Capítulo IV

A Arquitetura de Implementação GTA/MM

IV.1 Introdução

Este capítulo apresenta um ambiente de desenvolvimento que possui uma arquitetura de implementação que privilegia o alto desempenho em sistemas de comunicação. A Arquitetura GTA/MM proposta neste trabalho objetiva servir como base para o desenvolvimento de sistemas de comunicação multimídia, em tempo real e com garantia de qualidade de serviço. Para tanto, a arquitetura visa fornecer suporte para a implementação de:

- protocolos de comunicação de alta velocidade de forma eficiente;
- mecanismos de controle de tráfego e policiamento responsáveis pela garantia de qualidade de serviço.

A arquitetura busca alcançar quatro características fundamentais: eficiência, flexibilidade, modularidade e portabilidade. Por eficiência entende-se em evitar o desperdício de recursos tais como memória, banda passante, etc., e no processamento otimizado da informação. A flexibilidade consiste em fornecer módulos de suporte com uma relativa diversidade de funções de forma a melhor atender às características do sistema de comunicação em desenvolvimento. A modularidade desejada consiste em permitir a utilização de diversos tipos de protocolos de comunicação, de mecanismos de controle de tráfego e de meios de comunicação, com os módulos básicos da arquitetura. O quesito de portabilidade é importante para possibilitar que o sistema de comunicação desenvolvido não se restrinja a um único ambiente computacional.

A arquitetura de implementação proposta (Figura IV.1) consiste de três módulos básicos utilizados no desenvolvimento de todo sistema de comunicação e de alguns módulos específicos, opcionais, que definem as características do sistema de comunicação a ser desenvolvido. Os módulos básicos são o Módulo de Gerenciamento de Memória (GM), o Módulo de Escalonamento de Tarefas (ET) e o Módulo de Gerenciamento de Temporizações (GT). Os módulos específicos são o Módulo de Gerenciamento do Sistema, os Protocolos de Comunicação, o Módulo de Sincronização e o Módulo de Garantia de Qualidade de Serviço que engloba o Módulo de Controle de Tráfego e Policiamento.

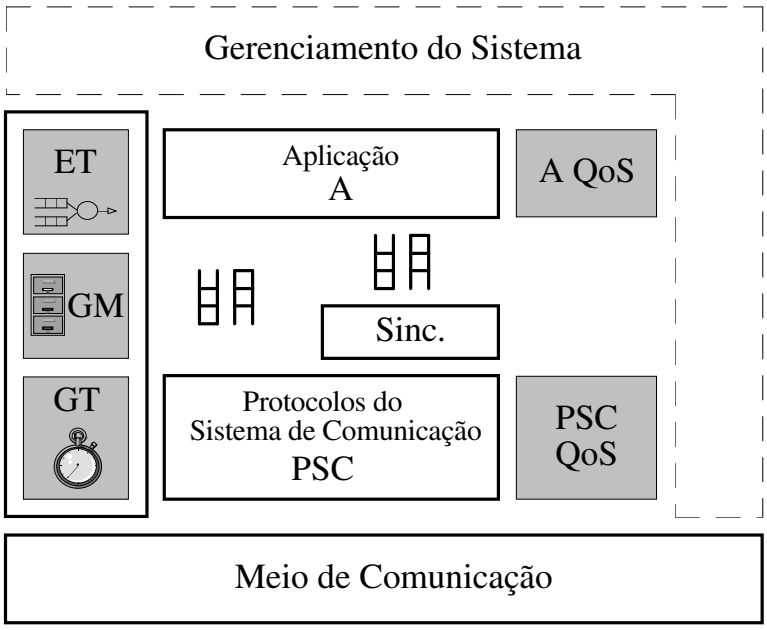


Figura IV.1: A arquitetura de implementação GTA/MM.

Os módulos específicos visam oferecer um conjunto de facilidades para a implementação do sistema. O Módulo de Gerenciamento do Sistema configura os demais módulos de acordo com as características do sistema desejado. Os Protocolos de Comunicação visam oferecer o conjunto de funcionalidades existentes nos principais protocolos de alta velocidade [NDu94, Nun95]. As funcionalidades são configuráveis ou por primitivas de serviços ou pelo módulo de gerenciamento do sistema. O Módulo de

Sincronização visa coordenar a ordenação temporal de informações multimídias (voz, vídeo, gráficos, dados, etc.). Através de Redes de Petri temporizadas [WQF94], este módulo implementa os tipos de sincronização inter e intramídia. O Módulo de Garantia de Qualidade de Serviço possui implementados os mecanismos *Virtual Clock* [Zha91], *Stop-and-Go* [Gol90], *Jitter-Earliest-Due-Date* [VZF91] e *Hierarchical Round Robin* [ZKe91]. Este módulo engloba ainda as implementações dos seguintes mecanismos de policiamento de tráfego: balde furado, balde furado com suavização do tráfego de saída, janela saltitante, janela saltitante sincronizada, janela deslizante contínua e janela móvel particionada.

A utilização dos recursos de forma eficiente é uma característica fortemente acoplada ao protocolo de comunicação utilizado, ao mecanismo de controle de tráfego, ao mecanismo de controle de admissão de conexão, aos requisitos de qualidade de serviço da aplicação, etc. No entanto, parte dos recursos do sistema de comunicação são utilizados no controle e gerenciamento dos módulos de suporte da arquitetura. Portanto, estes recursos também devem ser utilizados de forma eficiente, evitando-se o desperdício. Quanto ao processamento otimizado da informação, utilizou-se como base para o desenvolvimento desta arquitetura, uma outra arquitetura de implementação de sistemas de comunicação (Arquitetura de Implementação GTA/OSI), proposta pelo Grupo de Teleinformática e Automação (GTA) da UFRJ, a qual mostrou ser bastante eficiente no desenvolvimento de protocolos OSI [AND94, ALD93]. Baseado nesta arquitetura, foram desenvolvidos novos módulos apropriados para a implementação de mecanismos para garantia de QOS. Os novos módulos foram implementados de forma a minimizar o tempo de processamento e a oferecer o suporte necessário para a implementação de sistemas de comunicação em tempo real.

Uma grande diversidade de funções está presente em todos os módulos, tornando o sistema bastante flexível. No módulo de Gerenciamento de Memória, são oferecidas funções de alocação e liberação de: estruturas de dados inerentes à arquitetura, como estruturas de primitivas e de parâmetros; blocos de dados de tamanho fixo e dados de tamanho variável. No módulo de Escalonamento de Tarefas são oferecidas funções de inserção, execução e retirada

de tarefas de baixa, média e altíssima prioridade. O módulo de Gerenciamento de Temporizações, um dos mais requisitados em sistemas de comunicação em tempo real, oferece funções de iniciação, criação, interrupção, reiniciação e finalização de temporizadores simples ou auto-disparáveis, únicos ou múltiplos.

Uma grande modularidade do sistema foi obtida através da definição de interfaces padrões, bastante flexíveis, possibilitando a reutilização dos módulos básicos em várias configurações de sistemas de comunicação em tempo real.

A portabilidade do sistema foi plenamente alcançada em pelo menos dois ambientes de implementação distintos, nos quais foram desenvolvidas várias configurações de sistemas de comunicação em tempo real. O primeiro sistema é monoprocessado, monotarefa, IBM-PC compatível, com sistema operacional DOS. Em particular foram utilizadas máquinas IBM-PC-486 DX 50 MHz. O segundo sistema é monoprocessado, multitarefa, com sistema operacional UNIX/SunOS. Neste ambiente foram utilizadas Estações de Trabalho Sun SPARC 2. Ambos os ambientes possuem suas estações interconectadas por rede local tipo ETHERNET. Toda a arquitetura foi desenvolvida na linguagem de programação C, devido principalmente a sua portabilidade e eficiência.

As seções subsequentes descrevem a evolução desta arquitetura e detalham os seus módulos básicos. A seção IV.2 descreve a arquitetura original, implementada pelo GTA, sua finalidade e seus módulos de suporte. A seção IV.3 descreve a arquitetura de implementação proposta, enfocando a sua evolução com relação a arquitetura original. A seção IV.4 descreve a flexibilidade oferecida pelo Módulo de Gerenciamento de Memória na manipulação de estruturas de dados de tamanho fixo ou variável. A seção IV.5 descreve a eficiência do Módulo de Escalonamento de Tarefas na execução das diversas funções do sistema de comunicação. A seção IV.6 descreve o Módulo de Gerenciamento de Temporizações, ilustrando a grande necessidade do mesmo em sistemas de comunicação em tempo real.

Finalmente, a seção IV.7 apresenta tabelas que ilustram o tempo de processamento e a vazão máxima atingida pelos sistemas de comunicação em diversas configurações e ambientes.

