Linha    :String;
  X,
  Posicao,
  PrimCRLF :Integer;
Begin
  // Inicializa as variaveis
  UID := '';
  De := '';
  Data := '';
  Result := False;
  Assunto := '';
  Tamanho := '';

  if Not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao FETCH não pode
ser executada.');
      Exit;
    end;

  // Envia ao servidor o comando que busca as informações sobre os emails
  if Envia_Comando('FETCH', Email+' (BODY[HEADER.FIELDS (FROM SUBJECT DATE
MESSAGE-ID)] RFC822.SIZE)') = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        while Length(AUX) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
              DESCONHECIDA : begin
                               // Busca a informação do UID
                               if Pos('MESSAGE-ID', UpperCase(linha)) <> 0
then
                                 begin
                                   Delete(Linha, 1, 12);
                                   Uid := Linha;
                                   Continue;
                                 end;

                               // Busca a data do email
                               if Pos('DATE:', UpperCase(Linha)) <> 0 then
                                 begin
                                   Delete(Linha, 1, 6);
                                   Data := Linha;
                                   Continue;
                                 end;

                               // Trata os dados do remetente
                               if Pos('FROM:', UpperCase(Linha)) <> 0 then
```

```
                                    begin
                                      Delete(Linha, 1, 6);
                                      De := Linha;
                                      Continue;
                                    end;

                                  // Pega o Assunto do email
                                  if  Pos('SUBJECT:',  UpperCase(Linha))  <>  0
then
                                    begin
                                      Delete(Linha, 1, 9);
                                      Assunto := Linha;
                                      Continue;
                                    end;

                                  // Trata a informação do Tamanho
                                  Posicao := Pos('SIZE', UpperCase(Linha));
                                  if Posicao <> 0 then
                                    begin
                                      Delete(Linha, 1, Posicao+4);
                                      for X := 1 to length(Linha) do
                                        begin
                                          if linha[x] in Numero then
                                            Tamanho := Tamanho+linha[x]
                                          else
                                            Break;
                                        end;
                                      Continue;
                                    end
                                end;
              OK : Result := True;
            end;
          end;
    end;
end;
```

No processo 5.2.10, temos o comando LOGOUT, que é utilizado para fechar a conexão com o servidor.

PROCESSO 5.2.10: Comando LOGOUT

```
Function TFPrincipal.Logout :Boolean;
var
  Aux,
  Linha    :String;
  PrimCRLF :Integer;
begin
  Result := False;

  if Not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao LOGOUT não pode
ser executada.');
```

```
      Exit;
    end;

