UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Acahuã dos Santos Fialho

Sistema tempo-real embarcado utilizando o kit i.MX25

Porto Alegre
2013

Acahuã dos Santos Fialho

Sistema tempo-real embarcado utilizando o kit i.MX25

Trabalho de Diplomação apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Marcelo Götz

Porto Alegre

2013

Acahuã dos Santos Fialho

Sistema tempo-real embarcado utilizando o kit i.MX25

Este Trabalho de Diplomação foi julgado adequado para a obtenção dos créditos da Disciplina Projeto de Diplomação do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo orientador e pela banca examinadora.

Orientador:_____
Prof Dr. Marcelo Gotz,
Doutor pela Universität Paderborn – Paderborn, Alemanha

Banca Examinadora

Prof Dr. Marcelo Götz,
Doutor pela Universität Paderborn – Paderborn, Alemanha

Prof Dr. Walter Fetter Lages
Doutor pelo Instituto Tecnológico de Aeronáutica, ITA – São José dos Campos, Brasil

Prof Dr. Valner João Brusamarello
Doutor pela Universidade Federal de Santa Catarina, UFSC, Santa Catarina, Brasil

Chefe do DELET: _____
Prof. Dr. Altamiro Amadeu Susin

Porto Alegre, julho de 2013

Dedico este trabalho a meus pais, João Batista da Cunha Fialho e Simone Postiglione dos Santos, que sempre me apoiaram e especialmente durante o período do meu Curso de Graduação estiveram ao meu lado.

AGRADECIMENTOS

Agradeço ao Prof. Marcelo Götz, pelo seu apoio e orientação para a realização deste projeto.

Agradeço ao colega Rene Augusto Benvenuti e ao colega Ricardo Coelho, pelo auxílio no entendimento do trabalho.

Agradeço aos meus amigos e familiares, por estarem presentes nos momentos bons e nos momentos ruins, e por me darem a força necesária para a conclusão deste trabalho.

RESUMO

Este trabalho tem por finalidade atualizar a versão do sistema operacional da placa imx.25pdk, com foco no melhoramento do determinismo temporal para sistemas em tempo real. Durante esse trabalho, serão visto conceitos sobre sistemas embarcados, sistema de arquivos, sistemas em tempo real, e Linux. Será mostrada a solução encontrada para melhorar a performance da placa em questão, bem como a criação de testes para comprovar a diferença no tempo de latência da mesma. Todos os detalhes de instalação, a modificação nos drivers, entre outras informações práticas para a repetição deste trabalho serão expostos em forma de tutorial nos apêndices deste trabalho.

Palavras-chave: Sistema Embarcado em Tempo Real. Kernel. LTIB. CyclistTest

ABSTRACT

This study aims to upgrade the operating system version of the card imx.25pdk, focusing on temporal determinism improvement for real-time systems. During this work, it will be seen concepts of embedded systems, file system, real-time systems, and Linux. It will be shown the solution to improve the card performance, as well as the creation of tests to demonstrate this improvement. All details of installation, modification of the drivers, and other practical information for the repetition of this work will be exhibited in the form of tutorial in the appendix of this work.

Keywords: Real-Time Embedded System. Kernel. LTIB.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	Application Programming Interface
ATK	Advance Tool Kit
CACSDS	Computer Aided Control System Design Software
LTIB	Linux Target Image Builder
LTS	Long Term Stable
NFS	Network File System
JFFS2	Journalling Flash File System version 2
RT	Real Time
FIFO	First In First Out
RR	Round Robin
TFTP	Tivial File Transfer Protocol

Sumário

1. INTRODUÇÃO

1.1 Visão Geral e Motivação

Buscando uma vida mais prática e confortável, o homem moderno tem tornado suas ferramentas diárias cada vez mais inteligentes, fazendo com que elas sejam cada vez mais autônomas e mais fáceis de serem utilizadas. Para tal, o uso de microprocessadores é indispensável, como por exemplo, em *tablets,* televisores, automóveis, aparelhos reprodutores de arquivos mp3, entre outros.

Porém, este fenômeno não se resume a uma demanda por uma maior quantidade de microprocessadores; ele consiste, também, em uma exigência de melhor desempenho dos mesmos. Tendências como a convergência tecnológica onde um equipamento realiza a função de vários outros, conforme Figura 1, além da portabilidade e autonomia, exigem melhores capacidades de processamento com pouco consumo de energia.

Figura 1 - Convergência tecnológica

Fonte – [1]

Os softwares necessários para satisfazerem tais exigências acabaram se tornando complexos demais para que sejam realizados em linguagens de programação quase ao nível de máquina (assembler, por exemplo), como era feito nos primórdios dos sistemas embarcados. Sendo assim, é cada vez mais frequente

o uso de sistemas operacionais que facilitam o desenvolvimento e a execução de aplicações de uma maneira estável e consistente sem que o usuário ou o desenvolvedor tenham que conhecer todos os detalhes do hardware.

Porém, como o desenvolvedor não tem acesso a todos os recursos do sistema, não há como prever quando e em quanto tempo uma tarefa será executada, sendo que tal imprevisibilidade é incompatível com alguns tipos de sistema. Tais sistemas, conhecidos como sistemas em tempo real, possuem exigências temporais sob a aplicação, e para tratá-los, há a possibilidade de se usar os sistemas operacionais em tempo real, conhecidos como RTOS (*Real-Time Operating System*).

1.2 Objetivos e Metodologia

Este trabalho tem como objetivo atualizar a versão do sistema operacional, focando nos aspectos em tempo-real no kit i.mx25pdk. Para tal, será necessário configurar a nova atualização do sistema operacional, e realizar modificações no mesmo para poder adaptá-lo ao kit imx25pdk.

Além disso, será instalado um aplicativo chamado de *cyclictest*, desenvolvido com o intuito de medir a qualidade do sistema operacional para tratar tarefas em tempo real.

Usando esse aplicativo, será desenvolvida uma rotina de testes capazes de verificar a performance do sistema e compará-lo com o desempenho do sistema anterior. Também será medida a diferença de desempenho entre os métodos de escalonamento presentes no sistema escolhido.

Finalmente, uma análise dos resultados será feita, indicando se é vantajosa ou não a atualização no sistema operacional.

1.3 Organização deste Trabalho

Os capítulos seguintes desta obra serão estruturados da seguinte maneira:

- **Contexto**: será explicado brevemente o que será feito neste trabalho, bem como sua relação com o trabalho realizado anteriormente na imx25pdk;
- **Base Teórica**: serão mostrados, de forma mais geral, conceitos e teorias importantes para o entendimento e para a implementação deste projeto.
- **Metodologia e Materiais**: será mostrado como a teoria se junta com a prática e também serão dadas explicações sobre o hardware e sobre os softwares envolvidos.
- **Testes e Resultados**: será explicada a metodologia de testes escolhida, e será feita uma análise dos resultados encontrados.
- **Conclusão**: resume as ideias discutidas e apresenta as conclusões tomadas acerca deste projeto. Também indica possíveis caminhos para os próximos trabalhos com a imx25pdk.
- **Apêndices:** escritos em forma de tutorial, eles possuem as informações práticas necessárias para a repetição deste projeto.

2. CONTEXTO

Este trabalho será realizado utilizando um kit de desenvolvimento chamado de i.mx25pdk, fornecido pela freescale. Este kit é um *Computer Aided Control System Design Software* (CACSDS), um tipo de equipamento que tem sido cada vez mais utilizado atualmente, pela facilidade que é desenvolver aplicações usando ele, bem como pela grande gama de possibilidades de aplicação. Ele pode ser visto na Figura 1.

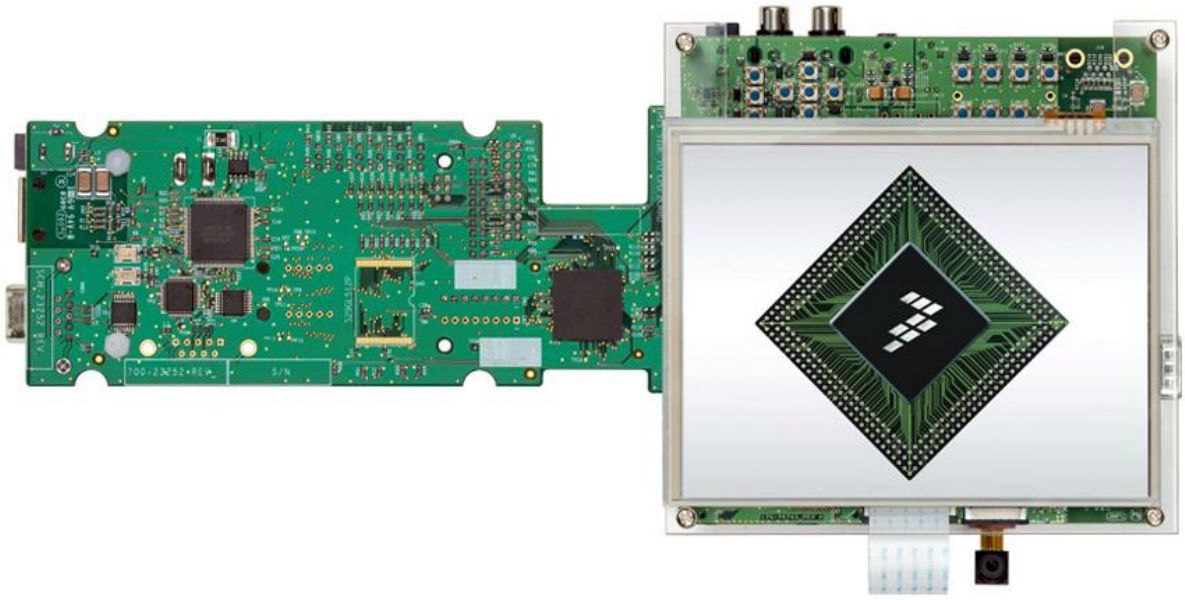

Figura 2 - A placa i.mx25pdk

Fonte : [2]

De acordo com [15], a atualização no sistema operacional, que possua um pacote em tempo real mais adequado, acarretaria num melhor desempenho da placa para sistemas em tempo real. Desta forma, a primeira parte do trabalho foi uma busca por alguma forma de trocar o sistema operacional da placa em questão, um trabalho árduo, pois esta nova versão do sistema operacional não é fornecida, nem tem a sua compatibilidade assegurada pela freescale, empresa criadora da placa.

Primeiramente, foi pesquisado sobre o sistema operacional Android, que está sendo muito usado atualmente. Nós fóruns da freescale, os poucos relatos encontrados afirmavam que a placa ficava demasiadamente lenta devido ao grande consumo de memória RAM por parte do sistema operacional Android. Desta forma,

optou-se por utilizar uma versão mais atualizada do sistema operacional Linux, que possua um pacote tempo real.

3. BASE TEÓRICA

3.1 Sistema Operacional

Em várias situações, é necessário que haja diversos processos diferentes rodando no mesmo hardware, o que adiciona a complexidade de compartilhar os recursos (como o uso da(s) memória(s), do processador, dos periféricos, etc.) entre tais processos. Desta forma, para diminuir a complexidade do desenvolvimento de uma aplicação, a responsabilidade em gerenciar o compartilhamento desses recursos é retirada dos processos e passa a ser responsabilidade do sistema operacional.

Outra questão importante é a independência do hardware em relação às aplicações. Se para cada modificação ou atualização do hardware, todas as aplicações tivessem que ser atualizadas, o volume de trabalho seria proibitivamente grande. De tal forma, surge a necessidade de uma camada intermediária entre ambos, sendo esta fornecida pelo sistema operacional.

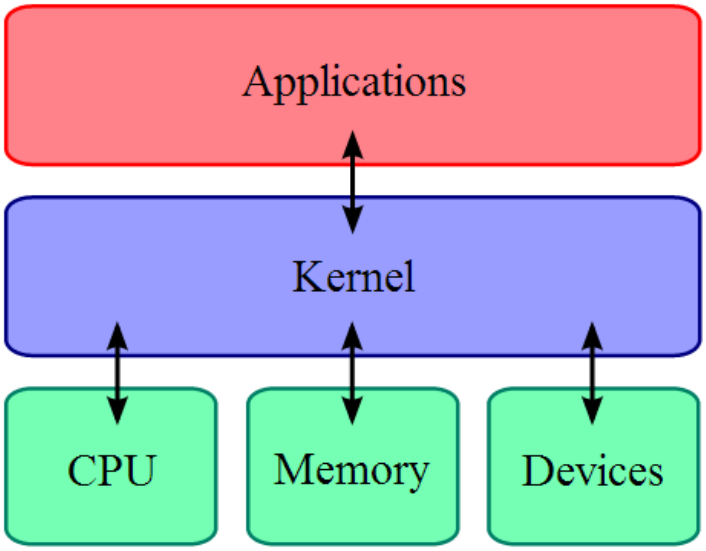

Figura 3 - Exemplo de arquitetura para um sistema operacional Linux

Fonte - [3]

3.2 Sistemas Embarcados em Tempo Real

3.2.1 Uma Visão Geral

Durante esse projeto, serão utilizados diversos conceitos de sistemas embarcados em tempo real, sendo então inevitável uma discussão teórica sobre as peculiaridades desse tipo de sistema. Primeiramente, o que define um sistema em tempo real, e quais são suas diferenças técnicas para com um sistema normal? Como lidar com tais diferenças técnicas?

Os sistemas embarcados em tempo real são sistemas que tratam tarefas com propriedades particulares. Uma tarefa tempo real tem como peculiaridade a existência dessas três seguintes propriedades:

- Um tempo inicial: um tempo a partir do qual a tarefa pode começar a ser executada;
- Um tempo final (*deadline*): a tarefa deve terminar a execução antes desse tempo;
- Uma ordem de dependências: dentro dessas restrições de tempo, também deve haver casos em que deve ser respeitada uma ordem de execução. Por exemplo, Tarefa B deve ser executada antes da Tarefa C e depois da Tarefa A.