IV.2 A Arquitetura de Implementação GTA/OSI

Transferência de dados confiável em ambientes industriais, de satélites ou em redes e implementação de alto desempenho de protocolos OSI foram os principais objetivos do GTA. Isto motivou o desenvolvimento de uma arquitetura de implementação, denominada Arquitetura GTA/OSI. Desde o primeiro trabalho de implementação de protocolos do sistema [BBS91], ficou claro que uma implementação cuidadosa era necessária e que a cópia de memória deveria ser evitada. Portanto, a arquitetura de implementação GTA/OSI foi definida [BDu91] de forma a se obter alto desempenho.

A Arquitetura GTA/OSI (Figura IV.2) se baseia em três módulos específicos (gerenciamento de memória, escalonamento de tarefas e gerenciamento de temporizações) e em interfaces padronizadas que serão descritas nas seções que seguem.

IV.2.1 Estruturas de Dados das Interfaces

O modelo de implementação da Arquitetura GTA é ilustrado na Figura IV.2. Esta arquitetura define uma interface padrão (Figura IV.3) entre camadas visando normalizar o acesso aos serviços oferecidos pelas camadas adjacentes, permitindo flexibilidade, modularidade e acoplamento fraco entre camadas. Esta independência simplificou a divisão das tarefas de implementação, manutenção e atualização dos protocolos de cada camada e levou a um melhor gerenciamento do sistema.

Há essencialmente duas maneiras de se implementar a comunicação entre camadas: chamada de procedimentos e trocas de mensagens síncronas/assíncronas. Chamada de procedimentos é certamente o mecanismo mais eficiente, mas implica em um sistema altamente

acoplado. Comunicação síncrona entre camadas não é adequada para ambientes monotarefa. Optou-se por uma comunicação assíncrona que se serve de um par de filas FIFO de mensagens (entrada/saída).

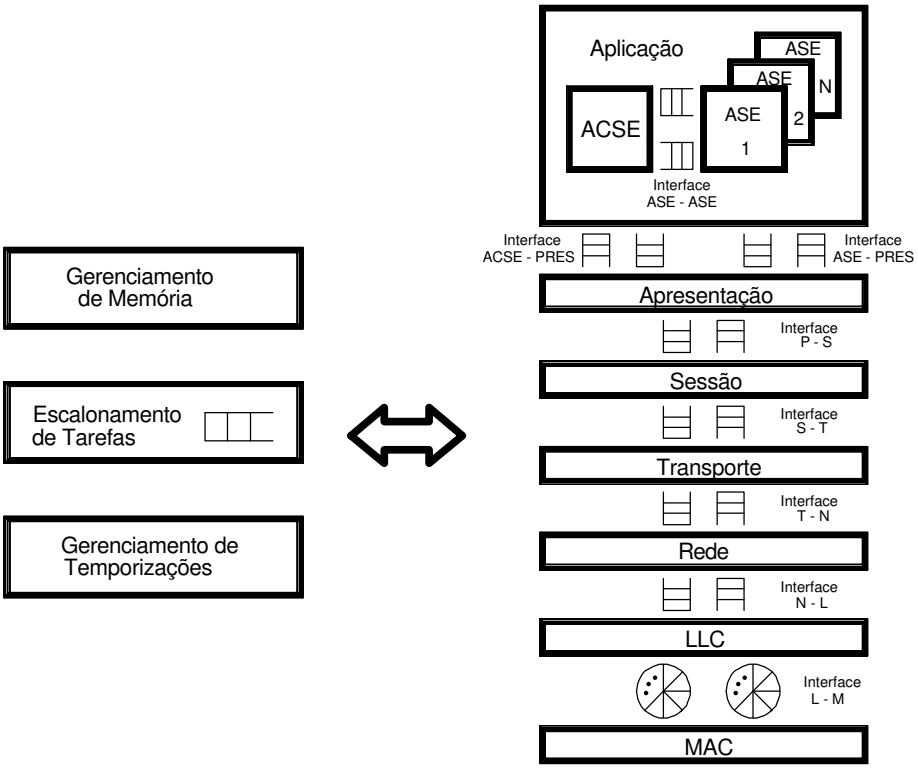


Figura IV.2: A Arquitetura de Implementação Original.

A interface entre camadas adotada consiste de um par filas FIFO por conexão e de um conjunto de rotinas de acesso às estruturas de dados. Cada fila é formada por um elemento básico composto de estruturas de dados encadeadas, definidas de forma a se obter um desempenho otimizado evitando a cópia de dados em um mesmo sistema. A única exceção é a interface entre as subcamadas de Enlace MAC e LLC, que inclui também um par de *buffers* circulares necessários para a transferência de dados através do meio físico, onde os dados a serem enviados à placa controladora são copiados para uma forma contígua. A placa controladora de comunicação é responsável pela transmissão no meio físico. De forma a permitir uma implementação eficiente desta interface, foi implementado um módulo de gerenciamento de memória específico para a arquitetura GTA.

Cada elemento da FIFO (Figura IV.3) é representado por uma estrutura de tamanho fixo chamada *Estrutura Primitiva* contendo os três campos seguintes:

- Ponteiro_Estrutura_Primitiva — usado para ligação com o próximo elemento da FIFO;
- Nome_Primitiva — nome da primitiva de serviço de entrada/saída;
- Ponteiro_Parâmetros_Primitiva — ponteiro para uma estrutura que contém os parâmetros atuais da primitiva que é chamada *Estrutura Parâmetros.*

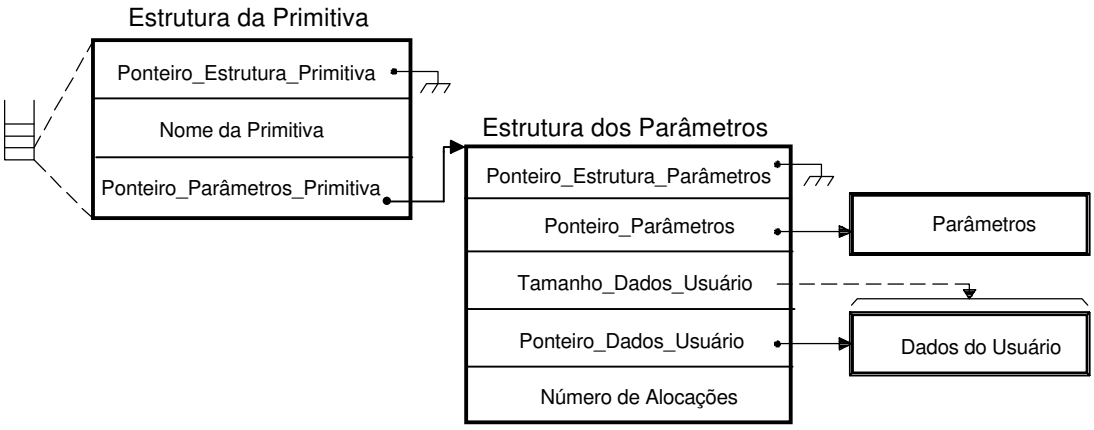


Figura IV.3: Estrutura de Dados das Interfaces.

O estabelecimento de uma conexão corresponde à instanciação de um par de filas na interface, por onde passarão todas as primitivas de serviço associadas àquela conexão.

Todas as Interfaces entre camadas seguem este modelo a exceção da interface Aplicação/Apresentação. Esta excepcionalidade se deve a característica particular da Camada Aplicação que é composta por uma variedade de Elementos de Serviço de Aplicação (ASE) Comuns (ACSE, ROSE, RTSE) e Específicos (MHS, FTAM, ...), que podem ser associados de diversas maneiras entre si e com a Camada Apresentação (Figura IV.2). A comunicação da Camada Aplicação com a Camada Apresentação é feita pelo ACSE e outro elemento de serviço. Logo, algumas filas específicas foram criadas: as filas ACSE-PRES e ASE(*n*)-PRES na interface Aplicação-Apresentação e as filas ASE(*n*)-ACSE na interface entre o Elemento de

Serviço de Controle de Associação e os demais e Elementos de Serviço, onde *n* identifica o elemento de serviço.

IV.2.2 Informações de Controle do Protocolo

Um processamento comum a todas as camadas, relacionado à comunicação entre entidades pares, é a inserção/remoção de informações de controle de protocolo (PCI — *Protocol Control Information*) em unidades de dados de serviço (SDUs — *Service Data Units*). Há vários métodos de inserção de PCI em SDUs [Svo89a, BDu91], por exemplo, cópia, alocação máxima e encadeamento.

