INSTITUTO FEDERAL DE SANTA CATARINA

BRUNO ANTÔNIO DE PINHO

Protocolo Ponto-a-Ponto sem-fio usando kernel de tempo real

São José - SC

Julho/2019

PROTOCOLO PONTO-A-PONTO SEM-FIO USANDO KERNEL DE TEMPO REAL

São José - SC

Julho/2019

Bruno Antônio de Pinho
Protocolo Ponto-a-Ponto sem-fio usando kernel de tempo real/ Bruno Antônio de Pinho. – São José - SC, Julho/2019-
33 p. : il. (algumas color.) ; 30 cm.

Orientador: Marcelo Maia Sobral

Monografia (Graduação) – Instituto Federal de Santa Catarina – IFSC
Campus São José
Engenharia de Telecomunicações, Julho/2019.

1. Palavra-chave1. 2. Palavra-chave2. 2. Palavra-chave3. I. Orientador. II. Instituto Federal de Santa Catarina. III. Campus São José. IV. Título

BRUNO ANTÔNIO DE PINHO

PROTOCOLO PONTO-A-PONTO SEM-FIO USANDO KERNEL DE TEMPO REAL

São José - SC, 15 de outubro de 2015:

Marcelo Maia Sobral, Dr.
Orientador
Instituto Federal de Santa Catarina

Eraldo Silveira e Silva, Dr.
Instituto Federal de Santa Catarina

RESUMO

Redes sem fio são extremamente dependentes do tempo de resposta a já que quando o tempo limite de resposta de um pacote é ultrapassado o mesmo é descartado. Sistemas operacionais em tempo real são especializados no tempo de resposta sendo mais preciso que sistemas operacionais de propósito geral. Este trabalho tem como objetivo a implementação de um device driver para protocolo um protocolo ponto-a-ponto para redes sem fio em um sistema operacional em tempo real a fim de utilizar suas características em tempo de resposta de modo a se obter um melhor desempenho do protocolo

Palavras-chave: RTOS. Xenomai. portocolo ponto-a-ponto. Tempo Real. Device Driver.

ABSTRACT

This is the english abstract.

Keywords: latex. abntex. text editoration.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

As redes sem-fio estão sendo cada vez mais utilizadas na comunicação entre dispositivos dos mais variados tipos e tamanhos (Notebooks, *smartphones*, *Internet of Things* (IoT), sensores, entre outros) em diferentes ambientes, como residências, edifícios, florestas e campos de batalha. Por permitirem a mobilidade. Diferentes padrões e tecnologias de rede sem-fio surgiram nos últimos anos para acomodar esta vasta gama de aplicações e coberturas. Foram concebidas das redes celulares de larga cobertura, passando pelas redes locais (WLANs), neste cenário, o padrão `IEEE802.11` obteve um enorme sucesso. Pela sua diversidade em termos de capacidade e cobertura e devido ao baixo custo dos dispositivos de rede, como solução para redes residenciais e de campus (redes locais e metropolitanas), ou mesmo, em enlaces ponto-a-ponto de média distância em soluções corporativas. Esta vasta aplicabilidade do padrão `IEEE802.11` tem sido a chave do seu sucesso comercial. O padrão `IEEE802.11` especifica o protocolo de controle de acesso ao meio (do inglês *Medi Acess Control* (MAC)) e diferentes camadas físicas de alcance e velocidades diversas. Avanços recentes nas técnicas de processamento de sinais permitem que se atinjam taxas de transmissão de até 14 Gbps no padrão `IEEE802.11ax` que opera nas bandas de 2,4 e 5 Ghz. Apesar do aumento contínuo da capacidade dessas redes (RUBINSTEIN; REZENDE, 2002).

O tempo de espaçamento entre quadros tem uma grande importância em coordenar o acesso para o meio de transmissão. A família de protocolos `IEEE 802.11` usa quatro espaçamentos entre quadros diferentes, três dos quais são utilizados para determinar o meio de acesso. A variação desses espaçamentos determina os diferentes níveis de prioridade para os diferentes tipos de trafico. A lógica por trás disso é que trafego com alta prioridade não tem de esperar contanto que o meio de transmissão esteja livre, por tanto se há um trafego de alta prioridade esperando, ele ira ocupar a rede antes de um trafego com prioridade baixa tentar transmitir (GAST, 2005).

Sistemas operacionais tradicionais dão impressão de ser multitarefas dividindo o seu tempo de processamento em fatias de tempo, cada processo tem uma ou mais fatias de tempo alocadas conforme sua prioridade, após esse tempo acabar passa-se para o próximo processo dando a impressão que múltiplas aplicações estão rodando simultaneamente (MAZIERO, 2019). Esta abordagem tem algumas desvantagens no entanto como, por exemplo uma aplicação com uma prioridade alta e com um longo tempo de processamento pode fazer com que aplicações menos prioritárias fiquem longos períodos sem ser processadas.

Sistema operacionais em tempo real do inglês *Real Time Operational Systems* (RTOS) é um sistema operacional mais especializado onde o tempo de resposta é mais importante do que executar centenas de tarefas simultaneamente. O tempo de resposta não precisa necessariamente ser o mais rápido possível, como o nome pode sugerir, mas deve ser previsível, logo, *Real-Time* pode ser uma resposta de vários minutos ou nano segundos, dependendo da forma que seu sistema funciona. As tarefas do RTOS contam com tempo limite para serem executadas, então é comumente chamado de *Time Critical* (MAZIERO, 2019).

O Xenomai é um RTOS baseado em Linux, possuindo um *kernel* Linux tradicional e um *kernel* para aplicações em tempo real chamado Cobalt, o qual trata aplicações que precisam de um curto tempo de resposta de maneira independente do *kernel* Linux. (XENOMAI, 2019).

