Avaliação e comparação do patch Preempt_RT com o RTAI

Aluno: Daniel Barlavento Gomes
Orientador: Paulo Abadie Guedes

Conceitos de “Tempo Real”

- Sistema de tempo real
  - Determinismo
  - Previsibilidade
    - Correção temporal
  - Classificados em Soft (Brandos) e Hard (Rígidos)
- Tarefas de Tempo Real
  - Periódicas
  - Aperiódicas
  - Parâmetros de tempo
  - Prioridade

Sistemas Operacionais de Tempo Real

- Objetivos de um Sistema Operacional de Tempo Real (SOTR)
  - Ser um lorde inglês, meticuloso e pontual, todos os seus passos são previsíveis, faça chuva ou faça sol
- Por quê tantas funcionalidades em um SOTR?
  - Pra te atender melhor!
- Ele não só tem que ser rápido?
  - Não, devem ser pontuais! São como os lordes
- Isso realmente presta? Em que situação?
  - Sim, como todo bom lorde, estão em quase todos os lugares onde existe dinheiro,
  - E estão salvando vidas neste exato momento

Sistemas Operacionais de Tempo Real vs Núcleos de Tempo Real

- SOTR
  - Facilidade de desenvolvimento
  - Suporte a diversos protocolos de comunicação
  - Suporte a diversos dispositivos periféricos
  - Suporte a políticas de segurança
  - Pode ter partes soft real-time
    - Suporte a interface gráfica, etc.
- NTR
  - Velocidade de execução
  - Suporte a restrições temporais extremamente rígidas
  - Tamanho muito reduzido
  - Suporte a diversos microcontroladores
  - Ideais para sistemas embarcados

Latência

- O que é?
  - Tempo decorrido entre o estímulo e a resposta correspondente
- Principais causas em SO
  - Escalonamento
  - Interrupção
  - Inversão de Prioridade
  - Inversão de Interrupção
- Pra quê saber disso?
  - A latência influencia diretamente a previsibilidade do sistema
  - Inconstância = Imprevisível

Objetivos

Avaliar a capacidade do patch Preempt_RT de transformar um PC em um computador capaz atender aos requisitos de uma aplicação de tempo real rígida e comparar os resultados com o RTAI, um sistema maduro, testado e consolidado.

Objetivos

- Verificação da viabilidade de uso do patch
- Configuração e geração do kernel (identificação dos vilões da latência)
- Escrita dos testes para Preempt_RT e RTAI
- Aplicação dos testes
- Avaliação e comparação dos resultados

Trabalhos Relacionados

- Moreira (2007)
  - Compara de forma quantitativa e qualitativa alguns sistemas operacionais disponíveis na época
  - Utilização dos testes propostos como base para os testes deste trabalho
  - Utilização dos resultados obtidos como referência
- Litayem (2011)
  - Avaliação de desempenho do patch Preempt_RT e Xenomai utilizando Cyclictest e UnixBench
- Hallberg (2017)
  - Utiliza o programa Cyclictest na medição de desempenho de kernels padrão e de tempo real para uma aplicação específica

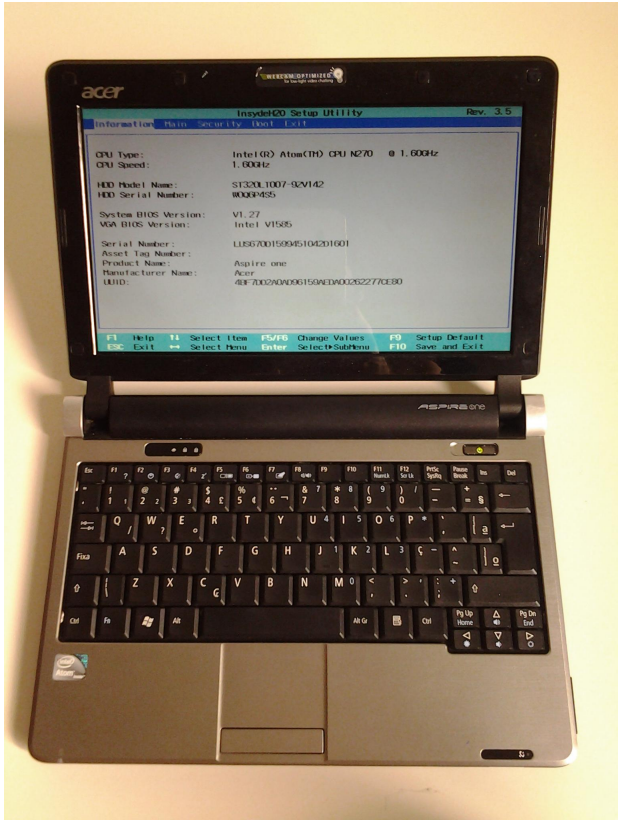
O Patch Preempt_RT

- Projeto suportado oficialmente pelos desenvolvedores do kernel Linux
  - Suporte da comunidade
  - Boa documentação
  - Desenvolvimento bastante ativo
  - Não altera a arquitetura do sistema
- Usa as bibliotecas definidas pelo padrão POSIX
- Suporta nativamente chamadas de sistema sem prejuízos à execução das aplicações de tempo real
- Possibilidade de escrita de programas que executam no modo kernel e modo usuário
- Todo o gerenciamento e comunicação entre processos nativos do Linux
- Suporte a escalonamento FIFO, Round Robin, e EDF

O RTAI

- Acrônimo de Real-Time Application Interface
- Fork do projeto RTLinux
- Utiliza uma solução Dual Kernel
- Camada de abstração de hardware (HAL) usando ADEOS
- Aplicações de tempo real são executadas por um microkernel
- Suporte a várias politicas de escalonamento (EDF, RM, FIFO, RR)
- Controle completo de tarefas de tempo real
- Comunicação entre tarefas por meio de troca de mensagens
- Permite aplicações de tempo real no espaço de usuário

O Hardware Utilizado



- Netbook Acer Aspire One D250-1023
- Processador: x86 Intel Aton N270 @ 1,60 GHz
- Cache L2: 512KB
- Ram: 1GB DDR2-533
- Disco Rígido: 320GB, 5400 RPM, SATA
- Versão da BIOS: 1.27

O Hardware Utilizado - Qualquer um serve?

