BRUNO DE ANDRADE AMATUSSI
LUIS FERNANDO S. RIGONI
ELTON FOGGIATTO

µSO

SISTEMA OPERACIONAL PARA MICROCONTROLADOR MSP430

Documentação de Projeto apresentado à disciplina de Microprocessadores II do Curso de Engenharia de Computação do Centro de Ciências Exatas e de Tecnologia da Pontifícia Universidade Católica do Paraná – PUCPR, como parte integrante da nota do segundo semestre.

Prof. orientador: Afonso Ferreira Miguel

CURITIBA
2009

SUMÁRIO

1. LISTA DE FIGURAS

2. INTRODUÇÃO

Diante da infinidade de microprocessadores e sensores diferentes disponíveis no mercado, este projeto consiste no desenvolvimento de um sistema operacional (ou kernel), para ser usado como plataforma de desenvolvimento padrão de aplicações para serem embarcadas em microcontroladores. Funcionalidades comuns que estariam facilmente encapsuladas é o controle de motores, leitura de sensores e controle de atuadores.

Um sistema operacional consiste de um ou um conjunto de programas que compõem o software básico do dispositivo e cuja finalidade é a de executar os aplicativos e de servir como interface mediadora entre o hardware e o usuário.

Os envolvidos no desenvolvimento desse projeto estão os autores como responsáveis pelo desenvolvimento do projeto, e o professor Afonso Ferreira Miguel (http://www.afonsomiguel.com/), no papel de orientador.

A idéia do projeto surgiu em meados de 2008, quando um dos autores cursou a disciplina de Sistemas Operacionais ministrada pelo professor Carlos Maziero (http://www.ppgia.pucpr.br/~maziero/doku.php). A hipótese de usar os códigos desenvolvidos durante o curso para criar uma forma didática de abstrair o desenvolvimento de aplicações microcontroladas para dispositivos embarcados inspirou a equipe.

3. JUSTIFICATIVA

O projeto “µSO” se diferencia por ser um sistema operacional básico para sistemas embarcados, no qual tem uma aplicabilidade enorme, onde o mesmo sistema poderia controlar diversas aplicações com necessidades e recursos diferentes utilizando único processo de gerenciamento, já que cada solução teria sua implementação de forma independente e otimizada para um único processo, o q economizaria tempo e dinheiro no desenvolvimento das aplicações.

O Projeto “µSO” é sistema que pode baratear o desenvolvimento de aplicações para sistemas embarcados, dependendo da necessidade de vários processos, ou de mais de um microcontrolador, abrindo uma possibilidade para a redução de custos na fase de prototipação e produção de um projeto comercial.

4. METODOLOGIA

A metodologia de desenvolvimento do µSO foi estruturada de uma forma a gerar o mínimo possível de erros diante do pouco tempo disponível. O plano foi em primeiro lugar fazer pesquisas sobre programação em linguagem C em dispositivos e a escolha do microcontrolador adequado às necessidades de memória.

Tendo sido feita a escolha pelo MSP430 e tendo a maior parte do código do sistema operacional já escrita, todo trabalho foi portá-lo para a plataforma, já que antes havia sido desenvolvido para arquitetura PC, baseado no Linux.

As razões pela escolha do MSP430 foram duas: atendia aos requisitos de memória e, já existe disponível nos laboratórios de computação da PUCPR um kit de desenvolvimento. A escolha do kit já disponível reduziu o tempo de implementação, já que não se teve problemas de prototipação e funcionamento dos circuitos eletrônicos.

O desafio foi reescrever as bibliotecas usadas que não havia semelhante para o dispositivo.

O fluxo de trabalho da equipe era mensal. A cada mês uma rápida reunião era realizada para verificar prazos e verificar o andamento do desenvolvimento.