  // Solicita a desconexão com o servidor
  if Envia_Comando('LOGOUT', '') = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        While Length(Aux) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
             OK : Begin
                    closesocket(FDSocket);
                    FDSocket := INVALID_SOCKET;
                    Result := True;
                  end;
          end;
        end;
    end;
end;
```

5.3 FUNCIONAMENTO DO PROTÓTIPO

A seguir será apresentado o funcionamento do protótipo desenvolvido. Será apresentada também cada interface (janela) e sua respectiva função e característica.

A figura 5.3.1, apresenta a tela principal do protótipo. Nesta janela se encontra o menu principal e logo abaixo desse, aparece a relação das mensagens novas contidas no servidor configurado.

Cada linha representa uma mensagem e cada mensagem possui as seguintes informações:

a) "UID", indica a identificação única desta mensagem junto ao servidor;

b) "De", informa o endereço do remetente da mensagem;

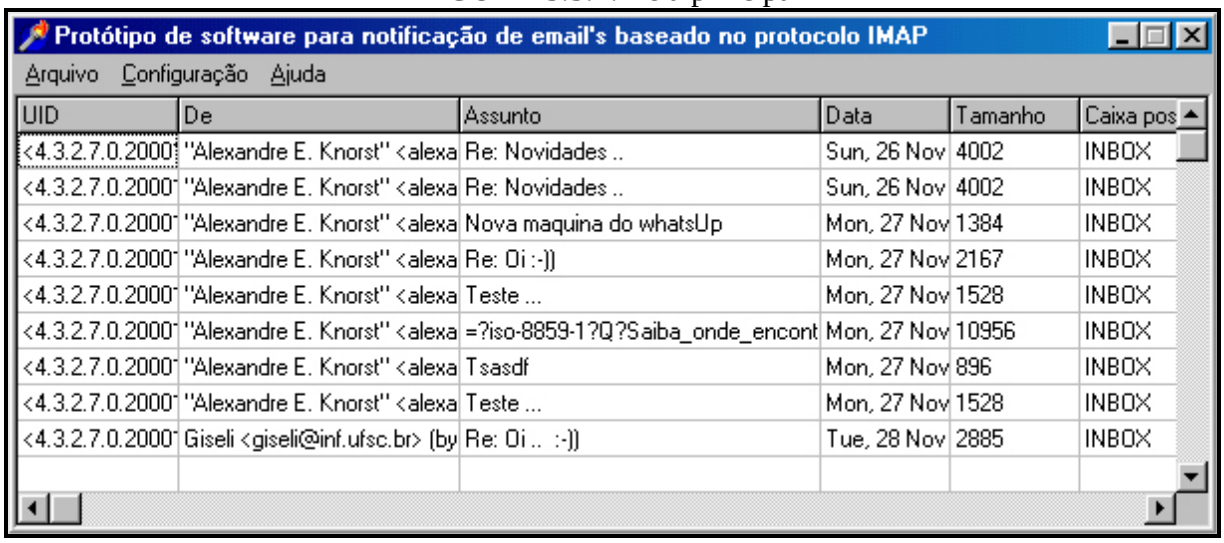
c) "Assunto", indica o assunto do que se trata o conteúdo da mensagem;

d) "Data", indica o horário em que esta mensagem foi recebida pelo servidor;

e) "Tamanho", representa a quantidades em bytes que compõem a mensagem;

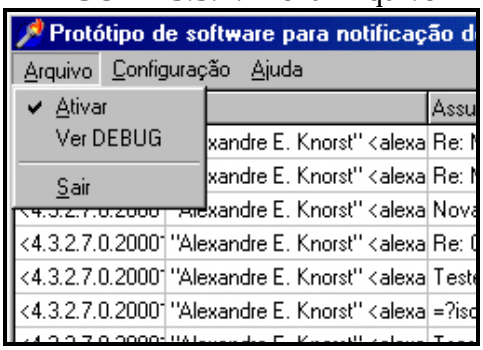
f) "Caixa Postal", indica em qual caixa postal se encontra esta mensagem

FIGURA 5.3.1: Tela principal



A figura 5.3.2, mostra as opções do menu "Arquivo", as opções encontradas neste menu são: "Ativar", quando esta opção do menu estiver assinalada no início da palavra, conforme mostra a figura 5.3.2, isso indica que o sistema de verificação está ativo. A opção "Ver DEBUG", abre a janela do DEBUG, nesta janela irão aparecer todas as transações realizadas entre o cliente e o servidor e a opção "Sair", finaliza o processo de verificação e sai do protótipo.

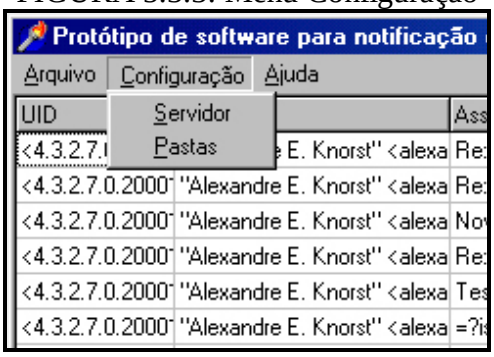
FIGURA 5.3.2: Menu Arquivo



A figura 5.3.3, mostra as opções do menu "Configuração". As opções deste menu são utilizadas para efetuar todas as configurações necessárias ao funcionamento do protótipo. A opção "Servidor", serve para especificar os dados necessários em relação ao servidor e a

opção "Pastas", serve para fazer a configuração de quais pastas deverão ser feitas as verificações por novas mensagens.

FIGURA 5.3.3: Menu Configuração



A figura 5.3.4, mostra a tela de configuração do Servidor, nesta tela o usuário deve especificar os dados relacionados ao servidor. Os campos que deverão ser especificados são:

a) "Servidor", neste campo deve ser informado o nome do servidor ou o endereço IP do servidor de correio IMAP;

b) "Porta", neste campo dever ser informado o número da porta utilizada pelo servidor para receber as conexões IMAP, e por padrão é a porta 143;

c) "Login", neste campo deve ser informado o nome de usuário cadastrado junto ao servidor;

d) "Senha", neste campo deve ser informado a senha do usuário especificado no campo "Login";

e) "Verificar por novas mensagens a cada", neste campo deverá ser especificado o intervalo de tempo entre cada verificação por novas mensagens junto ao servidor.

Para sair desta tela sem salvar as alterações, basta clicar no botão "Cancel", caso deseja que as alterações sejam salvas, clique sobre o botão "OK".

FIGURA 5.3.4: Tela de configuração do Servidor

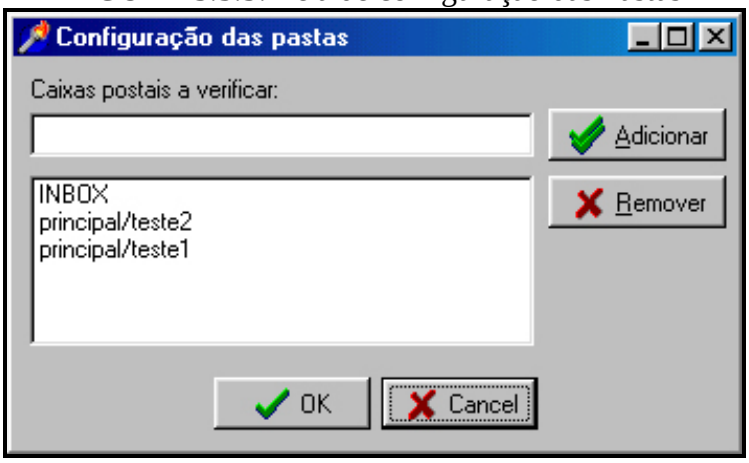
A figura 5.3.5, mostra a tela de configuração das pastas. Nesta tela o usuário deve especificar o nome das caixas postais em quais deverá ser feita a verificação por novas mensagens.