- Em alguns computadores podem não aceitar um kernel com as configurações propostas, como a remoção do suporte a ACPI, por conflito com a BIOS ou com o próprio hardware do sistema.
- O RTAI não suporta processadores anteriores ao processador Pentium.
- Não é possível compilar o kernel apos aplicação do patch **hal-linux-4.4.43-x86-6 (RTAI)**, caso a opção **AMD MCE features** esteja habilitada. Por consequência processadores AMD que necessitem desta opção não são suportados.
- Não existe um banco de dados contendo todo o hardware suportado, pra usar tem que testar!

A Distribuição Linux Utilizada

- Debian 8.8 (Jessie)
- 32 bits
- Instalação via netInstall (rede ethernet)
- Mínima instalação possível (sem GUI)
- Versão do Kernel padrão da distribuição: 3.16.0-4
- Versão do Kernel usado junto ao Preempt_RT: 4.4.17 (Vanilla)
- Versão do Kernel usado junto ao RTAI: 4.4.43 (Vanilla)

Produzindo um Kernel de Tempo Real

- Preparar o ambiente para a compilação
  - Baixar os pacotes contendo as ferramentas e bibliotecas necessárias para a compilação
  - Obter os fonte do kernel vanilla no kernel.org (deve ser uma versão suportada pelo patch!)
- Aplicar o patch certo aos fontes do kernel (Preempt_RT ou HAL - RTAI)
- Configurar o kernel (menuconfig, xconfig, etc). Será explicado mais a frente!
  - make menuconfig
- Compilar o kernel (pode esperar que vai demorar: “-j” é seu amigo!)
  - make
- Instalar o novo kernel
  - make modules_install, make install
- Reiniciar o sistema (se der tudo certo vai reiniciar)

Configuração do Kernel

- O programa Cyclictest é fornecido junto ao conjunto de teste rt-tests e pode ser baixada em **https://www.kernel.org/pub/linux/utils/rt-tests/**
- Possui uma função para rastreio de processos que aumentam a latência do sistema, usando o recursos de tracing do kernel
  - Opção ‘-f’ em combinação com ‘-b’
  - Marca os processos com latência superior ao valor definido
- Log pode ser visto em
  - /sys/kernel/tracing/trace
  - Os vilões estão marcados com uma ‘!’
- Quando executado em um em um sistema com RTAI o Cyclictest não funciona como uma aplicação de tempo real

Configuração do Kernel

Log do kernel tracer

```
# tracer: function
#
# function latency trace v1.1.5 on 3.16.0-4-686-pae
# --------------------------------------------------------------------
# latency: 0 us, #144306/1398292, CPU#1 | (M:desktop VP:0, KP:0, SP:0 HP:0 #P:2)
#    -----------------
#    | task: -0 (uid:0 nice:0 policy:0 rt_prio:0)
#    -----------------
#
#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
  <idle>-0       1.... 2814144us+: tick_nohz_idle_enter <-cpu_startup_entry
  <idle>-0       1.... 2814145us+: set_cpu_sd_state_idle <-tick_nohz_idle_enter
  <idle>-0       1d... 2814147us+: __tick_nohz_idle_enter <-tick_nohz_idle_enter
  <idle>-0       1d... 2814149us+: ktime_get <-__tick_nohz_idle_enter
  <idle>-0       1d... 2814150us+: read_hpet <-ktime_get
  <idle>-0       1d... 2814154us+: timekeeping_max_deferment <-__tick_nohz_idle_enter
  <idle>-0       1d... 2814156us+: rcu_needs_cpu <-__tick_nohz_idle_enter
  <idle>-0       1d... 2814157us+: rcu_cpu_has_callbacks <-rcu_needs_cpu
  <idle>-0       1d... 2814159us+: get_next_timer_interrupt <-__tick_nohz_idle_enter
  <idle>-0       1d... 2814161us+: _raw_spin_lock <-get_next_timer_interrupt
  <idle>-0       1d... 2814169us : hrtimer_get_next_event <-get_next_timer_interrupt
  <idle>-0       1d... 2814171us+: _raw_spin_lock_irqsave <-hrtimer_get_next_event
  <idle>-0       1d... 2814172us+: _raw_spin_unlock_irqrestore <-hrtimer_get_next_event
  <idle>-0       1d... 2814174us+: nohz_balance_enter_idle <-__tick_nohz_idle_enter
  <idle>-0       1d... 2814176us+: calc_load_enter_idle <-__tick_nohz_idle_enter
  <idle>-0       1d... 2814178us+: hrtimer_start <-__tick_nohz_idle_enter
  <idle>-0       1d... 2814179us+: __hrtimer_start_range_ns <-hrtimer_start
  <idle>-0       1d... 2814181us : lock_hrtimer_base.isra.13 <-__hrtimer_start_range_ns
  <idle>-0       1d... 2814182us+: _raw_spin_lock_irqsave <-lock_hrtimer_base.isra.13
:
```

```
  <idle>-0       0d... 3261440us!: read_hpet <-ktime_get
  <idle>-0       0d... 3262408us!: read_hpet <-ktime_get
  <idle>-0       0d... 3263394us!: read_hpet <-ktime_get
  <idle>-0       0d... 3264394us!: read_hpet <-ktime_get
  <idle>-0       0d... 3265387us!: read_hpet <-ktime_get
  <idle>-0       0d... 3266383us!: read_hpet <-ktime_get
  <idle>-0       0d... 3267395us!: read_hpet <-ktime_get
  <idle>-0       0d... 3268394us!: read_hpet <-ktime_get
cyclicte-4768    1.... 3269507us!: cfb_imageblit <-bit_putcs
cyclicte-4768    1.... 3271073us!: cfb_imageblit <-bit_putcs
  <idle>-0       1d... 3271861us!: read_hpet <-ktime_get
  <idle>-0       0d... 3272471us!: read_hpet <-ktime_get
  <idle>-0       0d... 3273459us!: read_hpet <-ktime_get
  <idle>-0       0d... 3274457us!: read_hpet <-ktime_get
  <idle>-0       0d... 3275468us!: read_hpet <-ktime_get
  <idle>-0       0d... 3276468us!: read_hpet <-ktime_get
  <idle>-0       0d... 3277460us!: read_hpet <-ktime_get
  <idle>-0       0d... 3278458us!: read_hpet <-ktime_get
  <idle>-0       0d... 3279535us!: read_hpet <-ktime_get
  <idle>-0       0d... 3280532us!: read_hpet <-ktime_get
  <idle>-0       0d... 3281536us!: read_hpet <-ktime_get
cyclicte-4768    0dNh. 3283189us!: irq_exit <-smp_apic_timer_interrupt
cyclicte-4768    0.N.. 3284775us!: cfb_imageblit <-bit_putcs
  <idle>-0       0d... 3285783us!: read_hpet <-ktime_get
  <idle>-0       0d... 3286678us!: read_hpet <-ktime_get
  <idle>-0       0d... 3287533us!: read_hpet <-ktime_get
  <idle>-0       0d... 3288531us!: read_hpet <-ktime_get
  <idle>-0       0d... 3289532us!: read_hpet <-ktime_get
  <idle>-0       0d... 3290555us!: read_hpet <-ktime_get
  <idle>-0       0d... 3291481us!: read_hpet <-ktime_get
  <idle>-0       0d... 3292479us!: read_hpet <-ktime_get
  <idle>-0       0d... 3293472us!: read_hpet <-ktime_get
  <idle>-0       0d... 3294610us!: read_hpet <-ktime_get
  <idle>-0       0d... 3295523us!: read_hpet <-ktime_get
cyclicte-4768    0.... 3296987us!: cfb_imageblit <-bit_putcs
cyclicte-4768    0.N.. 3298856us!: cfb_imageblit <-bit_putcs
daniel@debian-rt:~$ _
```