5. AS RESPONSABILIDADES

Para que o projeto obtivesse sucesso em seu desenvolvimento foi necessário a participação ativa de todos os participantes do grupo e também dos professores, é necessário muita responsabilidade, seriedade e muita força de vontade em todos os eixos do grupo para que o projeto fosse bem desenvolvido. Cada integrante teve a sua responsabilidade e cumpriu com o máximo de comprometimento para com ele. Os professores estavam aptos a responder todas nossas dúvidas em relação ao projeto, e nos ajudar, dar novas idéias e apoio. E também dependemos das estruturas da PUC, que se tornou a principal responsabilidade, pois o kit de desenvolvimento utilizado no projeto estava disponível para uso somente dentro dos laboratórios da PUC, em respeito a normas de utilização do local.

6. OS OBJETIVOS

O projeto µSO tem como objetivos a implementação de um sistema operacional viável para uso em microcontroladores que possa ser usado como plataforma de desenvolvimento de aplicações.

A principal característica esperada é que fosse possível executar mais de uma atividade independentemente ao mesmo tempo, gerenciando o tempo em que cada tarefa ocupasse o processamento. Isso dará ao sistema a propriedade se ser pré-emptivo.

A seguir é elencado a Especificação de Requisitos, com todas as características que este sistema atenderá.

- O kernel do sistema operacional será capaz de gerenciar várias tarefas (que contêm os contextos) e terá métodos que permitem manipulá-los.
- Haverá métodos que configuram de forma mais abstrata os timers, as interrupções e alguns dispositivos externos, como displays LCD e a comunicação serial.
- Métodos para leitura de sensores, analógicos e digitais, conectados ao dispositivo.

7. NÃO ESTÁ INCLUSO NO ESCOPO DESTE PROJETO

O projeto tem uma ampla diversidade de tecnologias que poderiam ser feitas, mas que não serão implementadas nessa versão.

As tecnologias que poderiam ser feitas são: Controle de dispositivos eletromecânicos, implementação de código de tarefas que visem algo alem de demonstrar o funcionamento do sistema operacional, uso de outro microprocessador Alem do MSP430, controle de mais funcionalidades, como comunicação serial ou paralela ou qualquer meio de comunicação externo ao kit de desenvolvimento

8. O PROJETO

O Projeto µSO gerencia varias tarefas independentes, sendo estas apresentadas em formas de funções dentro do código. O funcionamento do sistema é apresentado a seguir.

Ao inicializar o sistema, inicializa-se a função que prepara o sistema para o uso, que em seguida inicializa função *dispatcher* (despachante), responsável pelo gerenciamento das *tasks* (ou tarefas). Sua função é determinar qual tarefa será chamada a ser executada e realiza o salvamento do contexto de registradores em estruturas de dados. Quando essa tiver sua execução finalizada é feita a restauração do contexto do sistema para que cada tarefa possa continuar a ter sua execução continuada exatamente do ponto onde sua execução foi interrompida.

O tempo de execução de uma tarefa é um parâmetro configurável no código-fonte. Para o controle do tempo foi usado o *timer* do microcontrolador para gerar interrupções em determinados intervalos de tempo, sendo que cada interrupção decrementa a variável da tarefa corrente que registra seu tempo restante em execução. Quando esse contador chega a zero, retorna-se a função despachante que realiza o salvamento de contexto da tarefa, escolhe a próxima tarefa, retoma o contexto da tarefa seguinte aos registradores e passa-se a executar a tarefa exatamente do ponto em que ela foi parada.

Para a função de teste de funcionamento do sistema, foram implementadas duas funções simples, “Ping” e “Pong”. Uma função tem como função acender um led do kit de desenvolvimento, que fica acesso até o termino da função e inicio da função seguinte, que por sua vez apaga o led até o fim da tarefa e continuar a primeira tarefa.

9. OS RESULTADOS ESPERADOS

Como resultados deste projeto, serão apresentados ao professor os seguintes itens/funcionalidades:

1. Demonstração do código implementado e seu funcionamento no ambiente de desenvolvimento.
2. Demonstração dos softwares de teste implementados dentro do kit de desenvolvimento Texas MSP430;
3. CD com arquivos, fotos, códigos-fonte, esquemáticos, diagramas;
4. Vídeo do funcionamento;
5. Documentação do projeto dos itens acima.
6. Mostra do funcionamento aos professores e colegas.