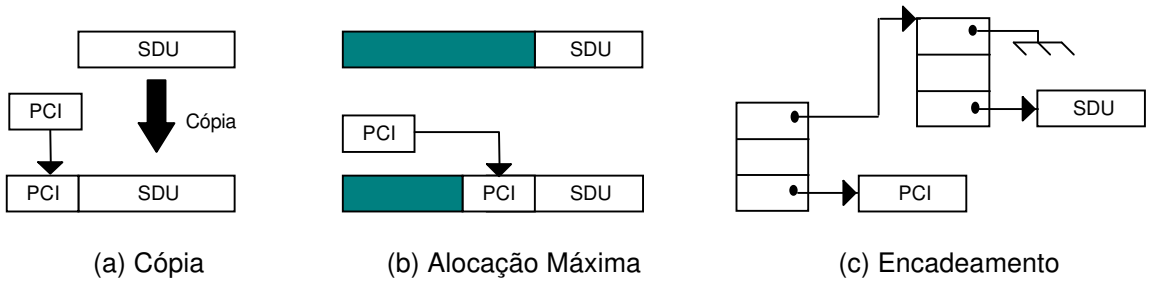


Figura IV.4: Técnicas de Inserção de PCI.

A inserção de PCI por cópia (Figura IV.4a) consiste em:

1. alocar uma região de memória com um comprimento correspondente aos tamanhos do PCI mais SDU;
2. preencher o PCI e copiar a SDU;
3. liberar a SDU anterior.

A cópia de dados de *buffers* entre camadas representa em geral um tempo de processamento significativo [Svo89a, BBS91, WMo89]. Além do mais, o tempo de cópia é fortemente dependente do tamanho da SDU. Tudo isso tem um impacto crítico no desempenho.

A técnica de inserção de PCI por alocação máxima (Figura IV.4b) consiste em se alocar uma região de memória suficiente para armazenar a SDU da camada mais alta e os cabeçalhos de todas as camadas inferiores. Entretanto, este procedimento pode levar a um grande desperdício de memória, pois o tamanho máximo do quadro deve necessariamente ser reservado. Este esquema também introduz a necessidade de cópias para desempenhar algumas funções básicas, como segmentação e concatenação. Outro argumento contra o uso deste procedimento é a necessidade da camada superior conhecer o tamanho máximo do quadro que pode ser enviado. Este fato vai de encontro ao princípio de independência entre as camadas, suportado pelo Modelo OSI [DZi83, ISO85].

O método de inserção de PCI por encadeamento (Figura IV.4c) evita a cópia de dados e o desperdício de memória, consistindo em:

1. alocar e preencher uma região de memória correspondente a um PCI;
2. associar a essa região uma estrutura de encadeamento;
3. encadear esta estrutura à estrutura da SDU recebida.

Esta abordagem pareceu ser a melhor maneira de inserir PCI. Cada fila é formada por um elemento básico composto de estruturas de dados encadeadas, definidas de maneira a evitar cópia de dados, um fator de degradação relevante no desempenho de sistemas de comunicação. A Figura IV.3 mostra as estruturas de dados implementadas que se baseiam em ponteiros e separam estruturas de tamanho fixo das estruturas de tamanho variável para possibilitar um gerenciamento de memória mais eficiente. Para implementar esta característica, a *Estrutura Parâmetros* foi definida [BDu91, Bag92] contendo os cinco campos seguintes:

- Ponteiro_Estrutura_Parâmetros — usado para encadeamento com outra estrutura de parâmetros;
- Ponteiro_Dados_Usuário — ponteiro para (parte do) parâmetro de dados do usuário da primitiva;

- Tamanho_Dados_Usuário — comprimento em octetos do respectivo parâmetro de dados do usuário;
- Ponteiro_Parâmetros — ponteiro para os demais parâmetros da primitiva;
- Número_de_Alocações.

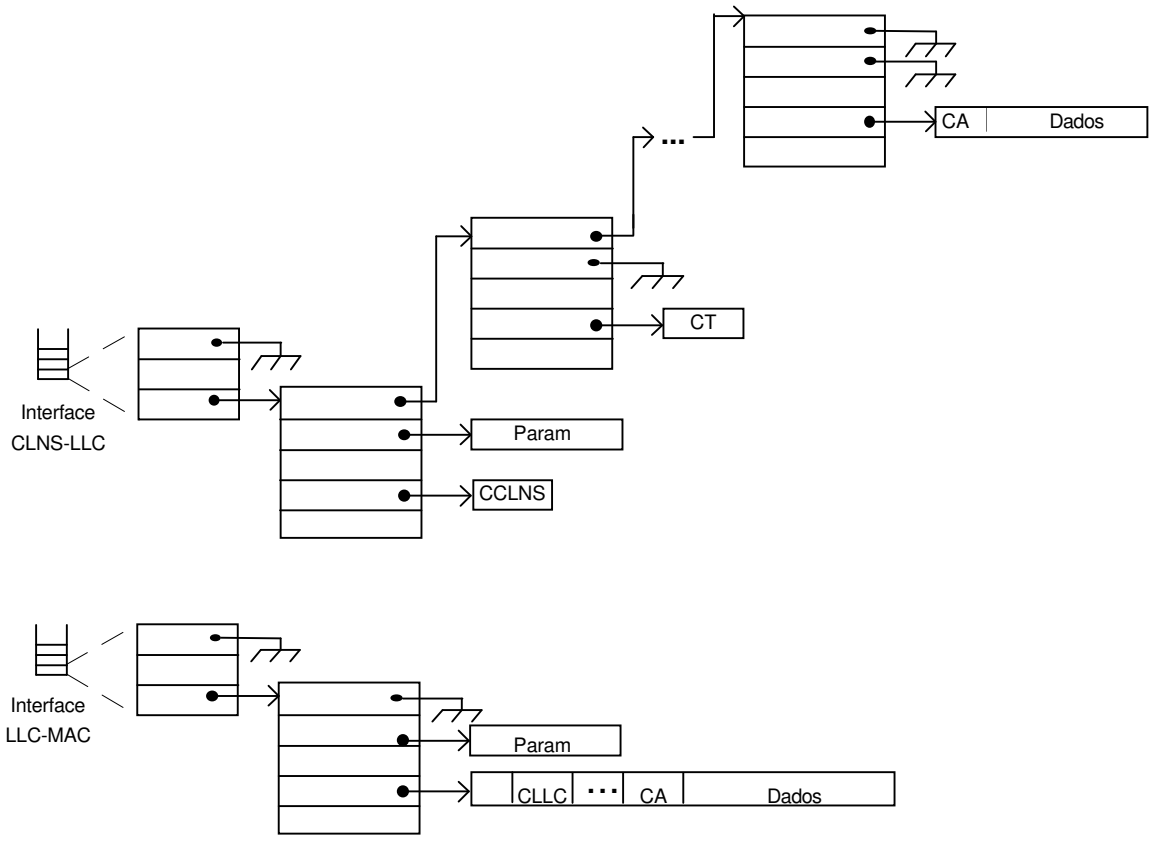


Figura IV.5: Transferência de Dados entre Camadas Adjacentes.

O campo *Número de Alocações* tem um importante papel no sistema. Ele evita grande parte das cópias que são necessárias nos procedimentos de retransmissão. Protocolos baseados em recuperação de erros por retransmissão são usados nas Camadas Enlace, Transporte, Sessão e Aplicação (RTSE). Este campo é incrementado/decrementado toda vez que uma camada aloca/dealoca aquela estrutura. O *buffer* de retransmissão correspondente é liberado da memória somente quando este campo chega a zero. O Módulo de Gerenciamento de Memória é responsável pelo controle desta propriedade.

Portanto, a transferência de dados entre camadas adjacentes da Arquitetura GTA é feita através de pares de filas FIFO e de estruturas de dados padronizadas para possibilitar, entre outras coisas, a inserção de PCIs por encadeamento. A Figura IV.5 ilustra como os dados se apresentam para serem processados pela subcamada de enlace LLC na transmissão e após este processamento, como os mesmos estão dispostos em uma região contígua de memória, apropriados para a transmissão física.

IV.2.3 Gerenciamento de Memória

A inserção/remoção de PCI de SDUs por cada camada (ou subcamada) exige necessariamente um gerenciamento de memória eficiente, com o propósito de alcançar um bom desempenho. Também é importante que a segmentação e remontagem de dados seja possível sem a necessidade de cópias, mas apenas através de manipulação de ponteiros. Estas restrições levaram à implementação de um Gerenciador de Memória específico e otimizado [Bag92].

De maneira a evitar excesso de fragmentação, este módulo divide a memória em regiões de transmissão e recepção. Isto é possível porque os fluxos de transmissão e recepção são independentes. Estas regiões são ainda subdivididas em três partes (*Estrutura Primitiva*, *Estrutura Parâmetros* e região de dados), alocações e dealocações de memória de tamanho fixo e variável. Algoritmos otimizados são utilizados, levando em consideração o comportamento específico de um sistema OSI.

