Relatório
Tempo Real - 2014.2
Universidade Federal do Rio de Janeiro

Gabriel Rodrigues de Lira
gabriellira@poli.ufrj.br
Felipe de Carvalho Gomes de Oliveira
felipecgo@poli.ufrj.br

24 de novembro de 2014

Índice

1 Descrição do Algoritmo

O PCP(Priority Ceiling Protocol) tem como princípio estender o PIP(Priority Inheritance Protocol) com a adição de uma regra de concessão para um pedido de bloqueio de um semáforo livre.

Com isso, o PCP impede que uma tarefa entre na região crítica se existe(m) semáforo(s), que possivelmente pode(m) bloqueá-la, esteja(m) trancado(s), evitando assim que ocorra um bloqueamento múltiplo. Ou seja, quando uma tarefa entrar numa seção crítica, ela nunca poderá ser bloqueada por uma tarefa de prioridade mais baixa até que esteja completa.

Com esse propósito, cada semáforo recebe uma "*priorityceiling*" igual à prioridade mais alta das tarefas que podem trancar este semáforo. Sendo assim, só é permitido a uma tarefa τ_i entrar na região crítica se sua prioridade for maior do que todas as prioridades dadas aos semáforos trancados, por outras tarefas, naquele instante.

O algoritmo funciona da seguinte forma:

Inicialmente, cada semáforo S_k recebe uma prioridade $C(S_k)$ igual a maior prioridade das tarefas que podem trancá-lo.Em seguida analisa-se qual a tarefa ativa com a maior prioridade. Feito isso, designa-se esta tarefa para o processador.

Para entrar numa seção crítica guardada por um semáforo S^*, τ_i precisa ter uma prioridade mais alta do que $C(S^*)$, caso contrário, o travamento de S^* é negadp e τ_i é bloqueada no semáforo S^* pela tarefa responsável pelo travamento de S^*. Nesse caso, a tarefa τ_i transmite sua prioridade para a tarefa τ_j que está travando o semáforo, isto é τ_j herda a prioridade de τ_i, e assim sendo a tarefa τ_j retorna a ser executada até terminar sua região crítica.

Terminada a região crítica de τ_j ela destrava o semáforo S^* e atualiza sua prioridade, caso τ_j não esteja bloqueando mais nenhuma tarefa ela retorna à sua prioridade normal, caso contrário ele herda a maior prioridade das tarefas que ainda está bloqueando.

Observação:

A herança de prioridades é transitória, ou seja caso a tarefa τ_3 esteja bloqueando a tarefa τ_2 que por sua vez bloqueia τ_1, a tarefa τ_3 herda a prioridade de τ_1.

2 Trechos importantes do código

Primeiramente é importante ressaltar que os trechos de códigos vistos abaixo foram escritos na linguagem do Matlab.

2.1 Variáveis importantes

```
handles.ns = 0;
handles.Soriginal = [];

.
.
.

chegada = str2num(get(handles.edit1, 'String'));

tempComp = str2num(get(handles.edit2, 'String'));

deadline = str2num(get(handles.edit3, 'String'));

prioridade = str2num(get(handles.edit4, 'String'));
```

No trecho acima pode-se observar algumas variáveis importantes para o bom funcionamento do protocolo estudado, em que:

- handles.ns é o número de semáforos utilizados por um determinado conjunto de tarefas;
- handles.Soriginal são os parâmetros de cada semáforo e sua relação com cada tarefa, ou seja, quanto tempo a tarefa irá utilizá-lo e apartir de que instante ela necessitará desse semáforo.

São as variáveis globais do código, isto é, podem ser acessadas por qualquer função do programa. Já as variáveis abaixo, são as variaveis locais da função principal do programa.

- chegada é um array que contém os tempo de chegada de cada tarefa;
- tempComp é um array que armazena os tempos de execução para cada tarefa;
- deadline array para armazenar os tempos limites(deadlines) de cada tarefa a ser executada;
- prioridade é um array que armazena as prioridades "normal"de cada tarefa;

É importante observar a função $get(handle.,'')$ vistas na atribuição de cada uma dessas últimas variáveis listadas, isso uma vez que esta função é responsável por adquirir o valor do campo correspondente àquela variável na interface.