11. CÓDIGO FONTE

ucontext.h

```
#ifndef __UCONTEXT__
#define __UCONTEXT__

typedef struct stack_t {
        char *ss_sp;
        //size_t ss_size;
        long ss_size;
        int ss_flags;
} stack_t;

typedef struct mcontext_t {
        int regPC; // program counter
        int regSP; // stack pointer
        int regSR; // status register
        int regR4;
        int regR5;
        int regR6;
        int regR7;
        int regR8;
        int regR9;
        int regR10;
        int regR11;
        int regR12;
        int regR13;
        int regR14;
        int regR15;
} mcontext_t;

typedef struct ucontext_t {
      struct ucontext_t *uc_link;
      //sigset_t uc_sigmask;
      stack_t uc_stack;
      mcontext_t uc_mcontext;
        int func;
        int initialized;
} ucontext_t;

int getcontext(ucontext_t *);
int setcontext( ucontext_t *);
void makecontext(ucontext_t *context, int bodyAdress, int n_parms, char
*parm);
//void makecontext(ucontext_t *context, int (*body)(), int n_parms, char
*parm);
//void makecontext(ucontext_t *context, int (*body)(), int, char *);
void setgearclock(int);
int swapcontext(ucontext_t *,  ucontext_t *);

#endif
```

ucontext.c

```
#include "ucontext.h"

extern int flag;
extern int memPC;

int r;

int getcontext(ucontext_t *context)
{
  asm
  (
    "MOV SR, r"
  );
  context->uc_mcontext.regSR = r;
  asm
  (
    "MOV R4, r"
  );
  context->uc_mcontext.regR4 = r;
  asm
  (
    "MOV R5, r"
  );
  context->uc_mcontext.regR5 = r;
  asm
  (
    "MOV R6, r"
  );
  context->uc_mcontext.regR6 = r;
  asm
  (
    "MOV R7, r"
  );
  context->uc_mcontext.regR7 = r;
  asm
  (
    "MOV R8, r"
  );
  context->uc_mcontext.regR8 = r;
  asm
  (
    "MOV R9, r"
  );
  context->uc_mcontext.regR9 = r;
  asm
  (
    "MOV R10, r"
  );
  context->uc_mcontext.regR10 = r;
  asm
  (
    "MOV R11, r"
  );
  context->uc_mcontext.regR11 = r;
  asm
  (
    "MOV R12, r"
  );
  context->uc_mcontext.regR12 = r;
  asm
```

```
  (
    "MOV R13, r"
  );
  context->uc_mcontext.regR13 = r;
  asm
  (
    "MOV R14, r"
  );
  context->uc_mcontext.regR14 = r;
  asm
  (
    "MOV R15, r"
  );
  context->uc_mcontext.regR15 = r;
  asm
  (
    "MOV SP, r"
  );
  context->uc_mcontext.regSP = r;
//  asm
//  (
//    "MOV PC, r"
//  );
//  if(!flag)
//  context->uc_mcontext.regPC = r;   // Com salvamento em ASM
  context->uc_mcontext.regPC = memPC;

  return 0;
}

int setcontext( ucontext_t *context)
{
  r = context->uc_mcontext.regSR;
  asm
  (
  "MOV r, SR"
  );
  r = context->uc_mcontext.regR4;
  asm
  (
  "MOV r, R4"
  );
  r = context->uc_mcontext.regR5;
  asm
  (
  "MOV r, R5"
  );
  r = context->uc_mcontext.regR6;
  asm
  (
  "MOV r, R6"
  );
  r = context->uc_mcontext.regR7;
  asm
  (
  "MOV r, R7"
  );
  r = context->uc_mcontext.regR8;
  asm
  (
  "MOV r, R8"
```

```
);
r = context->uc_mcontext.regR9;
asm
(
"MOV r, R9"
);
r = context->uc_mcontext.regR10;
asm
(
"MOV r, R10"
);
r = context->uc_mcontext.regR11;
asm
(
"MOV r, R11"
);
r = context->uc_mcontext.regR12;
asm
(
"MOV r, R12"
);
r = context->uc_mcontext.regR13;
asm
(
"MOV r, R13"
);
r = context->uc_mcontext.regR14;
asm
(
"MOV r, R14"
);
r = context->uc_mcontext.regR15;
asm
(
"MOV r, R15"
);

if(!context->initialized)
{
  r = (int)context->uc_stack.ss_sp;
  asm
  (
  "MOV r, SP"
  );
  r = (int)(context->func);
  context->initialized = 1;
  asm
  (
  "MOV r, PC"
  );
}
else
{
    flag = 1;
    r = context->uc_mcontext.regSP;
    asm
    (
    "MOV r, SP"
    );
    r = context->uc_mcontext.regPC;
    asm
```

```
      (
      "MOV r, PC"
      );

  }
  return 0;
}

void makecontext(ucontext_t *context, int start_routine, int n_parms, char
*parm)
{
  context->func = start_routine;
}

int swapcontext(ucontext_t *saida,  ucontext_t *entrada)
{
    getcontext(saida);
    setcontext(entrada);

    return 0;
}
```