Configuração do Kernel

- Kernel 32 bits
  - [ ] 64-bit kernel
- General setup > Timers subsystem
  - [*] High Resolution Timer Support
- [*] Enable Loadable module support **(RTAI)**
  - [ ] Module versioning support **(RTAI)**
- Processor type and features
  - [ ] Symmetric multi-processing support
  - Processor family > (X) Pentium-Classic
  - Preemption Model > (X) Fully Preemptible kernel (RT) **(Preempt_RT)**
  - [*] Interrupt pipeline **(RTAI)**
  - Time frequency > (X) 1000HZ
  - [ ] AMD MCE features (**RTAI**)

Configuração do Kernel

- Power Management and ACPI options
  - [ ] ACPI (Advanced Configuration and Power Interface) Support
  - CPU Frequency scaling
    - [ ] CPU Frequency scaling
  - CPU Idle
    - [ ] CPU Idle PM support
- File systems > Pseudo filesystems
  - [*] /proc file system support **(RTAI)**
- Kernel hacking
  - [ ] Debug preemptible kernel
  - [ ] Debug the x86 FPU code

Configuração do Kernel

De modo geral:

- Desabilitar todos os serviços de gerenciamento de energia
  - Caso precise de algum terá que lidar com latências altas e até imprevisíveis
  - Baterias não vão durar
- Desabilitar os serviços de debug
  - A não ser que você realmente saiba como eles se comportam
  - A função tracer do kernel pode ser deixada habilitada (para usar com o Cyclictest)
- Evolução do kernel
  - Algumas opções podem não existir ou existir com nomes diferentes dependendo da versão do kernel
  - As opções evoluem junto com o kernel, assim o uso de uma ferramentas como o Cyclictest é muito útil para rastrear causadores de latência

Produzindo um Kernel de Tempo Real - RTAI

Após a aplicação do patch, compilação, e instalação do kernel, o rtai exige sua própria compilação e instalação.

- Criar o diretório que receberá a instalação (o padrão é /usr/realtime)
- Configurar o RTAI (make menuconfig)
  - Não mexa em nada, além do diretório onde se encontra o kernel com o patch HAL e o local onde será feita a instalação. Outras configurações são misteriosas e não são bem documentadas.
- Compilar e instalar
  - make ; make install

Estrutura Básica de um Programa de tempo real

- Alocação e travamento de memória para evitar paginação
- Configuração das tarefas como tarefas de tempo real
  - Prioridade
  - Escalonamento
  - Periodicidade
- Execução de alguma computação
  - Ler valores de tempo
  - Dormir por um determinado intervalo

Estrutura Básica de um Programa - Preempt_RT

```c
/* Carregando as bibliotecas */
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <sched.h>
#include <sys/mman.h>
#include <string.h>

/* Definir a prioridade da tarefa */
#define PRIORIDADE (90)

/* Definir o tamanho da pilha do programa */
#define MAX_SAFE_STACK (8*1024)

/* Definir nanosegundos */
#define NSEG_POR_SEG (1000000000)

/* Inicializa toda a pilha do programa com zeros */
void stack_prefault(void) {
   unsigned char pilha[MAX_SAFE_STACK];
   memset(pilha, 0, MAX_SAFE_STACK);
    return;
}
```

```c
int main(int argc, char* argv[])
{
   unsigned int i = 0;
   struct timespec t;
   struct sched_param param;
   int periodo = 50000; /* 50us */

   /* Declara a sí mesma como um tarefa de tempo real */
   param.sched_priority = PRIORIDADE;
   if(sched_setscheduler(0, SCHED_FIFO, &param) == -1) {
      perror("sched_setscheduler falhou");
      exit(-1);
   }

   /* Trava a memória do programa para que não seja páginada */
   if(mlockall(MCL_CURRENT|MCL_FUTURE) == -1) {
      perror("mlockall falhou");
      exit(-2);
   }

   /* Pre-fault our stack */
   stack_prefault();

   /* Obtem o valor do tempo atual e salva em t */
   clock_gettime(CLOCK_MONOTONIC ,&t);

   /* Faz a tarefa iniciar após 1 segundo */
   t.tv_sec++;
```