Respeitando ao máximo o deadline, e a ordem de dependências, é necessário alocar as tarefas para que elas possam ser executadas pelo sistema. Alguns sistemas, conhecidos como *hard real time*, exigem o total respeito dessas propriedades, outros sistemas, conhecidos como *soft real time*, possuem certa tolerância quanto ao desrespeito a essas propriedades.

Por exemplo, o controlador de temperatura de uma centrífuga numa central hidroelétrica deve sempre respeitar suas restrições temporais, sendo de tal forma, um sistema crítico. Porém, um videoplayer pode permitir algumas falhas, que mesmo afetando sua performance, como por exemplo, vídeo não sincronizado com o áudio, ainda sim não causam uma falha com sérias consequências no sistema.

Em alguns sistemas, também é observado o tempo de execução, através de uma previsão do tempo máximo de execução da tarefa, chamado de WCET (*Worst Case Execution Time*).

Além disso, as tarefas podem ser periódicas, que ocorrem com uma frequência conhecida, ou podem ser aperiódicas. Embora tarefas aperiódicas ocorram de maneira não previsível, como o usuário realizando cliques no mouse, os sistemas embarcados em tempo real prezam pela previsibilidade, então essas tarefas também são tratadas de uma maneira que a sua execução se dê de uma maneira previsível.

3.2.2 O Escalonador

Para responder as propriedades temporais citadas anteriormente, é necessário realizar a alocação dessas tarefas, e a parte do sistema encarregada desta função é chamada de Escalonador. O Escalonador verifica e modifica o estado das tarefas, sendo que as tarefas podem estar nos seguintes estados: em execução, em espera ou bloqueada.

Uma característica importante a ser notada é o tempo de latência, que é o tempo entre a interrupção de uma tarefa, e o tempo de execução de outra.

Para tal, o Escalonador usa uma política de escalonamento, sendo que ela possui as seguintes características:

- Preemptiva ou não preemptiva.
- Estático ou dinâmico.
- Completo ou não completo.

No estático, a ordem de execução é definida durante a compilação, e no dinâmico a ordem de execução é definida durante a execução. Em razão de realizarem mais cálculos a cada iteração, o Escalonador com política de escalonamento dinâmica apresenta um maior tempo de latência, em geral.

Escalonador preemptivo significa que uma tarefa pode ser interrompida no meio de sua execução (o que depende da tarefa em si), e não preemptivo trata-se do contrário.

Escalonador completo significa que, dado um conjunto de tarefas, e um dado processador, se houver uma ou mais possibilidade de alocar as tarefas de tal forma que todas sejam executadas no tempo devido, então a política de escalonamento completa irá encontrar pelo menos uma maneira.

Neste trabalho, serão analisadas duas políticas de escalonamento: FIFO (First In First Out), e RR (Round-Robin).

FIFO é não preemptivo, estático, e não completo. Nele, a primeira tarefa que chegar é a primeira tarefa a ser executada. Um exemplo de escalonamento FIFO é mostrado na Figura 4.

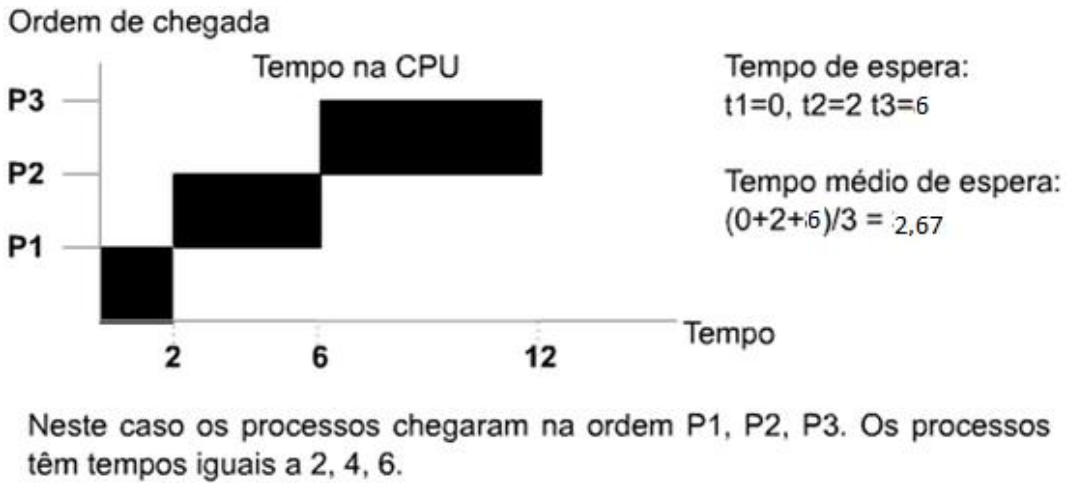

Figura 4 - Exemplo de um escalonamento FIFO

Fonte – [4]

RR é preemptivo, estático e não completo. Ele atribui frações de tempo para cada processo em partes iguais e de forma circular. Um exemplo de escalonamento RR é mostrado na Figura 5.

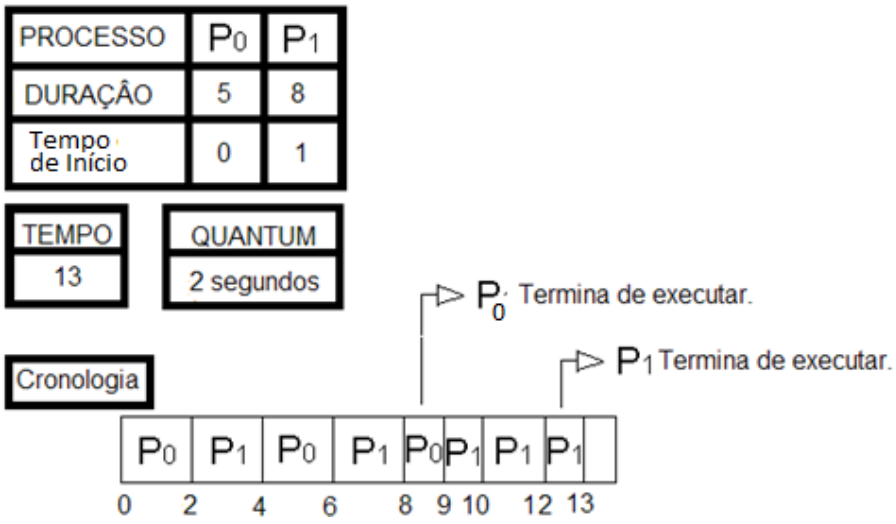

Figura 5 - Exemplo de um escalonamento Round-Robin

Fonte –[5]

3.3 Linux

Linux é um sistema operacional desenvolvido sob a licença GPL (*General Public License*). Sendo assim, pode-se ter acesso ao código e modificá-lo para poder melhor adaptá-lo as necessidades da aplicação em questão. O fato de ser gratuito, e de dar essa liberdade de ação, é muito atraente para fins de projetos acadêmicos. No entanto, certos problemas como código não funcional, erros de configuração, e documentação ausente ou insuficiente são dificuldades muito comuns de serem encontradas quando se trabalha com Linux.

A Figura 6 mostra uma arquitetura simplificada do Linux embarcado, destacando seus principais componentes.

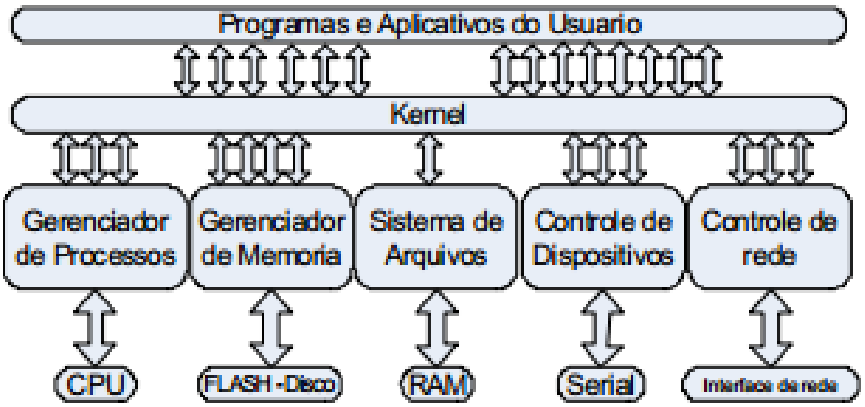

Figura 6 - Arquitetura Kernel

Da Figura 6, pode-se destacar o kernel, conhecido como núcleo do sistema operacional Linux. Ele é que é responsável por abstrair o funcionamento do hardware, por controlar a divisão de recursos entre os processos, entre outras funções.

De acordo com [12], um sistema de arquivos é um conjunto de estruturas lógicas que permite o sistema operacional controlar o acesso a um dispositivo de armazenamento como disco rígido, pen drive, cd-room, etc. É através do sistema de arquivos que os arquivos são criados, armazenados e organizados, sendo que, a maneira com que eles são guardados e manipulados varia de acordo com o tipo de sistema de arquivos escolhido. Há uma grande variedade de sistemas de arquivos disponíveis, e cada um deles tem diferentes vantagens e desvantagens. Neste trabalho serão analisados dois tipos de sistemas de arquivos: NFS e JFFS2.

O gerenciador de memória utiliza um esquema de memória virtual, de maneira que o endereço visto pelos programas de usuários não correspondem diretamente aos endereços em hardware. Ao gerenciador de processos, cabe a criação, a finalização, e o escalonamento dos processos.

O controle de dispositivos e o controle de rede são feitos por drivers de dispositivos, o que permite uma independência maior entre o hardware e o kernel. Os drivers são implementados em forma de módulos, sendo que eles podem ser dinamicamente carregados ou descarregados quando necessário. Os módulos não são necessariamente drivers, podendo ser a implementação de funções internas do kernel, por exemplo.

3.4 Compilação Cruzada

De acordo com [11], compilação cruzada (*cross compiling*) é compilar um arquivo para gerar um código que seja executável em uma plataforma diferente da qual a compilação cruzada está acontecendo. Tal método é útil, pois o computador pode ser usado para desenvolver aplicações para o sistema embarcado. Isso é importante tendo em vista que o computador possui muito mais recursos de depuração, além de ter capacidade de processamento suficiente para rodar um compilador (o que não é o caso de todo o sistema embarcado).

Para fazer a compilação cruzada, usa-se uma *toolchain*, que é um conjunto de programas que são utilizados no processo de compilação. Usando esses diferentes programas, são realizadas todas as etapas da compilação, tal como pode ser visto na Figura 7.

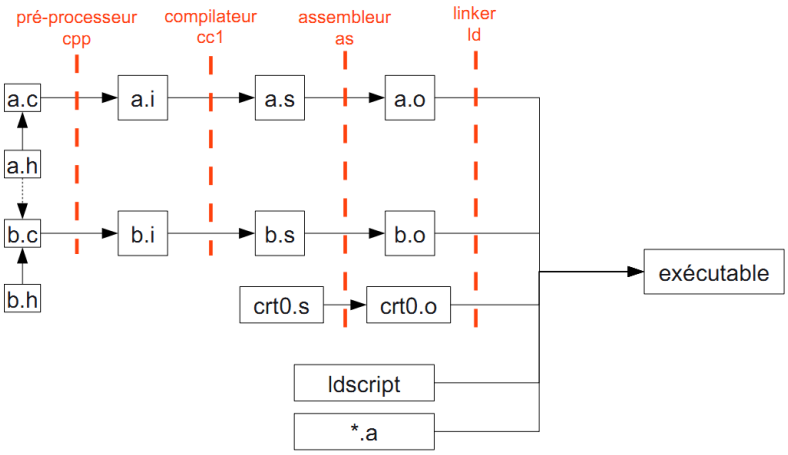

Figura 7 - A compilação

Fonte – [6]

Na pré-compilação são interpretadas as diretivas de compilação, como, por exemplo, a inclusão de bibliotecas (através da interpretação de #include), a supressão dos comentários, entre outras. Na compilação, esse texto pré-compilado é traduzido para a linguagem assembler de uma determinada arquitetura (arm, no caso deste projeto). Na parte de *assembling*, a linguagem assembly é interpretada e transformada em linguagem de máquina, gerando um programa-objeto. Finalmente, a parte de linkedição consiste em juntar os programas-objeto e bibliotecas necessárias para a criação de um arquivo que possa ser executado pela máquina.

4. METODOLOGIA E MATERIAIS

4.1 Sistema Operacional

4.1.1 Linux

Linux possui uma série de versões diferentes, sendo elas distribuídas no site www.kernel.org. A versão de Linux foi escolhida levando-se em conta três critérios:

- Uma versão de longo prazo, desta forma a versão está sujeita a receber atualizações, com novas funcionalidades, e melhoria de desempenho. Disto, pode-se notar pela Figura 8 que somente a 3.0, a 3.2, a 3.4 e a 3.9 se encaixam no perfil.
- A presença do pacote real time Preempt-RT, tendo em vista que este trabalho envolve sistemas em tempo real. Como se pode ver em https://www.kernel.org/pub/linux/kernel/projects/rt/, a versão 3.9 não atende a este requisito.
- Uma versão que tenha sido adaptada para o kit i.mx25pdk, tendo um arquivo de configuração próprio para o kit em questão, para aproveitar o trabalho já realizado pela comunidade Linux. Ao se examinar a versão 3.0, nota-se que ela não apresenta um arquivo de configuração pronto para o kit imx25pdk, desta forma, esta versão também é eliminada.

Após essa análise, pode se concluir que as únicas versões que cumprem os três itens são as versões 3.2 e a 3.4. A versão que foi escolhida para a realização deste trabalho foi a 3.4, e as instruções para compilá-la e para adicionar o pacote em tempo real Preempt-RT podem ser encontrados nos Apêndices C e K, respectivamente.