Um módulo de gerenciamento de memória inclui informações de controle utilizadas durante o processo de alocacão de memória: uma relativa às regiões alocadas (referências de início e de fim ou tamanho e referência de início) e outra relativa às regiões livres (referências de início e de fim ou tamanho e referência de início). Estas informações de controle podem ser armazenadas na forma de um cabeçalho de cada região alocada (como por exemplo o do gerenciamento de memória do compilador Borland C), o que no entanto, inviabiliza a técnica

de encadeamento desejada sem o uso de cópia de dados. Assim, o Gerenciamento de Memória, desenvolvido por Baginski [Bag92], armazena as informações de controle em separado. Este modelo permite também um uso mais ordenado da memória tornando o gerenciamento mais rápido e eficiente. Dividiu-se ainda cada região em subregiões reservadas à alocação de Estruturas de Primitivas, Estruturas de Parâmetros e Dados.

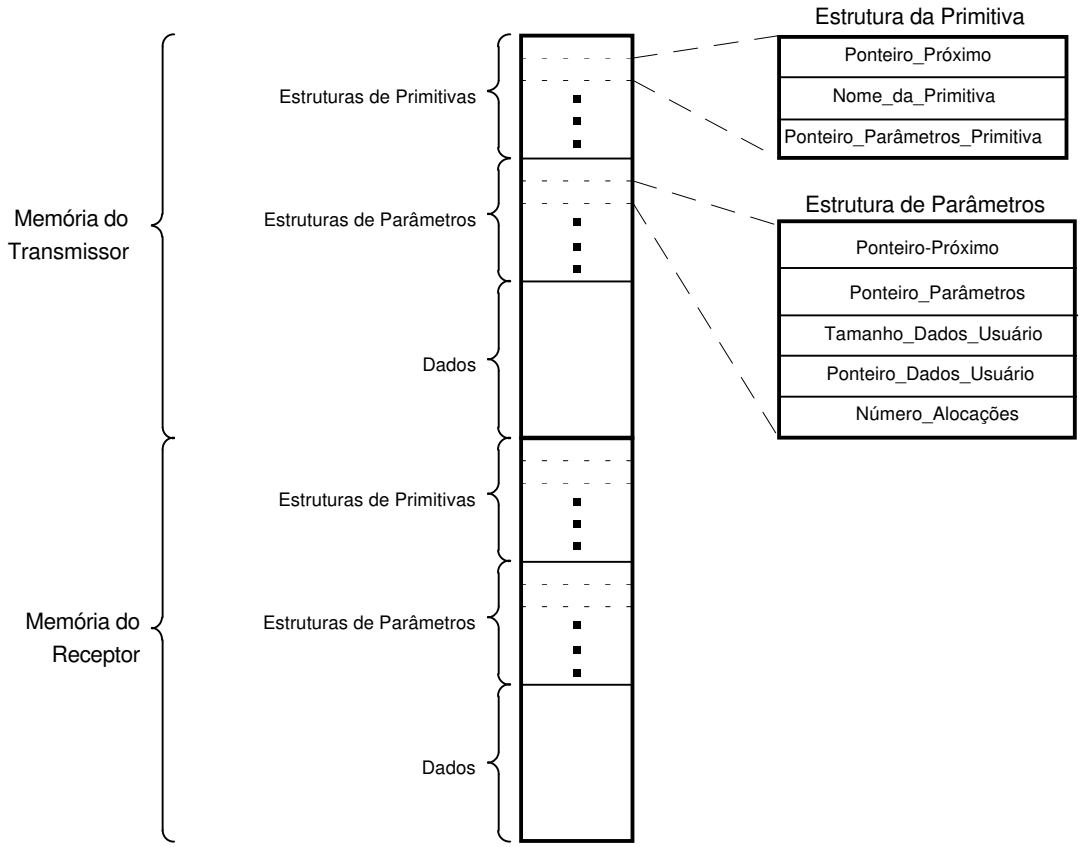


Figura IV.6: Gerenciamento de Memória.

IV.2.4 Escalonamento de Tarefas

A primeira versão do sistema tinha uma rotina principal que verificava se as filas da interface estavam não vazias, isto é, se havia algum elemento (primitiva de serviço) a ser processado. Este procedimento consumia muito tempo de processamento. Uma grande melhoria no desempenho foi atingida com a criação de um módulo Escalonador de Tarefas

[Bag92]. Este módulo consiste em uma fila única que armazena a seqüência de tarefas a serem executadas (Figura IV.7).

Escalonador de Tarefas Versão 1	Escalonador de Tarefas Versão 2
loop Executar_Tarefa_1; Executar_Tarefa_2; - - - Executar_Tarefa_N; forever;	loop Executar_Aplicação; While (Fila_Escalonador_Nao_Vazia) Executar_Rotina_Fila_Escalonador; forever;

Figura IV.7: Versões do Escalonador de Tarefas.

Um protocolo é facilmente definido por sua Máquina de Estados, onde a partir de um certo evento (recebimento de uma primitiva de serviço) e do estado atual, algumas tarefas podem ser executadas, causando a mudança do estado e originando um ou mais novos eventos (envio de primitivas para as camadas adjacentes). Dessa forma, na arquitetura descrita, toda vez que uma primitiva é colocada em uma das filas da interface (criação de um evento), a tarefa correspondente a essa primitiva é colocada na fila do escalonador. A atomização necessária à execução da máquina de estados também é facilmente obtida.

Uma outra característica a nível de implementação foi uma padronização da estrutura do *software* que realiza as Máquinas de Estado dos protocolos. Ao se inserir um determinado evento (primitiva) na fila FIFO da interface entre camadas adjacentes, insere-se também na fila do Escalonador de Tarefas, o módulo que deve tratar este evento. Com isso, conforme o estado do protocolo, identifica-se o evento. É comum no tratamento de um evento surgir a necessidade de gerar um novo evento. Para tanto, deve-se testar o suporte da criação deste novo evento no ambiente de implementação, e com o sucesso nos testes de Fila Não Cheia,

Alocação da Estrutura da Primitiva, Alocação da Estrutura de Parâmetros da Primitiva e Alocação de Memória, continua-se o procedimento de tratamento do evento original e preenche-se os parâmetros das estrutruras de dados referentes ao novo evento. Em seguida, este novo evento é inserido na fila da interface, sendo também inserido na fila do Escalonador de Tarefas o módulo que o tratará. Caso, durante o tratamento deste evento, algum dos testes de suporte da criação de um outro evento falhe, envia-se para a fila do Escalonador de Tarefas o módulo de tratamento em questão. A estrutura do *software* da Camada Sessão não segue exatamente os passos descritos acima, sendo cada procedimento da Máquina de Estados do Protocolo ativado por uma tabela cujos índices são o nome do evento e o estado do protocolo.

IV.2.5 Gerenciamento de Temporizações

A implementação de protocolos normalmente inclui procedimentos baseados em temporizadores que são usados em rotinas de recuperação de erros, controle de tempo máximo dos procedimentos de estabelecimento e liberação de conexões, etc.

Existem diversas formas de se implementar temporizadores. A mais simples consiste em manter para cada temporizador um conjunto de variáveis de controle: Ligado (booleano), Contador (inteiro), Estourado (booleano) e Tempo Máximo (constante). Através de interrupções a tempos constantes este conjunto de variáveis é manipulado ativando ou não os procedimentos relacionados ao estouro do temporizador. A cada interrupção, se o temporizador estiver ligado, se a variável contador foi inferior ao tempo máximo, incrementa-se o contador. Quando o contador for igual ao tempo máximo, a variável estourado é ativada. Esta forma de implementação é limitada quanto ao número máximo de temporizadores e ineficiente com relação ao número de testes lógicos a cada interrupção.

Uma outra forma de implementação mais complexa, porém mais eficiente foi proposta por Varghese e Lauck [VLa87], onde uma analogia a um relógio real é simulada (Figura IV.8) através de um contador em módulo. A implementação do Gerenciamento de Temporizações

segue esta proposta. Esta implementação provou ser bastante eficiente, evitando testes desnecessários a cada interrupção, permitindo portanto a criação de um grande número de temporizadores dinamicamente.

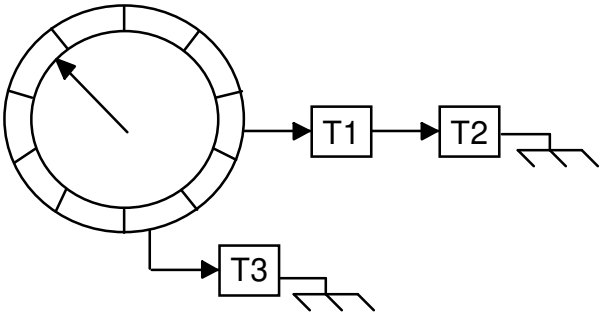


Figura IV.8: Temporização proposta por Varguese & Lauck.