No campo "Caixas postais a verificar", o usuário deve digitar o nome da caixa postal a qual gostaria que fosse incluída na relação das caixas postais a serem verificadas, e em seguida clicar no botão "Adicionar". O nome da caixa postal informada pelo cliente deve já ter sido configurado anteriormente junto ao servidor, ou seja, deve já existir no servidor, pois, o protótipo não faz nenhum verificação da existência da mesma. Abaixo deste campo encontra-se a relação das caixas postais já configuradas previamente. A caixa postal principal recebe o nome de "INBOX".

Para remover um nome da relação das caixas postais, basta selecionar o nome desejado e em seguida clicar sobre o botão "Remover".

Para sair desta tela sem salvar as alterações, basta clicar no botão "Cancel", caso deseja que as alterações sejam salvas, clique sobre o botão "OK".

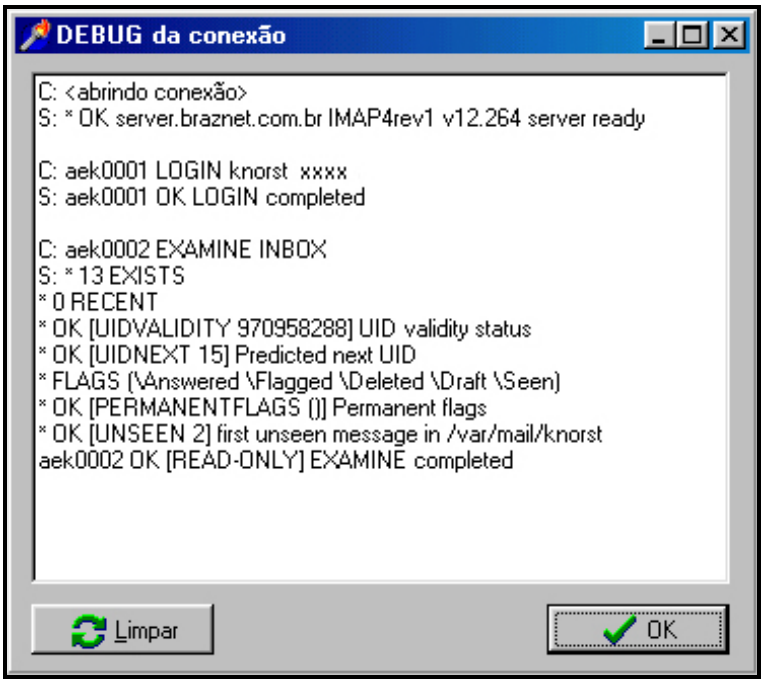
FIGURA 5.3.5: Tela de configuração das Pastas



Na figura 5.3.6, temos a tela de DEBUG, através da tela de DEBUG podemos visulizar todos os comandos que foram enviados do cliente para o servidor, que são todas as linha que iniciam com o caracter "C", como também todas as informações recebidas do servidor e que são as linhas que iniciam com o caracter "S".

Ao clicar sobre o botão "Limpar", o protótipo irá excluir todas as informações de DEBUG já armazenadas até aquele momento. O botão "OK" simplesmente fecha esta janela.

FIGURA 5.3.6: Tela de DEBUG



6 CONCLUSÃO

A elaboração deste trabalho consiste no cumprimento de cada umas das etapas previstas no cronograma da proposta apresentada e aprovada.

Apesar do protocolo ter sido especificado por volta do ano de 1986, encontrou-se grande dificuldade na obtenção de referências bibliográficas que abordassem sobre o assunto IMAP.

Para a conclusão deste trabalho foi necessário um estudo aprofundado de todo protocolo IMAP, envolvendo os comandos e as suas respectivas respostas. Este estudo foi feito baseado na RFC 2060 ([CRI1996]).

Além do mais foi necessário um estudo sobre o protocolo TCP/IP, que é utilizado como protocolo de rede, para a comunicação entre e cliente e o servidor.

Para a implementação do protótipo foi necessário o estudo das API do *socket*, que serve como interface entre a aplicação e a camada de transporte da arquitetura TCP/IP, como também o estudo de todas as suas funções em detalhes, as quais, neste trabalho sendo apresentados de forma bem discreta sem entrar em muitos detalhes de seu funcionamento.

O desenvolvimento foi feito utilizando o ambiente de desenvolvimento Delphi 4.0, pelo fato deste ambiente já possuir as bibliotecas de *socket* necessárias para o desenvolvimento do protótipo aqui proposto e pela facilidade na criação e implementação da interface com o usuáiro.

Apesar de todas as facilidades oferecidas pelo ambiente, o desenvolvimento do protótipo foi realizado utilizando a biblioteca nativa, que faz acesso direto ao WinSock do Windows, não se utilizando nenhum componente previamente desenvolvido.

6.1 EXTENSÕES

O protótipo atual não permite nenhuma manipulação com as mensagens e caixas postais armazenadas remotamente, como sugestão para futuras implementações podemos ter:

a) Permitir que se exclua mensagens do servidor;

b) Permitir alterar as FLAG da mensagem;

c) Criar e excluir caixas postais;

d) Mover mensagens de uma caixa postal para outra, utilizando o comando COPY;

Outra sugestão em relação ao protótipo, seria a implementação de uma opção para que o mesmo também funcione utilizando o protocolo POP3.

7 ANEXOS

Abaixo podemos encontrar o código fonte do protótipo desenvolvido. Porém, o código está distribuído em diversos arquivos.

a) Principal.pas: Como o próprio nome já diz, este arquivo é o módulo principal e implementa a interface principal do protótipo;

b) Udebug.pas: Responsável pela interface de visualização do DEBUG;

c) ConfPastas.pas: Responsável pela interface de configuração das pastas;

d) ConfServidor.pas: Responsável pela interface de configuração das informações sobre o servidor;

e) WinSock.pas: API do Socket. Parte integrante do ambiente de desenvolvimento Delphi 4.0.

7.1 PRINCIPAL.PAS