3.0	22 July 2011[123]	3.0.80[124]	Greg Kroah-Hartman[125][110]	7th **long-term** stable release from July 2011 to October 2013 (Base for Real-Time-Tree)[125][110]
3.1	24 October 2011[126]	3.1.10[127]	Greg Kroah-Hartman	**EOL** (Maintained from October 2011 to January 2012)[127]
3.2	5 January 2012[128]	3.2.46[129]	Ben Hutchings[130][110]	8th **long-term** stable release from March 2012 to 2016, used in *Ubuntu 12.04 LTS*, *Debian 7 Wheezy* and *Slackware 14.0*[130][110]
3.3	19 March 2012[131]	3.3.8[132]	Greg Kroah-Hartman	**EOL** (Maintained from March 2012 to June 2012)[132]
3.4	21 May 2012[133]	3.4.47[134]	Greg Kroah-Hartman[135][110]	9th **long-term** stable release from May 2012 to October 2014[135][110]
3.5	21 July 2012[136]	3.5.7[137]	Greg Kroah-Hartman	**EOL** (Maintained by Kroah-Hartman until the release of 3.6.1, from July 2012 to October 2012).[137] Unofficial extended support until March 2014.[138]
3.6	1 October 2012[139]	3.6.11[140]	Greg Kroah-Hartman	**EOL** (Maintained from October 2012 to December 2012)[140]
3.7	11 December 2012[141]	3.7.10[142]	Greg Kroah-Hartman	**EOL** (Maintained from December 2012 to March 2013)[142][143]
3.8	19 February 2013[144]	3.8.13[145]	Greg Kroah-Hartman	**EOL** (Maintained from February 2013 to May 2013)[145] Unofficial extended support until August 2014.[146]
3.9	29 April 2013[147]	3.9.4[2]	Greg Kroah-Hartman	Latest stable version
3.10	TBD	3.10-rc4[3]	Linus Torvalds	Latest release candidate version

Legend: Old version | Older version, still supported | **Latest version** | Latest preview version

Figura 8 - Versões do kernel Linux

Fonte : [7]

4.1.2 Pacote Tempo Real: Preempt-RT

Antes da versão 2.6, o escalonador Linux tinha dificuldades de lidar com diversas tarefas em tempo real, tendo em vista que utilizava algoritmos de complexidade O(n), ou seja, algoritmos cujo tempo gasto para escalonar as tarefas aumentava linearmente com o número de tarefas presentes no sistema. Conforme [8], desde a versão tempo real 2.6, foi adotado algoritmos de complexidade O(1), ou seja, o tempo gasto para escalonar as tarefas independe do número de tarefas sendo executadas.

Além disso, o kernel era não preemptivo, desta forma, uma tarefa de baixa prioridade poderia ser executada por um longo período de tempo deixando uma tarefa de grande prioridade bloqueada. Na versão 2.6, somente algumas partes críticas do kernel Linux são consideradas não preemptivas.

Cada CPU possui uma lista de 140 prioridades que são servidas utilizando o algoritmo FIFO ou Round Robin, conforme [8], sendo que no caso da placa imx25pdk, há somente uma CPU disponível.

As prioridades indicam a ordem de execução das tarefas. As tarefas de maior prioridade são executadas preferencialmente, sendo que, caso uma tarefa de menor prioridade esteja sendo executada, e uma tarefa de maior prioridade deseja se executar, o escalonador irá preemptar a tarefa de menor prioridade para que a tarefa de maior prioridade possa se executar. Desta forma, o algoritmo em si não é exatamente FIFO ou Round Robin, sendo, para o Linux Preempt-RT, FIFO ou Round Robin somente a forma como são tratadas as filas de uma mesma prioridade. Essas 140 prioridades são divididas em 100 níveis para as tarefas em tempo real e 40 níveis para as tarefas normais, como pode ser visto na Figura 9.

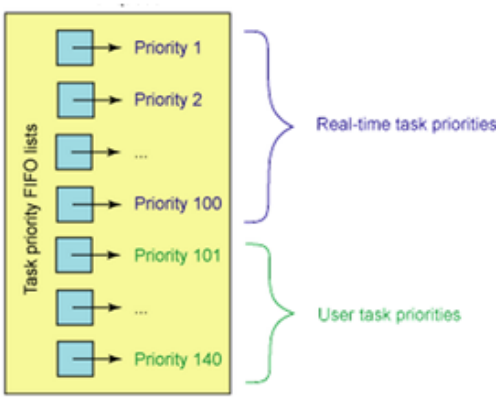

Figura 9 - As prioridades no linux tempo real

Fonte [8]

4.2 Freescale

4.2.1 I.mx25pdk

Essa placa é baseada no ARM926EJ-S™, um microprocessador de arquitetura ARM (*Advanced RISC Machines*). Esse tipo de microprocessador é construído sobre o design RISC (*Reduced Instruction Set Computer*), ou seja, utiliza de um conjunto reduzido e otimizado de instruções, porém com mais rapidez de execução que um microprocessador de arquitetura CISC (*Complex Instruction Set Computer*). Pela Figura 10, pode-se ver o diagrama de blocos simplificados da placa em questão.

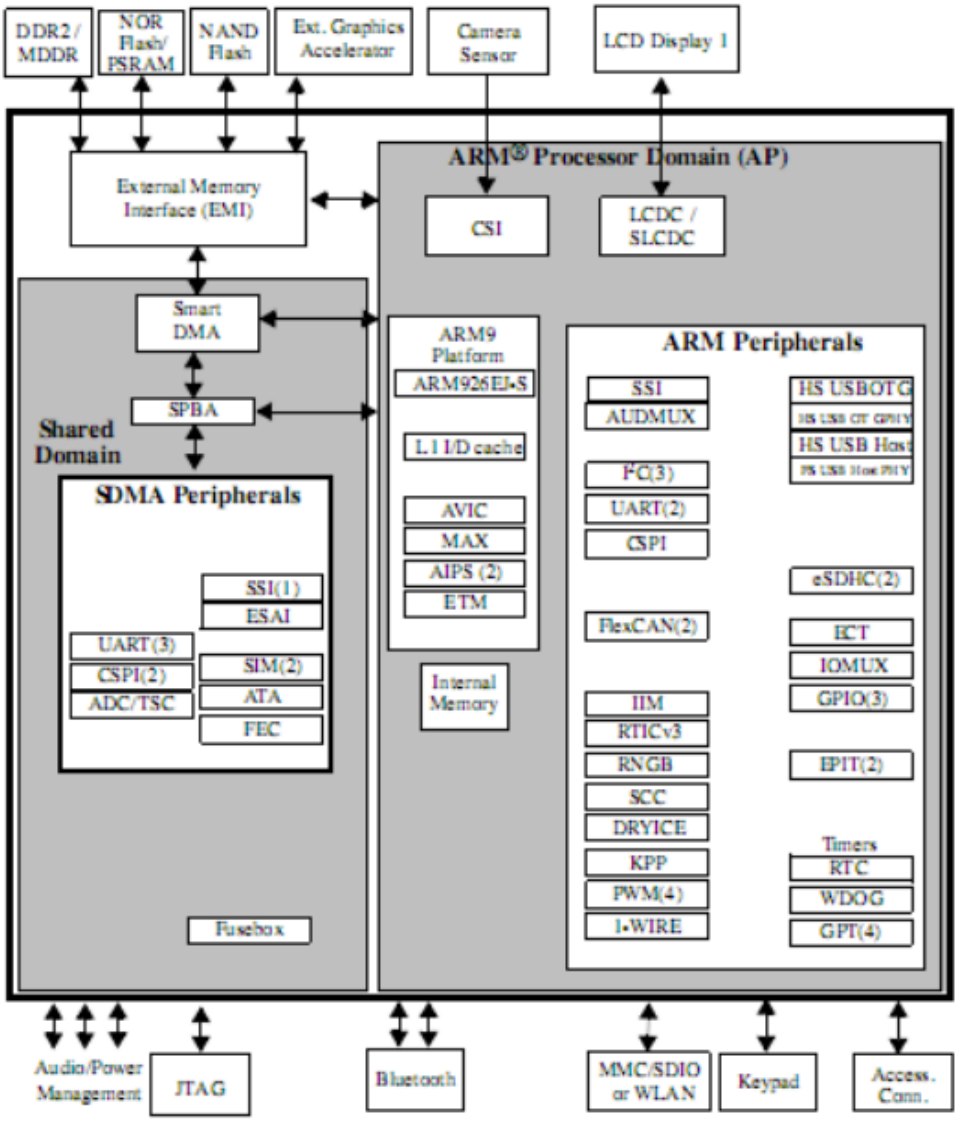

Figura 10 - Arquitetura da i.mx25pdk

Fonte – [9]

Ela possui uma grande conectividade, permitindo o seu uso em diversas aplicações, o que pode ser observado na Tabela 1.

Tabela 1 - Conteúdo da imx25pdk

i.mx25pdk	
Funcionalidade	Item
Core	ARM926EJ-S
Freq. Maxima CPU	400MHz
Flash	2GB
SDRAM	256MB
Controlador LCD	Contém
CAN	Contém
Ethernet	Contém
12 bit ADC	Contém
SD/SDIO/MMC	Contém
I2C	Contém
SSI/I2S	Contém
UART	Contém

Trata-se de uma placa voltada para o mercado automobilístico, com uma boa performance para um custo relativamente baixo. Além disso, ela possui uma grande eficiência energética.

Durante este trabalho, será falado sobre a conexão ethernet, o USB, e a porta serial, sendo que esta pode ser vista na tabela pelo nome de SSI (*Synchrnous Serial Interface*). Também se deve comentar sobre as peculiaridades da memória Flash, pois é importante conhecê-la para melhor entender o funcionamento dos sistemas de arquivos.

A memória Flash é do tipo NAND, que é uma memória de rápida leitura, e maior densidade de armazenamento, se comparadas com a Flash NOR, porém, o acesso é obrigatoriamente sequencial. A memória NAND é dividida em blocos, cada bloco composto de um determinado número de páginas, sendo que, um dos problemas associados a essa memória é que, se um determinado bloco for usado muitas vezes, ele acaba ficando corrompido, e não pode mais ser utilizado.

4.2.2 Ferramentas fornecidas pela Freescale

O *bootloader* é o primeiro programa a ser executado pela placa, sendo o software responsável realização da inicialização da memoria e dos dispostiviso, além de carregar a imagem do kernel e do sistema de arquivos para a placa, utilizando o protocolo tftp. Para realizar tal função, a freescale disponibiliza o reedboot, que já vem instalado no kit i.mx25pdk, sendo um *bootloader* simples e leve. O Anexo F contém as instruções para a utilização do reedboot.

O protocolo TFTP (*Trivial File Transference Protocol*) é um protocolo semelhante ao FTP (*File Transference Protocol*), porém mais simplificado. O TFTP é baseado no protocolo de transporte UDP (*User Datagram Protocol*), não possui ferramentas de segurança como autenticação nem encriptação de dados, diferente do FTP que é baseado no TCP (*Transmission Control Protocol*), e fornece tais ferramentas. Ele também é usado para transferir os arquivos de boot para a placa, e informações sobre seu uso podem ser encontradas no Anexo D.

O LTIB (*Linux Image Target Builder*) é a ferramenta que possibilita a geração de imagem do kernel (para versões inferiores a 3.0) para que então o *bootloader* possa carregá-la no kit. Ela também é responsável por gerar o sistema de arquivos que será utilizado durante esse trabalho.

Maiores detalhes sobre o uso dessas ferramentas estão disponibilizadas no Anexos A,e B deste trabalho.

4.3 O Sistema de Arquivos

4.3.1 NFS

NFS significa *Network File System*, como o nome indica, ele permite que se utilize o sistema de arquivos conectado a uma rede. Desta forma, o sistema de arquivos fica no PC e é acessado pelo kernel da placa conforme a necessidade, através de protocolos cliente-servidor.

Ele é flexível, pois pode se modificar o sistema de arquivos diretamente no PC host, sem a necessidade de se gerar uma imagem e de carregá-la na placa, como no caso do JFFS2. Desta forma, é mais rápido e fácil de realizar a depuração do sistema. Além disso, por manter o sistema de arquivos no PC, ela não usa a memória NAND da placa, o que aumenta a vida útil da mesma. A configuração do NFS está escrita no Anexo E deste trabalho.

Por outro lado, o fato de funcionar por NFS pode influenciar em testes e simulações já que o sistema de arquivos não fica na Flash e, além disso, o sistema fica dependente do PC host.

4.3.2 JFFS2

De acordo com [13], JFFS2 (Journallling Flash File System version 2) é um sistema de arquivos baseado em um log circular de operações unidos a um mecanismo de *Garbage Collection*, sendo compatível com memória flash NAND, que é a memória Flash usada pelo kit i.mx25pdk.

Para evitar a corrupção dos blocos da memória NAND, o JFFS2 faz uma organização que ao gravar um arquivo na memória ele não grava todo o conteúdo em um só bloco, mas ele espalha o conteúdo por toda a memória de maneira que os blocos sejam todos usados de forma igual.

A desvantagem do JFFS2 é que sua montagem é demorada, não sendo recomendado seu uso em espaços de memória muito grandes.

4.4 Os Drivers do Linux 3.4 Para o i.mx25pdk

4.4.1 Problemas Encontrados

Durante este trabalho, percebeu-se que alguns dos drivers apresentavam problemas. Entre os problemas que foram identificados, o SDMA não consegue encontrar o seu firmware, o USB não consegue ser montado, a conexão NFS não funciona, e a memória NAND não estava particionada.

Tais problemas são cruciais, visto que, sem conexão NFS é impossível de se montar o NFS, e sem partições estabelecidas na memória NAND, não é possível de se montar o JFFS2. E sem um sistema de arquivos, não há como se rodar o Linux na placa.