tasklist.h

```
#ifndef __TASKLIST__
#define __TASKLIST__

#ifndef NULL
#define NULL ((void *) 0)
#endif

#include "task.h"

void list_append (task_t **list, task_t *task);

task_t *list_remove (task_t **list, task_t *elem);

void list_print (char *name, task_t *list);

int list_size (task_t *list);

#endif
```

tasklist.c

```
#include<stdio.h>

#include "tasklist.h"

void list_append (task_t **list, task_t *task)
{
    if(!task) // Verifica se a tarefa existe
    {
        printf("Erro. Tarefa inconsistente.\n");
        return;
    }
```

```
    if(task->next || task->prev) // Verifica se a tarefa jÃ¡ pertence Ã 
outra lista
    {
        printf("Erro. Tarefa jÃ¡ iniciada.\n");
        return;
    }

    task->next = task->prev = task;

    if(! *list) // Verifica se a lista de tarefas existe. Se nÃ£o, cria
        (*list) = task;

    task_t *last;
    last = (*list)->prev;

    task->prev = last;
    last->next = task;
    task->next = (*list);
    (*list)->prev = task;
}

task_t *list_remove (task_t **list, task_t *elem)
{
    if(!elem) // Verifica se a tarefa existe
    {
        printf("Erro. Tarefa inconsistente.\n");
        return NULL;
    }

    if(! *list) // Verifica se a lista de tarefas existe
    {
        printf("Erro. Lista inconsistente.\n");
        return NULL;
    }

    task_t *aux;
    for(aux = *list; aux != elem; aux = aux->next) // Verifica se a tarefa
pertence Ã  lista indicada
    {
        if(aux == (*list)->prev)
        {
            printf("Erro. Tarefa nÃ£o encontrada.\n");
            return NULL;
        }
    }

    if(aux->next == aux)
    {
        (*list) = NULL;
        aux->next = aux->prev = NULL;
        return aux;
    }
    if(aux == (*list))
        (*list) = aux->next;

    task_t *before, *after;
    before = aux->prev;
    after = aux->next;
    before->next = after;
    after->prev = before;
```

```
    aux->next = aux->prev = NULL;

    return aux;
}

void list_print (char *name, task_t *list)
{
    task_t *aux;
    aux = list;

    printf("%s: [ ", name);
    do
    {
        if(!aux)
            break;
        printf("%i<%i>%i ", aux->prev->id, aux->id, aux->next->id);
        aux = aux->next;
    } while(aux != list);
    printf("]\n");
}

int list_size (task_t *list)
{
    if(!list)
        return 0;

    int size;
    task_t *aux;

    for(size = 1, aux = list; aux != list->prev; aux = aux->next, size++);

    return size;
}
```