Esse trabalho se propõe a fazer um protocolo MAC ponto-a-ponto (do inglês *Point-To-Point* (PTP)) para links de média distância (até 10 km), usando um RTOS para obter temporização precisa para as ações do protocolo (transmissões temporizadas, *backoffs* e *timeouts* de espera por quadros). Para tal

será desenvolvido um *driver* o qual se comunicara com a interface e sistema operacional implementando o protocolo. Inicialmente pretende-se fazer apenas a implementação do protocolo padrão sem a alteração de parâmetros que influenciem no seu comportamento devido ao tempo do projeto.

1.1 Objetivos

Este trabalho tem como objetivo a implementação de um protocolo de transmissão ponto-a-ponto sem fio utilizando um RTOS (Xenomai).

Como objetivos específicos tem-se:

- Implementar um protótipo para um protocolo MAC PTP em um kernel de tempo-real.
- Planejamento e execução de testes para avaliação do desempenho da implementação em RTOS.

1.2 Motivação

A motivação deste trabalho se dá no fato que protocolos de comunicação, principalmente protocolos de redes sem fio são extremamente dependentes do tempo e os possíveis ganhos em desempenho que uma implementação em um sistema operacionais em tempo real possa trazer, tendo em vista que seu foco principal está na precisão do tempo de resposta, evitando que o mesmo gaste tempo desnecessariamente com outras tarefas.

1.3 Organização do texto

O texto está organizado da seguinte forma: No Capítulo 2 é apresentado a fundamentação teórica onde serão discutidos conceitos e ferramentas que serão necessários para a implementação desse trabalho. Neste capitulo serão discutidos os conceitos sobre sistemas operacionais, sistemas operacionais em tempo real (RTOS),as diferenças entre *Hard Real Time* e *Soft Real Time*, o que são *Drivers* e como funcionam, o sistema operacional Xenomai, seus *kernels* e suas *Application Programming Interfaces* (API's), em seguida serão discutidos conceitos sob protocolos e em especifico os protocolos PPP e `IEEE802.11n`, e por fim o chipset `AR9271` para qual pretende se implementar o *driver*. No Capítulo 3 é apresentada a proposta, que falara sobre o escopo desse trabalho e por fim sera apresentada a metodologia, onde será discutido como será feita a implementação do *driver*, os testes e ferramentas que serão utilizados para averiguar o desempenho da implementação do *driver*.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas Operacionais

Segundo Maziero (2019) o sistema operacional é uma camada de software que opera entre o *hardware* e os programas aplicativos voltados ao usuário final. Trata-se de uma estrutura de *software* ampla, muitas vezes complexa, que incorpora aspectos de baixo nível (como *drivers* de dispositivos e gerência de memória física) e de alto nível (como programas utilitários e a própria interface gráfica). Os objetivos básicos de um sistema operacional podem ser sintetizados em duas palavras-chave: "abstração" e "gerência".

Por si só o sistema operacional não executa nenhuma tarefa útil porém ele fornece um ambiente onde outras aplicações possam vir a executar tarefas coordenando e controlando o hardware entre as diversas aplicações que possam estar sendo executadas. Assim sendo o sistema operacional gerencia recursos que as aplicações necessitam para a execução de uma determinada tarefa como tempo de processamento, espaço de memória, espaço para armazenamento de arquivos, dispositivos de entrada/saída (do inglês *Input/Output* (I/O)).

Uma visão ligeiramente diferente de um sistema operacional enfatiza a necessidade de controlar os vários dispositivos de I/O e programas de usuário para evitar erros e o uso indevido do computador. Uma definição mais comum é que o sistema operacional é um programa que está sempre executando no computador (chamado de *kernel*), todo o resto consistindo em programas aplicativos gerenciados pelo *kernel* (SILBERSCHATZ; GALVIN; GAGNE, 2001).

O sistema operacional tem como objetivo abstrair o hardware já que ele é o responsável por controlar e gerenciar o uso do mesmo, assim os programas executados no sistema operacional se comunicam com as sistema operacional e o sistema operacional se comunica com hardware (figura 1), isso possibilita que um mesmo programa rode em maquinas diferente, desde que se utilize o mesmo sistema operacional, evitando ter que-se reescrever.

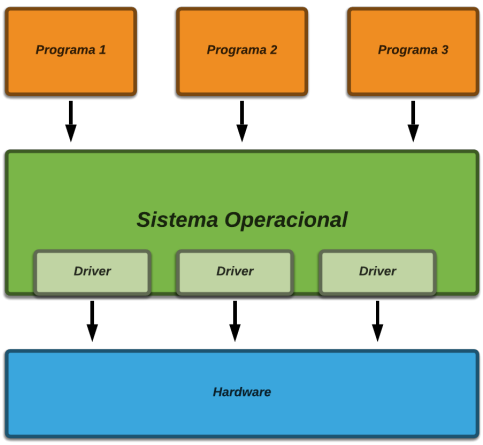

Figura 1 – Funcionamento do sistema operacional.

Exitem diferentes tipos sistemas operacionais, o quais se propõe a solucionar diferentes problema, eles podem ser classificados seguindo diversos parâmetros e aspectos como por exemplo tamanho velocidade,

suporte a recursos específicos, acesso à rede, etc. Discutiremos alguns desses tipos nas subseções a seguir.

2.1.1 Batch (lote)

Sistemas operacionais antigos executavam as aplicações em lotes, os programas eram colocados e uma fila com os dados e as informações necessárias para a execução, em seguida o sistema operacional agrupava os programas em lotes conforme o tipo de tarefa a ser executada conforme mostra a figura 2. Nesse tipo de sistema o processador recebe os programas e processa sem ter interação com o usuário tendo assim uma melhor utilização do sistema. Esse conceito ainda é aplicado em sistema onde a interação direta com o usuário não é necessária como por exemplo no caso de transações bancarias (MAZIERO, 2019).