4.4.2 Modificações Realizadas

Felizmente, como se pode observar no Apêndice I, foi possível realizar as partições na placa. Foram realizadas as partições, reservando o seguinte espaço na memória :

- Bootloader = 0x00000000 - 0x002FFFFF (Tamanho = 3Mb)
- kernel = 0x00300000 - 0x007FFFFF (Tamanho = 5Mb)
- rootfs = 0x00800000 - 0x107FFFFF (Tamanho = 256Mb)
- configure = 0x10800000 - 0x10FFFFFF (Tamanho = 8Mb)
- userfs = 0x11000000 - 0x7FFFFFFF (Tamanho = 1776 Mb)

Além disso, através de mudanças na configuração do Linux, o driver do USB é montado na placa, sendo que a placa reconhece quando um dispositivo USB é conectado a ela, porém, ainda não foi possível realizar troca de arquivos entre algum dispositivo USB e a placa.

Para a realização de trocas de arquivos, buscou-se, então, uma solução alternativa, o uso da porta serial. Como demonstrado no Anexo H, pode-se facilmente se realizar o envio e recebimento de arquivos depois de instalados os aplicativos necessários.

4.5 *Cyclictest*

Para testar o desempenho da placa, um único teste não seria suficiente, pois este sistema está sujeito a influência de um grande número de variáveis. A cada teste que fosse rodado, seria obtido um resultado diferente. Desta forma, o desempenho do sistema deve ser verificado através de uma extensa simulação, por um grande intervalo de tempo, que, então, fornecerá uma resposta estatística sobre o desempenho da placa. Deve-se observar que, quanto maior for o tempo de simulação, mais precisa será esta resposta.

Para tal finalidade, Thomas Gleixner desenvolveu a aplicação chamada de cyclictest. Considerado como o estado da arte em softwares para medição de desempenho do tempo de latência em sistemas operacionais Linux de tempo real, de acordo com [14], possui um algoritmo relativamente simples, que pode ser visto de maneira simplificada na Figura 11.

```
clock_gettime((&now))
next = now + par->interval

while (!shutdown) {

   clock_nanosleep((&next))

   clock_gettime((&now))
   diff = calcdiff(now, next)

   # update stat-> min, max, total latency, cycles
   # update the histogram data

   next += interval
}
```

Figura 11 - Código simplificado do cyclictest
Fonte – [10]

De acordo com [10], o programa mede o tempo atual, mede o tempo pelo qual ele supostamente irá ser suspenso, então ele é suspenso, e depois, ele mede a diferença entre o tempo em que ele deveria ter retornado a atividade e o tempo que ele realmente retornou. Desta forma, a diferença entre ambos é o período de latência.

Para a execução deste programa, uma série de opções serão selecionadas, tais como:

- Opção -n: uso do clock_nanosleep ao invés de intervalos de tempo posix, tornando a medição mais precisa;
- Opção -p99: define a prioridade da primeira Tarefa para 99, sendo esta uma prioridade RT. No caso da criação de três Tarefas, as prioridades seriam 99, 98 e 97, por exemplo;
- Opção -t: define o numero de threads, sendo que serão feitos testes com uma única Tarefa e também com mais de uma Tarefa;
- Opção –m: é usada para prevenir a memória cyclictest de ser paginada para a área de swap;
- Opção –D: indica de duração da simulação, por exemplo, -D 1h, indica que a simulação perdurará por uma hora.
- Opção –v: necessário para a realização de análises estatísticas.
- Opção --policy: indica qual o método de escalonamento que vai ser utilizado, sendo que neste trabalho serão utilizados os métodos FIFO (--policy=fifo) e RR (-policy=rr).

Um detalhe importante é que o volume de dados é muito grande, desta forma, se uma simulação rodar por muito tempo, ela produzirá um arquivo muito longo. Para facilitar a troca de dados entre a placa e o PC host, é recomendável que se compacte o produto final da simulação (neste trabalho, foi usado o gzip). Por exemplo, cyclictest –n –v –m –t 3 –D 1h –policy=fifo | gzip > test.gz.

Informações sobre a instalação do *cyclictest* podem ser encontradas no Anexo G.

5. TESTES E RESULTADOS

5.1 Versão 3.4 versus 2.6 sem carga

A fim de comparar o desempenho dos diferentes sistemas operacionais, foi feito um teste de uma hora de duração, com uma tarefa sendo executada. Primeiramente, foi analisado o desempenho do Linux freescale 3.4.42-rt57 #9 PREEMPT RT usando o método FIFO, sendo realizado um gráfico mostrando a latência encontrada ao longo do tempo, o que resultou em 3599999 medições (uma medição a cada microsegundo), conforme a Figura 12. Este gráfico, bem como os outros presentes nesta seção, foram feitos usando um script em phyton que pode ser encontrado no Anexo J deste trabalho.

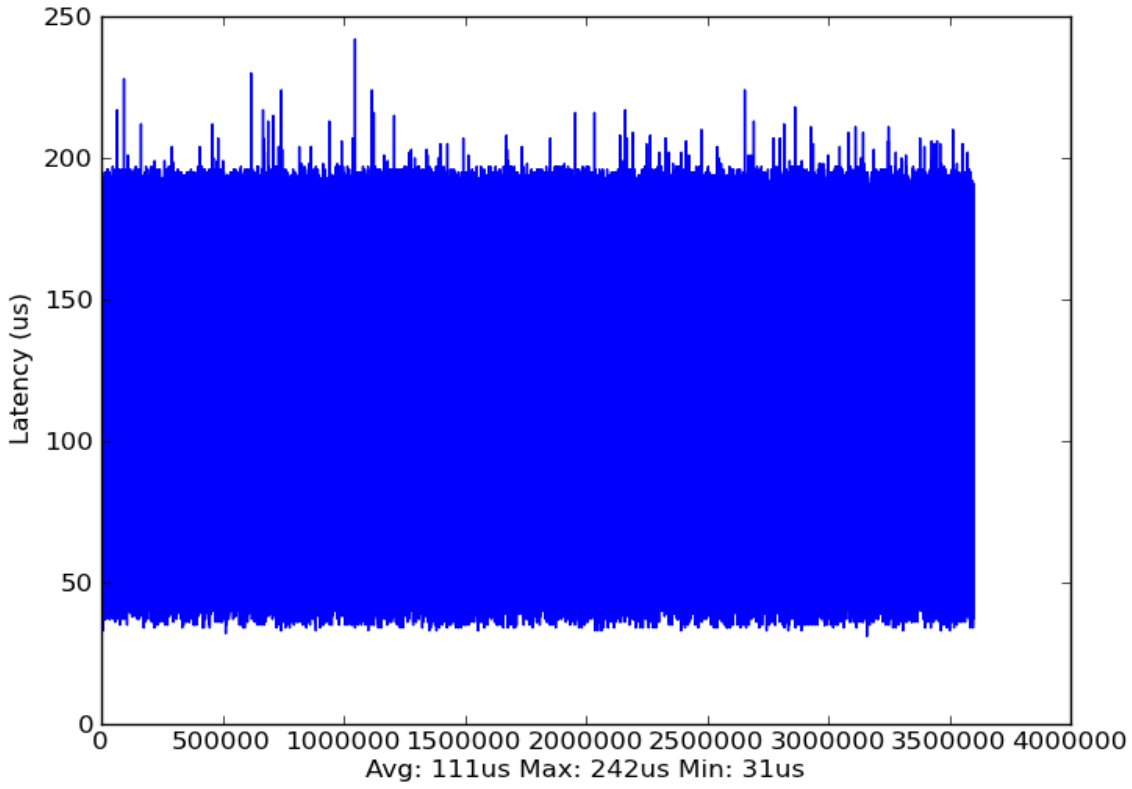

Figura 12 - Latência da v3.4 ao longo do tempo

Após isso, foi analisado o Linux freescale 2.6.31-207-g7286c01 #1 PREEMPT da mesma forma, conforme a Figura 13.

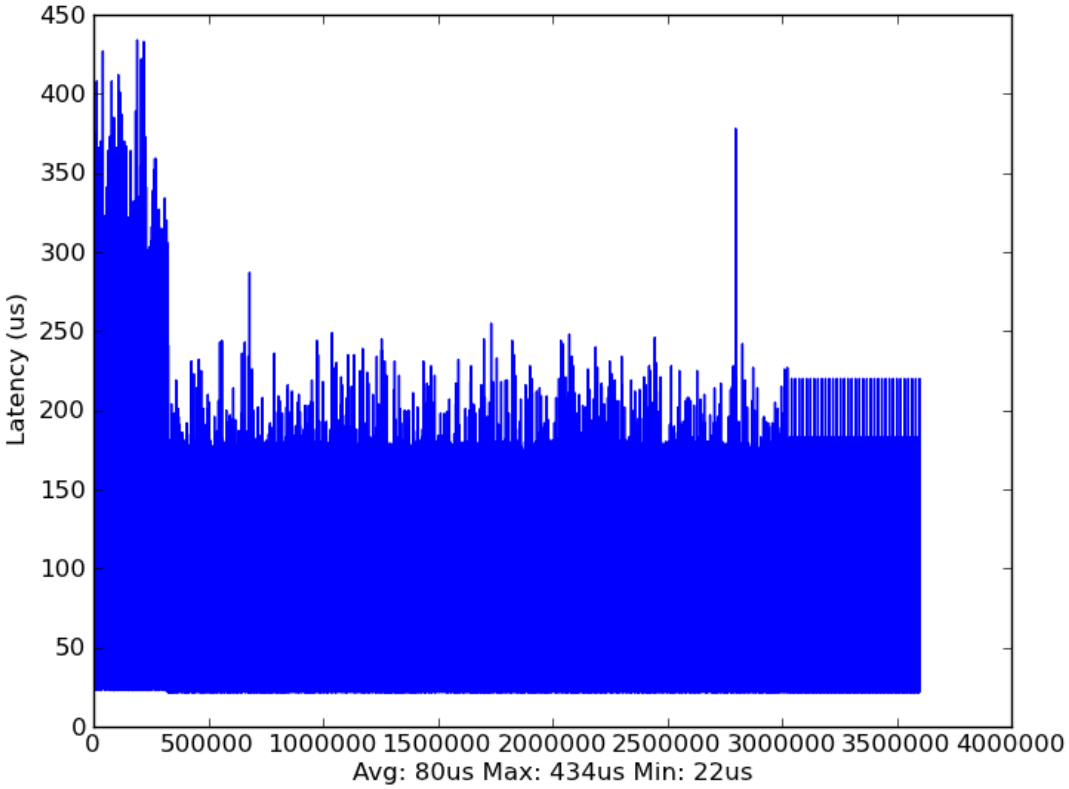

Figura 13 - Latência da v2.6 ao longo do tempo

Pelos resultados deste teste, pode-se observar que o Linux 3.4 apresentou um desempenho médio pior que o da versão 2.6 (tempo médio de 111us para a versão 3.4 contra 80 us para a versão 2.6), e também apresentou um tempo mínimo pior (31us para a versão 3.4 contra 22us para a versão 2.6). Porém, pelo gráfico pode-se notar que em várias ocasiões a versão 2.6 apresentou um valor de latência superior ao valor máximo encontrado na versão 3.4 (principalmente no início da simulação), desta forma, para o pior caso, a versão 3.4 possui um melhor desempenho (242us para a versão 3.4 contra 434us para a versão 2.6).

Para melhor visualizar a distribuição dos valores de latência, e saber, por exemplo, com que frequência os valores da versão 2.6 apresentam valores piores que o pior caso da versão 3.4, foi realizado um histograma para as duas versões, como pode ser visto pela Figura 14, e pela Figura 15.