Estrutura Básica de um Programa - Preempt_RT

```
55    while(1) {
56       /* Suspende a execução até o proximo período */
57       clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &t, NULL);
58
59       /* Aqui realiza alguma computação útil*/
60       i++;
61       printf("\nvalor de i: %d\n", i);
62
63       /* Calcula o momento da próxima ativação */
64       t.tv_nsec += intervalo;
65
66       /* Esse loop ajusta os valores de t
67        * para que fiquem distribuidos da
68        * forma correta entre os elementos da struct */
69       while (t.tv_nsec >= NSEG_POR_SEG) {
70          t.tv_nsec -= NSEG_POR_SEG;
71          t.tv_sec++;
72       }
73    }
74 }
```

Estrutura Básica de um Programa - RTAI

```
 1 /* Carregando as bibliotecas */
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4 #include <unistd.h>
 5 #include <fcntl.h>
 6 #include <sys/mman.h>
 7 #include <rtai_lxrt.h>
 8
 9 int main (void) {
10
11    /* Declara uma tarefa de tempo real */
12    RT_TASK *tarefa;
13
14    /* Configura os parametros de escalonamento */
15    tarefa = rt_task_init_schmod( id, 2, 0, 0, SCHED_FIFO, 0x1 );
16
17    /* Configura a tarefa como uma tarefa periódica */
18    rt_task_make_periodic_relative_ns( tarefa, delay, frequencia );
19
20    /* Configura a tarefa como rígida */
21    rt_make_hard_real_time();
```

Estrutura Básica de um Programa - RTAI

```
23    while( 1 ) {
24
25       /* Espera por um tempo igual ao definido por frequencia */
26       rt_task_wait_period();
27
28       computacao();
29    }
30
31    /* Remove a tarefa criada */
32    rt_task_delete( tarefa );
33    return 0;
34 }
```

Estrutura Básica de um Programa - RTAI

- Antes da compilação e executar:
  - Carregar módulos (e tem que ser numa ordem específica)
  - Definir as variáveis CFLAGS e LDFLAGS
- Um script para automatizar este processo foi criado: rtai-init

```
1  #!/bin/bash
2
3  # Carregando os modulos do RTAI
4  sudo insmod /usr/realtime/modules/rtai_hal.ko
5  sudo insmod /usr/realtime/modules/rtai_sched.ko
6  sudo insmod /usr/realtime/modules/rtai_fifos.ko
7  sudo insmod /usr/realtime/modules/rtai_sem.ko
8  sudo insmod /usr/realtime/modules/rtai_mbx.ko
9  sudo insmod /usr/realtime/modules/rtai_msg.ko
10 sudo insmod /usr/realtime/modules/rtai_shm.ko
11 sudo insmod /usr/realtime/modules/rtai_smi.ko
12 sudo insmod /usr/realtime/modules/latency_rt.ko
13
14 # Criando as variáveis para compilação
15 export CFLAGS=$(/usr/realtime/bin/./rtai-config --lxrt-cflags)
16 export LDFLAGS=$(/usr/realtime/bin/./rtai-config --lxrt-ldflags)
```

O Programa Utilizado nos Testes - Concepção

- Testes desenvolvidos
  - Uso do algoritmo de medição do Cyclictest
  - Fundamentado no benchmark desenvolvido em Moreira (2007)
    - Cinco tarefas periódicas com frequências de 1Hz, 2Hz, 4Hz, 8Hz e 16Hz
    - Cinco tarefas periódicas com as frequências anteriores mais duas aperiódicas
- Carga utilizada nos testes
  - Utilizam 100% do tempo da CPU
  - Intenso uso de de memória e disco rígido
  - Ping infinito para a própria máquina
  - Compilação do kernel

O Programa Utilizado nos Testes - Concepção

- O que os testes medem afinal?
  - Execução das tarefas sem perder seus deadlines independente da carga do sistema
  - Medir as latências do sistema
  - Medir o tempo de execução de cada tarefa
- Pra que serve isso?
  - Avaliar se um sistema atende as restrições temporais de uma aplicação ou não
  - Dada a simplicidade do algoritmo de medição, pode ser implementado na aplicação real para obter resultados mais fiéis

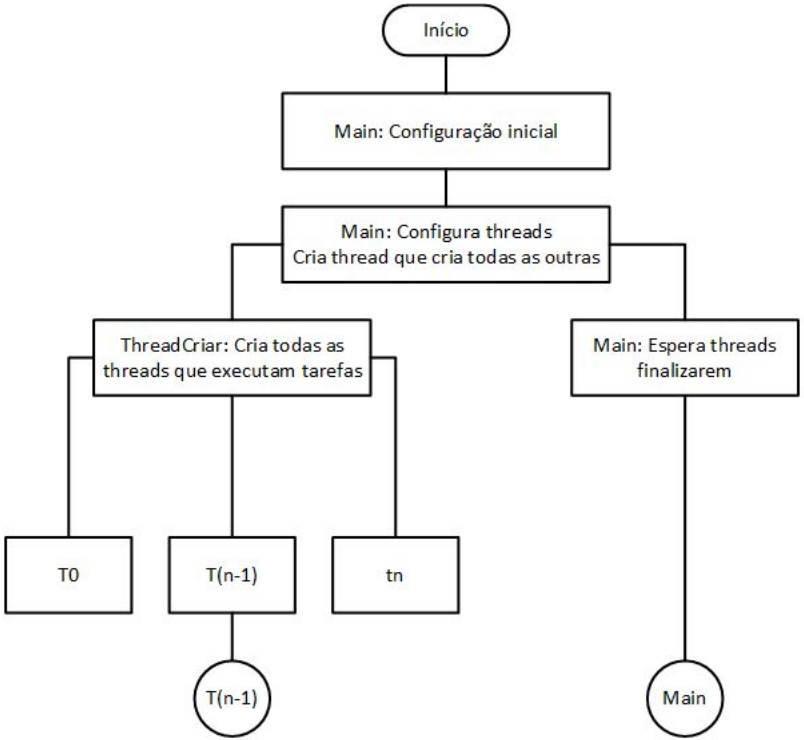
O Programa Utilizado nos Testes - As tarefas