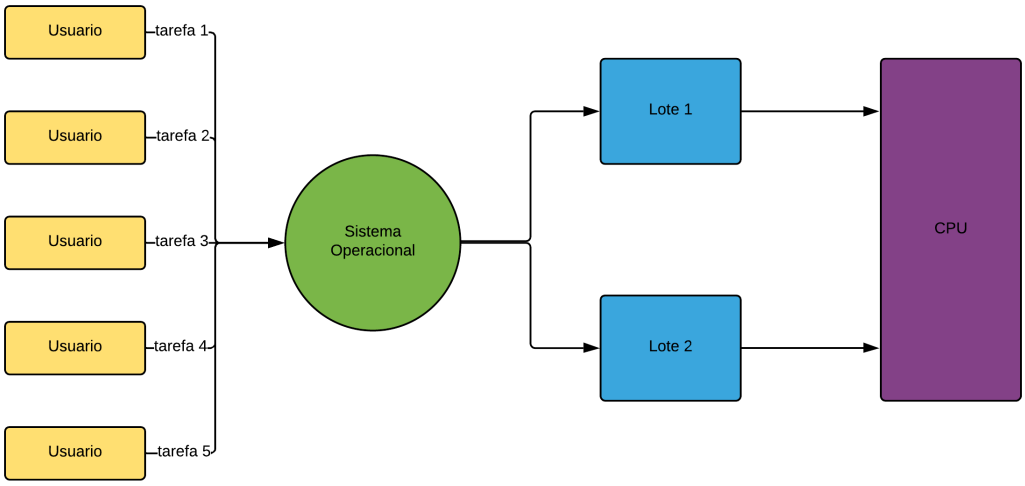

Figura 2 – Funcionamento do sistema operacional em lotes.

2.1.2 Tempo compartilhado

Esse tipo de sistema operacional permite que múltiplos usuários compartilhem um computador simultaneamente. O sistema da a impressão de ter um processamento dedicado a cada usuário alocando um curto período de processamento, com forme mostrado na figura 3, já que as tarefas desse tipo de sistema tendem a ser curtas, para cada aplicação e alternando rapidamente entre o diverso programas sendo executados. Sistema em tempo compartilhado conseguem isso fazendo escalonamento do uso da CPU e multiprogramação. Um processo geralmente é executado por um curtos período antes de encerrar ou necessitar interagir com o usuário, o sistema operacional de tempo compartilhado aproveita esse período de tempo em que o processo aguarda a interação com o usuário para executar outros processos evitando que a CPU fique inativa (SILBERSCHATZ; GALVIN; GAGNE, 2001).

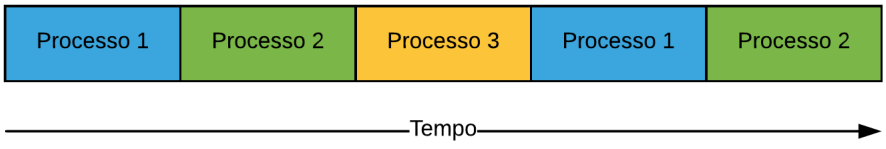

Figura 3 – Funcionamento do sistema operacional de tempo compartilhado.

2.1.3 Sistema paralelo

Até alguns anos atrás a eram sistema que tinham um único processador, ou seja possuíam apenas uma CPU. Porém equipamentos atuais contam com processadores de múltiplos núcleos, possuem múltiplas CPU's como é o caso por exemplo da série de processadores intel core. Tais sistemas têm mais de um processador em comunicação ativa, compartilhando o barramento, o *clock* e, as vezes, a memória e os dispositivos periféricos. Esses sistemas são chamados sistemas fortemente acoplado. Existem vários motivos para construir sistemas assim. Uma vantagem é a maior produção onde a aumentar o número de processadores, espera-se realizar mais trabalho em menos tempo (SILBERSCHATZ; GALVIN; GAGNE, 2001).

2.1.4 Embarcado

Sistema operacionais embarcados são construídos para operar em hardware com pouco recursos de processamento, armazenamento e energia (geralmente microcontroladores). Esse tipo de sistema operacional é geralmente utilizado em sistema de automação, controle automotivo, eletrodomésticos (como DVD, televisores, micro-ondas, etc.). Como esses sistemas operacionais são feitos para sistemas com poucos recursos seu tamanho tende a ser reduzido, e muitas vezes se apresentam como bibliotecas a serem adicionadas ao código do programa desenvolvido.

2.1.5 Em tempo real

Sistemas em tempo real são sistemas nos quais o tempo é essencial. Ao contrário da ideia usual, um sistema operacional de tempo real não precisa ser necessariamente ultrarrápido;sua característica essencial é ter um comportamento temporal previsível, ou seja, seu tempo de resposta deve ser previsível no melhor e no pior caso de operação. A estrutura interna de um sistema operacional de tempo real deve ser construída de forma a minimizar esperas e latências imprevisíveis, como tempos de acesso a disco e sincronizações excessivas. (MAZIERO, 2019). Esse tipo de sistema sera visto de maneira mais aprofundada na seção 2.2.

2.2 RTOS

Sistema operacionais em tempo real do inglês *Real Time Operational Systems* (RTOS) é um sistema operacional mais especializado onde o tempo de resposta é mais importante do que executar centenas de tarefas simultaneamente. O tempo de resposta não precisa necessariamente ser o mais rápido possível, como o nome pode sugerir, mas deve ser previsível, logo, *Real-Time* pode ser uma resposta de vários minutos ou nano segundos, dependendo da forma que seu sistema funciona. As tarefas do RTOS contam com tempo limite para serem executadas, então é comumente chamado de *Time Critical* (MAZIERO, 2019).

