```
//
//  Exemplo de simulação simples de escalonamento
//
//  compilar:   gcc simulador.c -o simul -D DEBUG
//  executar:   ./simul

#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/time.h>

#define NUM_PROCESSOS   20
#define TAM_MEMORIA   1999
#define NUM_TICKS       2

//#define DEBUG

int processo_id = -1, pid = 0;

enum estado_Tarefa {
    RODANDO,  PRONTA, BLOQUEADA, DORMINDO,
    SUSPENSA, MORTA
};

struct {
    int id,                 /* PID do processo....*/
     ci,                  /* contador de instrucoes*/
     bd,                  /* base de dados - memoria*/
     r0, r1, r2, r3, r4; /* registradores*/
} _cpu;

int memoria[ TAM_MEMORIA ];

struct lista_Processos {
    int prim,           /* cabeca da lista*/
        ultm;           /* cauda*/
} prontos;

struct {
    int id,                     /* PID do processo*/
     ci,                    /* contador de instrucoes*/
     bd,                    /* base de dados - memoria*/
     r0, r1, r2, r3, r4;     /* registradores*/
    enum estado_Tarefa estado;
    void (*processo)();         /*ponteiro para o processo*/
    int proximo;                /* link para proximo processo.*/
} tabela_Processos[ NUM_PROCESSOS ];


void kernel_panic( char *mensagem )
{
    printf( "KERNEL PANIC: %s...\n", mensagem );
    exit( 0 );
}


void salva_contexto()
{
  tabela_Processos[ pid ].ci = _cpu.ci; /*Salva o numero de instrucoes do processo*/
  tabela_Processos[ pid ].bd = _cpu.bd; /*Salva base de dados*/
  tabela_Processos[ pid ].r0 = _cpu.r0;/*Salva o registrador r0*/
  tabela_Processos[ pid ].r1 = _cpu.r1; /*Salva o registrador r1*/
  tabela_Processos[ pid ].r2 = _cpu.r2; /*salva o registrador r2*/
  tabela_Processos[ pid ].r3 = _cpu.r3; /*salva o registrador r3*/
  tabela_Processos[ pid ].r4 = _cpu.r4; /*Salva o registrador r4*/
}

void restaura_contexto() /*
                            Função que retorna os valores dos registradores,
                            número de instruções, base de dados de um processo que
                            foi salvo anteriormente
                         */
{
   _cpu.ci = tabela_Processos[ pid ].ci;
   _cpu.bd = tabela_Processos[ pid ].bd;
   _cpu.r0 = tabela_Processos[ pid ].r0;
   _cpu.r1 = tabela_Processos[ pid ].r1;
   _cpu.r2 = tabela_Processos[ pid ].r2;
   _cpu.r3 = tabela_Processos[ pid ].r3;
   _cpu.r4 = tabela_Processos[ pid ].r4;
}

#ifdef DEBUG /*Se definido DEBUG*/

```

```
85      void mostra_tabela_Processos( void )
86      {
87           int _ii;
88           printf("\tPID- PROX CI-- BD-- R0-- R1-- R2-- R3-- R4--\n" );
89           printf("\t==== ==== ==== ==== ==== ==== ==== ==== ====\n" );
90           for( _ii = 0; _ii <= processo_id; ++_ii ) {
91               printf("\t%4d %4d %4d %4d %4d %4d%4d %4d %4d\n", tabela_Processos[ _ii ].id,
92                                                               tabela_Processos[ _ii ].proximo,
93                                                               tabela_Processos[ _ii ].ci,
94                                                               tabela_Processos[ _ii ].bd,
95                                                               tabela_Processos[ _ii ].r0,
96                                                               tabela_Processos[ _ii ].r1,
97                                                               tabela_Processos[ _ii ].r2,
98                                                               tabela_Processos[ _ii ].r3,
99                                                               tabela_Processos[ _ii ].r4
100                                                              );
101          }
102     }
103
104 #endif
105
106 int escolhe_processo()
107 {
108   int meu_pid;
109
110   #ifdef DEBUG
111
112     printf("DEBUG: escolhe processo processo id:%3d proximo: %3d\n", pid, tabela_Processos[ pid
    ].proximo );
113     mostra_tabela_Processos();
114
115   #endif
116
117   meu_pid = tabela_Processos[ pid ].proximo;
118   if( meu_pid < 0 )
119       meu_pid = processo_id;  // estah mandando para o ultimo...