Na arquitetura implementada, o estouro de um temporizador é tratado como uma primitiva de serviço, ou seja, a indicação de estouro (evento) é colocada na fila apropriada da interface e a tarefa correspondente é colocada na fila do escalonador de tarefas.

IV.3 A Arquitetura de Implementação GTA/MM

Visando o desenvolvimento de sistemas de comunicação em tempo real com garantias de qualidade de serviço (QOS), esta seção define uma arquitetura de implementação flexível, eficiente e modular. Esta arquitetura é uma evolução da arquitetura descrita na seção IV.2, no sentido de otimizar e aumentar a flexibilidade dos módulos existentes, e de oferecer novos módulos necessários em sistemas com garantias de QOS.

Com relação a arquitetura GTA/OSI, a principal semelhança é a herança dos módulos básicos e das estruturas de dados. As grandes diferenças estão no desvinculamento em relação ao Modelo OSI e na criação dos módulos específicos. A arquitetura GTA/MM faz uso de protocolos de alta velocidade, de mecanismos de controle de tráfego e policiamento, de mecanismos de garantia de qualidade de serviço, etc.

A arquitetura GTA/OSI visa o alto desempenho de sistemas OSI e com esta finalidade criou-se estruturas de dados e módulos específicos para reduzir o tempo gasto com a "movimentação de dados" no sistema de comunicação. Embora um alto ganho tenha sido obtido [ALD93, AND93, AND94, BBS91, BDu91], a "movimentação de dados" continuava a ser um dos pontos mais críticos para a obtenção de alto desempenho.

Na arquitetura GTA/MM proposta, procurou-se otimizar ainda mais as funções que movimentam dados. A partir das medidas de desempenho efetuadas pelo GTA observou-se que a manutenção da estrutura de dados que permite a movimentação de ponteiros ao invés dos dados e de um módulo específico de gerenciamento de memória eram justificadas. No entanto, devido às características do algoritmo de alocação e dealocação de memória empregado, um uso ainda mais ordenado da memória propiciaria um ganho em desempenho. Assim, além da divisão já existente da região de memória em partes transmissora e receptora, os Módulos de Escalonamento de Tarefas e Gerenciamento de Temporizações tornaram-se independentes em relação ao uso da memória, gerenciando com total autonomia suas estruturas de dados.

Um outro significativo melhoramento proposto na Arquitetura GTA/MM é o gerenciamento de memória por blocos da região de memória responsável pelos dados a serem transferidos.

O escalonamento de tarefas para aplicações multimídias exige um escalonamento com prioridades a fim de atender aos dados que possuem restrições temporais. Além disso, como dito anteriormente, o novo Módulo de Escalonamento de Tarefas possui o seu próprio gerenciamento de memória e se serve de uma pré-alocação para a região das filas de tarefas.

O Módulo de Gerenciamento de Temporizações assume uma importância fundamental para as aplicações multimídias. Sua utilização passa a ser bastante freqüente devido as funções

de controle de fluxo por taxa, policiamento e garantia de QOS. Na arquitetura GTA/MM o Módulo de Gerenciamento de Temporizações inclui funções para a manipulação de temporizadores "auto-disparáveis" e gerencia as suas estruturas de forma independente, fazendo uso da pré-alocação.

A comunicação entre camadas na arquitetura GTA/MM é feita por chamada de procedimento. Esta opção se mostrou bem mais eficiente que a troca de mensagens por filas, embora o sistema fique mais fortemente acoplado.

Finalmente, o sistema de comunicação exige protocolos de alta velocidade. Protocolos de alta velocidade estão sendo implementados pelo GTA e deverão possuir diversas opções configuráveis conforme a aplicação [Nun95]. No entanto, algumas funções para alta velocidade foram incluídas no protocolo do sistema de comunicação. Foram adicionadas opções de controle de fluxo por taxa transmissão, além do convencional controle por janela de emissão. Foram implementadas funções para permitir o estabelecimento de conexão implícito, onde o originador da conexão tem a possibilidade de iniciar a transmissão sem ter recebido a confirmação do estabelecimento da conexão. Foram incluídas informações quanto ao seqüenciamento, tal que o usuário do receptor seja informado em caso de perda de um quadro. Neste caso, o protocolo do receptor poderia ou não requerer a retransmissão de tal quadro. Além disso, estão disponíveis as funções definidas no protocolo Transporte Classe 2.

Foram também implementados diversos mecanismos de controle de tráfego e policiamento, além de mecanismos para a garantia da qualidade de serviço em termos de atraso, vazão e *jitter*. A implementação destes mecanismos será descrita no capítulo V.

IV.4 O Módulo de Gerenciamento de Memória

O Módulo de Gerenciamento de Memória, de maneira a evitar excesso de fragmentação, divide a memória em regiões de transmissão e recepção. Este modelo permite

um uso mais ordenado da memória tornando o gerenciamento mais rápido e eficiente. Isto é possível porque os fluxos de transmissão e recepção são independentes. Estas regiões são ainda subdivididas em quatro partes. Duas estão reservadas para estruturas de dados de tamanho fixo das interfaces entre camadas adjacentes e entre os módulos básicos. Uma região é alocada para estruturas de tamanho variável usadas na passagem de parâmetros entre camadas de um mesmo sistema. A quarta região de memória é alocada de acordo com cada fluxo estabelecido e contém estruturas de dados de tamanho fixo, denominados blocos, que são usados pelos dados que serão efetivamente transmitidos. Diferentemente dos gerenciadores de memória de propósito geral, este gerenciador armazena as informações de controle de alocação separado dos dados, permitindo a implementação eficiente e sem cópias de funções de segmentação, remontagem, concatenação, separação, liberação de parte dos dados, entre outras.

As funções do Módulo de Gerenciamento de Memória são brevemente descritas a seguir.

- Inicializar_Memória_[Tx][Rx]: esta função é responsável pela alocação inicial de uma região de memória destinada às estruturas de tamanho fixo da arquitetura (estruturas de primitivas e estruturas de parâmetros) e às estruturas de tamanho variável. Nesta nova proposta do módulo de gerenciamento de memória, as únicas estruturas de tamanho variável manipuladas são as estruturas de parâmetros das primitivas de serviço. Os parâmetros das primitivas são diferentes de acordo com a primitiva (estabelecimento de conexão, liberação de conexão, etc.) e existem primitivas que não possuem parâmetros, como é o caso das primitivas de transferência de dados. Vale a pena ressaltar que o gerenciamento de memória é independente de acordo com o sentido do fluxo de dados. Em geral, as funções referentes ao fluxo de transmissão (Tx) são análogas às funções do fluxo de recepção (Rx), com poucas exceções que serão ressaltadas. Esta separação entre o gerenciamento dos fluxos diminui consideravelmente a fragmentação interna da memória [Bag92];

- Finalizar_Memória_[Tx][Rx]: esta função deve ser executada na finalização do sistema de comunicação, devolvendo ao sistema operacional a memória alocada. Deve-se tomar o cuidado de só executar esta função quando nenhuma conexão estiver estabelecida;

- Alocar_Memória_[Tx][Rx]: esta função permite a alocação de regiões de memória de tamanho variável. Seu algoritmo procura a primeira região livre que comporte a região solicitada. Variáveis de controle de início e fim desta região são inicializadas com a finalidade de identificar a região de memória ocupada. Este controle não é feito no cabeçalho da região. No sistema de comunicação implementado, esta função está sendo utilizada apenas para a transferência de parâmetros das primitivas entre entidades adjacentes, em um mesmo sistema de comunicação;

- Liberar_Memória_[Tx][Rx]: esta função permite a liberação, a partir de uma referência de início, de uma região de memória de tamanho variável menor ou igual ao tamanho da região alocada. Caso seja liberado uma região menor que à alocada, o controle de início desta região é atualizado. Caso a região liberada esteja contígua a uma região livre, o controle de início ou fim desta região livre é atualizado. Caso contrário, são atribuidos controles de início e fim para esta nova região livre. Como a alocação e liberação das regiões de memória não são necessariamente ordenadas, podem ocorrer diversas regiões livres resultantes desta fragmentação interna da memória;

- Alocar_Estrutura_Primitiva_[Tx][Rx]: esta função retorna a referência para a primeira estrutura de primitiva livre, e incrementa o índice de estrutura livre. Se a alocação e liberação de estruturas for razoavelmente ordenada, de forma que após a alocação de todas as estruturas, as primeiras a serem alocadas já tenham sido liberadas, o processamento desta alocação será minimizado;

- Liberar_Estrutura_Primitiva_[Tx][Rx]: esta função coloca a estrutura primitiva na condição de livre;

- Alocar_Estrutura_Parâmetros e Liberar_Estrutura_Parâmetros_[Tx][Rx]: estas funções são análogas às funcões de alocação e liberação de estruturas de primitivas, porém manipulam estruturas de parâmetros;