Para que se possa entender melhor os diferentes tipo de RTOS e métodos de agendamento e gerenciamento de recursos, se faz necessário que sejam definidos alguns termos de uso geral. Cada processo que foi agendado pelo sistema sera chamado de *job*. Um grupo de *jobs* que em conjunto provem alguma função ao sistema são chamados de *task*. O tempo de liberação (*release time*) de um *job* é o instante em que o mesmo se torna disponível para execução. o prazo de entrega (*deadline*), também chamado de prazo de entrega absoluto (*absolute deadline*), de um *job* será o instante de tempo em que o mesmo deve . O tempo de resposta é o tempo que leva do seu momento de liberação até o instante em que o *job*é completado. O tempo de resposta máximo é chamado de prazo de entrega relativo (*relative deadline*). Uma

constrição temporal é requerimento do comportamento de tempo do software é normalmente expresso pelo tempo de liberação e o tempo de entrega relativo ou absoluto. Um RTOS pode ser classificado

Em geral um sistema em tempo real pode ser classificado de acordo com sua constrição em dois tipos:

Hard Real Time

Um sistema é considerado *hard* se a falha e alcançar a sua constrição temporal é considerada uma falha critica. Um prazo de entrega desse tipo é imposto a um *job* por que o resultado produzido posteriormente ao prazo de entrega pode ter consequência desastrosos para o projeto. Esse tipo de requerimento necessita que todas as contrições temporais sejam validadas, essa inflexibilidade coloca muitas restrições na arquitetura de *hardware* e *software* utilizado além do planejamento e implementação desse tipo de aplicação. Portanto a implementação desse tipo de aplicação deve ser justificado. Como exemplos de sistemas *hard real time* podemos citar sistema onde falhas possam colocar vidas em risco: freios de trens, ajustes na posição de um avião, controle da refrigeração de uma usina termonuclear, etc.

Soft Real Time

Nesse sistema a perda de alguns prazos de entrega não apresentam nenhum problema sério ao sistema, apenas afetando sua performance a deixando pior conforme o numero de prazos de entrega perdidos. O desenvolvedor de um sistema desse tipo é raramente requerido que comprove rigorosamente que o sistema certamente irá alcançar a performance em tempo real. A validação de requerimentos menos rigorosa e constrições de tempo mais relaxada permitem que o desenvolvedor possa considerar outras métricas que sejam importantes para o sistema. Como exemplo de sistemas desse tipo temos transações online, *switches* telefônicos, jogos eletrônicos. Até mesmo sistemas de bolsas de ações, onde flutuações rápidas nos valores acontecem constantemente e a perda dos prazos implica na perda de dinheiro, ao contrario do que se espera são considerados *soft real time* isso por que se espera que o sistema o consiga acompanhar a mudança de preços. No entanto troca-se a ocasional perda de prazo pela disponibilidade do sistema e a maior quantidade usuários que o mesmo consegue atender (LIU; NARAYANAN; BAI, 2000).

2.3 Drivers

Programas em espaço de usuário não podem se comunicar diretamente com hardware devido ao fato do usuário não possuir certas permissões e a utilização de interrupções. Os *drivers* assumem esse papel fazendo comunicado entre o hardware e o restante do kernel e espaço de usuário. Os drivers são "caixas pretas"que fazem um *hardware* em especifico responder a uma *Application Programming Interface* (API); eles escondem os detalhes de como o dispositivo funciona. As ações do usuário são feitas através de um conjunto de chamadas padronizadas que são independentes de um *driver* em especifico, fazer o mapeamento dessas chamadas para um hardware em especifico é papel do *driver* (VENKATESWARAN, 2008).

2.3.1 Carácter

Um dispositivo de carácter pode ser acessado como um arquivo e o *driver*ira implementar esse comportamento. Esse tipo de *driver* implementa as chamadas de sistema *open, close, read* e *write*. O console de texto e as portas seriais são exemplos de dispositivos de caracteres. Diferentemente de um arquivo comum não se pode voltar para uma areá que já foi lida, esse tipo de *driver* é apenas um canal de informação e portanto a mesma só pode ser lida sequencialmente.

2.3.2 Bloco

Um dispositivo de bloco é um dispositivo que possui um sistema de arquivos, como um CD ou pendrive. Na maioria dos sistema Unix, o dispositivo de bloco só pode fazer operações de I/O que transfiram um ou bais blocos (geralmente 512 bytes ou uma potencia de dois maior). O Linux por sua vez não possui tal limitação e permite que as aplicações escrevam uma informação com um tamanho qualquer. Como resultado desse tratamento a única diferença entre dispositivos de blocos e carácter é como a informação é gerenciada internamente pelo *kernel* Linux.

2.3.3 Rede

Qualquer transação de rede é feita através de uma interface, o qual é um dispositivo que esta habilitado a trocar dados com um outro hospedeiro. Tipicamente, uma interface é um dispositivo de hardware, porém também pode ser puramente um dispositivo de software, como a interface *loopback*. Um interface de rede é responsável por enviar e receber pacotes de dados, sem saber como transações individuais se relacionam com os pacotes transmitidos. Muitas transmissões como TCP são orientadas a fluxo porém interfaces de rede são tipicamente orientadas a recebimento de pacotes. Por não ser orientada a fluxo uma interface de rede não é mapeável para o sistema de arquivos e embora Unix atribua um nome para a interface , como eth0, o mesmo não corresponde a uma entrada no sistema de arquivos(VENKATESWARAN, 2008).

2.4 Xenomai

O Xenomai foi criado em 2001 com o objetivo de facilita a migração de aplicações industriais vindas do ambiente proprietario para ambiente GNU/Linux, mantendo as garantias em tempo real. Para isso o Xenomai prove blocos genéricos para implementação de API em tempo real, conhecidos como *skins*, as quais conseguem imitar eficientemente API's proprietarias.

O Xenomai é um subsistema em tempo real que pode ser integrado com o *kernel* Linux para garantir tempos de respostas predizíveis para as aplicações. Em seu estado atual ele é baseado em abordagem de *kernel* duplo, com um pequeno *co-kernel* rodando simultaneamente com o *kernel* Linux no mesmo *hardware*. O Xenomai suporta a execução de programas em tempo real em espaço de usuário com proteção da *Memory Management Unit* (MMU) quando disponibilizado pelo *kernel* Linux, como um programa Linux comum. *Tasks* em tempo real são exclusivamente controladas pelo Cobalt durante o curso da operações de tempo critico para que as baixas latências sejam alcançadas na execução do código dentro do *kernel* Linux (KARIMYAGHMOUR; BEN-YOSSEF; GERUM, 2008).