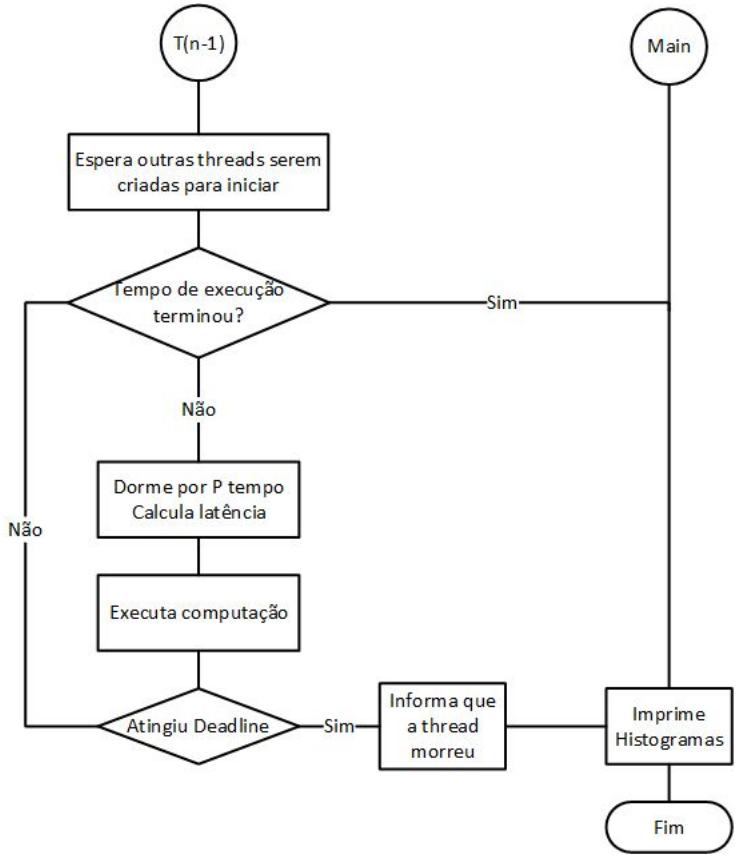
- Computação executada (Moreira 2007):
  - Conversão de graus para radianos
  - Cálculo da raiz quadrada através de séries de Taylor
  - Cálculo das raízes de um polinômio de grau 3
  - Quicksort de 20 inteiros
  - Multiplicação de duas matrizes 5x5
  - Valores de entrada gerados pela própria tarefa
- Tarefas periódicas (Moreira 2007):
  - 1Hz, 2Hz, 4Hz, 8Hz e 16Hz
  - Deadlines iguais ao período
  - Prioridade igual para todas

O Programa Utilizado nos Testes - As tarefas

- As tarefas aperiódicas
  - Executaram a conversão de graus em radianos
  - Ativação aleatória em intervalos de 40ms a 60ms
  - Deadline de 20ms
  - Prioridade igual entre si e maior que a das tarefas periódicas
- Escalonamento FIFO orientado a prioridades

O Programa Utilizado nos Testes - Funcionamento





O Programa Utilizado nos Testes - Funcionamento

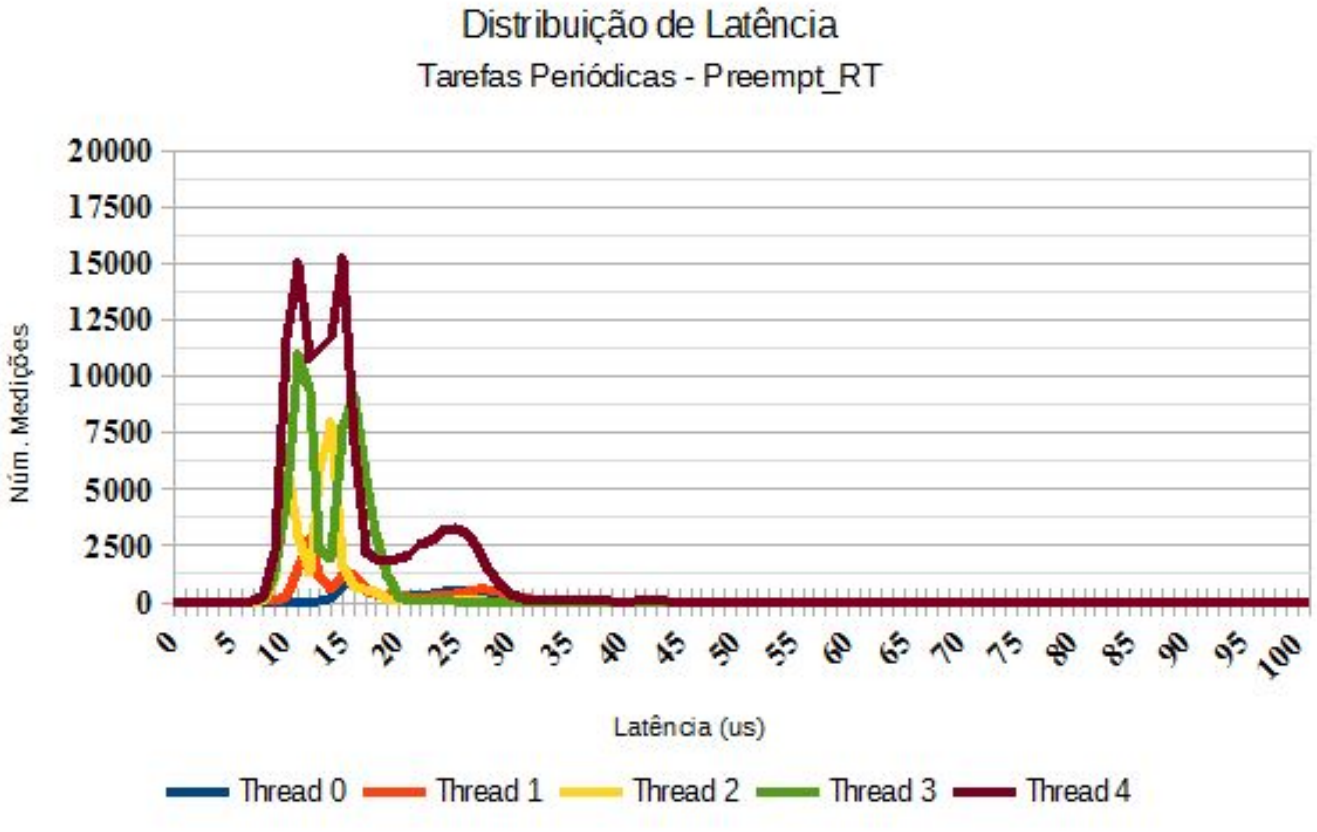
Algoritmo de medição

```
 1 pegaTempo( t1 );
 2 t1 += tempoQueDeveDormir;
 3 while(1) {
 4    dormeAte( t1 );
 5    pegaTempo( t2 );
 6    latencia = t2 - t1;
 7    t1 += tempoQueDeveDormir;
 8
 9    executaComputacao();
10
11    pegaTempo( t3 );
12    tempoComputacao = t3 - t2;
13    if( (t3 - t1) >= deadline ) {
14       termina();
15    }
16 }
17
18
```