- Inicializar_Blocos_[Tx][Rx]: esta função aloca do sistema operacional, um conjunto de blocos (regiões de memória de tamanho fixo) para serem utilizados para a comunicação de dados entre sistemas em uma conexão. Deve ser executada no estabelecimento de uma conexão para a reserva de memória para a mesma;

- Finalizar_Blocos_[Tx][Rx]: esta função deve ser executada ao término da conexão, liberando o conjunto de blocos utilizados pela mesma;

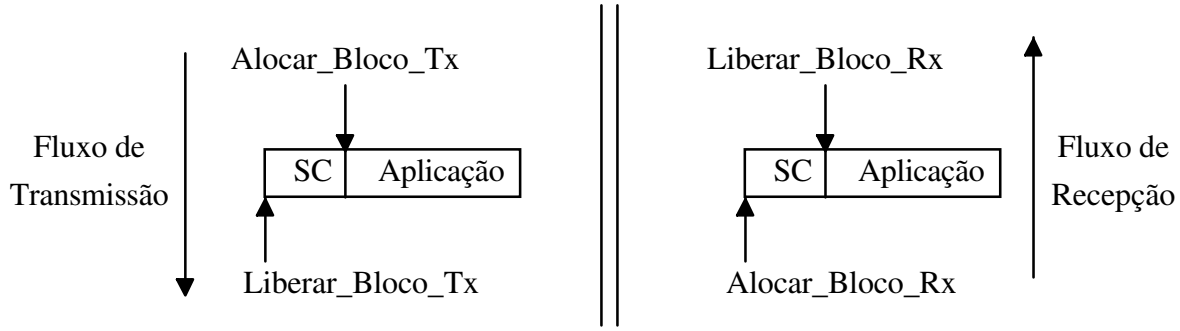


Figura IV.9: Procedimento de alocação e liberação de blocos pelos fluxos de transmissão e de recepção.

- Alocar_Bloco_[Tx][Rx] e Liberar_Bloco_[Tx][Rx]: estas funções são análogas às funcões de alocação e liberação de estruturas de primitivas e de parâmetros, porém manipulam blocos de dados de uma conexão específica. O número da conexão é parâmetro desta função. Na parte de transmissão do sistema implementado, esta função é normalmente chamada pela Aplicação. De forma a prover maior transparência para a

Aplicação, o ponteiro de retorno para os dados é incrementado do tamanho dos cabeçalhos das camadas inferiores do sistema de comunicação. A Figura IV.9 ilustra este procedimento tanto para a recepção quanto para a transmissão.

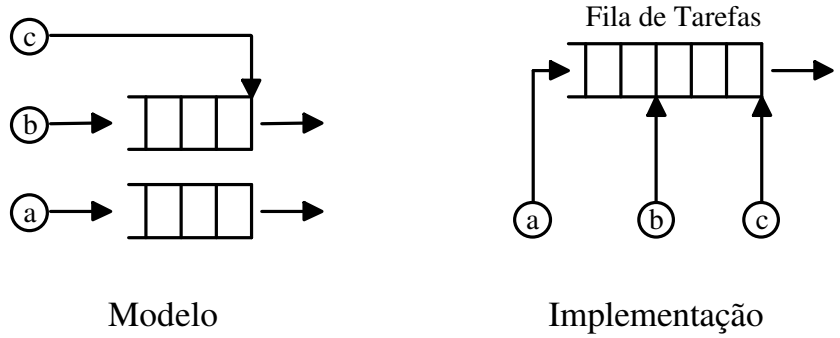
IV.5 O Módulo de Escalonamento de Tarefas

O Módulo de Escalonamento de Tarefas armazena e determina a seqüência das tarefas a serem executadas. Além disso, possui funções para permitir níveis de prioridade para os fluxos de tempo real. O comportamento da fila de tarefas é modelado como o de duas filas onde um elemento da fila de menor prioridade só é servido quando a fila de maior prioridade estiver vazia. De forma a priorizar o atendimento de tarefas de altíssima prioridade, como por exemplo o "estouro" de temporizadores para controle de tráfego e policiamento, estas tarefas são inseridas na primeira posição da fila de tarefas (ponto **c** da Figura IV.10). Este ponto é também o ponto de saída de todas as tarefas. Um segundo nível de prioridade é dado às tarefas dos fluxos em tempo real inseridas no ponto **b**. As demais tarefas de menor prioridades são inseridas no final da fila de tarefas (ponto **a**). Implementando o modelo de duas filas com prioridades através de uma única fila com três pontos de entrada e um único ponto de saída evita-se testes a cada retirada da fila para verificar se a fila de maior prioridade está vazia ou não.

As funções do Módulo de Escalonamento de Tarefas são:

- Inicializar_ET: esta função deve ser executada na inicialização do sistema de comunicação. Seu processamento consiste em inicializar as variáveis de controle da fila de tarefas e em reservar memória para os descritores de cada tarefa, além de inicializar as variáveis de controle desta região de memória. Assim, o gerenciamento dos elementos da fila de tarefas, descritores da tarefa, é feito de forma dinâmica, pré-alocada e independente do Módulo de Gerenciamento de Memória. No entanto, este

gerenciamento é feito de forma análoga ao gerenciamento de estruturas de tamanho fixo do Módulo de Gerenciamento de Memória;



a - ponto de entrada das tarefas de menor prioridade.
b - ponto de entrada das tarefas de maior prioridade.
c - ponto de entrada das tarefas de prioridade máxima
e de saída de todas as tarefas.

Figura IV.10: Módulo de Escalonamento de Tarefas.

Na arquitetura GTA/OSI, aproveitando-se do fato de o elemento da fila de tarefas possuir o mesmo tamanho e campos equivalentes aos da Estrutura_Primitiva (Figura IV.11), utilizou-se como elemento da fila de tarefas uma estrutura alocada pelo Gerenciamento de Estruturas de Primitivas Tx. Porém, esta simplificação ocasiona uma maior desordenação das estruturas de primitivas livres e alocadas, sendo esta região de memória mais frequentemente utilizada (em relação à região análoga para a recepção). Este aumento da utilização destas estruturas e o seu subseqüente desordenamento aumentava o tempo de alocação de uma estrutura e ainda possibilitava o transbordo desta região de memória.

A Figura IV.11 ilustra a forte semelhança entre as estruturas de primitivas e os elementos da fila de tarefas. Observe que os campos Ptr_Próximo são do tipo ponteiro, e sua finalidade em ambas as estruturas é o encadeamento com o próximo elemento da fila entre camadas ou de tarefas. Os campos Função e Ptr_Parâmetros também são do tipo ponteiro, sendo que o campo Função indica qual a tarefa que está na fila. Os campos Instância e

Nome_Primitiva são do tipo *byte* (ocupam 1 octeto), sendo que o campo Instância indica a que conexão a tarefa pertence.

Estrutura_Fila_ET {
Estrutura_Fila_ET *Ptr_Proximo;
void (*) ( ) Funcao;
byte Instancia;
};
	Estrutura_Primitiva {
	Estrutura_Primitiva *Ptr_Proximo;
	char * Ptr_Parametros;
	byte Nome_Primitiva;
	};

Figura IV.11 : Comparação das Estruturas de Primitivas e da Fila de Tarefas.

Como mencionado, a utilização de Estruturas de Primitivas Tx como elementos da fila de tarefas não se mostrou eficiente. Na arquitetura GTA/MM, este problema está solucionado através de um gerenciamento de elementos da fila de tarefas próprio. A versão implementada pré-aloca um total de 1024 elementos, os quais ocupam um total de 9 koctetos, possibilitando a existência de até 1024 tarefas na fila.