2.4.1 APIs

APIs é um conjunto de funções que visão facilitar a integração de um *software* existentes a *softwares* que venham a ser desenvolvidos. A figura 4 mostra as API disponibilizadas pelo o RTOS Xenomai.

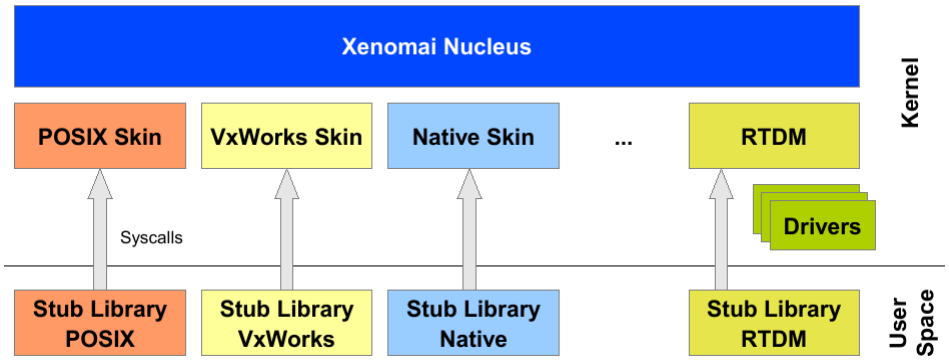

Figura 4 – API's do Xenomai (Fonte: (XENOMAI, 2019)).

2.4.1.1 POSIX

O Xenomai possui uma implementação da API *Portable Operating System Interface* (POSIX).POSIX é uma família de normas definidas pela IEEE para a manutenção de compatibilidade entre sistemas operacionais, sendo composta pela familia de padrões `IEEE 1003.x`. O POSIX define a API), juntamente com shells e interfaces utilitárias, para compatibilidade de software com variantes de Unix e outros sistemas operacionais.

2.4.1.2 *Real Time Driver Model* (RTDM)

O RTDM funciona como uma camada de abstração que conecta o processo de uma aplicação em tempo real ao serviço fornecido por um *driver*. Os *driver* podem ser classificados como:

Driver de Protocolos:

Esse tipo de *driver* expões uma interface de *socket*, e portanto se torna uma melhor opção para o gerenciamento de dispositivos orientados a mensagem com comunicação em tempo real. Uma biblioteca em espaço de usuário faz a interconexão entre as chamadas padrões de *socket* e as chamadas de sistema do Xenomai que repassam a informação para o *driver* que controla o protocolo solicitado.

Dispositivos Nomeados:

Esse tipo é parecido com o dispositivo de caracteres do modelo *driver* Linux, embora não tenha nenhuma equivalência com os modelos de *drivers* Linux vistos na seção 2.3 existindo somente na camada RTDM, sendo nomeados com rótulo arbitrário definido pelo *driver*. Nesse modelo assim como nos *drivers* de protocolo uma biblioteca em espaço de usuário faz a interconexão as chamadas de sistema do Xenomai e as chamadas padrão e sistema nesse caso são utilizadas as chamadas padrões de comunicação com as I/O que repassam a informação para o *driver* que corresponde ao nome passado inicialmente.

2.4.2 Kernels

O Xenomai atualmente (versão 3.x) utiliza uma arquitetura de *kernel* duplo baseado na camada de virtualização Adeos/I-pipe. O Xenomai encontra-se em diversas plataformas com baixo custo de desenvolvimento e manutenção. Essa abordagem tem como vantagens fundamentais:

- Ocorre um desacoplamento do Xenomai com o Linux, o que da mais liberdade aos desenvolvedor de selecionar o *kernel* utilizado como base, além de reduzir o impacto potencial que falhas e regressões no *kernel* possam ter no subsistema em tempo real.
- Protege as aplicações desenvolvidas para o Xenomai das aplicações comuns que não se comportem como esperado, devido as aplicações do Xenomai não serem controladas pelo mesmo *kernel* quando rodando em tempo real.
- Desenvolvedores para o Xenomai assim como desenvolvedores em sistemas embarcados conseguem rastrear problemas de tempo real, como por exemplo latências inesperadas, olhando as causas no espaço do *co-kernel*. Isso envolve inspecionar uma quantidade de código muito inferior do que se fosse usado o *kernel* padrão.
- *Co-kernels* são inerentemente leves, devido ao fato de eles dependerem do *kernel* hospedeiro prover os serviços que não são *time critical*, o que representa maior parte deles. Consequentemente o custo de implementar a garantia de tempo real no sistema é pago pelas aplicações que necessitam do mesmo. Ou seja o *kernel* Linux não tem que lidar com os problemas de tempo real, retirando a virtualização e a mascara de interrupção, que são operações baratas em termo de processamento.

Porém, esse modelo exclui todos os *drivers* e bibliotecas Linux que dependem do *kernel* padrão para a realização de operações em tempo real, já que eles não são servidos pelo *co-kernel*. Em inúmeras aplicações, esse requerimento pode causar problemas, particularmente quando a implementação envolve atividades que não sejam em tempo real mas sejam *time critical*, ou simplesmente por terem *drivers* comuns tempos de respostas amarrados conseguiriam suprir as necessidades do sistema a um custo menor de desenvolvimento (KARIMYAGHMOUR; BEN-YOSSEF; GERUM, 2008).

