P1 – Trocas de contexto

Henrique Luís Mazzuchetti

Nesse exercício foi dado esse código fonte para analisar e responder algumas questões:

```
#include <stdio.h>
//#include <printf.h>
#include <stdlib.h>
#include <ucontext.h>

// operating system check
#if defined(_WIN32) || (!defined(__unix__) && !defined(__unix) && (!defined(__APPLE__) || !defined(__MACH__)))
#warning Este codigo foi planejado para ambientes UNIX (LInux, *BSD, MacOS). A compilacao e execucao em outros ambientes e responsabilidade do usuario.
#endif

//#define STACKSIZE 32768       /* tamanho de pilha das threads */
#define STACKSIZE 4096      /* tamanho de pilha das threads */
#define _XOPEN_SOURCE 600   /* para compilar no MacOS */

ucontext_t ContextPing, ContextPong, ContextMain;

/****************************************************/

int flag;
int memPC = 0;

//-D DEBUG -D gcc CFLAGS =-g -Wall -I. -mcpu=cortex-m4 -mfloat-abi=hard
//-D DEBUG -D gcc CFLAGS =-g -Wall -I. -mcpu=cortex-m4 -mfloat-abi=hard
void BodyPing (void * arg)
{
   int i ;

   UARTprintf ("PING  %s iniciada\n", (char *) arg) ;

   for (i=0; i<6; i++)
   {
      UARTprintf ("PING  %s %d\n", (char *) arg, i) ;
      swap_context_asm (&ContextPing, &ContextPong);
   }
   UARTprintf ("%s FIM\n", (char *) arg) ;

   swap_context_asm (&ContextPing, &ContextMain) ;
}


/****************************************************/
void BodyPong (void * arg)
{
   int i ;

   UARTprintf ("PONG  %s iniciada\n", (char *) arg) ;

   for (i=0; i<6; i++)
   {
      UARTprintf ("PONG  %s %d\n", (char *) arg, i) ;
      swap_context_asm (&ContextPong, &ContextPing);
   }
   UARTprintf ("%s FIM\n", (char *) arg) ;

   swap_context_asm (&ContextPong, &ContextMain) ;
}

/****************************************************/
```

```c
//int main (int argc, char *argv[])
void testel(void){ //mainl(void){

//{
   char *stack ;

//  int a;
//  a = 10;

   UARTprintf ("Main INICIO\n");

   get_context_asm (&ContextPing);

   stack = malloc (10) ;
   if (stack)
   {
      ContextPing.uc_stack.ss_sp = stack ;
      ContextPing.uc_stack.ss_size = STACKSIZE;
      ContextPing.uc_stack.ss_flags = 0;
      ContextPing.uc_link = 0;
   }
   else
   {
      perror ("Erro na criação da pilha: ");
//       exit (1);
   }

   makecontext (&ContextPing, (int) (*BodyPing), 1, "    Ping");

   get_context_asm (&ContextPong);

   stack = malloc (STACKSIZE) ;
   if (stack)
   {
      ContextPong.uc_stack.ss_sp = stack ;
      ContextPong.uc_stack.ss_size = STACKSIZE;
      ContextPong.uc_stack.ss_flags = 0;
      ContextPong.uc_link = 0;
   }
   else
   {
      perror ("Erro na criação da pilha: ");
//       exit (1);
   }

   makecontext (&ContextPong, (int)(*BodyPong), 1, "        Pong");

   swap_context_asm (&ContextMain, &ContextPing);
   swap_context_asm (&ContextMain, &ContextPong);

   UARTprintf ("Main FIM\n");

//   exit (0);
    return;
}
```

Figura 1 – Código fonte do context.c.

1. **Explique o objetivo e os parâmetros de cada uma das quatro funções abaixo:**

   get_context_asm(&a): Função que trabalha com registradores em Assembly, que captura o contexto atual do projeto e salva esse contexto (valores dos registradores do processador) no endereço apontado pela variável “a”.

   setcontext(&a): Restaura um contexto que foi salvo numa determinada variável “a” e restaura, ou seja, recoloca nos registradores da CPU. Isso inclui recolocar a posição que foi salva no *program counter* e no *stack pointer*. Significa que é possível pular para uma área de código completamente diferente após a execução dessa função.

   swap_context_asm(&a,&b): Função que trabalha com registradores em Assembly, que salva o contexto atual na variável “a” e restaura o contexto da salvo anteriormente na variável “b”. Basicamente salva o contexto em “a” e salta para o contexto “b”.

   makecontext(&a, …) : Esta função não cria um contexto. Ela utiliza o contexto salvo em “a” e ajusta alguns valores internos do contexto salvo em “a”.

As variáveis “a” e “b” são do tipo ucontext_t e armazenam contextos de execução. Essas variáveis dependem da plataforma que está utilizando, quais registradores e flags existem no processador.

2. Explique o significado dos campos da estrutura ucontext_t que foram utilizados no código.

ucontext_t: as variáveis do tipo uconext_t armazenam contextos de execução.
uc_stack.ss_sp: seta o StackPointer na posição passada por um ponteiro de caracteres.
uc_stack.ss_size: informa tamanho total da pilha.
uc_stack.ss_flags: inicializa as flags que serão utilizadas na pilha.
uc_link: aponta para o contexto que será resumido quando este contexto retornar, ou seja informa o contexto sucessor.

3. Explique cada linha do código de contexts.c que chame uma dessas funções ou que manipule estruturas do tipo ucontext_t.

O programa inicializa fazendo a inclusão da biblioteca de *user context* através de <ucontext.h> e adiciona 3 variáveis ucontext_t para guardar os contextos ContextPing, ContextPong e ContextMain. São criadas as funções de thread BodyPing, BodyPong e o programa principal. O programa principal chama o getcontext, que armazena o contexto atual na variável ContextPing. Caso ele consiga alocar, o endereço da pilha será colocado no uc_stack.ss_sp, o tamanho da pilha no uc_stack.ss_size e flags serão ajustadas. Caso não consiga alocar irá retornar erro. Posteriormente a função makecontext é chamada, ajustando o contexto que foi salvo em getcontext e mexendo no valor do *program counter* para que ele aponte para a função BodyPing, e será colocado na pilha apontada pelo stackpointer a string “Ping”. Na sequência é feita a mesma coisa com o ContextPong, passando “Pong” para a pilha.

É feito o swapcontext onde é salvo o contexto atual de execução na variável ContexMain e ativa o contexto salvo em ContextPing. Isso irá acionar a função BodyPing que irá imprimir “Início” e irá fazer um for de 0 a 3 imprimindo a string que recebeu como parâmetro, adicionando “i”, que corresponde a: “Ping 0”, “Ping 1”, “Ping 2” e “Ping 3”. Porém, entre as passagens do laço *for* é feita a troca de contexto, saindo da variável ContextPing e entrando na variável ContextPong. Portando a execução altera entre “Ping” e “Pong”, e no final volta para Main, como pode ser visto na execução:

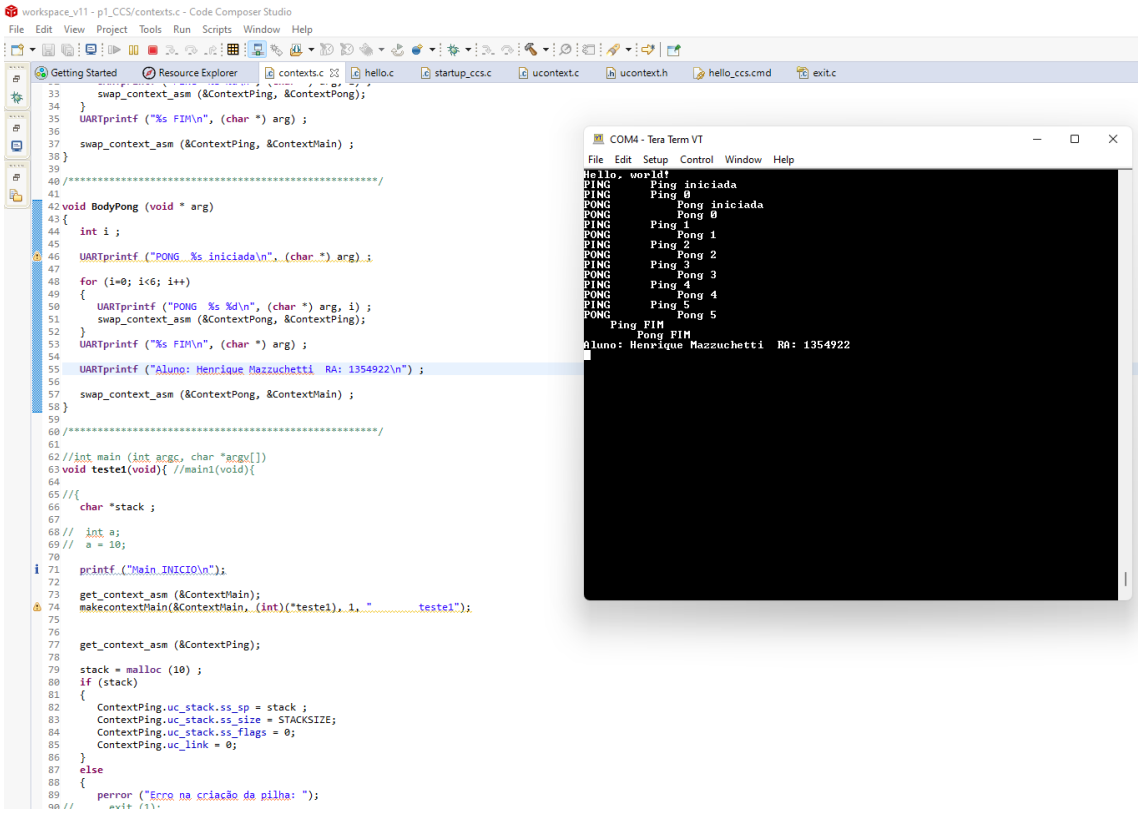

Figura 2 – Screenshot do Debug do CCS com o funcionamento no terminal TeraTerm.

4. Para visualizar melhor as trocas de contexto, desenhe o diagrama de tempo dessa execução.

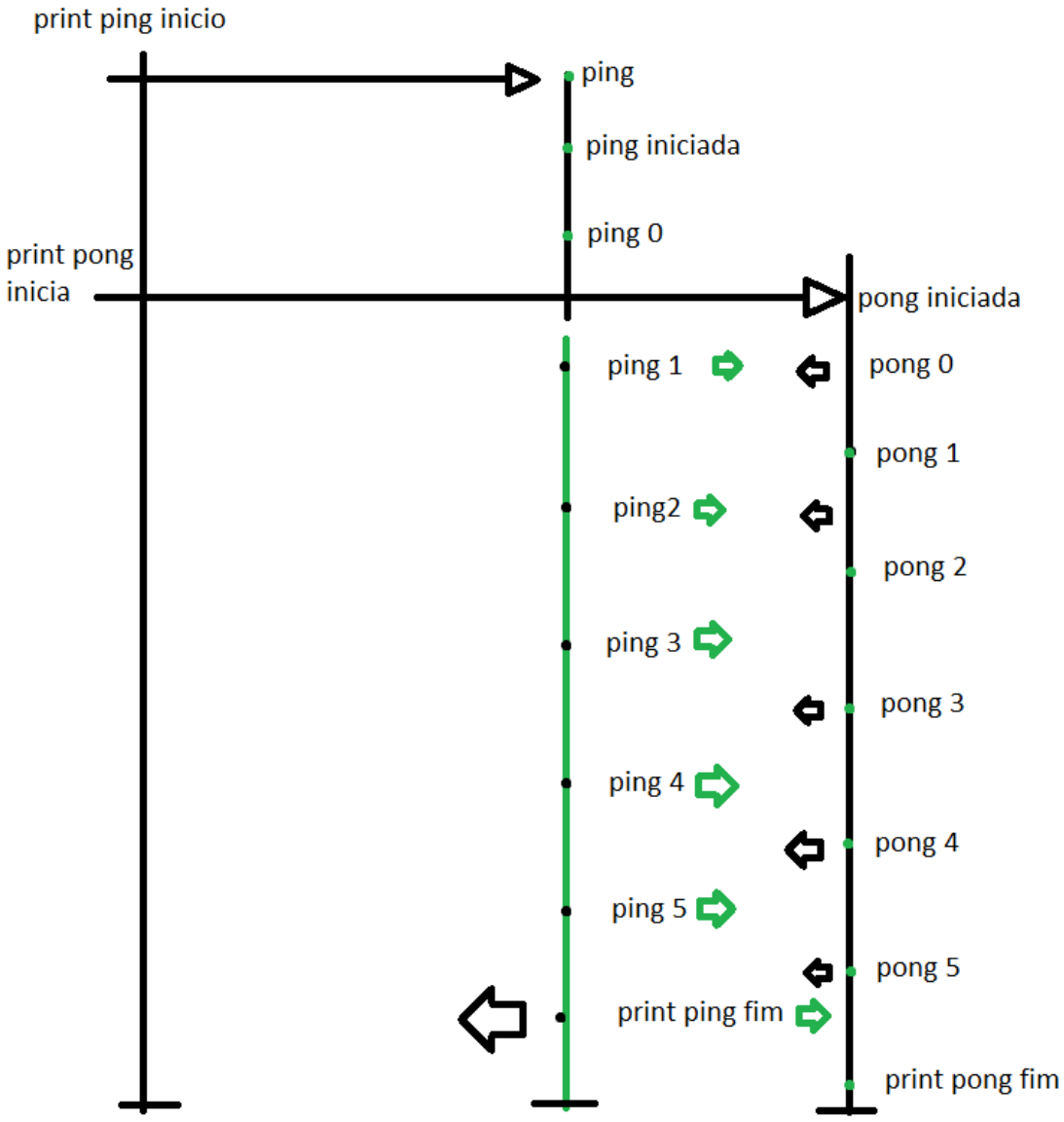

Figura 4 – Diagrama de tempo de execução.