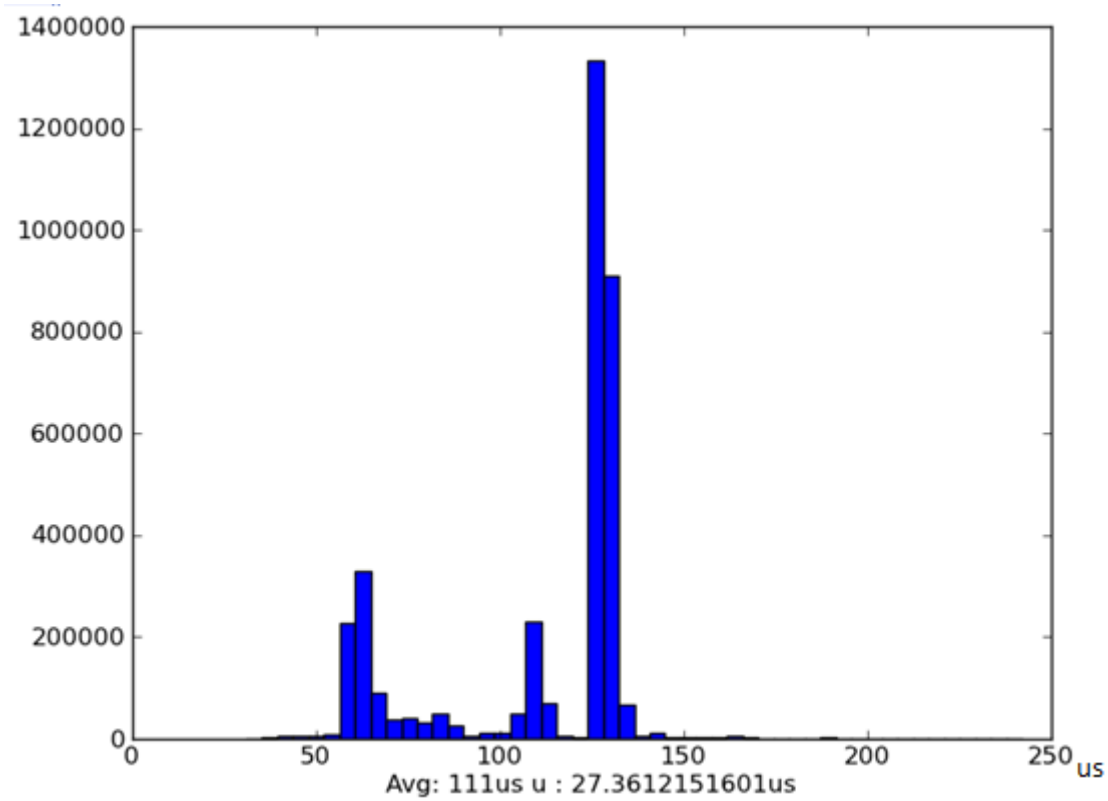

Figura 14 - Histograma da latência da v3.4

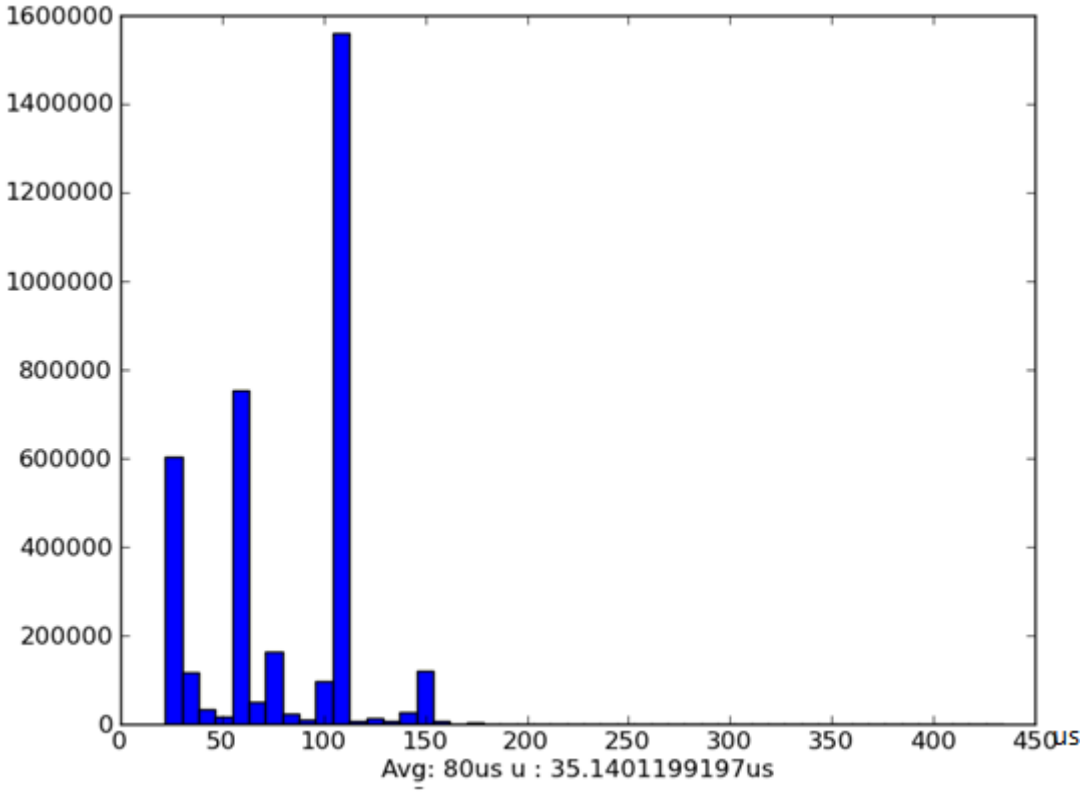

Figura 15 - Histograma da latência da v2.6

Olhando o histograma da versão 3.4, pode-se notar que a grande maioria dos valores se encontra entre 50us e 150us, sendo a maior concentração entre 100us e 150us. Na versão 2.6, a grande maioria dos valores se encontra entre 20us e 150us, sendo a maio concentração entre 20us e 110us. Desta forma, pode-se observar que é raro que a versão 2.6 obtenha resultados muito ruins (pior que o pior caso da versão 3.4).

A constatação da pequena frequência de ocorrência destes eventos, bem como a observação pela Figura 13 de que o mau desempenho da versão 2.6 ocorre principalmente no início da simulação, levou a suspeita de que o desempenho inicial ruim da versão 2.6 tenha se dado em razão de algum evento raro e desconhecido. A fim de investigar tal suspeita, um novo experimento foi realizado, com pequenas alterações em relação a este.

5.2 Versão 3.4 versus 2.6 com carga

Neste teste foi adicionada uma “carga” à simulação, um pequeno programa em C, que se executa em loop infinito realizando pequenos cálculos. Tentando aumentar o tamanho da amostra, desta vez foi feita uma medição de duas horas, resultando em 7199999 medições, como pode ser observado pela Figura 16 e pela Figura 17.

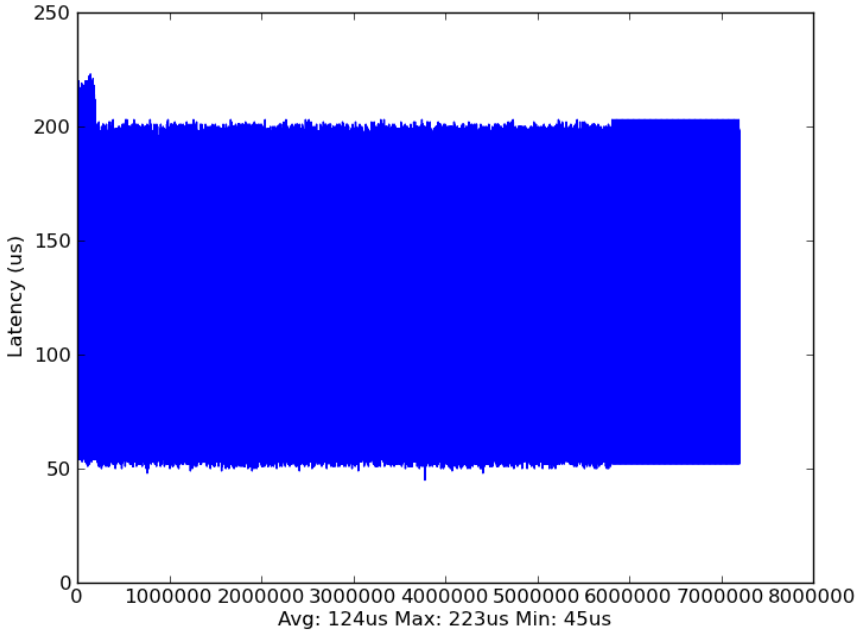

Figura 16 - Latência da v3.4 ao longo do tempo com carga

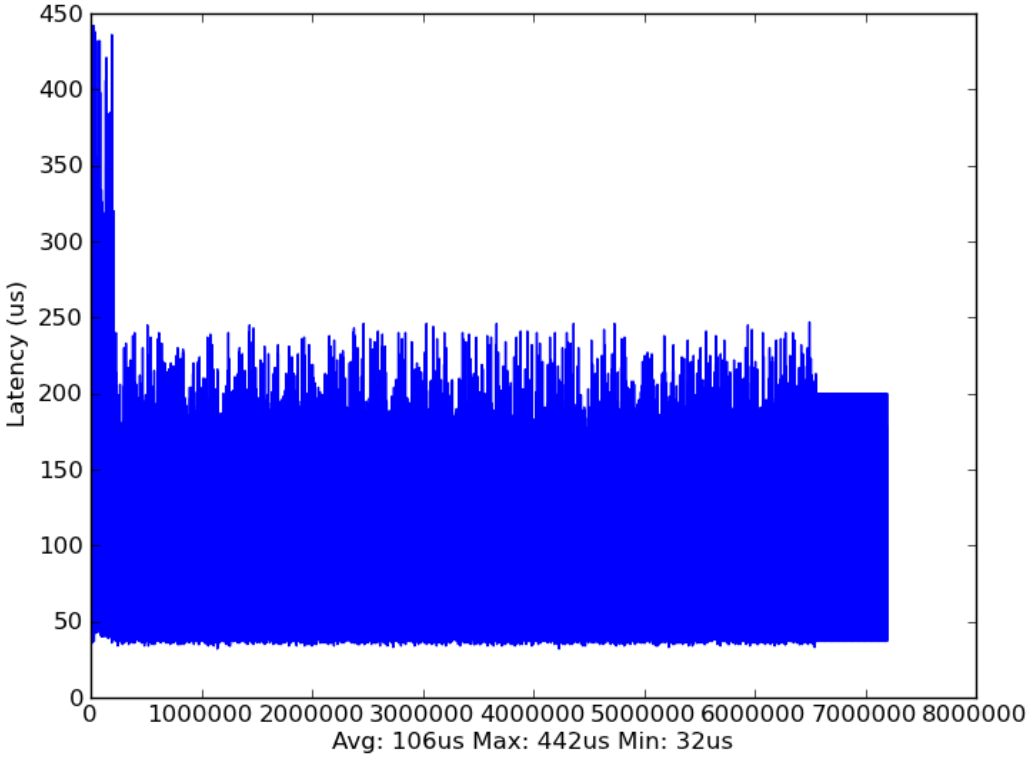

Figura 17 - Latência da v2.6 ao longo do tempo com carga

Pelos resultados deste teste, pode-se observar que o Linux 3.4 apresentou um desempenho médio pior que o da versão 2.6 (tempo médio de 124us para a versão 3.4 contra 106 us para a versão 2.6), e também apresentou um tempo mínimo pior (45us para a versão 3.4 contra 32us para a versão 2.6). Porém, pelo gráfico pode-se notar que em várias ocasiões a versão 2.6 apresentou um valor de

latência superior ao valor máximo encontrado na versão 3.4, desta forma, para o pior caso, a versão 3.4 possui um melhor desempenho (223us para a versão 3.4 contra 442us para a versão 2.6).

Em comparação ao teste anterior, também pode ser observado que os resultados são levemente piores, em geral, com exceção do valor de latência máximo da versão 3.4, que inclusive diminuiu (de 242us no primeiro teste, para 223us no segundo teste).

Para melhor visualizar a distribuição dos valores de latência, foi realizado um histograma para as duas versões como pode ser visto pela Figura 18, e pela Figura 19.