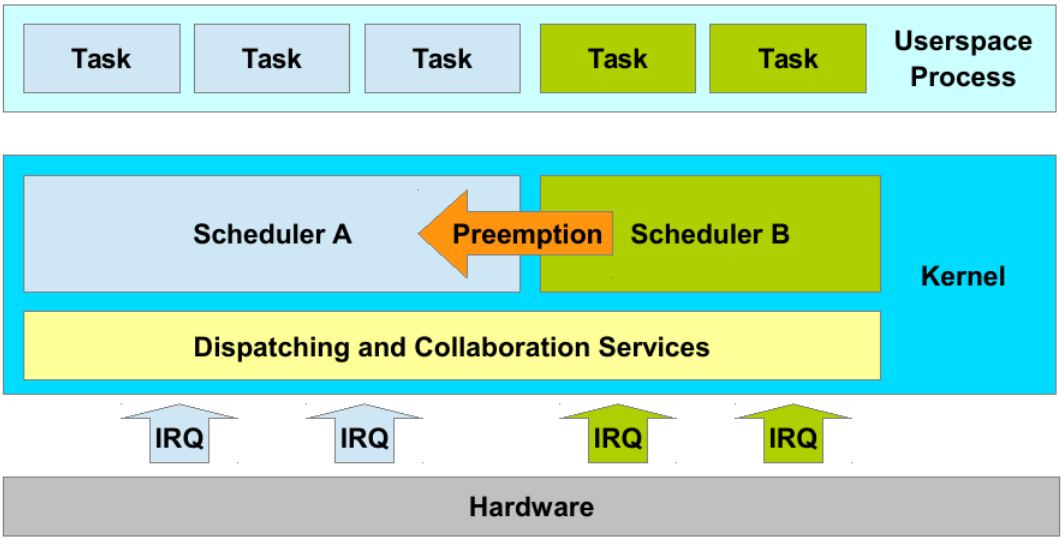

Figura 5 – Interação dos programas com o *kernel* Cobalt.

2.4.2.1 Linux

O Linux é baseado no sistema Unix os qual tipicamente possui um *kernel* monolítico, isso quer dizer que ele existe como uma única e grande imagem executável que é executada em um único espaço de endereço requerendo um sistema com uma unidade de gerenciamento de memória (do inglês MMU) paginada; este hardware permite que o sistema imponha proteção de memória e forneça um espaço de endereço virtual exclusivo para cada processo. Contudo o *kernel* Linux utiliza algumas ideias dos

microkernels como: um design modular, suporte *threads* no *kernel* e a capacidade de carregar arquivos binários (modulo) separadamente na imagem do kernel.

Devido ao fato do Linux não ser baseado em variante especifica do Unix, os desenvolvedores do *kernel* tem liberdade para escolher a solução que acham melhor para um determinado problema o que acaba levando a uma divergência entre o Unix clássico e o Linux. A exemplo de algumas dessas diferenças temos:

- O Linux tem suporte ao carregamento dinâmico de módulos.
- Possui suporte a multiprocessador simétrico do inglês *Symmetrical Multiprocessor* (SMP).
- O Linux não diferencia entre *threads* e processos comuns. Todos os processos sã tratados da mesma forma.
- Prove um modelo de dispositivo orientado a objetos, enventos *hot-plug* e um sistema de arquivos para dispositivos em espaço de usuário.
- Algumas características do Unix que os desenvolvedores do *kernel* consideram mal concebidas, como *STREAMS*, ou padrões que são impossíveis de se implementar de maneira eficiente são ignorados.

2.4.2.2 Cobalt

O *kernel* Cobalt complementa o *kernel* Linux no sistema de configuração de *kernel* duplo. O Cobalt se encarregara de todas as atividades de tempo critico, como lidar com as interrupções e agendamento de *threads* em tempo real. O *kenerl* Cobalt tem uma maior prioridade que todas as atividades nativas do *kernel* Linux.

Na configuração de *kernel* duplo, todas as API's de tempo real do Xenomai proveem interface com o *kernel* Cobalt, somente essas API's são consideradas capazes de reposta em tempo real, incluindo o subconjunto dede serviços POSIX 1003.1c (libcobalt) implementado pelo Xenomai, a figura 6 mostra a interação entre os programas e o *kernel* Cobalt (XENOMAI, 2019).

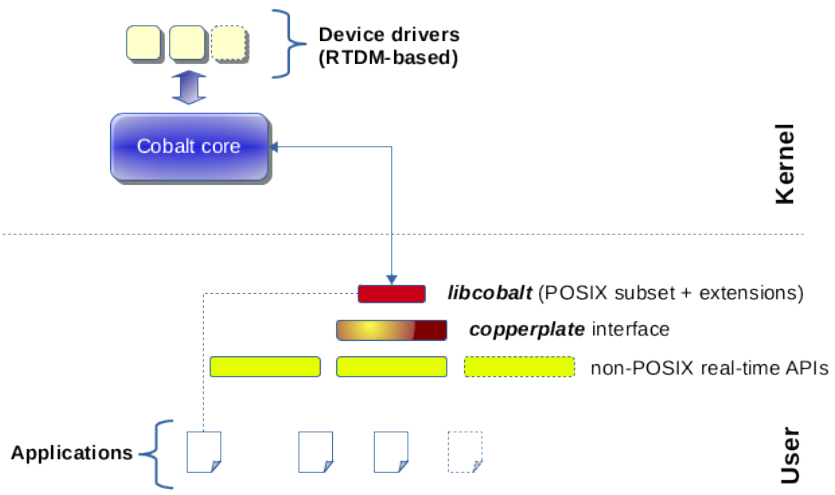

Figura 6 – Interação dos programas com o *kernel* Cobalt (Fonte: (XENOMAI, 2019)).