Resultados

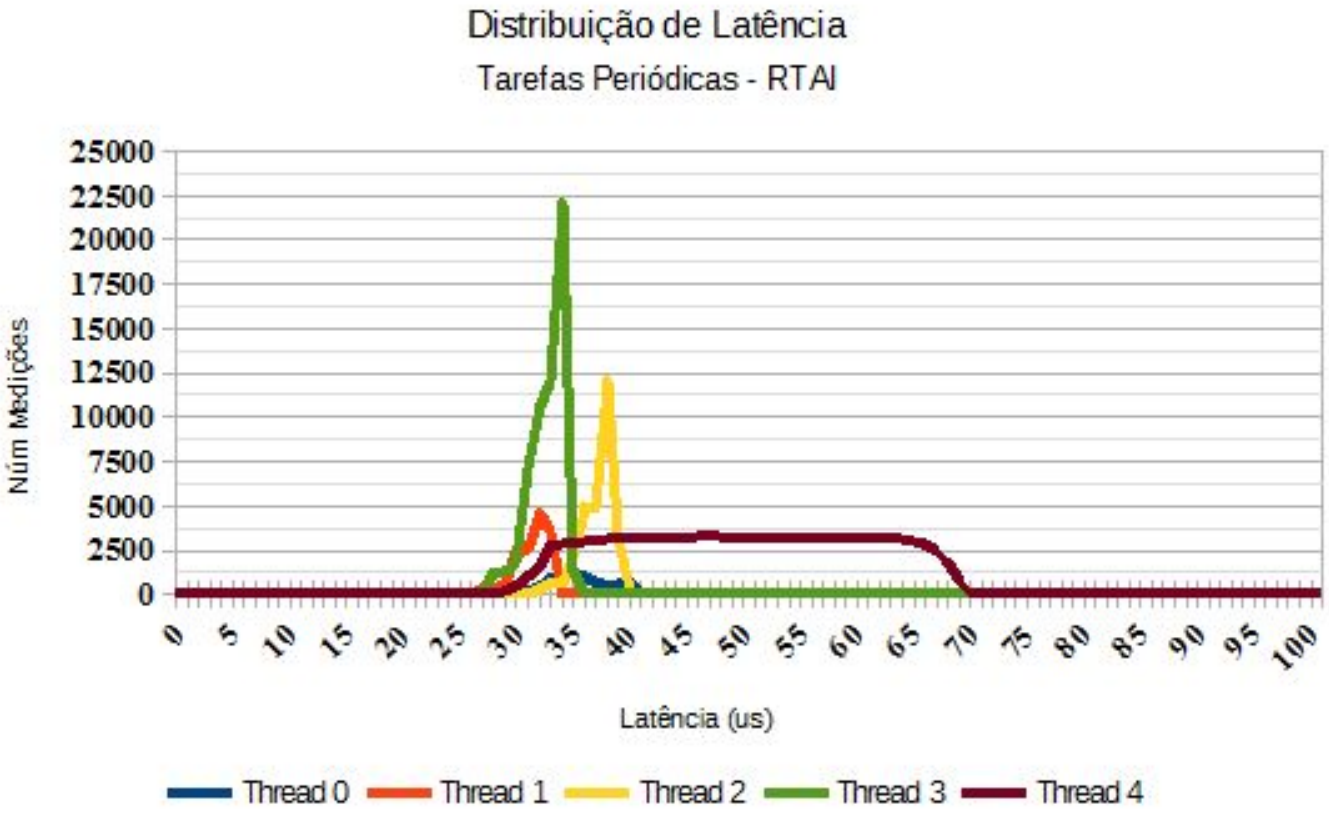
- As primeiras versões precisaram de ajustes
  - Threads não iniciavam no mesmo instante
  - Porte do código para RTAI
- Após os ajustes a execução ocorreu como o esperado
  - Nenhum deadline foi perdido
  - Valores de latência próximos dos valores obtidos nos trabalhos referênciados
  - Valores de latência de ambas as soluções foram equivalentes
- Mas nem tanto ...
  - Valores do tempo de computação das tarefas executadas no teste Periódicas + Aperiódicas no sistema com o patch Preempt_RT foram muito maiores que os do RTAI
  - RTAI produziu alguns valores de latência estranhos

