Sistemas Operacionais
Projeto 01

Rafael Hideo Toyomoto - 1722085

ucontext_t: As variáveis **a** e **b** são do tipo **ucontext_t** e armazenam contextos de execução.

```
1 typedef struct ucontext {
2    struct ucontext *uc_link;
3    sigset_t         uc_sigmask;
4    stack_t          uc_stack;
5    mcontext_t       uc_mcontext;
6    ...
7 } ucontext_t;
```

A estrutura **ucontext_t** é utilizada para representar um contexto de usuário, onde a variável **uc_link** é um ponteiro para o próximo contexto a ser restaurado quando o atual terminar seu período execução, **uc_stack** é a stack utilizada pelo contexto e **mcontext_t** é a representação específica de máquina do contexto em questão.

getcontext(&a): salva o contexto atual na variável **a**.

Essa função tem como objetivo fornecer uma cópia do valor do contexto da tarefa ou processo em execução.

É realizado a passagem da variável **a**, do tipo **ucontext_t**, por referência, onde a função irá armazenar esses valores, e após sua execução basta utilizar a variável em questão para realizar a leitura desses valores.

setcontext(&a): restaura um contexto salvo anteriormente na variável **a**.

Essa função será utilizada para restaurar um contexto de execução desejado.

É realizada a passagem de um ponteiro para um contexto anterior, contexto salvo na variável **a** do tipo **ucontext_t**. A função realizará a restauração de contexto alterando o ponteiro de contexto de execução para esse passado.

swapcontext(&a, &b): salva o contexto atual em **a** e restaura o contexto salvo anteriormente em **b**.

Essa função basicamente realizará o processo das funções **getcontext(&a)** e **setcontext(&a)**.

A variável **a** será utilizada para armazenar o contexto atual e a variável **b** representa o contexto que deseja se restaurar.

makecontext(&a, ...): ajusta alguns valores internos do contexto salvo em **a**.

Nessa função, a variável **a** deverá ser um contexto obtido com a função **getcontext(&a)**, e ela servirá para modificar valores desse contexto **a**.

Um exemplo é quando realizado a chamada dessa função da seguinte maneira: **makecontext(&a,func())**. Nesse caso, o parâmetro **func()** irá determinar uma função que será executada quando esse contexto for executado, e o retorno dessa função determina o momento em que será realizada a troca de contexto utilizando o ponteiro **uc_link** salvo em **a** para saber o próximo contexto.

---

context.c: Explicação das linhas que utilizando alguma função ou utilizando a estrutura citada anteriormente.

```
1  ucontext_t ContextPing, ContextPong, ContextMain;
2
3  /*****************************************************/
4
5  void BodyPing (void * arg)
6  {
7      int i ;
8
9      printf ("%s iniciada\n", (char *) arg) ;
10
11     for (i=0; i<4; i++)
12     {
13         printf ("%s %d\n", (char *) arg, i) ;
14         // Salva o contexto atual (o próprio Ping) em Ping
15         // Troca para Pong (faz 1 iteração do laço do BodyPong, idêntico a ↩
               esse)
16         // Ao final de BodyPong, ele realizará o processo de retornar esse ↩
               contexto para execução
17         swapcontext (&ContextPing, &ContextPong);
18     }
19     printf ("%s FIM\n", (char *) arg) ;
20
21     // Após as 4 iterações do laço acima, o processamento retorna para a ↩
           ContextMain (contexto inicial)
22     // No ContextMain, o retorno fará a chamada do swapcontext(&ContextMain, &↩
           ContextPong)
23     swapcontext (&ContextPing, &ContextMain) ;
24 }
25
26 /*****************************************************/
27
```

```
28 void BodyPong (void * arg)
29 {
30     int i ;
31 
32     printf ("%s iniciada\n", (char *) arg) ;
33 
34     for (i=0; i<4; i++)
35     {
36         printf ("%s %d\n", (char *) arg, i) ;
37         // Salva o contexto atual (o próprio Pong) em Pong
38         // Troca para Ping (faz 1 iteração do laço do BodyPing, idêntico a esse)
39         // Ao final de BodyPing, ele realizará o processo de retornar esse contexto para execução
40         swapcontext (&ContextPong, &ContextPing);
41     }
42     printf ("%s FIM\n", (char *) arg) ;
43 
44     // Após as 4 iterações do laço acima e o final de Ping (chamado antes, termina antes), esse contexto é chamado mais uma vez
45     // Ele irá retornar nessa linha, que apenas devolverá o processamento para o ContextMain
46     swapcontext (&ContextPong, &ContextMain) ;
47 }
48 
49 /****************************************************/
50 
51 int main (int argc, char *argv[])
52 {
53     char *stack ;
54 
55     printf ("Main INICIO\n");
56 
57     // Salva a estrutura do contexto atual em Ping para inicializar um novo contexto
58     getcontext (&ContextPing);
59 
60     stack = malloc (STACKSIZE) ;
61     if (stack)
62     {
63         // Atribui valores para Ping
64         // Ponteiro para a base da pilha (Stack Pointer)
65         ContextPing.uc_stack.ss_sp = stack ;
66         // Tamanho da pilha (32768 bytes)
67         ContextPing.uc_stack.ss_size = STACKSIZE;
68         // Flag que indica se a stack está em uso (controlada pelo sistema) OU está desabilitada para uso
```

```
69          ContextPing.uc_stack.ss_flags = 0;
70          // Ponteiro para o próximo contexto
71          ContextPing.uc_link = 0;
72      }
73      else
74      {
75          perror ("Erro na criação da pilha: ");
76          exit (1);
77      }
78
79      // Modifica o contexto Ping para executar a função BodyPing
80      makecontext (&ContextPing, (void*)(*BodyPing), 1, "    Ping");
81
82      // Salva a estrutura do contexto atual em Pong para inicializar um novo ↩
            contexto
83      getcontext (&ContextPong);
84
85      stack = malloc (STACKSIZE) ;
86      if (stack)
87      {
88          // Atribui valores para Pong
89          ContextPong.uc_stack.ss_sp = stack ;
90          ContextPong.uc_stack.ss_size = STACKSIZE;
91          ContextPong.uc_stack.ss_flags = 0;
92          ContextPong.uc_link = 0;
93      }
94      else
95      {
96          perror ("Erro na criação da pilha: ");
97          exit (1);
98      }
99
100     // Modifica o contexto Pong para executar a função BodyPong
101     makecontext (&ContextPong, (void*)(*BodyPong), 1, "        Pong");
102
103     // Salva o contexto atual em Main e modifica para o contexto Ping (Executa↩
            a função BodyPing)
104     swapcontext (&ContextMain, &ContextPing);
105     // Salva o contexto atual em Main e modifica para o contexto Pong (Executa↩
            a função BodyPong)
106     swapcontext (&ContextMain, &ContextPong);
107
108     printf ("Main FIM\n");
109
110     exit (0);
111 }
```

Diagrama de execução: