MIGRAÇÃO DE UM SOFTWARE DE CONTROLE DE ATITUDE E DE ÓRBITA PARA UM SISTEMA OPERACIONAL E UM PROCESSADOR DE TEMPO REAL

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA (PIBITI/CNPq/INPE)

João Marcos Alves Ballio Barreto (ETEP, Bolsista PIBITI/CNPQ)
jonnyabbarreto@hotmail.com

Francisco Carlos de Amorim III (MECTRON, co-Orientador)
amorim@mectron.com.br

Marcelo Lopes de Oliveira e Souza (DMC/ETE/INPE, Orientador)
marcelo@dem.inpe.br

São José dos Campos
Julho de 2012

RESUMO

Com os novos inventos da sociedade, como satélites, máquinas de automação, naves, entre outros, foi necessário criar um novo tipo de computação, que considere e respeite as restrições temporais requeridas por tais aplicações: a **Computação em Tempo Real.** Esta é majoritariamente aplicada aos **sistemas de controle** e minoritariamente aplicada aos sistemas de **mídia múltipla ou iterativa**. E isto porque nestas, o atraso ou adiantamento de qualquer tarefa pode, por exemplo, acarretar: 1) a perda de uma **transação** em um banco; 2) a perda da **missão** de um satélite; 3) a perda de um **veículo** não tripulado; ou até mesmo 4) a perda de **vidas humanas**. Por isso tornam-se tão importantes os estudos e pesquisas feitas a respeito de sistemas que utilizam a computação em tempo real. Este trabalho estuda a migração de um software de controle de atitude e de órbita para um sistema operacional e um processador de tempo real. Para isto, o trabalho estuda: 1) as características de sistemas operacionais em tempo real; 2) como escolher um sistema operacional que permita a mudança para um sistema operacional de tempo real; 3) como escolher um processador para uma futura migração do software de controle de atitude e de órbita, após testes e análises dos resultados numéricos. Em particular o trabalho estuda o sistema operacional RTEMS e o processador ERC32 a serem usados na PMM. Este trabalho está em andamento, mas espera-se atingir todas as etapas até realizar a análise comparativa entre: 1) os resultados obtidos pelo sistema operacional de tempo real rodando o software migrado sobre um **emulador** do processador; e 2) os resultados obtidos pelo sistema operacional de tempo real rodando o software migrado sobre um **processador** de tempo real para avaliar se o processador, o sistema operacional, e o software migrado, estarão aptos para serem verificados e validados visando seu uso em vôo.

Sumário

Lista de Figuras

1. Introdução

1.1. Contexto

Este trabalho é a continuação de uma sequência de trabalhos iniciada por Marcelo de Lima Bastos Moreira [1], aperfeiçoada por Francisco Carlos de Amorim III [2], e parcialmente continuada pelo ex-bolsista Denis Francisco Simões Donizete [3]. Por se tratar de um assunto de grande complexidade, este trabalho ainda está em andamento.

Os testes feitos num processador demandam mais tempo se comparados com os testes feitos num seu emulador. Por isto, o documento escrito por Amorim III [2], menciona o desenvolvimento e os testes feitos sobre: 1) um Sistema Operacional de Tempo Real (SOTR) RTEMS versão 4.0.0; instalado sobre 2) um emulador SPARC Instructions Simulator (SIS) versão 3.0.5; auxiliado por 3) um Sistema de Compilação Cruzada para o ERC32 (ERC32CCS) versão 2.0.6; instalado sobre 4) o Sistema Operacional (SO) Linux Mandriva 10.1 com kernel versão 2.6.17; instalado sobre 5) um PC com um processador INTEL Pentium Dual Core D820 com relógio de 2.8 GHz.

Por várias razões, neste trabalho, o desenvolvimento e os testes serão feitos sobre: 1) um Sistema Operacional de Tempo Real (SOTR) RTEMS versão 4.10.2; instalado sobre 2) um emulador SPARC Instructions Simulator (SIS) versão 3.0.5; auxiliado por 3) um Sistema de Compilação Cruzada para o ERC32 (ERC32CCS) versão 2.0.6; instalado sobre 4) o Sistema Operacional (SO) Linux Fedora 14 com kernel versão 2.6.17; instalado sobre 6) o Vmware Player; instalado sobre 7) o Sistema Operacional Windows XP; instalado sobre 7) um PC com um processador INTEL Pentium Dual Core D820 com relógio de 2.8 GHz. A Figura 1 descreve a arquitetura de instalação para este trabalho.

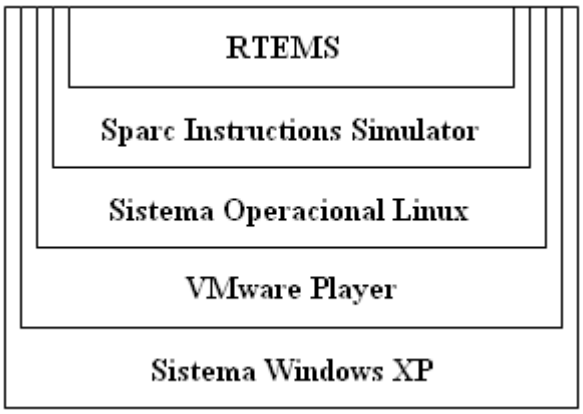

Figura 1 – Arquitetura de Instalação dos Programas deste Trabalho.