Preempt_RT x RTAI



Thread	Lat M.	Lat m.
0	39	13
1	43	8
2	30	8
3	25	7
4	44	7

Preempt_RT x RTAI



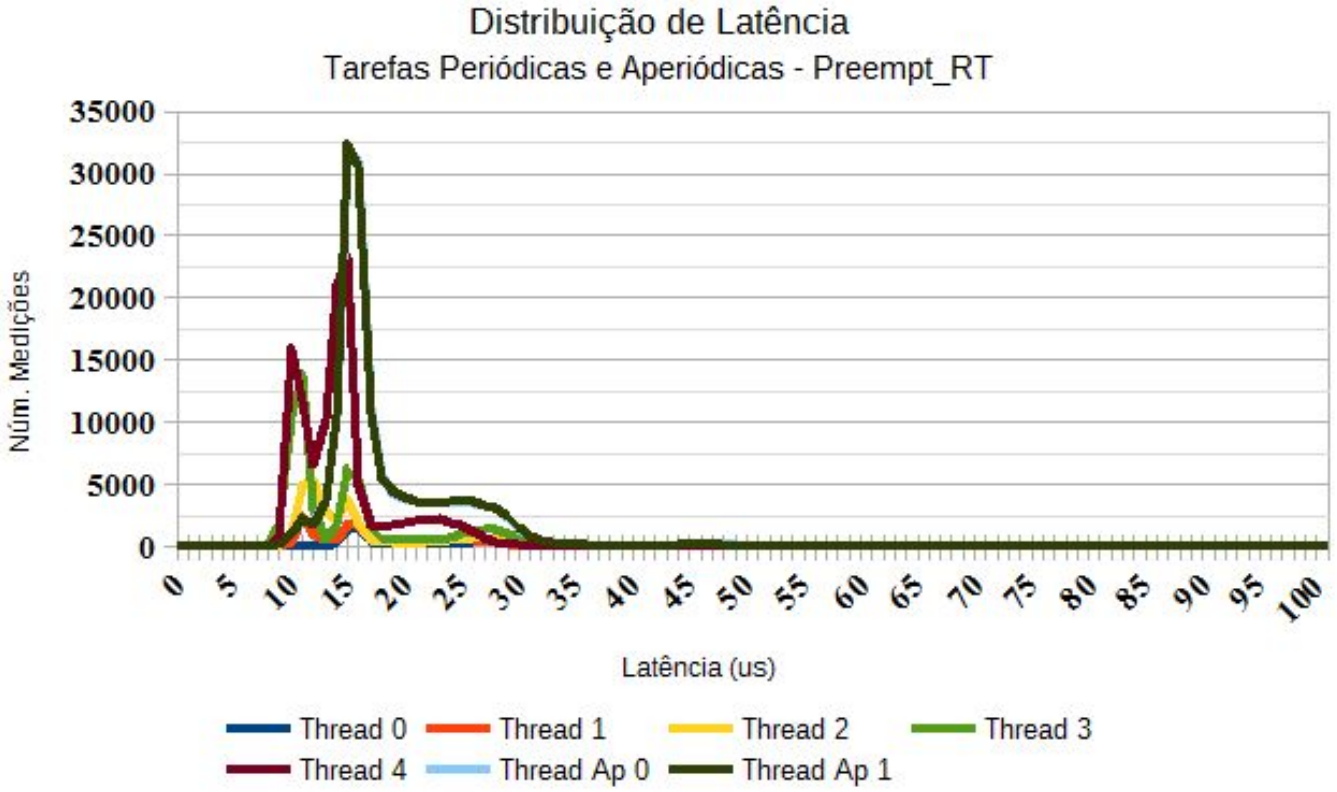
Thread	Lat M.	Lat m.
0	41	28
1	38	22
2	44	27
3	40	24
4	74	20

Preempt_RT x RTAI

Tempo máximo de computação (em us) - Tarefas Periódicas

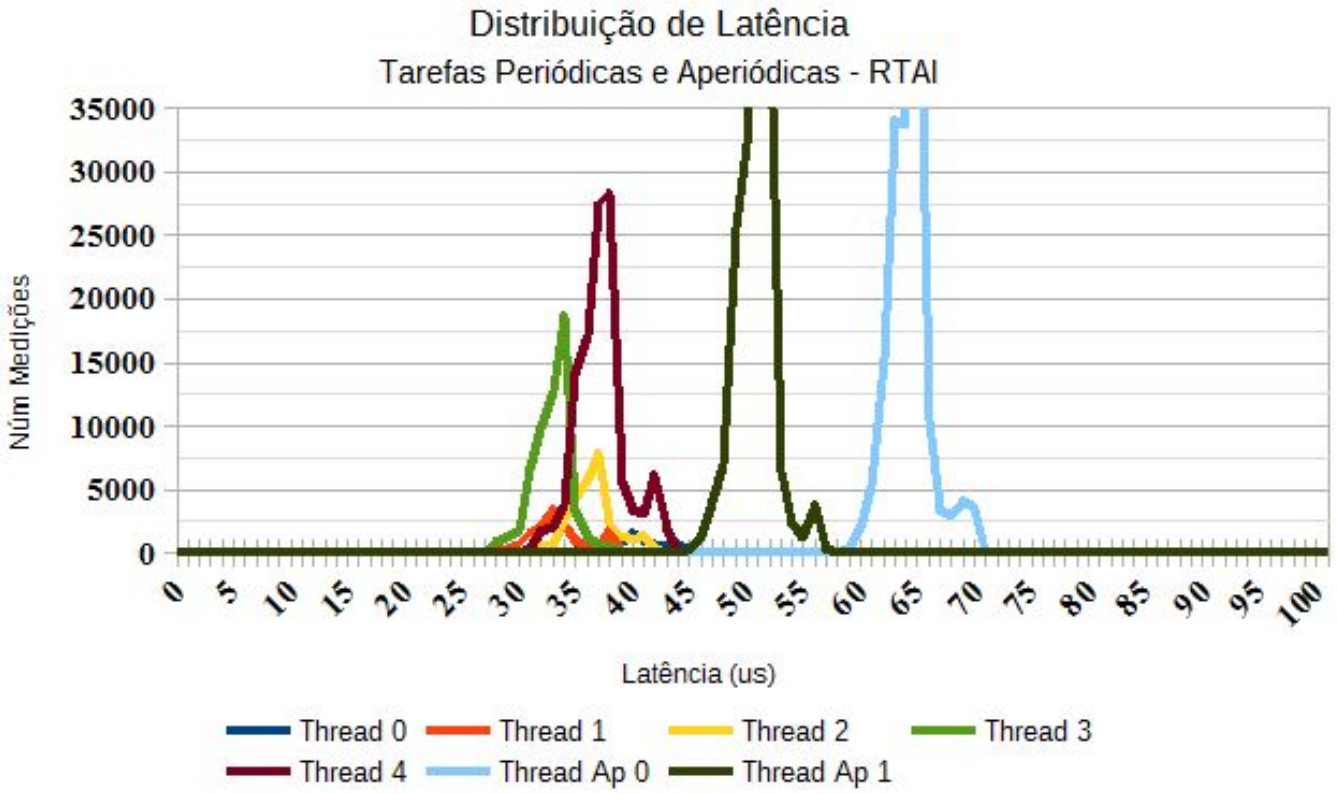
	Thread 0	Thread 1	Thread 2	Thread 3	Thread 4
Preempt_RT	18	38	39	27	39
RTAI	19	45	37	28	43

