Relatório Projeto P01 – Trocas de Contexto

Thiago Gonçalves Classen

Bacharelado Sistemas de Informação - UTFPR

15 de setembro de 2017

1. Funções de Manipulação de Contexto

1.1. Função getcontext(&a)

A função **getcontext** inicializa a estrutura apontada por **&a** para o contexto da thread de usuário atual. O parâmetro precisa ser do tipo **ucontext_t**, ele receberá o contexto do usuário e o conteúdo dos registradores da thread, o signal mask e a pilha de execução em seu estado atual.

1.2. Função setcontext(&a)

Restaura o contexto salvo no endereço apontado pelo parâmetro **&a.** Se realizado com sucesso ela não retorna, pois o sistema continua a execução no ponto especificado pelo contexto restaurado. O parâmetro deve ser previamente criado pelas funções **getcontext()** ou **makecontext().** Se o argumento foi criado através do **getcontext()**, o programa continua a execução a execução continua como se o **getcontext()** tivesse sido recém chamado. Caso o contexto tenha sido gerado com **makecontext()**, o programa continua a execução com a função passada para o **makecontext()**. No retorno da função a thread continua logo a após a chamada do **setcontext()** . Se o atributo **uc_link** da estrutura **ucontext_t** representada por **a** for 0, então este é o contexto principal e a thread será finalizada quando ocorrer algum retorno.

1.3. Função swapcontext(&a, &b)

Salva o contexto atual em **&a** e troca o contexto de usuário para **&b.**

1.4. Função makecontext(&ContextPing, function, id, arg)

Modifica o conteúdo de um contexto previamente inicializado por **getcontext()**. Quando este contexto é resumido utilizando **swapcontext()** ou **setcontext()**, o programa continua sua execução chamando a função passada por *function*, e utiliza os parâmetros em *arg* para sua chamada.

Antes da chamada ser realizada, o contexto sendo alterado deve ter uma pilha alocada para ele.

O atributo uc_link determina qual contexto será chamado quando este contexto retornar. Este atributo também deve ser inicializado antes da chamada de **makecontext()**.

2. Estrutura ucontext_t

2.1. *uc_link

Ponteiro que indica qual contexto será carregado após a finalização deste.

2.2. uc_sigmask

Conjunto de sinais de sistema que são bloqueados enquanto este contexto estiver ativo.

2.3. uc_stack

Pilha a ser usada por esse contexto.

2.4. uc_mcontext

Representação de máquina do estado do contexto atual.

3. Linhas do Código.

```c
// declara as estruturas de contexto a serem utilizadas.
ucontext_t ContextPing, ContextPong, ContextMain;

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

void BodyPing (void * arg)
{
   int i ;

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

   for (i=0; i<4; i++)
   {
      printf ("%s %d\n", (char *) arg, i) ;
      // troca o contexto de ContextPing para ContextPing
      swapcontext (&ContextPing, &ContextPong);
   }
   printf ("%s FIM\n", (char *) arg) ;
    // troca com ContextMain
   swapcontext (&ContextPing, &ContextMain) ;
}

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

```c
void BodyPong (void * arg)
{
   int i ;

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

   for (i=0; i<4; i++)
   {
      printf ("%s %d\n", (char *) arg, i) ;
      // troca o contexto de ContextPing para ContextPing
      swapcontext (&ContextPong, &ContextPing);
   }
   printf ("%s FIM\n", (char *) arg) ;
   // troca com ContextMain
   swapcontext (&ContextPong, &ContextMain) ;
}
```

```c
int main (int argc, char *argv[])
{
   char *stack ;

   printf ("Main INICIO\n");

   getcontext (&ContextPing); // inicia a estrutura com os parametros da thread atual.

   stack = malloc (STACKSIZE) ;
   if (stack)
   {
      ContextPing.uc_stack.ss_sp = stack ; // seta pilha de execução.
      ContextPing.uc_stack.ss_size = STACKSIZE; // tamanho da pilha de execução.
      ContextPing.uc_stack.ss_flags = 0; //
      ContextPing.uc_link = 0; // define este contexto como principal.
   }
   else
   {
      perror ("Erro na cria��o da pilha: ");
      exit (1);
   }
    // modifica o contexto para executar a função BodyPing, atribui id=1 e seta o parametro para BodyPing.
   makecontext (&ContextPing, (void*)(*BodyPing), 1, "    Ping");

   getcontext (&ContextPong); // inicia a estrutura com os parametros da thread atual.

   stack = malloc (STACKSIZE) ;
   if (stack)
   {
      ContextPong.uc_stack.ss_sp = stack ;  // seta pilha de execução.
      ContextPong.uc_stack.ss_size = STACKSIZE; // tamanho da pilha de execução.
      ContextPong.uc_stack.ss_flags = 0; // flags (??)
      ContextPong.uc_link = 0; // define este contexto como principal.
   }
   else
   {
      perror ("Erro na cria��o da pilha: ");
      exit (1);
   }
   // modifica o contexto para executar a função BodyPong, atribui id=1 e seta o parametro para BodyPong.
   makecontext (&ContextPong, (void*)(*BodyPong), 1, "        Pong");

   // troca o contexto principal para ContextPing
   swapcontext (&ContextMain, &ContextPing);
   // troca o contexto principal para ContextPong
   swapcontext (&ContextMain, &ContextPong);

   printf ("Main FIM\n");
```

4. Diagrama Tempo de Execução