A seqüência de passos para se chegar à execução do RTEMS, não está detalhada no trabalho de Amorim III [2]. Por isto, os passos tiveram que ser adaptados seguindo alguns passos deixados pelo trabalho de Amorim III [2] e continuados por Donizete [3]. Este teve que começar a colocar em seqüência os passos para a instalação dos SOTR RTEMS. E esta continua sendo desenvolvida neste trabalho.

1.2. Motivação

Este trabalho se baseia na grande necessidade de conhecimentos sobre o SOTR RTEMS que será executado no processador ERC 32, da Plataforma MultiMissão (PMM), projeto que está sendo desenvolvido pelo Instituto Nacional de Pesquisas Espaciais (INPE).

Este estudo abrange uma área que fica em destaque dia após dia, a Computação em Tempo Real, motivada pela necessidade de executar tarefas com características temporais especificadas, especialmente prazos fatais de execução (“deadlines”).

Particularmente, o atraso ou adiantamento das tarefas (como dito anteriormente) pode gerar perdas, como: 1) a perda de uma transação em um banco; 2) a perda da missão de um satélite; 3) a perda de um veículo não tripulado; ou até mesmo 4) a perda de vidas humanas.

Alguns exemplos de sistemas nos quais são utilizados o conceito de Tempo Real, são: sistemas embarcados, sistemas de automação, sistemas de controle, entre outros.

1.3. Objetivo

O objetivo deste trabalho é estudar a migração de um software de controle de

atitude e de órbita para um sistema operacional e um processador de tempo real. Para isto são necessários os seguintes passos:

1) Entender o processo de Integração, Verificação e Validação de Softwares, e também entender suas aplicações em Softwares de Controle de Atitude e de Órbita.
2) Estudar pelo menos um dos softwares e ambientes originais existentes, que foram utilizados no projeto da PMM pela Divisão de Mecânica Espacial e Controle (DMC) do INPE, e a “Migração de Softwares” de “SCAOs” para um “Sistema Operacional e um Processador de Tempo Real”.
3) Testar e simular o software migrado nas mesmas condições do software original para um SCAO de um satélite de interesse como a PMM.
4) Documentar toda a pesquisa feita, até então, para que se possa dar continuidade, e não retomar o estudo do zero.

2. Fundamentação Teórica

2.1. Integração, Verificação e Validação (IVV)

Para o sucesso de um grande projeto, como é o caso da PMM, não podemos apenas contar com um excelente grupo de trabalho, um excelente grupo pode resultar em benefícios para projeto, mas pode vir a não concretizar um sistema confiável, ou um sistema menos suscetível a falhas, ou até não entregar um projeto no devido prazo estabelecido.

Os processos de Integração, Verificação e Validação visam melhorar no desenvolvimento de soluções para problemas diversos, estes processos também podem ser chamados de “Migração”.

Existem benefícios obtidos quando estes processos são aplicados, entre eles estão: a diminuição do tempo de entrega do projeto finalizado, a diminuição do custo de um projeto, a diminuição de possíveis falhas que pudessem vir a ocorrer nas fases de término do projeto.

Vale ressaltar que estes métodos são aplicáveis a todos os projetos na atualidade, desde os mais complexos e robustos até os mais simples.

É importante saber que a perfeita cronologia entre essas três etapas resultará em uma perfeita linearidade de resolução do projeto.

2.1.1. Decomposição em Partes

Todo o projeto possui áreas de que são agrupadas logicamente ou fisicamente, e estas podem ser dividas para uma melhor compreensão dos fatos e de cada trabalho realizado pelo grupo.

Para que os processos de IVV possam ser aplicados a um projeto, este deverá passar por uma decomposição como é o caso da Figura 2, na qual especialistas se dividem para atender cada parte que foi decomposta. Também para facilitar sua resolução pode cada parte ser decomposta novamente e sofrer outro processo de IVV, cabe a cada equipe decidir a melhor forma de resolver as partes.

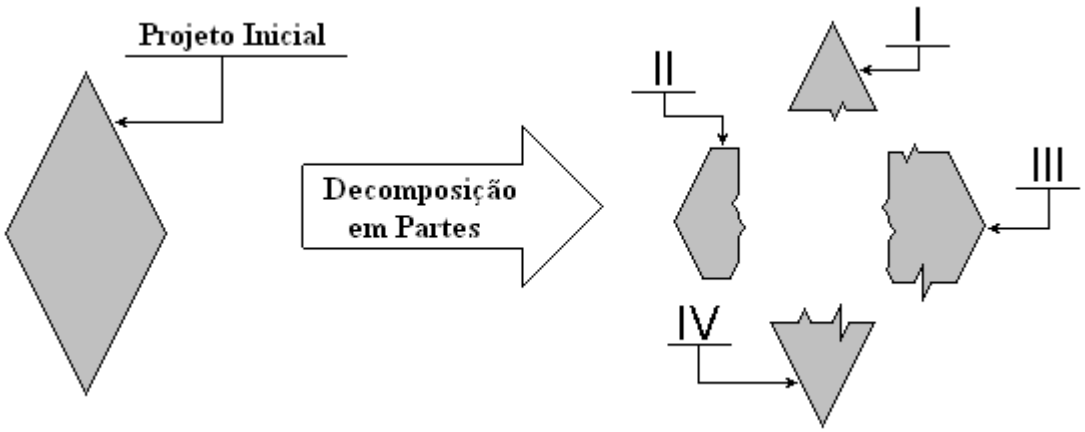

Figura 2 – Exemplo Teórico de Decomposição em Partes.