task.h

```
#ifndef __TASK__
#define __TASK__

#ifndef NULL
#define NULL ((void *) 0)
#endif

#include "ucontext.h"

//#define STACKSIZE   32768
//#define STACKSIZE   16384
//#define STACKSIZE   16384
//#define STACKSIZE   4096
#define STACKSIZE   16

#define QUANTUM         10

typedef enum Status
{
    Nova,
    Pronta,
    Rodando,
    Suspensa,
```

```
    Terminada
} Status;

typedef struct task_t
{
   unsigned int id;
   struct task_t *prev;
   struct task_t *next;
   ucontext_t context;
   Status status;
   unsigned int ticks_left;
   unsigned int tick_start; /* Indica o tick inicial da tarefa. */
   unsigned int tick_finish; /* Indica o tick final da tarefa. */
   unsigned int ticks_run; /* Indica o tempo, em ticks, de precessamento da
tarefa. */
   unsigned int activations; /* Indica o numero da ativacoes da tarefa. */
} task_t;

int systime ();

void task_init ();

//int task_create (task_t * task, void (*start_routine)(void *),  void *
arg);

void task_exit (int exit_code);

int task_yield (task_t *task);

int task_id ();

#endif
```

task.c

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#include "time.h"
#include "task.h"
#include "tasklist.h"
#include "ucontext.h"
#include "msp430x44x.h"

extern int flag;
task_t  main_task,
        *curr_task,
        dispatcher_task;

task_t  *ready_tasks;

unsigned int    id_cont = 0;

int    kernelMode = 0;

int    memPC = 0;
```

```
unsigned long int   clkTickCounter = 0; /* Contador de ticks. */

int systime()
{
      return clkTickCounter;
}

int flag;

#pragma vector = TIMERA1_VECTOR
__interrupt void ticks_handler()
{
    TAR = 65503; // contador 1ms

    clkTickCounter++;
    curr_task->ticks_run++; /* Incremeta o tempo de processamento da tarefa
atual. */

    TACTL &= 0xFFFE; // baixa flag interrupção

    if(kernelMode == 1)
        return;

    if(curr_task->ticks_left == 0)
    {
        task_yield(NULL);
    }
    else
        curr_task->ticks_left--;
}

task_t* scheduler_fifo()
{
    task_t *aux;
    aux = ready_tasks;

    if(aux != NULL)
        ready_tasks = ready_tasks->next;

    return aux;
}

int dispatcher ()
{
    kernelMode = 1;

    task_t *next;
    while((next = scheduler_fifo()))
    {
        next->ticks_left = QUANTUM;
        task_yield(next);
        switch(next->status)
        {
            case Terminada: list_remove(&ready_tasks, next);
                            free(next->context.uc_stack.ss_sp);
                            break;
            default:        break;
        }
    }
```

```
    #ifdef DEBUG
    printf("Dispatcher sendo encerrado.\n");
    #endif

    task_exit(0);
    return 0;
}

void setgearclock(int msec)
{
  // Configure TIMER_A for generating interrupts every 1 second
  // TIMER_A count frequency == ACLK/8 == 32768/8 == 4096

    TACCR0 = 32;      // TIMER A 32 LIMITE

    TACTL = 274;       //MUDAMOS O ID DE 11 PARA 00 POIS DIVIDE POR 1 PARA
1ms
                                  // TACTL = 466;    // 000000 TASSEL 01 ID
11  UP Contator 01 0 0 Interrupt Enable 1 0
                // CLOCK = ACLK
                // DIVIDER = /8
                // UP MODE (THE TIMER COUNTS UP TO TACCR0)
                // INTERRUPT ENABLED
                // NO TIMER_A INTERRUPT PENDING

    FLL_CTL1 |= 24;  // MASTER CLOCK == ACLK (32768Hz)
    _BIS_SR(GIE); // GENERAL INTERRUPT ENABLE
}

void task_init ()
{
    kernelMode = 1;