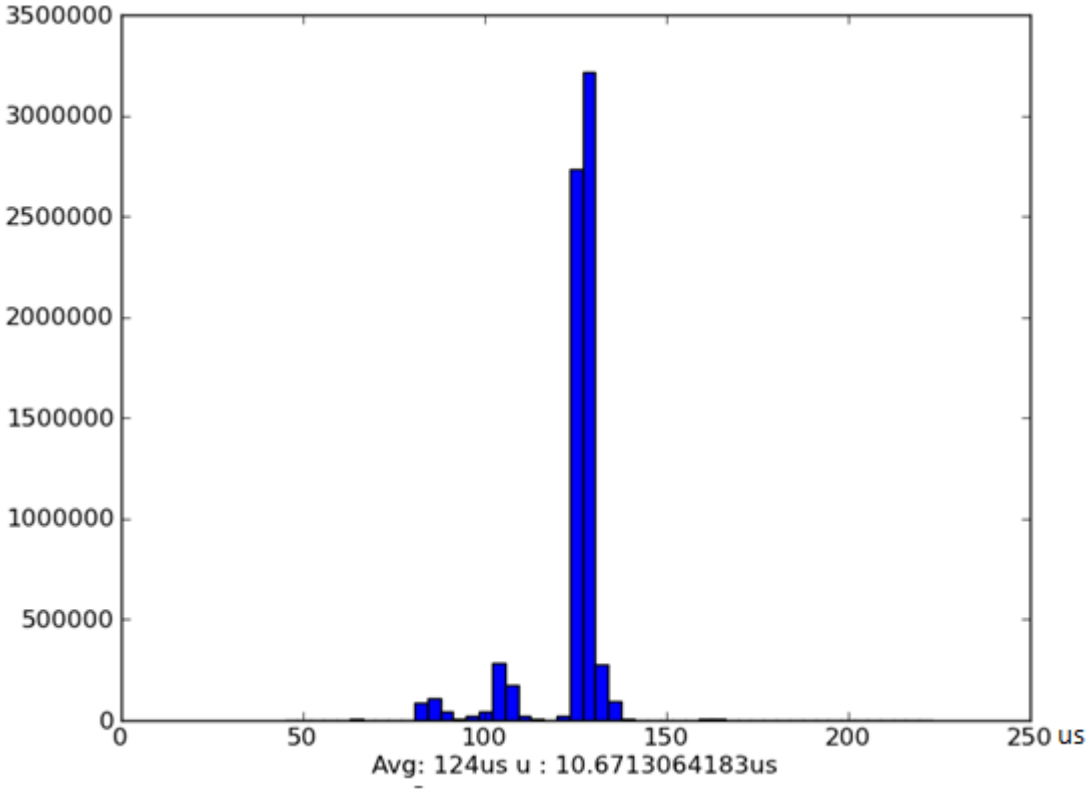

Figura 18 - Histograma da latência da v3.4 com carga

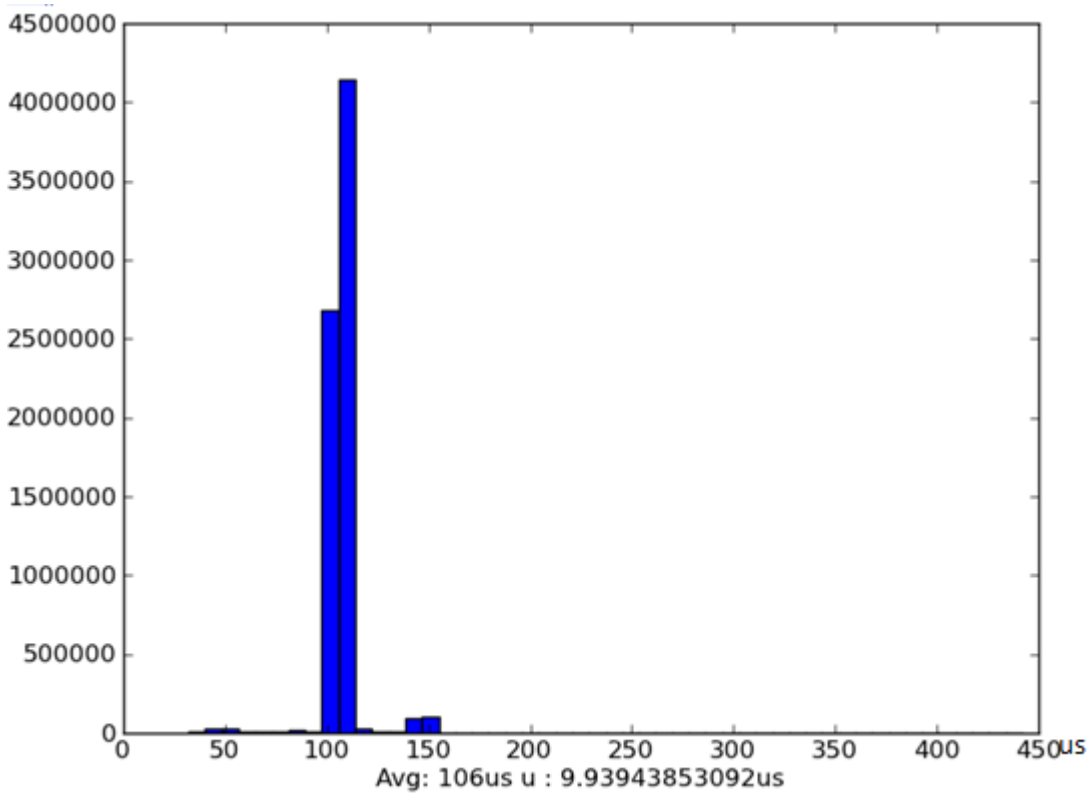

Figura 19 - Histograma da latência da v2.6 com carga

Pelo histograma da versão 3.4, pode-se notar que a grande maioria dos valores se encontra entre 70us e 130us, sendo a maior concentração entre 110us e 130us. Na versão 2.6, a grande maioria dos valores se encontra entre 100us e 170us, sendo a maio concentração entre 100us e 120us. Da mesma forma que no teste anterior, pode-se observar que é um tanto quanto raro que a versão 2.6 obtenha resultados muito ruins (pior que o pior caso da versão 3.4).

5.3 Round Robin versus First In First Out

A fim de comparar o desempenho dos diferentes métodos de escalonamento do Linux freescale 3.4.42-rt57 #9 PREEMPT RT foi feito um teste de uma hora de duração, com três tarefas sendo executadas. Primeiramente, foi analisado o método *Round Robin*, como pode ser visto na Figura 20. Após isso, foi analisado o método *First In First Out*, como pode ser observado na Figura 21.

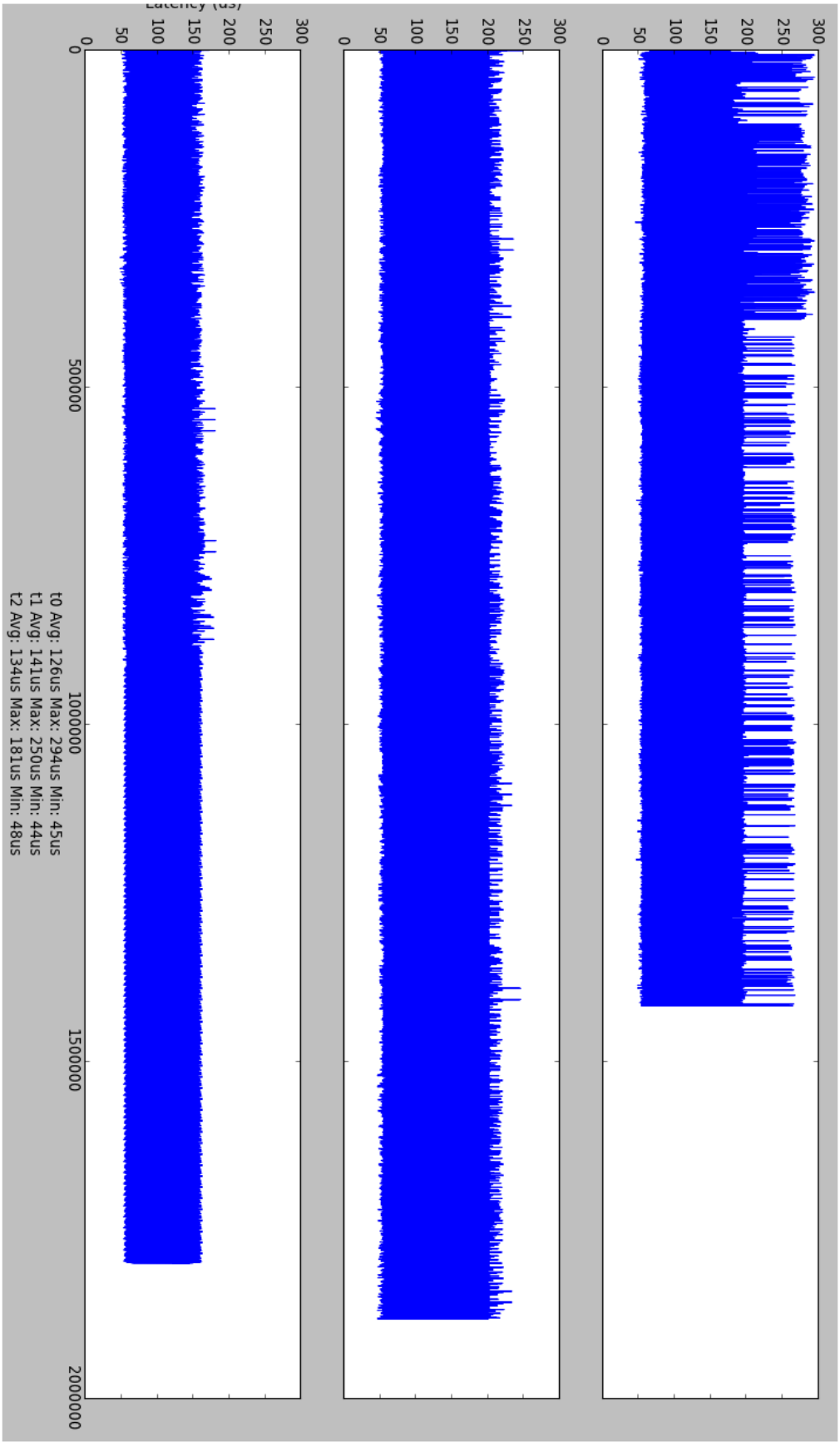

Figura 20 - Latência das tarefas para o método RR

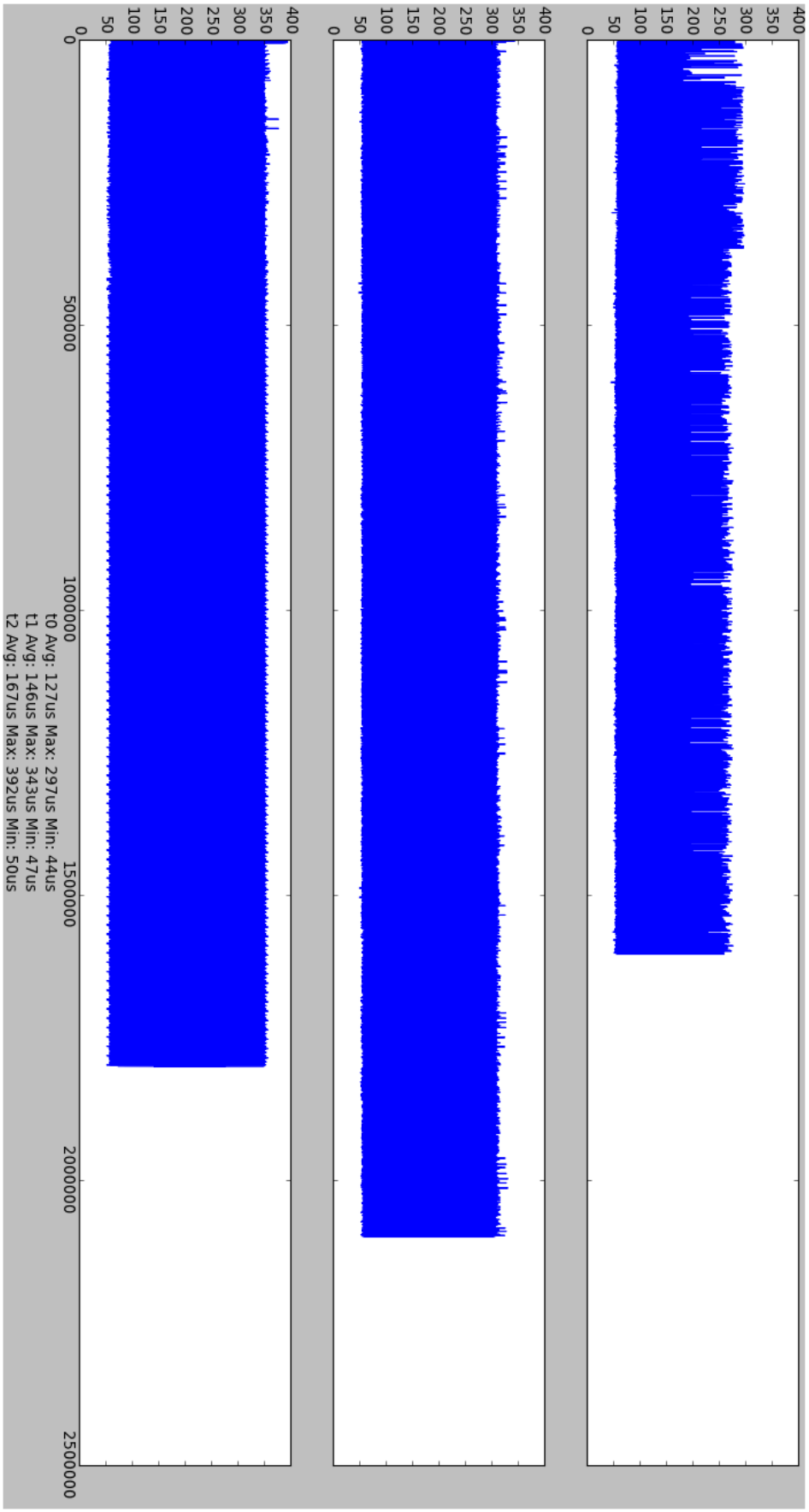

Figura 21 - Latência das tarefas para o método FIFO

No terceiro experimento, analisando os métodos de escalonamento, pode-se notar que seus resultados são semelhantes, sendo que uma diferença que deve ser citada é o valor médio de latência. O que acontece é que, diferente do First In First Out quanto uma tarefa só é parada pelo escalonador caso seja bloqueada por um sincronizador (um semáforo, por exemplo), por uma tarefa de maior prioridade, ou por vontade própria (*sleep()* ou *pthread_yeld()*, por exemplo), no método Round Robin (RR) a tarefa também é parada após um determinado tempo de execução. Desta forma, o recurso é repartido de forma mais igualitária entre as tarefas, e é por isso que os valores de tempo médio são relativamente semelhante entre as tarefas, diferente do que acontece no método FIFO.

Para melhor visualizar a distribuição dos valores de latência, foi realizado um histograma para as duas versões como pode ser visto pela Figura 22, e pela Figura 23.