2.2 Botão de inserção

Nesta seção analisa-se o funcionamento do botão *Inserir*.

```
function pushbutton2_Callback(hObject, eventdata, handles)
w = handles.ns + 1;
initial_temp = cellstr(get(handles.listbox1,'String'))
a = get(handles.edit6, 'String');
temp = [initial_temp; a];
set(handles.listbox1, 'String', temp);
b =  str2num(a);
handles.ns = w;
handles.Soriginal(:,:,w) = str2num(get(handles.edit6, 'String'));
guidata(hObject, handles);
```

Este botão, como o nome sugere serve para a inserção de um semáforo na execução das tarefas, para isso é necessário que se escreva os parâmetros do semáforo, isto é, tempo em que cada tarefafaz o requerimento para utilizá-lo; duração da execução de cada tarefa utilizando este semáforo e por fim o "status"da tarefa em relação ao semáforo sendo:

1. 0 → para tarefas inativas;
2. 1 → para tarefas ativas;
3. 2 → para tarefas finalizadas ou inexistentes em relação ao semáforo.

2.3 Loop principal

Nesta seção é mostrado o trecho do código onde "a mágica acontece", isso tendo em vista que é nesse trecho que todas as comparações e definições de prioridades ocorrem como pode ser visto abaixo:

```
if length(tam)< 3;
    tam(3) = 1;
end
for i=1:tam(3)
    prioridadeBloqueada(i)=101;

    [cores(i,1),cores(i,2),cores(i,3)]=HSVtoRGB(1+(i-1)*(358/(tam(3))),.41,.86);

    for j=1:tam(1)
        if S(j,2,i)>0
            C(i)=j;
            break;
        end
    end
end

C(length(C)+1)=100;

tempo=0;

for i=1:length(chegada)
    prioridadeAtiva(i)=100;
end


ativos=zeros(length(chegada));


largura=max((sum(sum(S(:,2,:)))+sum(tempComp))*1.2,(max(deadline)+2));


tic;


cla
hold on
grid on
set(gca,'xtick',[0:1:largura])

xlim([0,largura])
ylim([0,20*(1+length(chegada))])
set(gca,'YTick',[0:20:20*length(chegada)])

for i=1:length(chegada)
    plot([chegada(i),chegada(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i)
     +15],'-.b','LineWidth',2)
    plot([deadline(i),deadline(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i
     )+15],'-.r','LineWidth',2)
end

drawnow


SmaiorCeiling=length(C);


while (true)

    paraContador=toc;


    tic;


    tempo=tempo+paraContador;