//    setvbuf (stdout, 0, _IONBF, 0);

    if(getcontext(&main_task.context) != 0)
        return;

    char *stack;
    stack = malloc (STACKSIZE);
    if (stack)
    {
        main_task.context.uc_stack.ss_sp = stack;
        main_task.context.uc_stack.ss_size = STACKSIZE;
        main_task.context.uc_stack.ss_flags = 0;
        main_task.context.uc_link =  0;
        main_task.id = 0;
        main_task.status = Rodando;
        main_task.ticks_left = 0;
        main_task.tick_start = systime();
        main_task.activations = main_task.ticks_run = 0;
    }
    else
    {
        perror ("Erro na criaÃ§Ã£o da pilha: ");
        exit (1);
    }

    curr_task = &main_task;

    #ifdef DEBUG
```

```
    printf("task_init: tarefa main iniciada com sucesso.\n");
    #endif

    task_create(&dispatcher_task, &dispatcher, NULL);
    dispatcher_task = *list_remove(&ready_tasks, ready_tasks);

    setgearclock(1);
    kernelMode = 0;
}
//int task_create(task_t* task, int (*start_routine)(),  void* arg)
int task_create(task_t* task, int start_routine,  void* arg)
{
    if(!task)
        return -1;

    if(getcontext (&task->context) != 0)
        return -1;

    kernelMode = 1;

    task->status = Nova;

    char *stack;
    stack = malloc (STACKSIZE);
    if (stack)
    {
        task->context.uc_stack.ss_sp = stack;
        task->context.uc_stack.ss_size = STACKSIZE;
        task->context.uc_stack.ss_flags = 0;
        task->context.uc_link =  0;
        task->id = ++id_cont;
        task->ticks_left = 0;
        task->tick_start = task->tick_finish = systime();
        task->activations = task->ticks_run = 0;

        task->context.initialized = 0;
    }
    else
    {
        perror ("Erro na criaÃ§Ã£o da pilha: ");
        exit (1);
    }

    //makecontext (&task->context, *start_routine, 1, arg);
    makecontext (&task->context, start_routine, 1, arg);

    #ifdef DEBUG
    printf("task_creat: criou tarefa %i.\n", task->id);
    #endif

    list_append(&ready_tasks, task);
    task->status = Pronta;

    kernelMode = 0;
    return task->id;
}

void task_exit (int exit_code)
{
    kernelMode = 1;
```

```
    #ifdef DEBUG
    printf("task_exit: tarefa %i sendo encerrada.\n", task_id());
    #endif

    curr_task->tick_finish = systime();

    printf("Task %02i exit: %04ims run time, %04ims CPU time, %03i
activations.\n",
        task_id(),   (curr_task->tick_finish   -   curr_task->tick_start),
curr_task->ticks_run, curr_task->activations);

    task_t *next;

    if (curr_task == &dispatcher_task)
        next = &main_task;
    else
        next = &dispatcher_task;

    task_t *old;
    old = curr_task;
    curr_task = next;

    old->status = Terminada;
    next->status = Rodando;

    swapcontext(&old->context, &next->context);

    if(curr_task == &dispatcher_task)
        kernelMode = 1;
    else
        kernelMode = 0;
}

int task_yield (task_t *task)
{
    kernelMode = 1;

    if(task == NULL)
        task = &dispatcher_task;

    #ifdef DEBUG
    printf("task_yield: trocando contexto %i -> %i.\n", task_id(), task-
>id);
    #endif

    task_t *old;
    old = curr_task;
    curr_task = task;

    old->status = Pronta;
    curr_task->status = Rodando;

    curr_task->activations++;

    kernelMode = 0;
    // salvar pc aqui move de memPC pra PC ou de PC pra memPC?
    asm
    (
      "mov PC,memPC"
    );
    // testar flag aqui
```