```
unit Principal;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, Grids, ExtCtrls, Winsock, IniFiles, StdCtrls;

type

  TResposta = (Sem_Rotulo, OK, NO, BAD, DESCONHECIDA, VAZIO);

  TFPrincipal = class(TForm)
    MainMenu1: TMainMenu;
    Arquivo1: TMenuItem;
    Configurao1: TMenuItem;
    Ajuda1: TMenuItem;
    Ativar1: TMenuItem;
    N1: TMenuItem;
    Sair1: TMenuItem;
    Servidor1: TMenuItem;
    Pastas1: TMenuItem;
    Sobre1: TMenuItem;
    StrGrid_MENSAGENS: TStringGrid;
    Timer: TTimer;
    VerDEBUG1: TMenuItem;
    procedure Sair1Click(Sender: TObject);
    procedure Ativar1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
```

```
    procedure Servidor1Click(Sender: TObject);
    procedure Pastas1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure VerDEBUG1Click(Sender: TObject);
    procedure TimerTimer(Sender: TObject);
  private
    { Private declarations }
    Login,
    Senha,
    Rotulo,
    Comando,
    Servidor  :string;
    Intervalo,
    Porta     :Integer;
    FDSocket  :TSocket;
    Function Conecta :Integer;
    Function RecebeText :String;
    Function NovoRotulo: String;
    Function Iniciliza_Configuracao :Boolean;
    Function LookupName(Nome :String; var SockAddr: TInAddr):Boolean;
    Function  InicializaSockAddr(Nome  :String;  Porta  :Integer;  var  Socket
:TSockAddrIn):Boolean;
    Function Envia_Comando(QualComando, Parametros: string):Integer;
    Function Efetua_LOGIN :Boolean;
    Function Conectado :Boolean;
    Function TipodeResposta(Resposta :String): TResposta;
    Function    Examine(CaixaPostal    :String;    var    Exists,    Recent
:Integer):Boolean;
    Function Search(var Emails :String):Boolean;
    Function  Fetch(Email  :String;  var  UID,  De,  Data,  Assunto,  Tamanho
:String):Boolean;
    Function Logout :Boolean;
    Procedure Processa;
  public
    { Public declarations }
  end;

var
  FPrincipal  :TFPrincipal;
  NSeq        :Integer = 1;
  WSAData     :TWSAData;
  Running     :Boolean;

Const
  MAXBUFF = 1024;
  ERRO_INTERNO = 'TEU_ERRO';

  Numero = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'];

implementation

uses ConfServidor, ConfPastas, UDebug;

{$R *.DFM}

Function  TFPrincipal.LookupName(Nome  :String;  var  SockAddr:  TInAddr)
:Boolean;
var
  Host    :PHostEnt;
```

```
begin
  Result := False;
  Host := gethostbyname(PChar(Nome));
  FillChar(Result, SizeOf(Result), 0);
  if Host <> nil then
    begin
      with SockAddr, Host^ do
        begin
          S_un_b.s_b1 := h_addr^[0];
          S_un_b.s_b2 := h_addr^[1];
          S_un_b.s_b3 := h_addr^[2];
          S_un_b.s_b4 := h_addr^[3];
        end;
      Result := True;
    end;
end;

Function TFPrincipal.InicializaSockAddr(Nome :String; Porta :Integer; var
Socket :TSockAddrIn):Boolean;
begin
  Result := False;
  if LookupName(Nome, Socket.sin_addr) then
    begin
      Socket.sin_port := htons(Porta);
      Socket.sin_family := PF_INET;
      Result := True;
    end;
end;

Function TFPrincipal.Conecta :Integer;
var
  Inicializado :Boolean;
  SockAddrIn   :TSockAddrIn;
begin
  result := 99999;

  // Veirifica se ja estah conectado ou nao ..
  if Conectado then
    begin
      MessageDlg('Atualmente já existe uma conexão aberta, fecha a conexão
primeiro', mtError, [mbOK], 0);
      exit;
    end;

  Repeat
    // Carrega as informacoes de inicializacao ..
    Inicializado := Iniciliza_Configuracao;
    if Not Inicializado then
      begin
        if   MessageDlg('A   configuração   do   servidor   ainda   não   foi
realizada.'+#10+#13+
                        'Desejas Efetuar a configuração agora ??',
                        mtWarning, [mbYes, mbNo], 0) = mrYes then
          begin
            FServidor.ShowModal;
          end
        else
          break;
      end;
```

```
  until Inicializado;

  if Inicializado then
    begin
      Result :=  0;

      FormDEBUG.DEBUG('C: <abrindo conexão>');

      //  Inicializa o WinSock
      if WSAStartup($0101, WSAData) <> 0 then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          Exit;
        end;

      // Cria o Socket
     FDSocket := socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
      if FDsocket = INVALID_SOCKET then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          Exit;
        end;

      // Seta o tamanho do BUFFER de Send e Recv
      SetSockOpt(FDSocket, SOL_SOCKET, SO_RCVBUF, PChar(IntToStr(MAXBUFF)),
Length(IntToStr(MAXBUFF)));
      SetSockOpt(FDSocket, SOL_SOCKET, SO_SNDBUF, PChar(IntToStr(MAXBUFF)),
Length(IntToStr(MAXBUFF)));

      // Faz a inicializacao do Socket
      if not InicializaSockAddr(Servidor, Porta, SockAddrIn) then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          FDSocket := INVALID_SOCKET;
          Exit;
        end;

      // Abre a conexao com o Servidor
      if Connect(FDSocket, SockAddrIn, Sizeof(SockAddrIn)) <> 0 then
        begin
          Result := WSAGetLastError;
          FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
          FDSocket := INVALID_SOCKET;
          Exit;
        end
      else
        RecebeText;
    end;
  if Result <> 0 then
    FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
end;

// Lê os dados do Buffer enviado pelo servidor
Function TFPrincipal.RecebeText :String;
Var
  Len    :Integer;
```

```
begin
  SetLength(Result, MAXBUFF);
  Len := Recv(FDSocket, Pointer(Result)^, MAXBUFF, 0);
  if Len <> SOCKET_ERROR then
    SetLength(Result, Len)
  else
    Result := ERRO_INTERNO;
  FormDEBUG.DEBUG(’S: ’+Result);
end;