```

```
    for i=1:length(chegada)
        if ~ativos(i) && chegada(i)<=tempo
            ativos(i)=1;
            prioridadeAtiva(i)=prioridade(i);
        end
    end


    [minimo,proximoProcesso]=min(prioridadeAtiva);

    for i=1:tam(3)
        if S(proximoProcesso,3,i)==0 & compFeita(proximoProcesso) >= S(proximoProcesso,1,i)
            for j=1:tam(3)
                if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
                    SmaiorCeiling=j;
                end
            end

            if prioridadeAtiva(proximoProcesso)<C(SmaiorCeiling) | any(S(proximoProcesso,3,:)==1)
                S(proximoProcesso,3,i)=1;

            else

                prioridadeAtiva(find((S(:,3,SmaiorCeiling))==1))=proximoProcesso-0.1;

                prioridadeBloqueada(SmaiorCeiling)=proximoProcesso-0.1;
            end
        end
    end

    if minimo<100

        if ~(any(S(proximoProcesso,3,:)==1))

            compRestante(proximoProcesso)=compRestante(proximoProcesso)-paraContador;
            if compRestante(proximoProcesso)<0
                compRestante(proximoProcesso)=0;
                tempo=tempo+compRestante(proximoProcesso);
                paraContador=paraContador-compRestante(proximoProcesso);
            end

            compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;

            rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-proximoProcesso),paraContador,10],'FaceColor',[0.6 0.6 0.9],'LineStyle','None');
            drawnow

        else

            zonaAtiva=[0 0];
            for i=1:tam(3)
                if S(proximoProcesso,3,i)==1 & S(proximoProcesso,1,i)>=zonaAtiva(2)
                    zonaAtiva=[i S(proximoProcesso,1,i)];
                end
            end

            S(proximoProcesso,2,zonaAtiva(1))=S(proximoProcesso,2,zonaAtiva(1))-paraContador;
            compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;

            if S(proximoProcesso,2,zonaAtiva(1))<0
                S(proximoProcesso,2,zonaAtiva(1))=0;
                tempo=tempo+S(proximoProcesso,2,zonaAtiva(1));
                paraContador=paraContador-S(proximoProcesso,2,zonaAtiva(1));
            end

            rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-proximoProcesso),paraContador,10],'FaceColor',cores(zonaAtiva(1),:),'LineStyle','None');
            xlim([0,largura])
            ylim([0,20*(1+length(chegada))])
            set(gca,'YTick',[0:20:20*length(chegada)])
            drawnow

            if S(proximoProcesso,2,zonaAtiva(1))<=0
                %Marca fim da zona
                S(proximoProcesso,3,zonaAtiva(1))=2;

                prioridadeBloqueada(zonaAtiva(1))=101;

                prioridadeAtiva(proximoProcesso)=proximoProcesso;
                for i=1:tam(3)
                    if S(proximoProcesso,3,i)==1
                        prioridadeAtiva(proximoProcesso)=min(prioridadeAtiva(proximoProcesso),prioridadeBloqueada(i));
                    end
                end

                SmaiorCeiling=length(C);

                for j=1:tam(3)
                    if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
                        SmaiorCeiling=j;
                    end
```

```
154                     end
155
156                 end
157
158
159             end
160         end
161
162         if compRestante(proximoProcesso)<=0
163
164             prioridadeAtiva(proximoProcesso)=105;
165
166             if max(compRestante)<=0
167                 break;
168             end
169         end
```

2.4 Explicação do Programa

Para um melhor entendimento do programa uma breve explicação do mesmo se faz necessária.

Para a simulação, o usuário deve fornecer vetores com as informações dos tempos de chegada, tempo de computação, deadline e as informações sobre as zonas críticas.

As zonas críticas, associadas cada uma a um recurso exlusivo e um semáforo, devem ser fornecidas com tempos de chegada e de computação. O tempo de chegada é o tempo total de computação da tarefa que deve ter se passado para que ela entre numa zona crítica, esse tempo leva em conta outras zonas críticas pelas quais essa tarefa pode ter passado.

O simulador funciona em loop, que só acaba quando todas as tarefas terminam de ser computadas. A cada ciclo, o programa mede quanto tempo se passou desde o ciclo anterior, para simular o melhor possível um sistema em tempo real.

Ao início de cada loop o programa primeiro checa se alguma tarefa nova chegou, caso sim, ele a marca com uma tarefa ativa. Após isso, há uma checagem de qual é a tarefa ativa que possui maior prioridade (no caso o valor menor entre os dados pelo usuário) e essa é a tarefa que rodará nesse ciclo. A tarefa que roda no ciclo atual começa verificando se ela acabou de entrar em uma zona crítica. Caso tenha entrado, é preciso verificar se há algum ceiling $C(S^*)$ com prioridade maior ou igual à da tarefa atual entre os semáforos ativos, para bloquear e passar a prioridade da tarefa pra S^*, caso sim, ou permitir a entrada na zona crítica, caso não.

Após essa verificação, há a verificação de se há alguma zona crítica ativa ou não. Caso haja, o programa identifica qual é e desconta do seu contador de duração da zona crítica o tempo do ciclo, verificando se a zona crítica chegou ao fim, para poder liberar a tarefa de maior prioridade por ela bloqueada e reduzir a prioridade da tarefa atual de acordo com a lógica do PCP. Caso não haja zona crítica ativa, o tempo do ciclo é descontado do contador de duração da computação, verificando se a computação chegou ao fim, para desativar a tarefa.