Preempt_RT x RTAI



Thread	Lat M.	Lat m.
0	70	9
1	72	8
2	71	7
3	74	7
4	67	7
Ap 0	80	6
Ap 1	71	7

Preempt_RT x RTAI



Thread	Lat M.	Lat m.
0	46	32
1	39	25
2	43	28
3	39	21
4	45	23
Ap 0	72	52
Ap 1	59	37

Preempt_RT x RTAI

Tempo máximo de computação (em us) - Tarefas Periódicas e Aperiódicas

	Thread 0	Thread 1	Thread 2	Thread 3	Thread 4	Thread AP 0	Thread AP 1
Preempt_RT	70	81	83	70	79	62	56
RTAI	21	45	36	20	40	42	44

Conclusão

- Os testes mostram que os sistemas podem suportar com segurança tarefas com restrições temporais na casa dos milissegundos e em alguns casos microssegundos
- As latências do Preempt_RT foram mais homogêneas do ponto de vista das tarefas, mas geram algumas “surpresas” na presença de tarefas aperiódicas
- Os resultados do RTAI mostram um distribuição de latências estranhas do ponto de vista das tarefas, mas dentro de intervalos mais bem definidos
- O tempo de computação do RTAI foi melhor na presença de tarefas aperiódicas.
- O Preempt_RT foi melhor com tarefas periódicas

Conclusão

- O patch Preempt_RT é
  - Eficiente e fácil de usar e instalar
  - Boa documentação, totalmente compatível com o kernel vanilla
  - Desenvolvimento a pleno vapor
  - Latências espúrias podem comprometer seu desempenho
- Instalar e usar o RTAI pela primeira vez é um calvário
  - A documentação é escassa, dispersa e desatualizada!
  - Possui vários easter eggs! Faz coisas que não estão documentadas, como o suporte a EDF
  - Não é possível utilizar chamadas de sistema em uma tarefa de tempo real sem comprometer a previsibilidade
- Apesar das dificuldades, o RTAI …
  - Teve bom desempenho na execução das computações propostas
  - Traz poucas surpresas quanto a latência

Trabalhos Futuros

- Avaliar o suporte a escalonamento EDF
- Avaliar a possibilidade de uma solução conjunta RTAI + Preempt_RT
- Produzir uma prova de conceito que mostre a aplicação prática dos sistemas avaliados.
- Produzir uma análise da execução das duas soluções em sistemas multiprocessados e avaliar o suporte a SMP.
- Analisar como as duas soluções se comportam quando executadas em um sistema microcontrolado, num ARM, por exemplo.

Bibliográfia

- BERRY, Richard. Mastering the FreeRTOS Real Time Kernel. [S.l.]: Real Time Engineers Ltd., 2016.
- FARINES, Jean-Marie. Sistemas de Tempo Real. [S.l.]: Departamento de Automação e Sistemas da Universidade Federal de Santa Catariana, 2000.
- HART, Darren V. Internals of the RT Patch. Proceedings of the Linux Symposium, 2007.
- KOPETZ, Hermann. Real-Time Systems - Design Principles for Distributed Embedded Applications. [S.l.]: Kluwer Academic Publishers, 2002.
- LAPLANTE, Phillip A. Real-Time Systems Design and Analysis. [S.l.]: Wiley, 2012.
- MOREIRA, Andeson Luiz Souza. Análise de Sistemas Operacionais de Tempo Real. 2007. Mestrado Universidade Federal De Pernambuco.
- PRIBERAM. Priberam da Língua Portuguesa. 2017. Disponível em: <https://www.priberam.pt/dlpo/>. Acesso em: 17 out. 2017.
- TANENBAUM, Andrew S. Sistemas Operacionais Modernos. 3a edição. [S.l.]: Pearson Prentice Hall, 2009.
- ANH, Tran Nguyen Bao; TAN, Su-Lim. Real-time operating systems for small microcontrollers. IEEE micro, v. 29, n. 5, 2009.
- LITAYEM, Nabil; SAOUD, Slim Ben. Impact of the linux real-time enhancements on the system performances for multi-core intel architectures. International Journal of Computer Applications, v. 17, n. 3, 2011.
- REGNIER, Paul; LIMA, George; BARRETO, Luciano. Evaluation of interrupt handling timeliness in real-time linux operating systems. ACM SIGOPS Operating Systems Review, v. 42, n. 6, p. 52-63, 2008.

Bibliográfia

- GITE, Vivek. How to Compile and Install Linux Kernel v4.5 Source On a Debian / Ubuntu Linux. 2015. Disponível em: <https://www.cyberciti.biz/faq/debian-ubuntu-building-installing-a-custom-linux-kernel/>. Acesso em: 12 abr. 2017.
- FOUNDATION, Linux. Cyclictest. 2017. Disponível em: <https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest>. Acesso em: 3 ago. 2017.
- RACCIU, Giovanni; MANTEGAZZA, Paolo. RTAI 3.4 User Manual rev 0.3. 2006.
- SOUSA, Cristóvão. RTAI Tutorial. Coimbra University, 2009.
- ROWAND, Frank. Using and understanding the real-time cyclictest benchmark. Em: Embedded Linux Conference. 2013.
- HALLBERG, ANDRÉAS. Time Critical Messaging Using a Real-Time Operating System. Mestrado Chalmers University of Technology and University of Gotherburg