2.1.2. Integração

A integração fornece a união entre as partes que previamente foram decompostas.

A importância desta etapa é que todas as partes decompostas poderão ser mais bem visualizadas para caso haja uma busca minuciosa a procura de futuras ocorrências de erros. Caso o projeto tivesse uma resolução centralizada não se poderia decompor a resolução para melhor estudo.

Este estágio será responsável apenas pela união das partes, pois também facilitará o estágio seguinte.

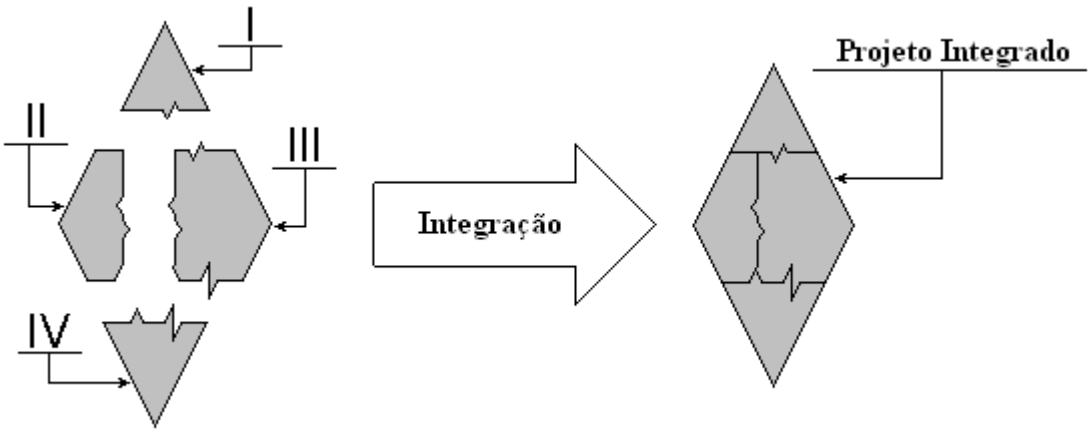

Figura 3 – Exemplo Teórico de Integração.

2.1.3. Verificação

A decomposição em Partes permite um melhor entendimento das partes, logo

sua resolução tende a ficar mais fácil. Contudo, temos o caso destas partes gerarem interferências uma nas outros, fazendo com que erros sejam gerados por estas interferências.

No estágio de verificação existe uma analise entre os códigos que previamente foram integrados. A verificação permitirá avaliar se uma entrada correta irá gerar uma saída correta, ou seja, permitirá uma visualização dos códigos conectados entre si, para verificar se todos funcionam de forma certa, na Figura 3 está um exemplo teórico de como seria uma verificação.

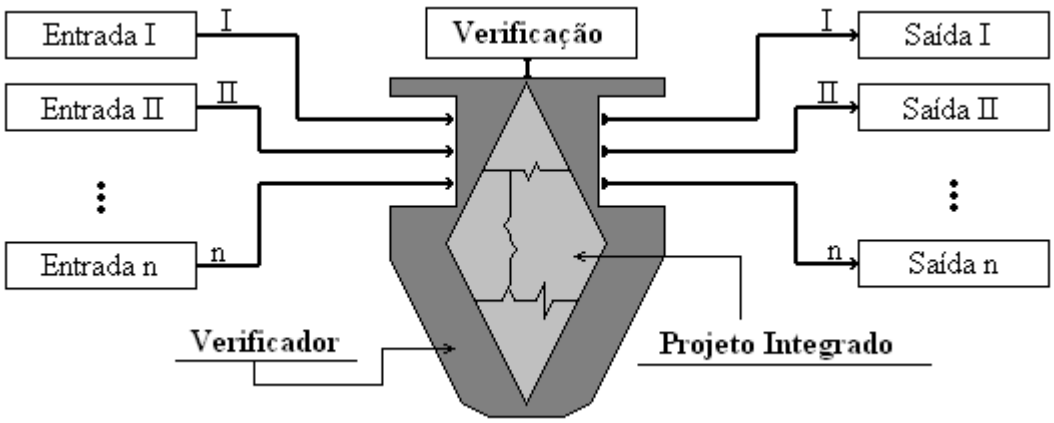

Figura 4 – Exemplo Teórico de Verificação.

A grande importância da verificação de um software é que ela reduz os problemas que podem ser gerados, na etapa de validação.

2.1.4. Validação

Um dos grandes perigos para a construção de um software é que tudo aquilo que posteriormente foi Decomposto, Integrado e Verificado pode vir a não ser válido ou adequado ao equipamento alvo onde se (no caso de um software) instalará o software, eu já esta Verificado, o que faria com que o projeto fosse inadequado para uso.