2.5 IEEE 802.11

O ponto principal da especificação da IEEE 802.11 é o MAC, pois é ele quem controla a a transmissão dos dados do usuário no ar. Ele prove as principais operações de enquadramento e as interações com a estrutura de rede cabeada. Diferentes camadas físicas podem prover velocidades de transmissão diferentes e todas elas devem ser interoperáveis. O IEEE 802.11 não diverge do antigo padrão IEEE 802 em nenhum modo modo extremo. O padrão IEEE 802.11 adapta com sucesso estilo de rede Ethernet para canais de radio. Assim como o Ethernet o IEEE 802.11 utiliza um método de acesso múltiplo por portadora (do inglês *carrier sense multiple access* (CSMA)) para acesso pra o meio de transmissão. Contudo as colisões desperdiçam capacidade de transmissão, então ao invés de detectar colisões (do inglês *Colision Detection* (CD) ou CSMA/CD) como no Ethernet o IEEE 802.11 evita colisões (do inglês *Caolision Avoidance* (CA) ou CSMA/CA). Assim como no Ethernet o IEEE 802.11 utiliza um método de acesso distribuido sem controle centralizado. Todas as estações IEEE 802.11 utilizam o mesmo método para ganhar acesso ao meio.

2.5.0.1 Acesso ao meio

O acesso ao meio sem fio é controlado pelas funções de coordenação. O acesso tipo Ethernet CSM/CA é provido pela função de coordenação distribuída (do inglês *Distributed Coordination Function* (DFC)). Se um serviço sem disputa é requerido, ele pode ser provido pela função de coordenação pontual (do inglês *Point Coordination Function* (PFC)), o qual é construído em cima do DFC, esses serviços são providos somente na rede de infraestrutura.

O DFC é base do mecanismo de acesso do CSMA/CA. Assim como no Ethernet, ele primeiro confirma se o canal de radio esta livre antes de transmiti. Para evitar colisões, estações usam um *backoff* aleatório após cada quadro, com o primeiro a transmitir ocupando o canal. Em alguns casos o DFC pode usar o método CTS/RTS para diminuir a possibilidade de colisões.

O PFC prove serviços sem disputa. Estações especiai chamadas coordenadores de ponto são usadas para que o meio seja provido sem disputa. Coordenadores de ponto en encontram nos pontos de acesso, portanto o PFC fica restrito a rede de infraestrutura. Para conseguir prioridade sobre os serviços com disputa o PFC permite que a estações transmitam quadros após um curto período, no entanto o PFC não é amplamente implementado.

2.5.1 Espaçamento entre quadros

O tempo de espaçamento entre quadros tem uma grande importância em coordenar o acesso para o meio de transmissão. A família de protocolos IEEE 802.11 usa quatro espaçamentos entre quadros diferentes, três dos quais são utilizados para determinar o meio de acesso. A variação desses espaçamentos determina os diferentes níveis de prioridade para os diferentes tipos de trafico. A lógica por trás disso é que trafego com alta prioridade não tem de esperar contanto que o meio de transmissão esteja livre, por tanto se há um trafego de alta prioridade esperando, ele ira ocupar a rede antes de um trafego com prioridade baixa tentar transmitir. Devido as diferente taxas de dados causado pelas diferentes camadas físicas o espaçamento entre quadros é um tempo fixo. Diferentes camadas físicas no entanto, podem especificar diferentes tempos de espaçamento. Como dito anteriormente tem-se quatro tipos de espaçamentos diferentes no IEEE 802.11. são eles:

espaçamento curto entre quadros

O *Short Interf Frame Space* (SIFS) é utilizado para transmissões com alta prioridade, como quadros RTS/CTS. Transmissões com alta prioridade começam após o SIFS ter acabado. Assim que essas transmissões inicia, o meio se torna ocupado, portanto quadros transmitidos após o SIFS tem prioridade sobre os demais.

espaçamento de PFC entre quadros

O *PFC Inter Frame Space* (PIFS) é usado pelo PFC durante as operações livre de disputa. Estações com dados para transmitir no período livre de disputa podem transmitir após o PIFS ter acabado.

espaçamento de DFC entre quadros

O *DFC Inter Frame Space* (DIFS) é a média miníma de tempo inativo para serviços baseados em disputa. Estações podem ter acesso a meio se ele estiver livre por um período maior que DIFS.

espaçamento extendido entre quadros

O *Extended Inter Frame Space* (EIFS) não é um intervalo fixo, ele é usado somente quando há um erro na transmissão do quadro (GAST, 2005).

espaçamento reduzido entre quadros

O *Reduced Inter Frame Space* (RIFS) é um espaçamento introduzido pelo `IEEE 802.11n`, sua função é equivalente ao SIFS, de fato seu objetivo é ser utilizado ao invés do SIFS como um método par aumentar a eficiência (GAST, 2012).

2.5.2 Prioridade

Operações atômicas começão como transmissões regulares, onde elas precisam esperar pelo DIFS antes de começarem, porém após essa etapa operações atômica passam a utilizar o SIFS em seu lugar. Isso quer dizer que após a primeira parte todas as outras partes irão ocupar o meio antes que outro tipo de quadro possa ser transmitido. Ou seja utilizando o SIFS e o **NAV!** (**NAV!**), as estações conseguem ocupar o meio pelo tempo que for necessário.

2.5.3 Backoff

Após uma transmissão de quadro ser completada e o DIFS tiver passado, estações podem transmitir dados com disputa. Um período chamado de janela de contenção segue após o DIFS. Esta janela é dividida em *slots*. O primeiro é o tamanho o qual depende do meio, camadas físicas mais rápidas utilizam *slot* de tempo menor. Estações pegam um *slot* aleatório e esperam por ele antes de tentar acessar o meio. Quando varias estações estão tentando transmitir a estação que pegou o primeiro *slot*, aquele com o menor numero aletório, vence. Esse número é pego de uma faixa maior cada vez que a transmissão falha. Quando a janela de contenção chega ao seu tamanho máximo, ela permanece nele até que possa ser reinicializada. Permitir longas janelas de contenção quando diversas estações estão competindo para ganhar acesso ao meio mantém os algorítimos MAC estáveis mesmo sobre a carga máxima. A janela de contenção é reinicializada para o seu tamanho mínimo quando quadros são transmitido com sucesso, ou o numero de tentativa associado é alcançado e o quadro é descartado (GAST, 2005).