- Finalizar_ET: esta função deve ser executada na finalização do sistema de comunicação, devolvendo ao sistema operacional a memória alocada. Deve-se tomar o cuidado de só executar esta função quando todas as aplicações estiverem inativas e a fila de tarefas estiver vazia;

- Alocar_Elem_Fila_ET: esta função retorna o endereço de um elemento da fila de tarefas livre ou nulo, caso todos os elementos estejam ocupados. O algoritmo utilizado privilegia alocações e dealocações seqüenciadas. Assim, esta função possui um bom desempenho se os elementos da fila de tarefas forem alocados e liberados de maneira razoavelmente ordenada. Seu processamento é idêntico ao das funções de gerenciamento de estruturas de tamanho fixo do Módulo de Gerenciamento de Memória;

- Liberar_Elem_Fila_ET: esta função basicamente recebe o elemento em uso e o libera, de forma a torná-lo livre;

- Enviar_Rotina_Fila_ET: esta função insere a tarefa na última posição da fila de tarefas (ponto **a**). Nesta fila existem ainda dois outros pontos de entrada de tarefas, conforme ilustrado na Figura IV.10;

- Enviar_Rotina_Tempo_Real_Fila_ET: esta função foi criada nesta arquitetura para prover um nível de prioridade superior para os fluxos em tempo real. Desta forma, o comportamento da fila de tarefas foi modelado como o de duas filas onde um elemento da fila de menor prioridade só é servido quando a fila de maior prioridade estiver vazia. A implementação das filas de tarefas consiste de uma fila FIFO com um segundo ponto de entrada (ponto **b**) para as tarefas prioritárias;

- Enviar_Rotina_Expressa_Fila_ET: esta função também foi criada nesta arquitetura, e a sua finalidade é priorizar o atendimento de tarefas de altíssima prioridade, como por exemplo o estouro de alguns temporizadores. Esta tarefa é portanto inserida na primeira posição da fila de tarefas (ponto **c**). Este ponto é também o ponto de saída de todas as tarefas;

- Executar_Rotina_Fila_ET: esta função retira a primeira tarefa da fila do escalonador e executa-a, passando como parâmetro da função o número da conexão a que ela pertence. Esta função só deve ser executada se a fila não estiver vazia, devendo ser utilizada conforme ilustrado na Figura IV.7;

- Cancelar_Rotina_Fila_ET: esta função foi criada nesta arquitetura para suprir uma necessidade exclusiva da nova versão do Módulo de Gerenciamento de Temporizações, que será detalhada na seção IV.6. Embora criada para este fim, ela pode ser utilizada por

qualquer outro módulo ou protocolo do sistema de comunicação. Seu processamento consiste em retirar da fila de tarefas a tarefa cujo nome e instância coincidem com os parâmetros especificados. Por uma questão de otimização que será melhor entendida na subseção a seguir, a procura pela tarefa a ser retirada da fila é feita a partir do primeiro elemento da fila (ponto **c**);

- Retirar_Rotina_Fila_ET: esta função é executada exclusivamente pela função Executar_Rotina_Fila_ET e seu processamento consiste em retirar a primeira tarefa da fila de tarefas.

IV.6 O Módulo de Gerenciamento de Temporizações

O Módulo de Gerenciamento de Temporizações é bastante eficiente. Ele é baseado na proposta de Varghese & Lauck [VLa87], no qual uma analogia a um relógio real é simulada através de um contador em módulo. Esta implementação privilegia o desempenho, evitando testes desnecessários a cada interrupção, permitindo portanto a criação dinâmica de um grande número de temporizadores. São fornecidas funções para disparar temporizadores simples e auto-disparáveis, reiniciar, interromper e sinalizar o estouro de um temporizador.

Nesta implementação foram assumidas algumas posições de forma a privilegiar o desempenho. Estruturas descritoras de tempo foram pré-alocadas e novamente o seu gerenciamento foi feito de forma independente do gerenciamento de memória, no entanto de forma análoga ao gerenciamento de estruturas de tamanho fixo. Com isso, foi limitado o número máximo de temporizadores que podem ser criados simultaneamente. Para representar a escala de tempo, foi utilizado um vetor de tamanho fixo. Com isso, a criação de um temporizador corresponde a atribuir a uma posição do vetor de tempo a referência de um descritor de tempo. Os descritores de tempo podem ser cascateados, possibilitando que vários temporizadores possam estourar em um mesmo instante de tempo. O vetor de tempo é

percorrido por um contador de tempo (ponteiro do relógio) conforme as interrupções do relógio (tics e tacs).

- Inicializar_Módulo_Temporizador: esta função deve ser executada na inicialização do sistema. É responsável por inicializar as variáveis de controle dos descritores de tempo e habilitar interrupções regulares que indiquem a evolução do tempo. No ambiente IBM-PC são utilizadas as interrupções do relógio de tempo real geradas 18.2 vezes por segundo ou em intervalos de cerca de 55 ms. Já nas estações de trabalho SUN, pode-se especificar o intervalo de geração das interrupções em microsegundos. Pode-se ainda optar por interrupções geradas pelo relógio de tempo real ou por um relógio de tempo virtual. No caso de se optar pelo relógio de tempo virtual, o tempo decorrido entre as interrupções é computado de acordo com o tempo de processamento do processo em questão. No arquivo tipglob.h deve-se informar o ambiente de execução (PC ou SUN), e no caso de SUN, o tipo de relógio a ser utilizado (TEMPO_REAL ou TEMPO_VIRTUAL);

- Finalizar_Módulo_Temporizador: esta função desabilita a geração de interrupções a intervalos regulares. Após a execução desta função o estouro de temporizadores é cancelado;

- Criar_Temporizador: esta função recebe como parâmetros o tempo de temporização (em unidades de tempo), o número da conexão e a função a ser executada devido ao estouro do temporizador. É responsável por alocar um descritor de tempo, preencher seus parâmetros, e posicioná-lo no "instante de tempo" correspondente ao instante atual acrescido do valor do tempo de temporização;

- Criar_Temporizador_AD: esta função é analoga a anterior, exceto pelo fato de criar temporizadores auto-disparáveis, ou seja, no estouro do temporizador, o descritor de tempo do mesmo é reutilizado na posição do instante de tempo correspondente ao

instante do estouro acrescido do valor do tempo de temporização. Isto resulta no "re-disparo" do temporizador ao final de seu estouro. Este processo continua indefinidamente até que o temporizador seja interrompido;

- Interromper_Temporizador: esta função cancela o descritor de tempo correspondente ao temporizador. Caso o temporizador já tenha estourado, porém não atendido, a função correspondente ao estouro deste temporizador estará na fila do escalonador de tarefas. Neste caso, a função Cancelar_Rotina_Fila_ET é automáticamente chamada. Como as funções relativas a estouros de temporizadores são enviadas para o início da fila de tarefas, a procura para cancelar uma rotina é feita a partir da primeira posição;

- Reiniciar_Temporizador: esta função reposiciona o descritor de tempo de forma a adiantar ou atrasar o estouro do temporizador, sendo útil na sintonia de temporizadores dinamicamente ajustáveis;

- Finalizar_Temporizadores_Conexão: esta função foi criada para garantir que ao término de uma conexão, todos os temporizadores relativos a esta conexão sejam interrompidos;

- Interrupção_Temporizador: esta é a principal função do gerenciamento de temporizações. É responsável pela atualização do contador de tempo a cada interrupção do relógio, pelo teste do estouro de um temporizador e pelo conseqüente envio da função relativa a um estouro para a fila do escalonador de tarefas. O estouro de um temporizador é detectado pela existência de um descritor de tempo na posição atual do contador de tempo.

IV.7 Arquitetura de Implementação e Protocolos

Faz parte dos objetivos deste trabalho a implementação de mecanismos que levam a protocolos de alta velocidade e a comprovação, através de medidas de desempenho, da eficiência da arquitetura proposta. No entanto, não é objetivo deste trabalho a proposição de um novo protocolo apropriado para o ambiente de alta velocidade.

Assim, este trabalho parte do perfil de protocolos constituído do Transporte (Classes 0 e 2), INP (Inactive Network Protocol), LLC Tipo 1 (Logical Link Control) e MAC IEEE802.3 (Medium Access Control) implementados segundo a Arquitetura GTA/OSI e evolui passo-a-passo para a Arquitetura GTA/MM.

O Modelo de Referência OSI foi proposto em uma época onde as comunicações eram bastante sujeitas a erros de transmissão e a taxa de transmissão se limitava a dezenas de kbps. O perfil de protocolos propostos visavam tornar o sistema de comunicação confiável e extremamente flexível. A grande quantidade de camadas e a repetição de diversas funções (multiplexação, segmentação, controle de fluxo e erros) em várias destas camadas constituem em fatores de degradação da eficiência do sistema de comunicação. Assim, os protocolos de Transporte classe 0 classe 2, denominados TP0 e TP2, foram alterados para atender as características de protocolos de alta velocidade (*light weighed protocols*). As funções de estabelecimento de conexão implícita e de controle de fluxo por taxa de transmissão foram inseridas e por outro lado as funções de concatenação/separação (raramente usadas mas obrigatórias) foram retiradas. Estes "novos" protocolos foram denominados SC0 e SC2.

Com este perfil definido, foi feita a primeira otimização do sistema (versão 1). As filas entre camadas foram retiradas da arquitetura, com exceção das filas para a camada Aplicação e das filas para o meio de comunicação. A interação entre camadas foi portanto feita através de chamadas de procedimento, sem nenhuma perda de funcionalidade ou de flexibilidade, havendo porém um conseqüente maior acoplamento entra as entidades adjacentes.