A etapa de validação fornece a adequação da resolução, para verificar se esta é valida para o fim desejo.

Todo o projeto desenvolvido desde as etapas de Integração e Verificação pode vir a não atender as especificações donde este projeto será instalado. Por isso nesta etapa tudo o que foi verificação passa por um processo de adequação, para permitir que

todas as partes sejam validas ao Dispositivo Alvo (Target Hardware).

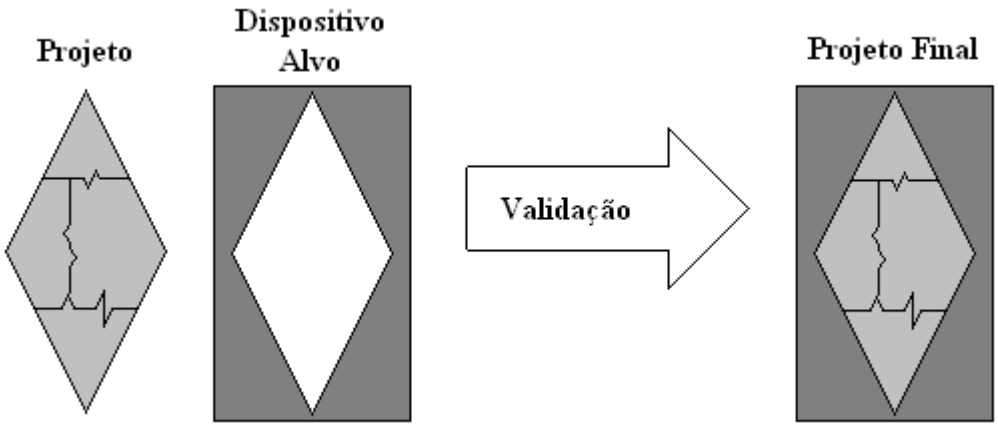

Figura 5 – Exemplo Teórico da Validação.

2.1.5. IVV nos SCAO da PMM

O caso da PMM se trata de um exemplo prático destes processos de Migração. Estes processos são utilizados na composição dos Softwares de Controle de Atitude e Órbita utilizados na PMM. Os softwares de controle foram decompostos para grupos de especialistas e cada especialista verifica e testa estes softwares em máquinas, logo após existe a integração em um SOTR, e desta forma pode-se então partir para os processos de verificação e validação. O processo de verificação será efetuado em um Sistema Operacional Linux, e após isto será validado no processador ERC 32.

2.1. Software de Controle de Atitude e de Órbita (SCAO)

Os softwares são seqüências lógicas predefinidas executadas por um computador para um determinado fim desejado, como é o caso dos softwares de controle de órbita e atitude.

2.1.1. Sistemas de Controle de Atitude

Os softwares de controle de atitude permitem que o veículo espacial, tenha uma rotação em torno de si, permitindo assim girar para um lugar desejado.

Podemos citar como exemplo um software que faz com que:

- Um satélite aponte sua câmera (exemplo fotográfica) para a terra;

- Os painéis solares de um satélite apontem em direção ao sol.

Lembrando que este software fica responsável por manter o “apontamento” do satélite independente de sua órbita atual e das perturbações externas.

2.1.2. Sistemas de Controle de Órbita

Os softwares de controle de órbita desempenham a função de controlar velocidade e localização espacial do veículo.

Desta forma garantindo que independente dos fatores como a gravidade de corpos celestes, perturbações externas, ou até mesmo a luz solar, venham a mover o veículo espacial da órbita desejada.

2.1.3. Criação dos Softwares de Controle

Muitos softwares para controle criados na atualidade são utilizados em lugares de alto risco (podendo ser risco materiais, ou até humanos), alem de serem mais robustos, necessitando de melhor desempenho do processador.

Por estes motivos existem softwares que permitem a criação do software de controle de atitude e órbita mais sua analise gráfica e a amostragem de seu desempenho. Diminuindo a quantidade de erros, que poderiam vir acontecer em fases de teste.

Os SCAOs são desenvolvidos em C (no caso da PMM), pois a linguagem C é compatível ao SOTR RTEMS.

2.2. Tempo Real

O conceito de “Tempo Real” na computação representa algo, que exige uma resposta num tempo determinado, ou num certo prazo limitado, geralmente quase sempre curto.

O exemplo disto é o caso dos recursos de um piloto automático, que necessitam de atualizações constantes e com frequência, como os recursos de velocidade, altitude e direção. Estas ações nunca poderão esperar que outras ações não tão importantes sejam verificadas.

2.2.1. Sistemas Operacionais de Tempo Real (SOTR)

Os Sistemas Operacionais em Tempo Real (SOTR) são desenvolvidos para atender as especificações temporais de todos os programas embutidos neles.

Existem dois tipos de SOTR, que possuem características iguais, mas o seu funcionamento é diferente. No caso da PMM, o que será usado é o “SOTR Crítico” devido a suas características.

2.2.1.1. Não Críticos

Os SOTR não críticos não garantem que todas as tarefas sejam executadas a tempo, e nem se serão executas, pois o descumprimento ocasional de um prazo, nestes sistemas, é válido. Nestes sistemas o descumprimento dos prazos é valido, pois não existe nem risco de perda de informação e nem de vidas.