120   return meu_pid;
121 }
122
123 void escalonador ( int signum )
124 {
125   static int contador = 0;
126   static volatile int reentrada = 0;
127
128   if( reentrada > 0 ) {
129     kernel_panic( "\n\nChamando escalonador 2x!!!\n\n" );
130   }
131
132   reentrada = 1;
133
134   #ifdef DEBUG
135
136      printf ( "DEBUG: relogio contador: %d\n", contador );
137
138   #endif
139
140   salva_contexto();
141   if( ++contador > NUM_TICKS ) {
142     pid = escolhe_processo();
143     contador = 0;
144   }
145   restaura_contexto();
146   reentrada = 0;
147   (tabela_Processos[ pid ].processo)();
148 }
149
150 void init_proc( void )
151 {
152   struct sigaction sa;
153                                       /*A chamada de sistema sigaction é usada para
154                                       alterar a ação tomada por um processo no
155                                       recebimento de um sinal específico
156                                       Signum especififca o sinal e pode ser qualquer
157                                       sinal válido, exceto SIGKILL e SIGSTOP
158                                       Se for não NULL, a nova ação para o sinal*/
159   struct itimerval relogio;
160
161   memset( memoria, 0, TAM_MEMORIA );
162   memset( &sa, 0, sizeof( sa ) );
163
164   sa.sa_handler = &escalonador;
165   sigaction( SIGVTALRM, &sa, NULL );   /* registra o manipulador do SIGVTALARM - escalonador*/
166
167   relogio.it_value.tv_sec = 0;
```

```
168     relogio.it_value.tv_usec = 200000;
169     relogio.it_interval.tv_sec = 0;
170     relogio.it_interval.tv_usec = 200000;
171     setitimer( ITIMER_VIRTUAL, &relogio, NULL );  /*registra o intervalo para cada SIGVTALARM*/
172   }
173
174   int cria_processo( void (*proc)() ) /*Função para criar um processo*/
175   {
176       #ifdef DEBUG /*Se definido DEBUG*/
177
178          printf( "DEBUG: cria processo processo id: %d\n", processo_id );
179          mostra_tabela_Processos();
180
181       #endif
182
183       if( processo_id < NUM_PROCESSOS-1 ) {
184         tabela_Processos[ ++processo_id ].id       = processo_id;
185         tabela_Processos[   processo_id ].ci       = 0;
186         tabela_Processos[   processo_id ].bd       = 0;
187         tabela_Processos[   processo_id ].r0       = 0;
188         tabela_Processos[   processo_id ].r1       = 0;
189         tabela_Processos[   processo_id ].r2       = 0;
190         tabela_Processos[   processo_id ].r3       = 0;
191         tabela_Processos[   processo_id ].r4       = 0;
192         tabela_Processos[   processo_id ].processo = proc;
193
194         if( processo_id > 0 )
195           tabela_Processos[ processo_id ].proximo  = processo_id-1;
196         else
197           tabela_Processos[ processo_id ].proximo  = -1;
198
199       } else
200         kernel_panic( "muitos processos" );
201   }
202
203   void processo_1( void )
204   {
205       switch( _cpu.ci ) {
206         case 0 : printf( "1: Este é o processo 1\n" );
207                  _cpu.ci++;
208                  break;
209         case 1 : _cpu.ci = 0;
210                  break;
211       }
212   }
213
214   void processo_2( void )
215   {
216       switch( _cpu.ci ) {
217          case 0 : printf( "2: Este é o processo 2\n" );
218                   _cpu.ci++;
219                   break;
220          case 1 : printf( "2: Fazendo alguma outra coisa\n" );
221                   _cpu.ci++;
222                   break;
223          case 2 : printf( "2: Ainda fazendo algo...\n" );
224                   _cpu.ci++;
225                   break;
226          case 3 :  _cpu.ci = 0;
227                   break;
228       }
229   }
230
231   void processo_3( void )
232   {
233       switch( _cpu.ci ) {
234          case 0 : printf( "3: Este é o processo 3\n" );
235                   _cpu.ci++;
236                   break;
237          case 1 : _cpu.ci = 0;
238                   break;
239       }
240   }
241
242   void processo_4( void )
243   {
244       switch( _cpu.ci ) {
245          case 0 : printf( "4: Este é o processo 4\n" );
246                   _cpu.ci++;
247                   break;
248          case 1 : _cpu.ci = 0;
249                   break;
250       }
251   }
```

```
252
253   int main ()
254   {
255     init_proc();
256     cria_processo( (void *) processo_1 );
257     cria_processo( (void *) processo_2 );
258     cria_processo( (void *) processo_3 );
259     cria_processo( (void *) processo_4 );
260     while( 1 )                              /*loop infinito*/
261        ;
262   }
263
```