// Inicializa as variáveis do protótipo
Function TFPrincipal.Iniciliza_Configuracao :Boolean;
var
  InitFile :TIniFile;
  Flag_aux :Integer;
begin
  InitFile := TIniFile.Create('EmailNot.ini');
  Flag_aux := InitFile.ReadInteger('Conexão', 'Configurado', 0);
  if Flag_aux = 150778 then
    begin
      Servidor := InitFile.ReadString('Conexão', 'Servidor', '');
      Porta := InitFile.ReadInteger('Conexão', 'Porta', 143);
      Login := InitFile.ReadString('Conexão', 'Login', '');
      Senha := InitFile.ReadString('Conexão', 'Senha', '');
      Intervalo := InitFile.ReadInteger('Conexão', 'Intervalo', 1);
      Timer.Interval := Intervalo * 60000;
    end;
  InitFile.Free;
  Result := Flag_aux = 150778;
end;

// Funcao utilizada para enviar dados e comandos ao servidor
Function         TFPrincipal.Envia_Comando(QualComando,         Parametros:
string):Integer;
var
  Aux_comando  :String;
begin
  Comando := QualComando;
  Aux_comando := NovoRotulo+' '+QualComando+' '+Parametros;
  FormDEBUG.DEBUG('C: '+Aux_comando);
  Aux_comando := Aux_comando+#10+#13;
  Send(FDSocket, Pchar(Aux_Comando)^, Length(Aux_comando), 0);
  Result := WSAGetLastError;
  if Result <> 0 then
    FormDEBUG.DEBUG('S: Erro -> '+IntToStr(Result));
end;

// Gera um novo rotulo
Function TFPrincipal.NovoRotulo: String;
var
  Aux :string;
begin
  Aux := IntToStr(NSeq);
  while Length(Aux) < 4 do Aux := '0'+Aux;
  Inc(NSeq);
  Rotulo := 'aek'+Aux;
  result := Rotulo;
end;
```

```
// Verificacao se a conexao estah ativa
Function TFPrincipal.Conectado :Boolean;
begin
  Result := FDSocket <> INVALID_SOCKET;
end;

// Efetua o login junto ao servidor
Function TFPrincipal.Efetua_LOGIN :Boolean;
var
  Resultado   :String;
  Logado      :Boolean;
begin
  Logado := False;
  if Conectado then
    begin
      repeat
        // Tenta efetuar o LOGIN junto ao Servidor
        if Envia_Comando('LOGIN', Login+' '+Senha) = 0 then
          begin
            Resultado := RecebeText;
            // Verifica se o LOGIN
            if  TipodeResposta(Resultado) = OK then
              Logado := True
            else
              begin
                if   MessageDlg('Sua   senha   e   login   nao   conferem
!!'+#10#13+'Desejar informar eles agora ??',mtConfirmation, [mbYes, mbNO],
0) = mrYes then
                  begin
                    Login   :=   InputBox('Autenticação',   'Informe   o   seu
Login', '');
                    Senha   :=   InputBox('Autenticação',   'Informe   a   sua
Senha', '');
                  end
                else
                  break;
              end;
          end
      until Logado;
    end;
  Result := Logado;
end;

// Identifica o tipo de resposta recebida pelo Servidor
Function TFPrincipal.TipodeResposta(Resposta :String): TResposta;
var
  Posicao  :Integer;
begin
  if Length(Resposta) <= 0 then
    begin
      Result := VAZIO;
      Exit;
    end;

  if Resposta[1]= '*' then
    Result := SEM_ROTULO
  else
    begin
      Posicao := Pos(' ', Resposta);
```

```
      Delete(Resposta, 1, Posicao);
      Posicao := Pos(’ ’, Resposta);
      Delete(Resposta, Posicao, Length(Resposta));
      if Uppercase(Resposta) = ’OK’ then
        Result := OK
      else if Uppercase(Resposta) = ’NO’ then
        Result := NO
      else if Uppercase(Resposta) = ’BAD’ then
        Result := BAD
      else
        Result := DESCONHECIDA;
    end;
end;

// Seleciona a caixa postal em modo READ-ONLY
Function  TFPrincipal.Examine(CaixaPostal  :String;  var  Exists,  Recent
:Integer):Boolean;
var
  Aux,
  Str_INT,
  Linha      :string;
  PrimCRLF   :Integer;
begin
  Result := False;
  Exists := 0;
  Recent := 0;

  // Verifica se está conectado com o servidor
  if not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao EXAMINE não pode
ser executada.');
      Exit;
    end;

  if Length(CaixaPostal) = 0 then
    Exit;

  // Seleciona a caixa postal em modo READ-ONLY
  if Envia_Comando('EXAMINE', CaixaPostal) = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        While Length(Aux) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
              SEM_ROTULO : begin
                             Delete(Linha, 1, 2); // Tira o '* '
                             if Pos('EXIST', Uppercase(Linha)) <> 0 then
                               begin
                                 Str_INT := Copy(Linha, 1, Pos(' ', Linha)
-1);
                                 Exists := StrToInt(Str_INT);
                               end
                             else
                               if Pos('RECENT', Uppercase(Linha)) <> 0 then
```

```
                                    begin
                                      Str_INT  :=  Copy(Linha,  1,  Pos('  ',
Linha) -1);
                                      Recent := StrToInt(Str_INT);
                                    end;
                               end; // SEM_ROTULO
              OK : Result := True;
            end; // CASE
          end;
    end;
end;

// Busca a relacao de emails novos, junto ao servidor na caixa postal
selecionada.
Function TFPrincipal.Search(var Emails :String):Boolean;
Var
  Linha,
  Aux    :String;
  PrimCRLF :Integer;
begin

  Result := False;
  Emails := '';

  if Not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao SEARCH não pode
ser executada.');
      Exit;
    end;