Alguns exemplos destes sistemas são os sistemas de áudio digital ou multimídia pertencem a essa categoria.

2.2.1.2. Críticos

Os SOTR críticos garantem que todas as tarefas serão concluídas, cada uma no seu tempo, e sempre no prazo certo. Nestes sistemas o descumprimento dos prazos pode levar a perda de veículos aéreos e até em vidas humanas.

Alguns exemplos destes sistemas são os encontrados nos ambientes de robótica, aviação, e também em sistemas de controle.

2.3. Processadores

Os processadores têm uma das funções mais importante em uma maquina, assim como o cérebro mantém todos os órgãos conectados e interagindo entre si, assim é o processador.

O processador permite que todos os atuadores, sensores, o software da PMM, e os demais periféricos da PMM, estejam conectados, e interagindo entre si. Por isso é de extrema importância um estudo detalhado para verificar o melhor clock do processador.

Também fica para a função do processador interpretar e executar todas as funções desejadas para este, ou executar os softwares designados para o processador.

2.3.1. Compiladores

Os compiladores são programas que possuem a função de converter um programa com linguagem de alto nível para um programa com linguagem de baixo nível. Desta forma um processador poderá interpretar, ou executar, um arquivo que tenha sido compilado.

Neste estudo estaremos convertendo um programa com Linguagem C, para um programa com Linguagem de Máquina.

2.3.1.1. GCC (GNU Compiler Collection)

O GCC foi desenvolvido para ser um compilador do sistema operacional GNU.

Este permite a compilação de várias linguagens de programação, incluindo a linguagem usada no SOTR RTEMS, a C.

Adota a política de código aberto, permitindo assim as alterações, que serão efetuadas para a compilação do SOTR RTEMS.

Para que seja possuí a utilização do SIS (que será explicado em seguida) será necessária a instalação do ERC32CCS, para desta forma podermos compilar o arquivo RTEMS para futura execução deste no simulador SIS.

2.3.2. Depuração

A depuração é uma das formas mais confiáveis para se analisar aquilo que foi programado.

A depuração permite analisar todo o código que foi programado. Desta forma é possível analisar o código a procura de erros lógicos, que existem no código.

2.3.2.1. GDB (GNU Debugger)

O GDB é o depurador do Projeto GNU, e por isto adota a política de código aberto, necessário para as adaptações para a depuração do RTEMS.

Este permite a depuração de várias linguagens de programação, incluindo a linguagem usada no RTEMS, a C.

2.3.2.2. Sparc Instructions Simulator (SIS)

O SIS é um simulador que emula o ERC 32 permitindo então fazer a execução do SOTR RTEMS. Desta forma teremos uma simulação mais próxima da que será feito futuramente no verdadeiro processador ERC32.

Este simulador foi escolhido, pois permite trabalhar em conjunto com o GDB, desta forma ao executarmos o RTEMS com o SIS e o GDB podemos simular um RTEMS no ERC32 fazendo a sua depuração para melhor detalhamento.

Tanto o GDB quanto o SIS serão instalados sobre a máquina virtual, Fedora 14.

2.4. ERC 32

Para que algo, como um veículo espacial, seja usado no espaço, todos os equipamentos deste devem ser desenvolvidos para agüentarem todas as periculosidades do lugar onde o equipamento funcionará.

Deste mesmo modo acontece com os processadores. Os processadores usualmente usados atendem a especificações para serem usados em condições normais, ou seja, dentro da atmosfera da terra.

Por não estarem dentro da atmosfera todo aparelho espacial esta sujeito a interferências da radiação proveniente da atividade solar e dos raios cósmicos. Alem de estarem expostos a eventos cósmicos mais raros, como as erupções de raios gama.

Por dois motivos o processador escolhido é o ERC 32 (32 - bit Embedded Real-time Computing Core):

- O Processador ERC 32 possui tolerância a níveis de radiação.
- O INPE conta com a existência de um kit VME SPARC RT da THARSIS INC. disponível para programação e desenvolvimento do ERC 32, emprestado pela Divisão de Eletrônica Aeroespacial (DEA) à Divisão de Mecânica Espacial e Controle (DMC).

Figura 6 – Placa ERC32 / VME Sparc da Tharsys. Fonte: Donizete (2011).

Figura 7 – Hack contendo o kit VME SPARC RT, THARSIS INC. Fonte: Donizete (2011).

2.5. SOTR RTEMS

O Sistema Operacional de Tempo Real RTEMS teve seu nome inicialmente como Real-Time Executive for Missile Systems, pelo motivo básico para o qual era desenvolvido, controlar em tempo real a trajetória de um míssil.