```
    if(!flag) // !flag = 0
      return swapcontext(&old->context, &task->context);
    else
      flag = 0;
}

int task_id ()
{
    return curr_task->id;
}
```

main.c

```
#include "io430.h"
#include "task.h"

task_t Ping, Pong;

void ping(void* arg)
{
  while(1)
  {
    P2OUT = 0x01;
  }
}

void pong(void* arg)
{
  while(1)
  {
    P2OUT = 0x00;
  }
}

int main ( void )
{
  WDTCTL = WDTPW + WDTHOLD; // desliga watchdog

  P2DIR = 0x01; // configura P2.0 como saida

  task_init();

  task_create(&Ping, ping, NULL);
  task_create(&Pong, pong, NULL);

  task_yield(NULL);
}
```

12. A EQUIPE DE DESENVOLVIMENTO

A equipe de desenvolvimento contará com Bruno de Andrade Amatussi, Luis Fernando Sobejeiro Rigoni e Elton Foggiatto, que serão responsáveis por realizar as tarefas necessárias para tornar o projeto viável.

13. CONCLUSÃO

O “µSO” se destaca por ser um projeto acadêmico que é inspirado no funcionamento de computadores com arquitetura x86, porém com a tecnologia que vai crescendo e com as mudanças nos padrões atuais, esse mesmo projeto acadêmico pode ter várias melhorias e competir com sistemas que visam o mesmo nível de independência.

Com a finalização deste projeto, que durou aproximadamente quatro meses, podemos dizer que conseguimos atender várias das metas levantadas ao início do projeto.

No transcorrer do projeto, encontramos várias dificuldades, porém, a maioria delas foi sanada quanto tivemos contato com o desenvolvimento em linguagem C/C++ para microcontroladores, isso ajudou-nos bastante no desenvolvimento de funções essenciais para o projeto.

O maior problema encontrado foi na adaptação do código C/C++ da arquitetura x86, trazido pelo Luís da disciplina de Sistemas Operacionais, para a arquitetura MSP430, algumas funções tiveram que ser totalmente re-escritas, o que nos proporcionou uma maior proximidade com as funcionalidades do microcontrolador selecionado inicialmente. Outros problemas encontrados foram os momentos de troca de contexto, onde há o salvamento de todos os registradores do microcontrolador e o tamanho do código fonte.

As soluções encontradas sanaram alguns problemas como: o problema da migração do código entre as arquiteturas teve como ponto principal da sua solução as aulas de desenvolvimento C/C++ em MSP430 ministradas pelo professor Afonso, a troca de contexto teve como diferencial o uso do timer em MSP430, e para o uso do código completo, obtivemos uma versão completa do IAR com suporte a maior uso da memória de código.

Após várias correções, a troca de contexto continuou apresentando problemas, e esses problemas preocuparam a equipe até dias antes da apresentação final do projeto, o que ainda impossibilitava o teste de parte do código. Apos a resolução do problema com a alteração do momento em que se guarda o valor armazenado no registrador PC (Program Counter), o programa passou a funcionar plenamente, e os testes com as funções PING (ascende led) e PONG (apaga led) puderam ser feitos normalmente. No sábado anterior a apresentação do projeto, obtivemos sucesso total na fase de teste, porém não pudemos fazer os testes finais com o kit MSP430, pois o nosso código era demasiado grande, e não compilou nos computadores da PUCPR, dessa forma voltamos à PUCPR na segunda-feira pela manhã para fazer os testes finais embarcando o "µSO" o que foi um desastre, por algum motivo que não conseguimos identificar. O salvamento da pilha que ocorre normalmente dentro da simulação não ocorre no microcontrolador e isso faz com que na hora de seu retorno ao despachante percamos o caminho de volta, travando o sistema.

Por fim, como o projeto possui cunho acadêmico, acreditamos que o aprendizado dentro da plataforma do microcontrolador MSP-430 foi bastante satisfatório, uma vez que em simulações feitas no aplicativo de desenvolvimento IAR Embedded Workbench IDE, obtivemos sucesso em todos os requisitos, com exceção da embarcação do sistema operacional, que teve problemas na troca de contexto entre as threads.

14. REFERÊNCIAS

Prof. Carlos A. Maziero <http://www.ppgia.pucpr.br/~maziero/doku.php>

The FreeRTOS Project <http://www.freertos.org/>

DATASHEET do microcontrolador MSP-430F44x disponível On-Line pesquisado em 16/10/2009 <http://www.datasheetcatalog.org/datasheet/texasinstruments/msp430f449.pdf>