  // Envia ao servidor o comando Search, que retorna o
  // numero de mensagens novas no Servidor
  if Envia_Comando('SEARCH', 'RECENT') = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        while Length(AUX) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
              SEM_ROTULO : begin
                             if Pos('SEARCH', Linha) <> 0 then
                               begin
                                 Delete(Linha, 1, 9);
                                 Emails := Linha;
                               end;
                           end;
              OK : Result := True;
            end;
          end;
    end;
end;

// Buscas as informacoes sobre o email, as quais serao apresentadas ao
usuário
Function  TFPrincipal.Fetch(Email  :String;  var  UID,  De,  Data,  Assunto,
```

```
Tamanho :String):Boolean;
Var
  Aux,
  Linha    :String;
  X,
  Posicao,
  PrimCRLF :Integer;
Begin
  // Inicializa as variaveis
  UID := '';
  De := '';
  Data := '';
  Result := False;
  Assunto := '';
  Tamanho := '';

  if Not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao FETCH não pode
ser executada.');
      Exit;
    end;

  // Envia ao servidor o comando que busca as informações sobre os emails
  if Envia_Comando('FETCH', Email+' (BODY[HEADER.FIELDS (FROM SUBJECT DATE
MESSAGE-ID)] RFC822.SIZE)') = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        while Length(AUX) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
              DESCONHECIDA : begin
                               // Busca a informação do UID
                               if Pos('MESSAGE-ID', UpperCase(linha)) <> 0
then
                                 begin
                                   Delete(Linha, 1, 12);
                                   Uid := Linha;
                                   Continue;
                                 end;

                               // Busca a data do email
                               if Pos('DATE:', UpperCase(Linha)) <> 0 then
                                 begin
                                   Delete(Linha, 1, 6);
                                   Data := Linha;
                                   Continue;
                                 end;

                               // Trata os dados do remetente
                               if Pos('FROM:', UpperCase(Linha)) <> 0 then
                                 begin
                                   Delete(Linha, 1, 6);
                                   De := Linha;
                                   Continue;
```

```
                                  end;

                                // Pega o Assunto do email
                                if  Pos('SUBJECT:',  UpperCase(Linha))  <>  0
then
                                  begin
                                    Delete(Linha, 1, 9);
                                    Assunto := Linha;
                                    Continue;
                                  end;

                                // Trata a informação do Tamanho
                                Posicao := Pos('SIZE', UpperCase(Linha));
                                if Posicao <> 0 then
                                  begin
                                    Delete(Linha, 1, Posicao+4);
                                    for X := 1 to length(Linha) do
                                      begin
                                        if linha[x] in Numero then
                                          Tamanho := Tamanho+linha[x]
                                        else
                                          Break;
                                      end;
                                    Continue;
                                  end
                              end;
              OK : Result := True;
            end;
          end;
    end;
end;

// Fecha a conexao com o servidor
Function TFPrincipal.Logout :Boolean;
var
  Aux,
  Linha    :String;
  PrimCRLF :Integer;
begin
  Result := False;

  if Not Conectado then
    begin
      FormDEBUG.DEBUG('Não conectado no momento - Operacao LOGOUT não pode
ser executada.');
      Exit;
    end;

  // Solicita a desconexão com o servidor
  if Envia_Comando('LOGOUT', '') = 0 then
    begin
      Aux := RecebeText;
      if Aux <> ERRO_INTERNO then
        While Length(Aux) > 0 do
          begin
            PrimCRLF := Pos(#13#10, AUX);
            Linha := Copy(Aux, 1, PrimCRLF -1);
            Delete(Aux, 1, PrimCRLF+1);
            case TipodeResposta(Linha) of
```

```
             OK : Begin
                    closesocket(FDSocket);
                    FDSocket := INVALID_SOCKET;
                    Result := True;
                  end;
            end;
          end;
    end;
end;

procedure TFPrincipal.Sair1Click(Sender: TObject);
begin
  Application.Terminate;
end;

procedure TFPrincipal.Ativar1Click(Sender: TObject);
begin
  Ativar1.Checked := not Ativar1.Checked;
  Timer.Enabled := Ativar1.Checked;
end;


procedure TFPrincipal.FormActivate(Sender: TObject);
begin
  StrGrid_MENSAGENS.Cells[0, 0] := 'UID';
  StrGrid_MENSAGENS.Cells[1, 0] := 'De';
  StrGrid_MENSAGENS.Cells[2, 0] := 'Assunto';
  StrGrid_MENSAGENS.Cells[3, 0] := 'Data';
  StrGrid_MENSAGENS.Cells[4, 0] := 'Tamanho';
  StrGrid_MENSAGENS.Cells[5, 0] := 'Caixa postal';
end;

procedure TFPrincipal.Servidor1Click(Sender: TObject);
begin
  FServidor.Showmodal;
end;

procedure TFPrincipal.Pastas1Click(Sender: TObject);
begin
  FPastas.Showmodal;
end;

procedure TFPrincipal.FormCreate(Sender: TObject);
begin
  Rotulo := '';
  FDSocket := INVALID_SOCKET
end;

procedure TFPrincipal.VerDEBUG1Click(Sender: TObject);
begin
  FormDEBUG.ShowModal;
end;

procedure TFPrincipal.Processa;
var
  Aux1,
  Aux2,
  Linha,
  Recent,
```

```
  Exists,
  Posicao,
  Tentativa     :Integer;

  De,
  UID,
  Aux,
  Data,
  Caixas,
  Tamanho,
  Assunto,
  CaixaPostal    :String;

  InitFile       :TIniFile;
begin
  // Verifica se o processo está sendo executado ou não
  if Running then
    Exit;

  Running := True;
  Linha := 1;
  Tentativa := 1;