Contudo após séries de estudos sobre o RTEMS foi verificado que este SOTR não somente se adequava a projetos para controlar mísseis, mas se adequava em outros projetos em que era necessário um SOTR. Em decorrência disto a sua sigla passou a significar Real-Time Executive for Multiprocessor Systems (RTEMS).

```c
#include <rtems.h>
rtems_task user_application(rtems_task_argument argument);
rtems_task init_task(rtems_task_argument ignored) {
    rtems_id tid;
    rtems_status_code status;
    rtems_name name;
    name = rtems_build_name( 'A', 'P', 'P', '1' )
    status = rtems_task_create(
        name, 1, RTEMS_MINIMUM_STACK_SIZE,
        RTEMS_NO_PREEMPT, RTEMS_FLOATING_POINT, &tid
        );
    if ( status != RTEMS_STATUS_SUCCESSFUL ) {
        printf( "rtems_task_create failed with status of %d.\n", status );
        exit( 1 );
    }
    status = rtems_task_start( tid, user_application, 0 );
    if ( status != RTEMS_STATUS_SUCCESSFUL ) {
        printf( "rtems_task_start failed with status of %d.\n", status );
        exit( 1 );
    }
    status = rtems_task_delete( SELF );
    printf( "rtems_task_delete returned with status of %d.\n", status );
    exit( 1 );
}
rtems_task user_application(rtems_task_argument argument) {
    while ( 1 ) { /* Laço Infinito */
        /* Codigo para as aplicações são implementados aqui. */
    }
}
#define CONFIGURE_TEST_NEEDS_CONSOLE_DRIVER
#define CONFIGURE_TEST_NEEDS_CLOCK_DRIVER
#define CONFIGURE_MAXIMUM_TASKS 2
#define CONFIGURE_INIT_TASK_NAME rtems_build_name( 'E', 'X', 'A', 'M' )
#define CONFIGURE_RTEMS_INIT_TASKS_TABLE
#define CONFIGURE_INIT
#include <confdefs.h>
```

Figura 8 – Exemplo de um SOTR RTEMS.

Podemos hoje em dia ver SOTR RTEMS em muitos outros lugares distintos, e um dos mais importantes para este estudo, é os que são adaptados e usados em satélites.

A Figura 8 segue um exemplo de um SOTR RTEMS simples, sem nenhum acréscimo de funções para o processador executar. As funções, para serem executadas pelo processador, são geralmente implementadas dentro do laço infinito.

2.5.1. Benefícios

O motivo pela qual o SOTR RTEMS foi escolhido para o projeto da Plataforma Multi-Missão (PMM), é que:

- É um software que não necessita de taxa de licença;
- Este software pode ser adaptado para qualquer projeto;

- É compatível com o processador ERC 32;
- Fornece recursos completos de gerenciamento de tarefas, interrupções, tempo e múltiplos processadores, além das características típicas de sistemas operacionais genéricos;
- E Pode ser implementado na linguagem C.

2.6. Linguagem de Programação

As Linguagens de Programação são usadas para desenvolver programas para um determinado fim desejado. Existem dois tipos de linguagens de programação: as linguagens de baixo nível, e as linguagens de alto nível.

2.6.1. Linguagem C

A linguagem C está sendo utilizada no projeto da PMM, pois o SOTR RTEMS possui compatibilidade com a linguagem C.

Existe também a compatibilidade com os programas de compilação e depuração, respectivamente GCC e GDB.

2.7. Migração de um SCAO

Para melhorar a criação de um software de controle de atitude e órbita é necessário contar com softwares que permitirão ajudar nesta criação.

Um software de controle de atitude e órbita pode ser criado em um bloco de notas, e testado em um compilador como o GCC (GNU Compiler Collection). Entretanto este método é demorado e não permite visualizações rápidas para o programador.

Para que a criação do software de controle de atitude e órbita seja muito mais fácil, alguns softwares vendidos na atualidade permitem que o programador, desenvolva com mais facilidade, pois estes permitem testes e visualizações com mais velocidade e praticidade. Programas como o MATRIXx, que permitem alem da simulação (consequentemente um estudo mais aprimorado dos casos), a geração do código para a linguagem C.

2.8. VMware Player

VMware Player é um programa para criação e emulação de máquinas virtuais.

Figura 9 – Ambiente do VMware Player.

Este programa permite que outros Sistemas Operacionais sejam instalados na mesma máquina (neste estudo será usado o Sistema Operacional Fedora 13).

Não necessariamente deva ser o VMware Player neste estudo, entretanto este software é gratuito e também é muito simples e prático de ser utilizar.

Na Figura 9 há o ambiente do programa VMware Player, com três maquinas virtuais instaladas.

2.9. SO Linux

Por necessidades de adaptação de código, foi necessário escolher um sistema operacional que se adapte ao SOTR RTEMS.

O Linux possui uma grande vantagem sobre os demais Sistemas Operacionais. Este adota a política de código aberto, permitindo tanto o aprimoramento deste, quanto sua adaptação para outros fins.

Por este motivo usa-se plataforma que derivam do SO Linux. Elas permitem adaptações no GCC e no GDB, que são ferramentas ou programas, que serão utilizados para a execução do RTEMS.

2.9.1. Fedora 13

O SO Fedora 13 foi escolhido para a instalação do RTEMS devido a sua

compatibilidade com o SO Linux. Possui características muito similares ao SO Linux e também possui grandes semelhanças com os códigos para instalação do RTEMS no manual “Getting Started with RTEMS”

Na Figura 10 há o Sistema Operacional Fedora 13 sendo executado sobre o emulador VMware Player.