Com essa forma de funcionamento, o simulador é capaz de simular o PCP ao longo de tantos loops quanto forem necessários para que todas as tarefas terminem de computar.

3 Exemplos de simulações

Nesta seção é possível observar algumas simulações feitas afim de exemplificar o PCP e assim verificar se a implementação do mesmo está correta.

3.1 Teste 1

3.1.1 Configurações

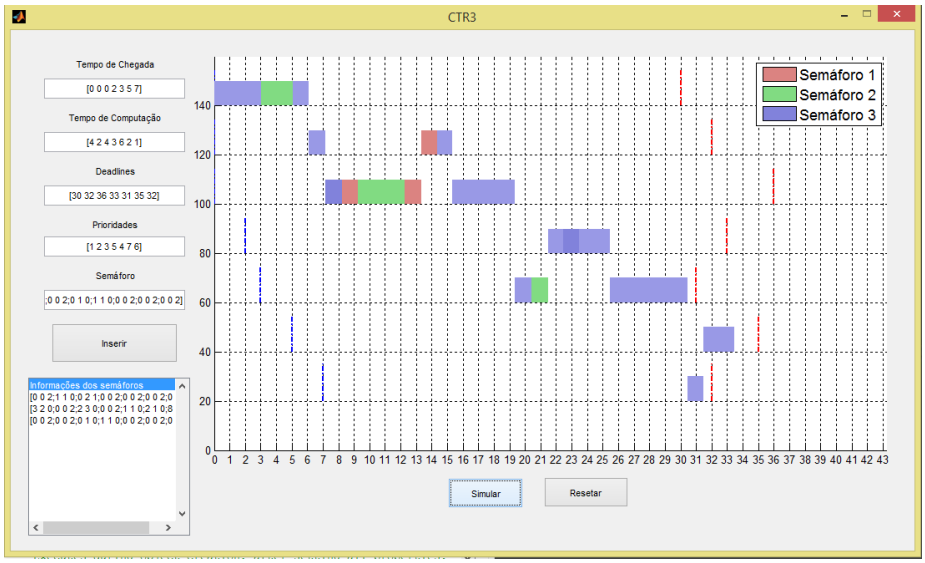
Tabela 1: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
a_i	0	0	0	2	3	5	7
C_i	4	2	4	3	6	2	1
D_i	30	32	36	33	30	40	31
P_i	1	2	3	5	4	7	6

Tabela 2: *Semáforo.*

Tarefas	Sa_1	Sa_C1	Status 1	Sa_2	Sa_C2	Status 2	Sa_3	Sa_C3	Status 3
τ_1	0	0	2	3	2	0	0	0	2
τ_2	1	1	0	0	0	2	0	0	2
τ_3	0	2	1	2	3	0	0	1	0
τ_4	0	0	2	0	0	2	1	1	0
τ_5	0	0	2	1	1	0	0	0	2
τ_6	0	0	2	2	1	0	0	0	2
τ_7	0	0	2	0	0	2	0	0	2