  // Limpa a relacao dos emails
  for Aux1 :=1 to StrGrid_MENSAGENS.RowCount do
    for Aux2 := 0 to StrGrid_MENSAGENS.ColCount do
      StrGrid_MENSAGENS.Cells[Aux2, Aux1] := '';

  // Tenta estabelecer conexao
  while not conectado do
    begin
      if Conecta = 0 then
        break
      else
        begin
          if tentativa = 5 then
            begin
              if MessageDlg('Foram feitas cinco tentativas de conexão ao
servidor sem sucesso'+#10#13+
                            'Continuar tentando conectar ??', mtWarning,
[mbYes, mbNo], 0) = mrYes then
                Tentativa := 1
              else
                begin
                  Timer.Enabled := False;
                  Running := False;
                  MessageDlg('Processo    de    conexão    abortado    !!',
mtInformation, [mbOk], 0);
                  exit;
                end;
            end;
          inc(Tentativa);
        end;
    end;

  if not Efetua_LOGIN then
    begin
      Timer.Enabled := False;
      Running := False;
```

```
      MessageDlg('Processo de verificação de LOGIN abortado !!',
mtInformation, [mbOk], 0);
      Exit;
    end;

  InitFile := TInifile.Create('EmailNot.ini');
  Aux1 := InitFile.ReadInteger('Pastas', 'Número de pastas', -1);
  for Aux2 := 0 to Aux1 -1 do
    begin
      CaixaPostal := InitFile.ReadString('Pastas', 'Pasta'+inttostr(Aux2),
'');

      // Verifica se existem novas mensagens na CaixaPostal
      if Examine(CaixaPostal, Exists, Recent) then
        begin
          if Recent > 0 then
            begin

              // Busca a relacao de novas mensagens
              if Search(Caixas) Then
                begin
                  while length(Caixas) > 0 do
                    begin
                      posicao := Pos(' ', Caixas);
                      if ((posicao = 0) and (length(caixas) > 0)) then
                        begin
                          aux := Caixas;
                          Caixas := '';
                        end else begin
                          aux := copy(caixas, 1, posicao -1);
                          delete(caixas, 1, posicao);
                        end;

                      // Busca os dados da novas mensagens
                      if Fetch(aux, UID, De, Data, Assunto, Tamanho) then
                        begin
                          StrGrid_MENSAGENS.Cells[0, Linha] := UID;
                          StrGrid_MENSAGENS.Cells[1, Linha] := De;
                          StrGrid_MENSAGENS.Cells[2, Linha] := Assunto;
                          StrGrid_MENSAGENS.Cells[3, Linha] := Data;
                          StrGrid_MENSAGENS.Cells[4, Linha] := Tamanho;
                          StrGrid_MENSAGENS.Cells[5, Linha] := CaixaPostal;
                          Inc(Linha);
                        end; // Fetch
                    end; // While
                end; // Search
            end; // Recent
        end; // Examine
    end; // For
  InitFile.Free;
  Logout;

  if StrGrid_MENSAGENS.Cells[0, 1] <> '' then
     begin
       beep;
       MessageDlg('Existem novas mensagens no Servidor !!!', mtInformation,
[mbOk], 0);
     end;
```

```
  Running := False;
end;

procedure TFPrincipal.TimerTimer(Sender: TObject);
begin
  if not Running then
    processa;
end;

end.
```

7.2 UDEBUG.PAS

```
Unit UDebug;

Interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons;

type
  TFormDEBUG = class(TForm)
    MemoDEBUG: TMemo;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    Procedure DEBUG(Texto :String);
  end;

var
  FormDEBUG: TFormDEBUG;

implementation

{$R *.DFM}

Procedure TFormDEBUG.DEBUG(Texto :String);
begin
  MemoDEBUG.Lines.Add(Texto);
end;

procedure TFormDEBUG.BitBtn2Click(Sender: TObject);
begin
  MemoDEBUG.Lines.Clear;
end;

procedure TFormDEBUG.BitBtn1Click(Sender: TObject);
begin
  Close;
end;
```

```
end.
```

7.3 CONFPASTAS.PAS

```
unit ConfPastas;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, IniFiles, Buttons;

type
  TFPastas = class(TForm)
    ListBox_PASTAS: TListBox;
    Label1: TLabel;
    Edit_PASTA: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure ListBox_PASTASDblClick(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FPastas: TFPastas;

implementation

{$R *.DFM}

procedure TFPastas.BitBtn1Click(Sender: TObject);
begin
  if Length(Edit_PASTA.Text) > 0 then
    ListBox_PASTAS.Items.Add(Edit_PASTA.Text);
  Edit_PASTA.Clear;
end;

procedure TFPastas.BitBtn2Click(Sender: TObject);
begin
  if ListBox_PASTAS.Selected[ListBox_PASTAS.ItemIndex] then
    ListBox_PASTAS.Items.Delete(ListBox_PASTAS.ItemIndex);
end;

procedure TFPastas.BitBtn3Click(Sender: TObject);
var
```

```
  InitFile :TInifile;
  NItems,
  Aux      :Integer;
begin
  InitFile := TInifile.Create('EmailNot.ini');
  InitFile.EraseSection('Pastas');
  NItems := ListBox_PASTAS.Items.Count;
  InitFile.WriteInteger('Pastas', 'Número de pastas', NItems);
  For Aux := 0 to NItems -1  do
    begin
      InitFile.WriteString('Pastas',                  'Pasta'+inttostr(Aux),
ListBox_PASTAS.Items.Strings[Aux]);
    end;
  InitFile.Free;
  close;
end;