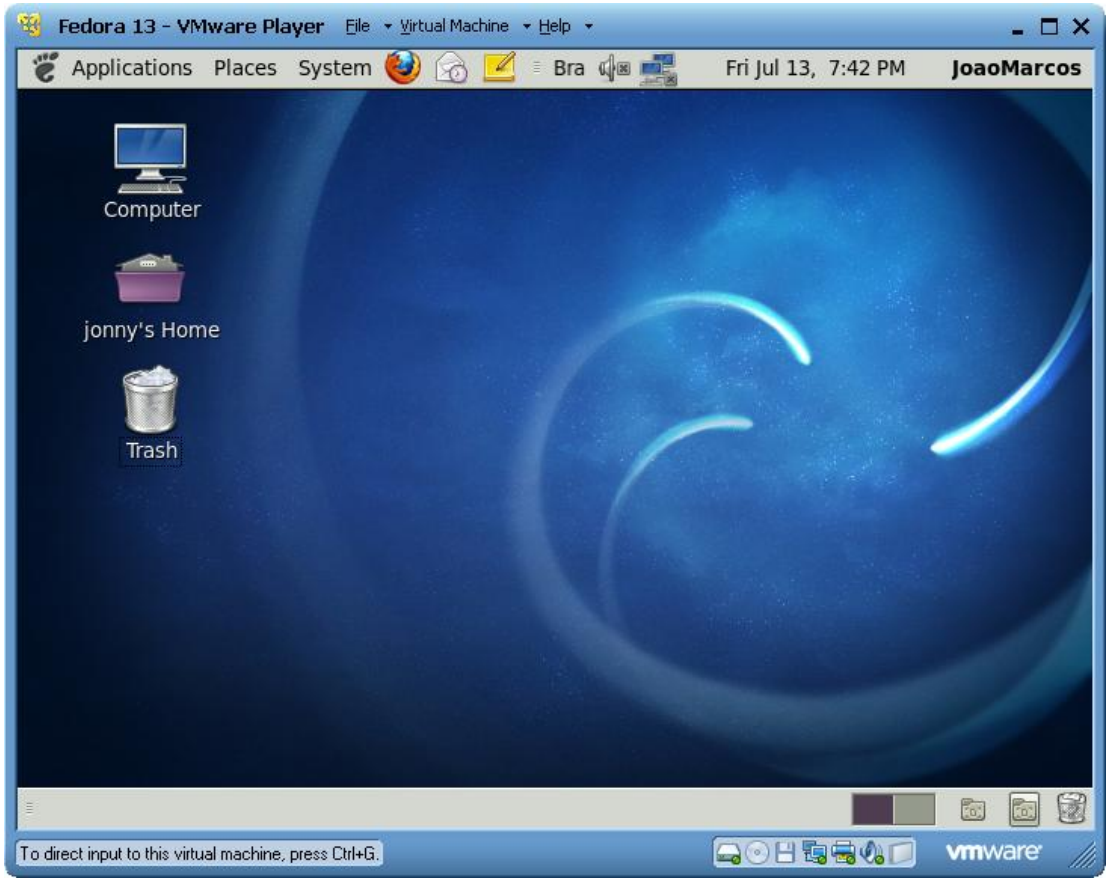

Figura 10 – SO Fedora no emulador VMware Player.

3. Metodologia Aplicada

3.1. Estudos sobre Temas e Referencias

Para que fosse dada continuidade aos estudos feitos por Amorim III, foi necessário um entendimento dos assuntos de:

- Computação em Tempo Real,
- Sistemas Operacionais em Tempo Real,
- Migração (Integração, Verificação e Validação),
- Softwares de Controle de Atitude e de Órbita,

Com o decorrer da metodologia e do estudo será necessário fazer a leitura também das referências [I], [II], [III], [IV], e [V].

3.2. Instalação do VMware Player

Por se tratar de um software livre o seu download é muito simples e também sua instalação.

Esses dois passos foram feitos. Tanto seu download quanto a sua instalação.

O software VMware Player foi instalado sobre um sistema operacional Windows XP, disponibilizado por Francisco Terceiro, localizado no INPE.

3.3. SO Fedora 13

O download e a instalação do sistema operacional Fedora 13, são extremamente simples.

Estes dois passos foram feitos.

3.4. Getting Started with RTEMS

A edição deste manual seguida neste estudo foi o “Edition 4.6.5, for 4.6.5”, e foi tentado fazer a instalação do GCC e do GDB sobre o SO Fedora 13.

4. Estudo de Caso

4.1. Resultados Preliminares

Primeiramente as leituras de algumas das referenciam foram de extrema importância, devido à necessidade de entendimento do assunto para aplicação da metodologia, nem todas as leituras foram ainda feitas devido ao grande enfoque na parte de preparação para executar o SOTR RTEMS.

Os passos referentes à instalação da Maquina Virtual já foram concluídos. O programa VMware Player já foi instalado, e por cima deste foi instalado o Sistema Operacional Fedora 13.

Entretanto o passo seguinte não foi realizado com sucesso, devido a alguns problemas.

Somente após alguns meses de tentativas, erros, análises e pesquisas, foi constatado que as versões do Fedora 13 não são compatíveis á instalação proposta pelo manual de instalação do RTEMS o “Getting Started with RTEMS Edition 4.6.5”.