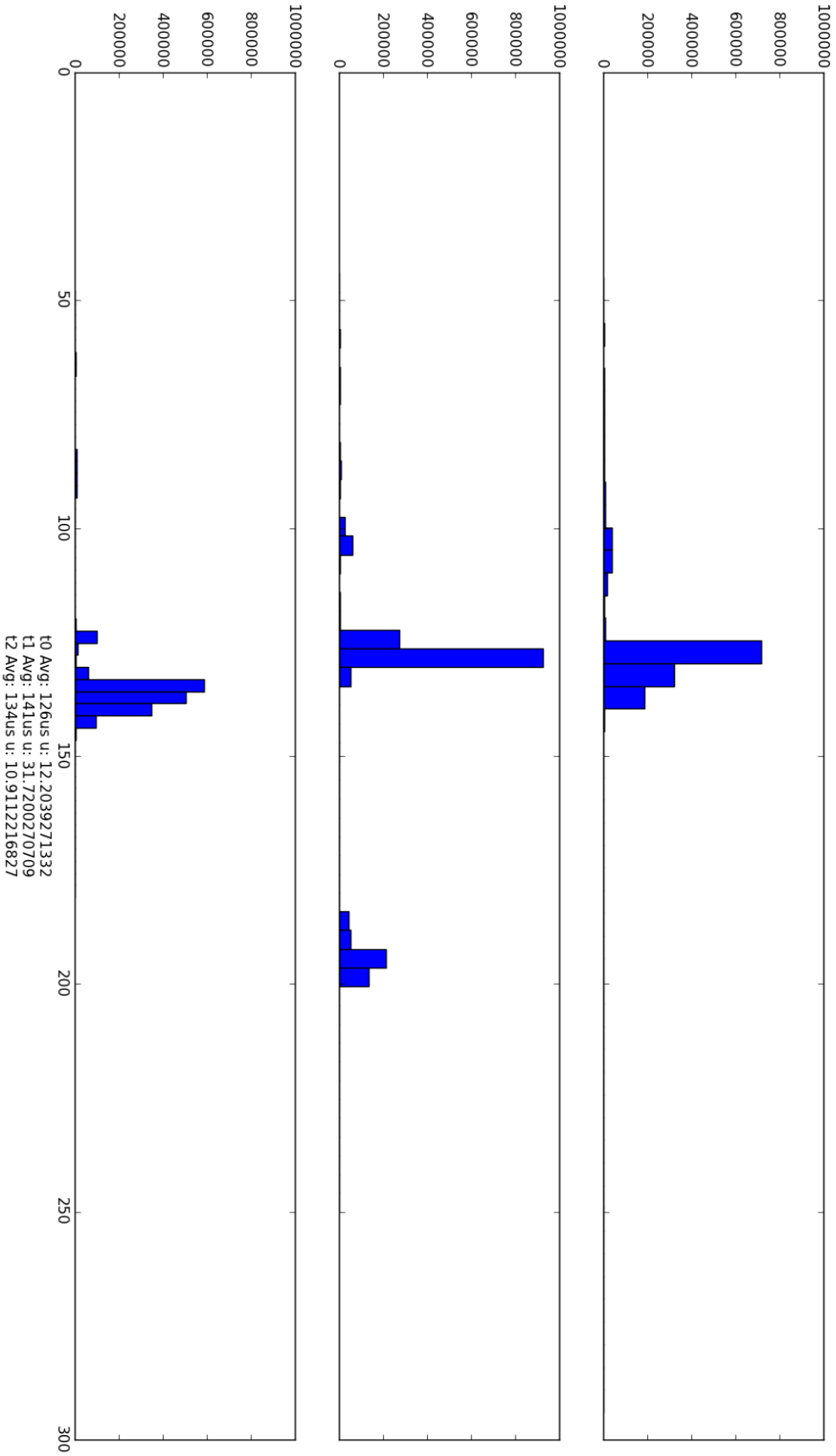

Figura 22 - Latência das tarefas para o método RR

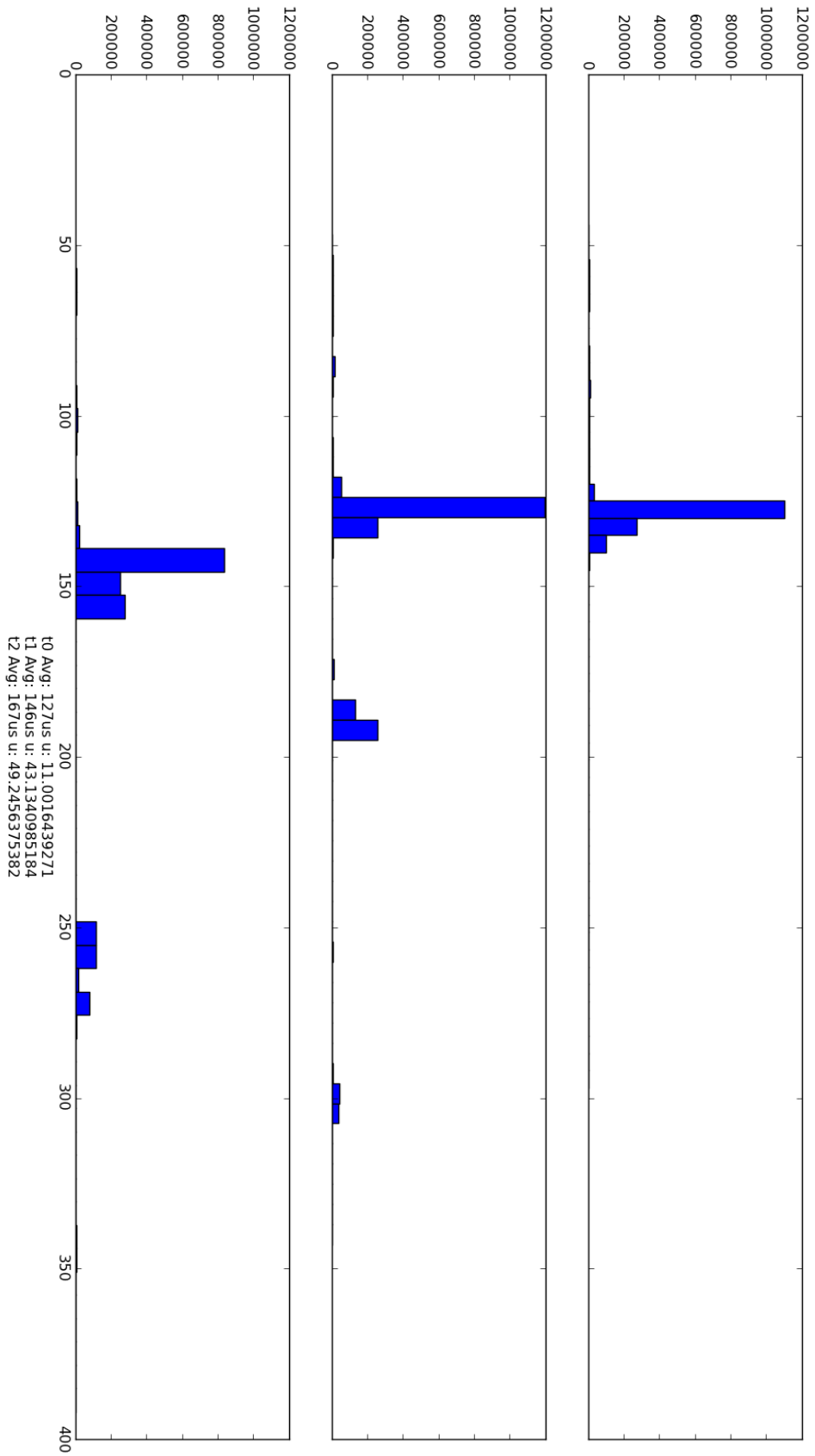

Figura 23 - Histograma das tarefas para o método FIFO

Pelo histograma, é possível observar que o método FIFO claramente prioriza as tarefas mais prioritárias, desta forma, quanto maior for a prioridade, melhor vai ser o desempenho. É por isso que, quanto maior a prioridade, mais a esquerda se situam as ocorrências do método FIFO. No entanto, dado as características do RR, essa situação não é vista de forma tão clara no seu método, já que ele compartilha melhor o recurso entre as tarefas.

6. CONCLUSÕES

Com o auxílio dos programas fornecidos pela freescale e pelos programas fornecidos pela comunidade Linux, além de um esforço em entender e corrigir os problemas encontrados nos mesmos, foi possível atualizar o sistema operacional da placa i.mx25pdk com sucesso. Foi possível também realizar uma série de testes de desempenho para o sistema operacional da placa, comparando o novo sistema operacional com o sistema antigo, bem como comparando diferentes métodos de escalonamento.

Para realizar tais comparações, é importante relembrar os diversos conceitos que foram visto acerca da teoria de sistemas embarcados, em particular sobre sistemas embarcados em tempo real. Algo importante de ser notado é que as exigências de desempenho de um sistema variam de acordo com a aplicação do mesmo. Tais diferenças são importantes na momento de decidir qual sistema operacional a ser usado com a placa imx25pdk, bem como para decidir o método de escalonamento a ser utilizado.

Pelo desempenho no primeiro e no segundo teste, pode-se observar que o desempenho médio da versão 2.6 é melhor que o desempenho médio da versão 3.4. Por outro lado, o pior caso (latência máxima) da versão 3.4 foi melhor que o da versão 2.6. Desta forma, se a aplicação em questão for uma aplicação crítica, *hard real time*, onde é imprescindível o respeito a todas as propriedades de tempo real, o pior caso da latência é o valor que será considerado, então, é recomendado o uso da versão 3.4. Porém, se a aplicação em questão for uma aplicação não crítica, *soft real time*, onde algum desrespeito às propriedades em tempo real é tolerável, pode-se optar por escolher a versão 2.6 tendo em vista que o seu desempenho médio é melhor.

Em relação ao método de escalonamento, conforme era esperado, o método Round Robin compartilha os recursos de forma mais igualitária entre as tarefas que o método FIFO, sendo que este é melhor para dar preferência as tarefas mais prioritárias.

Além da diferença de desempenho atual, também é importante notar que as versões 3.x continuam a ser sustentadas e atualizadas pela comunidade Linux, diferente da versão 2.6. Dessa forma, novas inovações, como, por exemplo, a

adição de novos algoritmos de escalonamento, entre outras, podem vir a ser realizadas. Por isso, a possibilidade de se ter acesso às novas modificações do Linux é outra fonte de motivação para que continue a se entender e se adaptar a versão 3 para o kit i.mx25pdk.

Para o futuro, é necessário que seja realizado um esforço para corrigir os problemas listados durante o relatório, como o driver do USB, a conexão ethernet com o PC *host*, além do teste e possível correção das outras funcionalidades da placa, que não foram analisadas tendo em vista que não eram necessárias para a realização deste trabalho.

Também é importante a realização de novos testes para melhor medir o desempenho do sistema, como por exemplo, como o sistema se comporta em caso de sobrecarga. Outra possibilidade seria a repetição dos mesmos testes utilizando um maior período de tempo, para melhorar a precisão dos mesmos, levando em conta que em algumas análises chega-se a deixar o cyclictest rodando por vários dias, o que não foi feito neste projeto devido à falta de tempo hábil.

REFÊRENCIAS

1. BELISLE,J.F. **Marketing, Analytics & Entertainment**. Disponível em:
http://jfbelisle.com/2009/05/three-types-of-convergence-is-the-future-friendly/
Acesso em: Junho de 2013.

2.FREESCALE. **i.MX25 Product Development Kit**. Disponível em:
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=IMX25PDK&tid=vanIMX25PDK
Acesso em: Junho de 2013.

3. WIKIPÉDIA, **Kernel (computing)**. Disponível em:
http://en.wikipedia.org/wiki/Kernel_(computing)
Acesso em: Junho de 2013.

4. WIKIPÉDIA, **FIFO (escalonamento)**. Disponível em:
http://pt.wikipedia.org/wiki/FIFO_(escalonamento)
Acesso em: Junho de 2013.

5. WIKIPÉDIA, **Round-Robin**. Disponível em:
http://pt.wikipedia.org/wiki/Round-robin
Acesso em: Junho de 2013.

6. GROUPE SEM, **ELEC 223**. Disponível em:
http://sen.enst.fr/ue/elec223
Acesso em: Junho de 2013.

7. WIKIPÉDIA,**Linux Kernel**. Disponível em:
http://en.wikipedia.org/wiki/Linux_kernel
Acesso em: Junho de 2013.

8. IBM, **Inside the Linux Escalonador.** Disponível em:
http://www.ibm.com/developerworks/linux/library/l-Escalonador/
Acesso em: Junho de 2013.

9. FREESCALE. **i.MX25 PDK Hardware User's Guide** – Disponível em:
http://www.freescale.com/files/32bit/doc/user_guide/pdk10_imx25_Hardware_UG.pdf
Acesso em: Junho de 2013.

10. ROWAND, F. **Using and Understanding the Real-Time Cyclictest Benchmark** – Disponível em : http://www.youtube.com/watch?v=f_u4r6ehZKY

Acesso em: Junho de 2013.

11. WIKIPÉDIA,**Cross Compiler**. Disponível em:
http://en.wikipedia.org/wiki/Cross_compiler
Acesso em: Junho de 2013.

12. DINIZ, Morganna, **Sistemas de arquivos**. Disponível em:
http://www.uniriotec.br/~morganna/guia/sistemas_de_arquivos.html
Acesso em: Junho de 2013.

13. WIKIPÉDIA,**JFFS2**. Disponível em:
http://en.wikipedia.org/wiki/JFFS2
Acesso em: Junho de 2013.

14. OPEN SOURCE AUTOMATION LAB, **You can tune a piano - and you can tuna Linux system, too!** Disponível em:
https://www.osadl.org/Single-View.111+M52212cb1379.0.html
Acesso em: Junho de 2013.

15. BENVENUTTI, Renê, **Plataforma Embarcada de Tempo Real Integrada ao Matlab Simulink.**

APÊNDICE A – Instalar uma versão adequada do Ltib

Primeiramente, deve-se baixar no site da freescale um arquivo contendo o ltib, sendo que deve-se ter cuidado em relação ao toolchain utilizado. Para compilar um versão do kernel mais recente que a 2.6.34 é necessária uma toolchain 4.4.1 ou superior (de acordo com instruções da própria freescale). No caso deste projeto, foi usado uma toolchain 4.4.4, encontrada no arquivo L2.6.35_11.04.01 disponibilizado no site da freescale.

Para a instalação do ltib, foi utilizado o seguinte script:

```
cd /bin
sudo rm sh
sudo ln -s bash sh

sudo apt-get install patch g++ rpm zlib1g-dev m4 bison libncurses5-dev libglib2.0-dev gettext build-essential tcl intltool libxml2-dev liborbit2-dev libx11-dev
ccache flex uuid-dev liblzo2-dev

sudo apt-get install x11proto-core-dev

sudo apt-get install libdbus-glib-1-dev libgtk2.0-dev libdbus-glib-1-dev

sudo chmod 777 /opt

sudo apt-get install ccache flex libglib2.0-dev gettext ja-trans

sudo apt-get install uboot-mkimage
```

Figura 24 - Script para instalação do ltib

Depois disso, basta descompactar o arquivo L2.6.35_11.04.01, entrar na pasta, e dar o comando ./install.

Tendo feito isso, esse comando irá gerar uma pasta freescale dentro da pasta /opt, e irá gerar uma pasta ltib dentro do diretório que for indicado durante a instalação.

Deve-se, então, editar o arquivo ltib (na pasta ltib), adicionando a linha indicada pelo “+”:

```
    $cmd.="--excludedocs";
+   $cmd .= "--force-debian " if $rpm =~ m/rpm-fs/ && `uname -a` =~ m/ubuntu/i;
    $cmd .= "--define '_tmppath $cf->{tmppath}' ";
```

Finalmente, basta digitar o comando ./ltib para terminar a instalação do softwate.