A segunda grande alteração (versão 2) foi a utilização de um gerenciamento de unidades de informação de tamanho fixo (blocos), resultando na inserção de PCIs pelo método da alocação máxima, diferindo da técnica de encadeamento usada originalmente. Desta forma, torna-se desnecessária a cópia dos dados para uma região contígua de memória para serem transmitidos, uma vez que os mesmos já se encontram em uma forma contígua. Portanto, foi eliminado o *buffer* circular pré-alocado que fazia parte da interface MAC-LLC. Além disso, o gerenciamento de blocos de dados de tamanho fixo é extremamente mais simples que o gerenciamento de dados de tamanho variável. Neste sistema, os dados a serem transmitidos são alocados na aplicação e somente liberados pelo meio de comunicação. Na recepção, o processamento análogo inverso é feito. Note que as camadas intermediárias não possuem a necessidade de alocar memória para inserir seus PCIs. Foi também padronizado a estrutura de dados que representa a primitiva de serviço de forma que o elemento da fila proveniente da aplicação possa ser enviado diretamente à fila da camada MAC, e vice-versa.

Na versão 3 do sistema de comunicação foi retirada a fila de transmissão para o meio, sendo os dados transmitidos também por chamada de procedimento.

A Figura IV.12 ilustra a evolução da arquitetura proposta.

IV.8 Medidas de Desempenho

Esta seção ilustra numericamente os avanços obtidos em cada passo da evolução da arquitetura. Para efeito de comparação são utilizados os protocolos Transporte Classe 0 e 2 do Modelo OSI implementados na arquitetura GTA/OSI e os os protocolos SC0 e SC2 implementados para a base do sistema de comunicação na arquitetura GTA/MM.

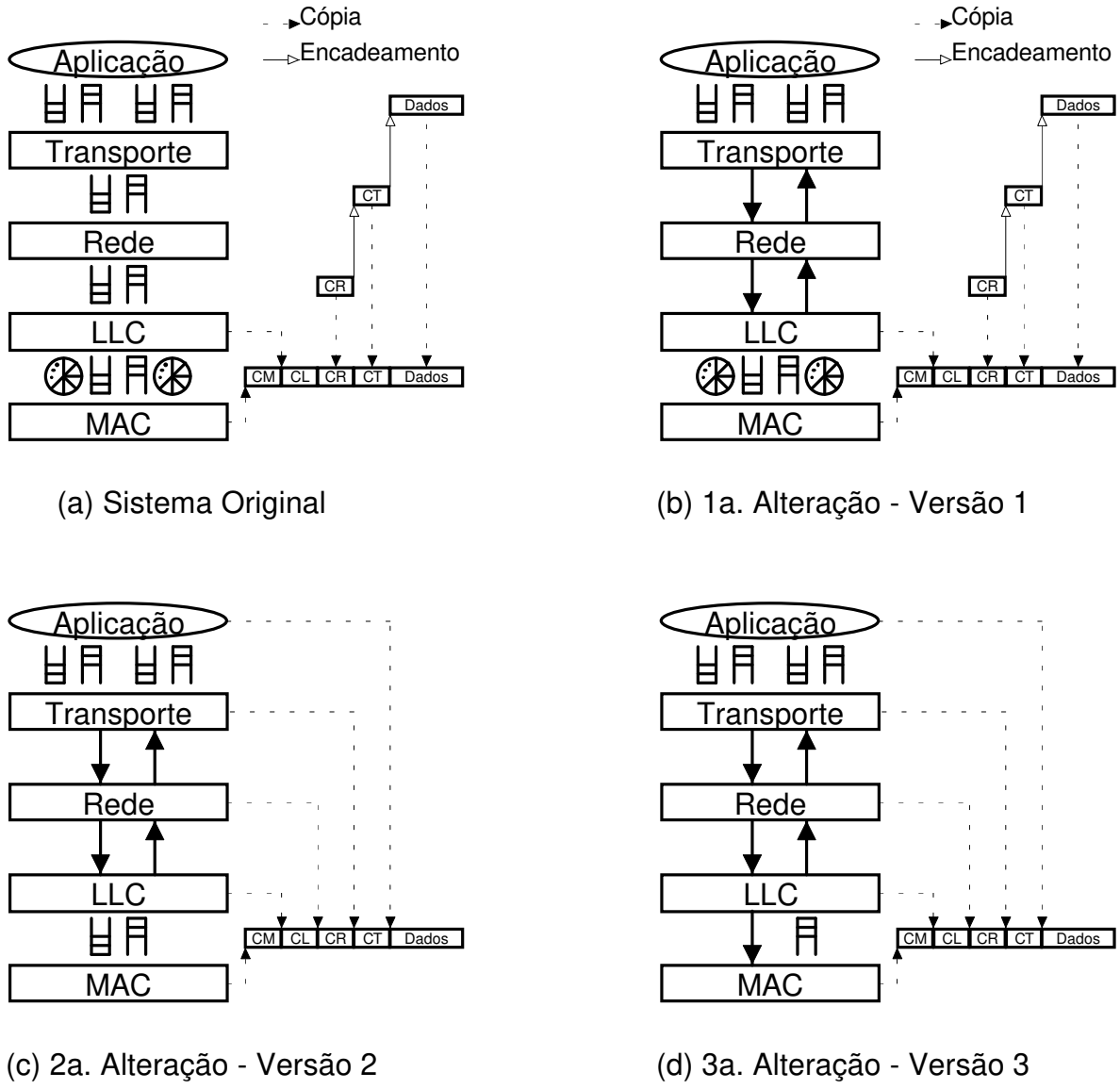


Figura IV.12: Evolução da Arquitetura de Implementação.

A análise de desempenho enfatizou a vazão atingida pelo usuário do sistema de comunicação. Vazão é definida como o total de informação útil transferida (mensagens sem cabeçalhos nem caudas) por unidade de tempo. O tempo total de transferência da informação deve ser medido a partir do instante que o transmissor envia a primeira mensagem até o receptor receber a última mensagem. Entretanto, como isto requer um sistema de medição distribuído, um outro esquema de medição foi empregado. Este esquema consiste em medir no transmissor o intervalo de tempo entre a transmissão da primeira mensagem e o recebimento do reconhecimento positivo da última mensagem. O sistema de medição faz uso de funções

baseadas no relógio de tempo real do IBM-PC, o que limita sua precisão a 55 ms. Para tornar o erro da medida desprezível, foi transferido um volume de informação composto de 10.000 quadros, o que torna o tempo de transferência muitas vezes superior ao período usado como unidade básica de tempo. As medidas de desempenho foram feitas em *loopback* computadores IBM-PC 486 SX 33MHz e em computadores IBM-PC 386 SX 25 MHz. Foram transferidos quadros de tamanho 1024 octetos.

A Tabela IV.1 ilustra os resultados de medidas de desempenho obtidos.

Observa-se dos resultados que a eliminação das filas e a substituição dos protocolos de transporte resultaram num ganho de cerca de 51% para a vazão de quadros de tamanho 1024 sem controle de fluxo e de 82% para quadros com controle de fluxo. A eliminação dos *buffers* circulares e o emprego da técnica de alocação máxima (versão 2) resultaram num ganho de 54% para a vazão de quadros de tamanho 1024 sem controle de fluxo e de 44% para quadros com controle de fluxo. Por fim a versão 3 proporcionou um ganho de 9% para a vazão de quadros de tamanho 1024 sem controle de fluxo e de 10% para quadros com controle de fluxo.

O sistema atual obteve um desempenho três vezes maior que o sistema original (até 215 % de ganho) no IBM-PC 486 SX 33 MHz para o protocolo com controle de fluxo.

Tabela IV.1: Medidas de Desempenho da Evolução da Arquitetura de Implementação.

	Arquitetura GTA/OSI			
	386 SX 25 MHz		486 SX 33 MHz	
	Tempo (segundos)	Vazão (Mbps)	Tempo (segundos)	Vazão (Mbps)
TC0	27.0879	3.02	20.8791	3.92
TC2	35.1099	2.33	27.8022	2.94

	Arquitetura GTA/MM versão 1			
	386 SX 25 MHz		486 SX 33 MHz	
	Tempo (segundos)	Vazão (Mbps)	Tempo (segundos)	Vazão (Mbps)
SC0	17,9670	4,56	14,1209	5,80
SC2	19,3407	4,24	15,1648	5,40

	Arquitetura GTA/MM versão 2			
	386 SX 25 MHz		486 SX 33 MHz	
	Tempo (segundos)	Vazão (Mbps)	Tempo (segundos)	Vazão (Mbps)
SC0	11,6484	7,03	8,3516	9,81
SC2	13,4615	6,09	9,6154	8,52

	Arquitetura GTA/MM versão 3			
	386 SX 25 MHz		486 SX 33 MHz	
	Tempo (segundos)	Vazão (Mbps)	Tempo (segundos)	Vazão (Mbps)
SC0	10,7143	7,64	7,7473	10,57
SC2	12.1978	6,72	8,8462	9,26