procedure TFPastas.FormActivate(Sender: TObject);
var
  InitFile :TInifile;
  NItems,
  Aux      :Integer;
begin
  InitFile := TInifile.Create('EmailNot.ini');
  ListBox_PASTAS.Items.Clear;
  NItems := InitFile.ReadInteger('Pastas', 'Número de pastas', -1);
  for Aux := 0 to NItems -1 do
    begin
      ListBox_PASTAS.Items.Add(InitFile.ReadString('Pastas',
'Pasta'+inttostr(Aux), ''));
    end;
  InitFile.Free;
end;

procedure TFPastas.ListBox_PASTASDblClick(Sender: TObject);
begin
  Edit_PASTA.Text                                                    :=
ListBox_PASTAS.Items.Strings[ListBox_PASTAS.ItemIndex];
end;

procedure TFPastas.BitBtn4Click(Sender: TObject);
begin
  Close;
end;

end.
```

7.4 CONFSERVIDOR.PAS

```
Unit ConfServidor;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, IniFiles, Spin;

Type
  TFServidor = class(TForm)
    Edit_SERVIDOR: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    SpinEdit_PORTA: TSpinEdit;
    Edit_LOGIN: TEdit;
    Login: TLabel;
    Edit_SENHA: TEdit;
    Label3: TLabel;
    Label4: TLabel;
    SpinEdit_INTERVALO: TSpinEdit;
    Label5: TLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FServidor: TFServidor;

implementation

{$R *.DFM}

procedure TFServidor.BitBtn2Click(Sender: TObject);
begin
  Close;
end;

procedure TFServidor.BitBtn1Click(Sender: TObject);
var
  InitFile :TIniFile;
begin
  InitFile := TIniFile.Create('EmailNot.ini');
  InitFile.WriteInteger('Conexão', 'Configurado', 150778);
  InitFile.WriteString('Conexão', 'Servidor', edit_SERVIDOR.Text);
  InitFile.WriteInteger('Conexão', 'Porta', SpinEdit_PORTA.Value);
  InitFile.WriteString('Conexão', 'Login', edit_LOGIN.Text);
  InitFile.WriteString('Conexão', 'Senha', edit_SENHA.Text);
  InitFile.WriteInteger('Conexão', 'Intervalo', SpinEdit_INTERVALO.Value);
  InitFile.Free;
  close;
end;

procedure TFServidor.FormActivate(Sender: TObject);
var
  InitFile :TIniFile;
begin
```

```
  InitFile := TIniFile.Create(’EmailNot.ini’);
  edit_SERVIDOR.Text := InitFile.ReadString('Conexão', 'Servidor', '');
  SpinEdit_PORTA.Value := InitFile.ReadInteger('Conexão', 'Porta', 143);
  edit_LOGIN.Text := InitFile.ReadString('Conexão', 'Login', '');
  edit_SENHA.Text := InitFile.ReadString('Conexão', 'Senha', '');
  SpinEdit_INTERVALO.Value  :=  InitFile.ReadInteger('Conexão',  'Intervalo',
1);
  InitFile.Free;
end;

end.
```

8 REFERÊNCIA BIBLIOGRÁFICA

[CRI1996] CRISPIN, M; Washington. **RFC-2060 - *IMAP: internet message access protocol*** 1996. Endereço Eletrônico: http://www.isi.edu/in-notes/rfc2060.txt. Data da consulta: 26/06/2000.

[CRO1982] CROCKER, David H.. **RFC 822 – Standard for the format of ARPA Internet text messages**. 1982. Endereco Eletrônico: http://www.landfield.com/rfcs/rfc822.html. Data da consulta: 15/09/2000.

[CYC1997] CYCLADES Brasil. **Guia Internet de conectividade.** São Paulo, 1997.

[DIG2000] DIGIWEB. **Digiweb Brasil – Suporte** 2000. Endereço Eletrônico: http://www.digiweb.psi.br/support/glossary/. Data da consulta: 20/10/2000.

[IES2000] IESG. ***IESG – Internet Engineering Steering Group*** 2000. Endereço eletrônico: http://www.ietf.org/iesg.html. Data da consulta: 20/09/2000.

[IET1989] IETF. **RFC-1122 – *Requirements for Internet Hosts – Communication Layers*** 1989. Endereço Eletrônico: http://www.landfield.com/rfcs/rfc1122.html. Data da consulta: 26/08/2000.

[IET2000] IETF. ***IETF – Internet Engineering Task Force*** 2000. Endereço Eletrônico: http://www.ietf.org/. Data da consulta: 14/10/2000.

Eletrônico: http://www.ietf.org/. Data da consulta: 14/10/2000.

[MAR1994a] MARTIN, James. LEBEN, Joe. ***TCP/IP networking: architecture, administration and programming.*** New Jersey : Prentice Hall, 1994.

[MAR1994b] MARTIN, James. CHAPMAN, Kaethleen Kavanagh. LEBEN, Joe. ***Local area networks: architecture and implementations.*** New Jersey : Prentice Hall, 1994.

[MYR1996] MYRES, J.; Mellon, Carnegie; Rose, M.; Dover Beach Consulting, Inc. **RFC-1939 – POP – post office protocol – version 3** 1996. Endereço Eletrônico: http://www.isi.edu/in-notes/rfc1939.txt. Data da consulta: 26/06/2000.

[POS1982] POSTEL, Jonathan B. **RFC821 – SMTP – simple mail transfer protocol** 1996. Endereço Eletrônico: http://www.isi.edu/in-notes/rfc821.txt. Data da consulta: 26/06/2000.

[MYE1982] MYERS, J.; Mellon, Carnegie. **RFC1731 - IMAP4 authentication mechanisms** 1982. Endereço Eletrônico: http://www.landfield.com/rfcs/rfc1731.html. Data da consulta: 10/09/2000.