Este problema foi gerado devido à incompatibilidade do RTEMS 4.6 (que o manual pede para ser instalado) com o Fedora 13.

4.2. Trabalhos Futuros

4.2.1. Instalação do SO Fedora 14

O candidato a ser instalado sobre o Windows XP, através do VMware Player será o Fedora 14, motivos pela qual:

- O Fedora 14 é compatível com a versão 4.10 do RTEMS (download gratuito e uma das versões mais atualizadas).
- E o Fedora 14 é está disponível na internet, sendo este também mais facilmente encontrado na internet.

4.2.2. Instalação do ERC32CCS e do SIS

Pretende instalar o ERC32CCS e o SIS do site da European Space Agency (ESA), pois esta fornece tanto o download quanto os manuais necessários.

4.2.3. Instalação do GDB

Como o depurador possui a função de apenas auxiliar na procura de erros, sua

instalação será efetuada para caso aconteça algum erro de lógica no RTEMS que futuramente será executado.

4.2.4. Escolher um Software de Controle de Atitude e de Órbita

Não será possível a criação de um software de controle de atitude ou de controle de órbita devido ao tempo em que leva para estes serem testados e validados, podendo chegar até meses.

Por isso será necessária a escolha de um software que será usado na PMM, para ser instalado na PMM e, em seguida, ser testado e verificado.

Como a obtenção dos softwares será livre para este estudo, torna-se este caminho mais viável a de se desenvolver um SCAO.

4.2.5. Criação do Sistema Operacional RTEMS

O RTEMS, por ser um sistema operacional de código aberto, já está desenvolvido, entretanto é necessário adapta-lo para o caso da PMM.

É necessário então que os SCAOs sejam implementados no RTEMS, atribuindo suas prioridades e suas restrições temporais.

4.2.6. Compilação e Depuração

Como o GCC e o GDB já estão prontos para serem usados, e o RTEMS já está pronto para testes, então:

- O arquivo contendo o SOTR RTEMS será compilado gerando um arquivo compilado.
- E logo após com este arquivo poderá este ser executado para obtenção dos arquivos que permitirão a leitura e análise.

4.2.7. Analise e Interpretação

E desta forma poderemos então verificar se os dados gerados pelo Sistema Operacional e futuramente pelo Processador ERC 32 seguem os Requisitos da Missão, e seguem as normas estabelecidas para os tais.

5. Conclusão

Apesar das grandes dificuldades encontradas no processo de construção do RTEMS, é visível que a cada novo estudo realizado sobre este assunto fica mais perto da implementação do RTEMS no ERC 32.

De acordo com os relatórios lidos e estudos feitos, tudo aponta para uma bem sucedida instalação dos programas Fedora 14 e RTEMS 4.10.

Sendo assim após isso teremos então em mãos os dados gerados pela execução do RTEMS (com um SCAO, da PMM), no SO Fedora.

E com estes dados gerados em mãos poderemos, então, haver a geração de um relatório de desempenho, de um dos SCAO da PMM.

6. Referências

1) MOREIRA, M.L.B., SOUZA, M.L.O. “Design of an Attitude Control System for the Multi-Mission Platform and its Migration to a Real Time Operating System”. XV Congresso SAE Brasil, São Paulo, SP, 2007 (MS).

2) AMORIM III, F.C., SOUZA, M.L.O. Automatic Generation, Migration, and Tests of a Real Time Code to an Embedded Controller. XVI Congresso SAE Brasil, São Paulo, SP, 2008 (AS).

3) DONIZETE, D.F.S., AMORIM III, F.C., SOUZA, M.L.O. “Relatório Parcial de Projeto de Iniciação Científica (PIBIC/CNPq/INPE)”, INPE, SJC, SP, 2011.

4) ON-LINE APPLICATIONS RESEARCH CORP. “RTEMS C User’s Guide”. Ed.4.7.1, Huntsvile, Al, USA, 2006(OL).

5) NATIONAL INSTRUMENTS. “MATRIXx: AutoCode User Guide”. Austin, Texas, USA, 2004 (NI).

6) SOUZA, M.L.O., TRIVELATO, G.C. “Simulators, and Simulations: Their Characteristics and Applications to the Simulation and Control of Aerospace Vehicles. SAE Brasil Conf., 2003 (ST).

7) KUGA, H.K., RAO, K.R., CARRARA, V. Introdução à Dinâmica Orbital (2ª. ed.). INPE, SJC, SP, 2008.

8) DEITEL, H.M., DEITEL, P.J., CHOFFNES, D.R., “Sistemas Operacionais”, 3ª ed., Editora Pearson / Prentice Hall, Brasil, SP, São Paulo, 2005.

9) ANDREW S.T., “Sistemas Operacionais Modernos”, 2ª ed., Editora Pearson / Prentice Hall, Brasil, SP, São Paulo, 2003.

10) SILBERSCHATZ A., GALVIN P.B., GAGNE G., “Fundamentos de Sistemas Operacionais”, 6ª ed., Editora LTC, Brasil RJ, Rio de Janeiro, 2004.

11) ON-LINE APPLICATIONS RESEARCH CORP. “Getting Started with RTEMS”. Ed.4.6.6. Huntsvile, Al, USA.

12)