APÊNDICE B – Adicionar um novo kernel ao ltib

Para adicionar um kernel no ltib, primeiramente deve-se criar um arquivo spec.in, cujo nome deve ser kernel.<version>.spec.in identificando este kernel, como mostrado na Figura 25.

```
# Template = kernel-common.tmpl

%define pfx /opt/freescale/rootfs/%{_target_cpu}
%define pkg_name linux

Summary         : Linux kernel (core of the Linux operating system)
Name            : kernel
Version         : 3.2.43
Release         : 0
License         : GPL
Vendor          : kernel.org + others
Packager        : Peter Barada
Group           : System Environment/Kernel
Source          : %{pkg_name}-%{version}.tar.bz2

BuildRoot       : %{_tmppath}/%{name}
Prefix          : %{pfx}

%Description
%{summary}

%Prep
%setup -n %{pkg_name}-%{version}
```

Figura 25 - Arquivo de spec.in

Esse arquivo deve ser colocado em /<ltib folder>/ltib/dist/lfs-5.1/kernel e em /<ltib folder>/ltib/configs/plataform/imx/

Após isso, deve-se editar o arquivo main.lkc contido na pasta /<ltib folder>/ltib/configs/plataform/imx/ como mostrado na Figura .

```
comment "Choose your Kernel"
choice
    prompt "kernel"
    default KERNEL_31
    help
        This menu will let you choose the kernel to use with your board.
        If you don't want to build a kernel, unselect this option.

    config KERNEL_31
        bool "Linux 2.6.31-imx"
        help
            This is the 2.6.26 kernel and patches for MXC and i.MX platforms.
    config KERNEL_28
        bool "Linux 2.6.28-imx"
        help
            This is the 2.6.28 kernel and patches for MXC and i.MX platforms.
    config KERNEL_3072
        bool "Linux 3.0.72-imx"
        help
            This is the 3.0.72 kernel and patches for MXC and i.MX platforms.
    config KERNEL_3243
        bool "Linux 3.2.43-imx"
        help
            This is the 3.2.43 kernel and patches for MXC and i.MX platforms.
```

Figura 26 - Primeira alteração no main.lkc

```
config PKG_KERNEL
    string
    default "kernel-2.6.31"         if KERNEL_31
    default "kernel-2.6.28"         if KERNEL_28
    default "kernel-3.0.72"         if KERNEL_3072
    default "kernel-3.2.43"         if KERNEL_3243
    default "kernel-3.4.41"         if KERNEL_3441
    default "kernel-git"            if KERNEL_GIT
    default "kernel26-dir-build"    if KERNEL_DIR_BUILD
```

Figura 27 - Segunda alteração no main.lkc

Finalmente, deve-se copiar o kernel (compactado no formato tar.bz2) para a pasta /opt/freescale/pkgs

APÊNDICE C – Compilar o kernel

Para compilar o kernel, usando o ltib, basta usar o comando ./ltib –c e selecionar as configurações necessárias (a placa alvo, o kernel a ser compilado, etc.) .

O Ltib deve ser usado para versões inferiores a versão 3.0, para versões superiores (como a 3.4, usada neste trabalho), deve-se baixar o kernel de algum site (como o kernel.org, por exemplo), descompactar o arquivo, e ir diretamente na pasta root, e usar os seguintes comandos:

- export ARCH=arm
- export CROSS_COMPILE=arm-none-linux-gnueabi-
- export PATH="$PATH:/opt/freescale/usr/local/gcc-4.4.4-glibc-2.11.1-multilib-1.0/arm-fsl-linux-gnueabi/bin/" (esse comando varia dependendo da versão da toolchain que foi instalada).

Depois, deve-se realizar um make do arquivo de configuração (por exemplo, make imx_v4_v5_defconfig).

Finalmente, deve-se fazer make menuconfig (para selecionar as opções adequadas, como a placa alvo e os drivers corretos de comunicação ethernet), e então make uImage.

Tanto uImage quando zImage vão estar localizados em <kernel diretório>/arch/arm/boot.

APÊNDICE D – Estabelecer comunicação tftp com a placa

Para se instalar o tftp, deve-se utilizar os seguintes comandos:

- sudo apt-get install openbsd-inetd
- sudo mkdir /tftpboot
- sudo chmod a+x /tftpboot

Após isso, no diretório /etc/xinetd.d/, deve-se editar o arquivo tftp (sudo gedit /etc/inetd.conf), adicionando as seguintes linhas:

```
service tftp
{
  socket_type = dgram
  protocol = udp
  wait = yes
  user = root
  server = /usr/sbin/in.tftpd
  server_args = /tftpboot
  disable = no
  per_source = 100 2
  flags = IPv4
}
```

Finalmente, usa-se o comando sudo /etc/init.d/xinetd restart

APÊNDICE E – Configurar o NFS

Durante a compilação do ltib (depois de ter instalado o mesmo, e usado o comando ./ltib, como visto anteriormente), uma pasta chamada rootfs é criada dentro do ltib, sendo ela a pasta que servira de sistema de arquivos para a nossa placa.

```
sudo apt-get install nfs-kernel-server
sudo ln -s /home/acahua/freescale/ltib/rootfs /tftpboot/rootfs
sudo gedit /etc/exports
#add the following line /tftpboot/rootfs/ *(rw,no_root_squash,no_subtree_check,async)
sudo /etc/init.d/nfs-kernel-server restart
```

Figura 28 - Scrip para a instalação do NFS

Além disso, caso o kernel tenha sido compilado usando make, deve-se assegurar que foi selecionado, dentro das configurações, a opção referente ao NFS. Se o kernel também foi compilado usando o ltib, deve-se selecionar a opção “Options” em “Target Image Generation”, e, finalmente, seleciona-se “NFS only”.

Caso tenha sido optado pelo sistema JFFS2, deve-se então criar o sistema de arquivos em modo JFFS2 para que ele possa ser passado para a placa. Isso é feito usando o comando: “sudo mkfs.jffs2 -r rootfs -e 64 -s 0x1000 -n -o rootfs.jffs2”. Deve-se atentar para selecionar o modo JFFS2 na compilação do kernel, tanto pelo método usando o ltib quanto pelo método usando o comando make.

APÊNDICE F – Configurar o Bootloader (Redboot)

Freescale recomenda a utilização de um dos seguintes bootloaders: uboot (que utiliza uImage) ou redboot(que utiliza zImage). Como o redboot já vem instalado na placa, por questões de praticidade, ele foi o escolhido para a continuação do desenvolvimento do trabalho.

Ao ligar a placa, se não for efetuado o comando “Ctrl+c” o bootloader se executara automaticamende, usando a ultima configuraçao salva. Para ver a configuração que ele está usando atualmente, usa-se o comando “fc –l”, e para editá-la, usa-se o comando “fc”.

Caso o NFS esteja sendo utilizado, deve-se usar os seguintes comandos :

- load -r -b 0x100000 /tftpboot/zImage
- exec -c "noinitrd console=ttymxc0,115200 root=/dev/nfsroot nfsrootdebug nfsroot=192.168.0.1:/home/acahua/freescale/ltib/rootfs/ rw ip=192.168.0.2"

Caso o JFFS2 esteja sendo utilizado, deve-se usar os seguintes comandos:

- fis init –f (Limpa a memoria Flash)
- load -r -b 0x100000 /tftpboot/zImage
- fis create -f 0x300000 kernel
- load -r -b 0x100000 /tftpboot/rootfs.jffs2
- fis create -f 0x800000 root
- fis load kernel
- exec -c "noinitrd console=ttymxc0,115200 root=/dev/mtdblock2 rw ip=none rootfstype=jffs2"

Finalmente, passa-se o kernel para a placa. Nesta etapa, é importante notar que quando a placa inicia, ela reinicia o ip do host, então devemos continuamente manter configurar o ip do host usando o comando “ifconfig eth0 ip”, sendo eht0 a porta 0 (porta que está conectada a placa) e “ip” o ip dado ao host. Outra maneira, é criar uma configurar uma conexão ethernet no linux com o referido ip. No caso do NFS, é obrigatorio ter conexao com a internet;

Para configurar o Redboot para que o boot seja automatico, basta colocar na configuracao os 2 ultimos comandos apresentados anteriormente (tanto para o JFFS2 quanto para o NFS).

APÊNDICE G – Instalação do CyclicTest

O download do codigo source do cyclictest pode ser realizado no seguinte site: https://rt.wiki.kernel.org/index.php/Cyclictest. Após isso, deve-se descompactar o arquivo, entrar na pasta do cyclictest, e então realizar os seguintes comandos:

- export CC=arm-none-linux-gnueabi-gcc
- export LD=arm-none-linux-gnueabi-ld
- export AR=arm-none-linux-gnueabi-ar

Feito isso, pode-se compilar o arquivo usando o comando make all. Com o cyclictest compilado, é necessário copiar o arquivo para o sistema de arquivos da nossa placa. Isso pode ser feito antes de colocar o sistemas de arquivos na placa, simplesmente copiando o arquivo para a pasta /<diretorio do sistema de arquivos>/usr/bin. Outra maneira, seria passar o arquivo para a placa usando a porta serial.

APÊNDICE H – Instalação da troca de arquivos por serial

Primeiramente, o pc host deve possuir o pacote lrzsz, então, deve ser feito o seguinte comando: sudo apt-get install lrzsz.

Após isso, deve ser instalado o lrzsz na placa. Pode-se fazer o download do arquivo no seguinte site: http://ohse.de/uwe/software/lrzsz.html. Após isso, deve-se descompactar o arquivo e utilizar o comando ./configure na pasta do lrzsz. Com esse comando, são criados diversos arquivos Makefile.

Deve-se editar os Makefiles da pasta root, lib, e src, fazendo as seguintes modificações:

CC = arm-none-linux-gnueabi-gcc

CPP = arm-none-linux-gnueabi-gcc –E

Alem disso, deve-se editar a pasta /intl fazendo as seguintes modificacoes:

AR = arm-none-linux-gnueabi-ar

CC = arm-none-linux-gnueabi-gcc

RANLIB = arm-none-linux-gnueabi-ranlib

Agora, basta dar o comando make, e make install na pasta do lrzsz.

Feito isso, é necessário copiar o arquivo para o sistema de arquivos da nossa placa. Isso pode ser feito antes de colocar o sistemas de arquivos na placa, simplesmente copiando o arquivo para a pasta /<diretorio do sistema de arquivos>/usr/bin. Outra maneira, seria passar o arquivo para a placa usando a porta serial.

APÊNDICE I – Criação das partições da memoria Flash

Deve-se editar o arquivo arch/arm/mach-imx/mach-mx25_3ds.c, modificando a estrura mxc_nand_platform_data, e adicionando a estrutura mxc_nand_partitions, como mostrado abaixo:

```
static struct mtd_partition mxc_nand_partitions[] = {
{
  .name = "nand.bootloader",
  .offset = 0,
  .size = 3 * 1024 * 1024},
{
  .name = "nand.kernel",
  .offset = MTDPART_OFS_APPEND,
  .size = 5 * 1024 * 1024},
{
  .name = "nand.rootfs",
  .offset = MTDPART_OFS_APPEND,
  .size = 256 * 1024 * 1024},
{
  .name = "nand.configure",
  .offset = MTDPART_OFS_APPEND,
  .size = 8 * 1024 * 1024},
{
  .name = "nand.userfs",
  .offset = MTDPART_OFS_APPEND,
  .size = MTDPART_SIZ_FULL},
};

static const struct mxc_nand_platform_data
mx25pdk_nand_board_info __initconst = {
        .width          = 1,
        .hw_ecc         = 1,
        .flash_bbt      = 1,
        .parts          = mxc_nand_partitions,
        .nr_parts       = ARRAY_SIZE(mxc_nand_partitions)
};
```

APÊNDICE J – Script para o tratamento dos dados

Foi feito o seguinte script usando a linguagem *phyton* para tratar os dados e gerar os gráficos necessários para a realização deste trabalho. Esta linguagem foi escolhida tendo em vista que se trata de uma linguagem muito intuitiva, fácil de programar, contendo uma biblioteca de funções muito útil para o problema em questão. Além disso, a linguagem e seu interpretador estão disponíveis tanto para PC Linux quanto para PC Windows, sendo um programa leve, cujo download pode ser feito no seguinte site: http://www.python.org/getit/.

O script toma como entrada um arquivo de texto (.txt), sendo que para alterar o arquivo basta modificar a linha indicada pelo “+” (no caso, está sendo analisado o arquivo “test_34rtrr_3thread.txt”), como pode ser observado pela Figura 29.

```
from scipy.stats import norm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import numpy
+with open('test_34rtrr_3thread.txt', 'r') as f:
    myNames = f.readlines()
    threads = []
    latencies = []
    numbers = []
    for i in range(4,len(myNames)-1):
        aux = myNames[i].replace(' ','').replace('\n','')
        r = 0
        thread = ''
        num = ''
        latency = ''
        if aux <> '':
            for char in aux:
                if char == ':':
                    r += 1
                elif r == 0:
                    thread+=char
                elif r == 1:
                    num+=char
                elif r == 2:
                    latency+=char
            threads.append(int(thread))
            numbers.append(int(num))
            latencies.append(int(latency))
n = max(threads)+1
t = [[] for x in xrange(n)]
for i in range((len(numbers))):
    t[threads[i]].append(latencies[i])
f, ax = plt.subplots(n,sharex=True)
a = []
b = []
c = []
d = []
xl = ''
xlhist = ''
for i in range(n):
    ax[i].plot(t[i])
    a.append("Avg: "+str(sum(t[i])/len(t[i]))+"us ")
    b.append("Max: "+str(max(t[i]))+"us ")
    c.append("Min: "+str(min(t[i]))+"us \n")
    xl += "t"+str(i)+": "+a[i]+b[i]+c[i]
plt.xlabel(xl)
fhist, axhist = plt.subplots(n,sharex=True)
for i in range(n):
    axhist[i].hist(t[i])
    d = "u: "+str(numpy.std(t[i]))+"\n"
    xlhist += "t"+str(i)+": "+a[i]+d[i]
plt.xlabel(xlhist)
plt.show()
```

Figura 29 - Script pata tratamento dos dados

APÊNDICE K – Adicionando o pacote tempo real

O download do pacote pode ser efetuado no seguinte site: https://www.kernel.org/pub/linux/kernel/projects/rt/. Para adicioná-lo ao kernel, basta descompactar o arquivo, entrar na pasta do kernel e usar o seguinte comando: patch -p1 < localização_do_patch.