3 PROPOSTA

Para este trabalho propõe a implementação de um *driver* para o protocolo protocolo sem fio PTP baseado no padrão `IEEE 802.11n`. O protocolo proposto ira funcionar no modelo mestre-escravo, nesse modelo o mestre enviara periodicamente uma mensagem controle (chamada de "*poll*"), a qual tem como objetivo autorizar que o escravo transmita informação por um determinado período de tempo. terminado esse período o escravo devera entrar em silencio e o mestre voltara a transmitir. Para implementação desse protocolo pretende-se utilizar o RTOS Xenomai e sua API RTDM para obter uma temporização precisa das atividades do protocolo, no caso do protocolo proposto tais atividades consistem da transmissão mensagem de *poll* transmitida pelo mestre e a duração do período em que o escravo está autorizado a transmitir (período de *poll*). Pra que tais alterações do protocolo `IEEE 802.11` sejam possíveis o *driver* ira se basear em um *chipset* que possibilite modificar valores de parâmetros do MAC `IEEE 802.11`, como o cwmin, *Inter Frame Space* (IFS) e desativamento das mensagens de ACK, por tal motivo optou-se pelo uso de um *chipset* Atheros da família `AR9XXX` (ex: `AR9271` ou `AR9342`). Por fim pretende-se demonstrar o funcionamento do protocolo PTP desenvolvido utilizando o *Customer Premise Equipment* (CPE) Intelbras `WOM 5000`, que possui o *chipset* `AR9342`. Porém não foi investigada a portabilidade do Xenomai para essa plataforma, por tanto tal demonstração dependera se o mesmo será possível.

3.1 Metodologia

Este procedimento será dividido em três etapas: sendo elas a especificação do protocolo e sua modelagem como *driver* RTDM para o RTOS Xenomai, a implementação do *driver* para o protocolo PTP baseado no `IEEE 802.11n` utilizando a API RTDM e por fim os testes utilizando o wireshark e iperf afim de averiguar o funcionamento da implementação feita e corrigir os erros encontrados.

Inicialmente será feito a especificação do protocolo com o intuito de definir o funcionamento e as características do protocolo a ser implementado como por exemplo troca de mensagens e tempos entre quadros. Em seguida será feito a modelagem do protocolo seguindo o modelo RTDM, essa tem como objetivo descrever como o procolo irá se comportar com base na API essa etapa será critica pois todo o desenvolvimento será feito com base nela.

Para o desenvolvimento será feita a implementação do protocolo PTP baseado no `IEEE 802.11n` com base no modelo feito para a API na etapa anterior. O *driver* desenvolvido deverá se comunicar com um hardware especifico, para tal se optou pelo família `AR9XXX` o qual possui suporte para o padrão utilizado além de ser bem documentada.

Para fim de testes pretende-se usar a ferramenta *iperf* e o *wireshark*. O *iperf* funciona como uma aplicação cliente servidor, a qual gera um tráfico de dados conforme os parâmetros passados para o servidor. Dessa forma é possível usar essa aplicação de forma a simular uma carga no canal de comunicação ponto-a-ponto e averiguar o desempenho do sistema com uma implementação em um sistema Linux normal e no *software* desenvolvida no Xenomai é possível averiguar se implementação esta obedecendo o padrão do protocolo ou se houve algum problema durante a implementação que está ocasionando perdas para fim de testes pretende se usar a princípio o cenário mostrado na figura 7 onde teremos um link ponto-a-ponto sem fio entre dois computadores, para testes inicialmente pretende se testar a taxa de vazão de *upload* e *download* com e sem trafego no sentido contrário de comunicação. O *wireshark* por sua vez será utilizado a fim de capturar e analisar os pacotes a fim de averiguar se a implementação está de acordo com o protocolo `IEEE 802.11n` ou se algum problema que deve ser corrigido. além da latência para qual

será usado o *ping* e o tempo para estabelecimento da conexão, caso haja necessidade ou perceba que são necessárias novas informações poderão ser feitos outros testes a fim de se obter os dados desejados.

Figura 7 – Cenário de teste.

3.2 Cronograma

Pretende-se seguir o cronograma mostrado na tabela

Atividade	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
Modelagem do protocolo	X	X				
Desenvolvimento do Protocolo			X	X	X	
Testes					X	X
Escrita do Documento					X	X

REFERÊNCIAS

GAST, M. *802.11 wireless networks: the definitive guide*. [S.l.]: "O'Reilly Media, Inc.", 2005. Citado 2 vezes nas páginas 19 e 30.

GAST, M. *802.11 n: a survival guide*. [S.l.]: "O'Reilly Media, Inc.", 2012. Citado na página 30.

KARIMYAGHMOUR, J. M.; BEN-YOSSEF, G.; GERUM, P. *Building Embedded Linux Systems, 2E*. [S.l.]: O'Reilly Media, 2008. Citado 2 vezes nas páginas 25 e 27.

LIU, F.; NARAYANAN, A.; BAI, Q. Real-time systems. Citeseer, 2000. Citado na página 24.

MAZIERO, C. *Sistemas Operacionais: Conceitos e Mecanismos*. [S.l.: s.n.], 2019. Citado 4 vezes nas páginas 19, 21, 22 e 23.

RUBINSTEIN, M. G.; REZENDE, J. F. Qualidade de serviço em redes 802.11. *XX Simpósio Brasileiro de Redes de Computadores (SBRC2002)*, p. 26, 2002. Citado na página 19.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Sistemas operacionais: conceitos e aplicações*. [S.l.]: Campus, 2001. Citado 3 vezes nas páginas 21, 22 e 23.

VENKATESWARAN, S. *Essential Linux Device Drivers*. First. Upper Saddle River, NJ, USA: Prentice Hall Press, 2008. ISBN 9780132396554. Citado 2 vezes nas páginas 24 e 25.

XENOMAI. *Home*. 2019. Disponível em: <https://www.xenomai.org/>. Citado 4 vezes nas páginas 9, 19, 26 e 28.