3.1.2 Resultado da Simulação



3.2 Teste 2

3.2.1 Configurações

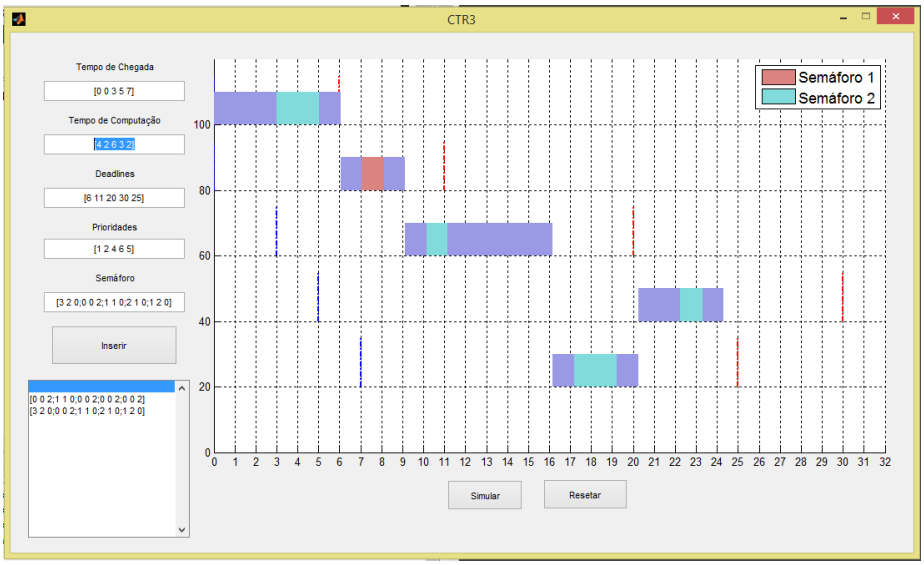
Tabela 3: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5
a_i	0	0	3	5	7
C_i	4	2	6	3	2
D_i	6	11	20	30	25
P_i	1	2	4	6	5

Tabela 4: *Semáforo.*

Tarefas	Sa_1	Sa_C1	Status 1	Sa_2	Sa_C2	Status 2
τ_1	0	0	2	3	2	0
τ_2	1	1	0	0	0	2
τ_3	0	0	2	1	1	0
τ_4	0	0	2	2	1	0
τ_5	0	0	2	1	2	0

3.2.2 Resultado da Simulação



3.3 Teste 3

3.3.1 Configurações

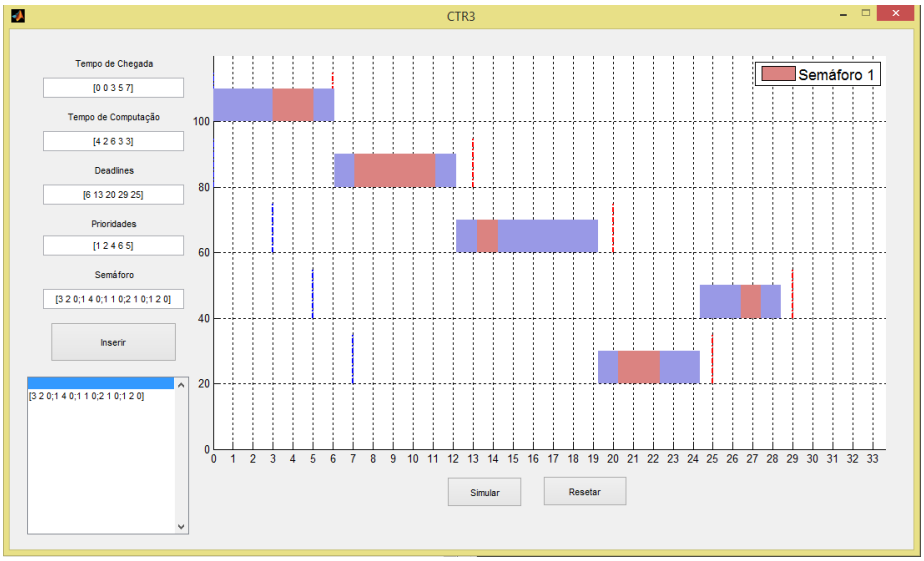
Tabela 5: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5
a_i	0	0	3	5	7
C_i	4	2	6	3	3
D_i	6	13	20	29	25
P_i	1	2	4	6	5

Tabela 6: *Semáforo.*

Tarefas	Sa_1	Sa_C1	Status 1
τ_1	3	2	0
τ_2	1	4	0
τ_3	1	1	0
τ_4	2	1	0
τ_5	1	2	0

3.3.2 Resultado da Simulação



3.4 Teste 4

3.4.1 Configurações

Tabela 7: *Tarefas.*

Parâmetros	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
a_i	1	0	0	0	0	0	0	0
C_i	4	2	7	5	6	3	3	9
D_i	30	32	36	33	37	43	45	55
P_i	1	2	3	5	4	7	6	8

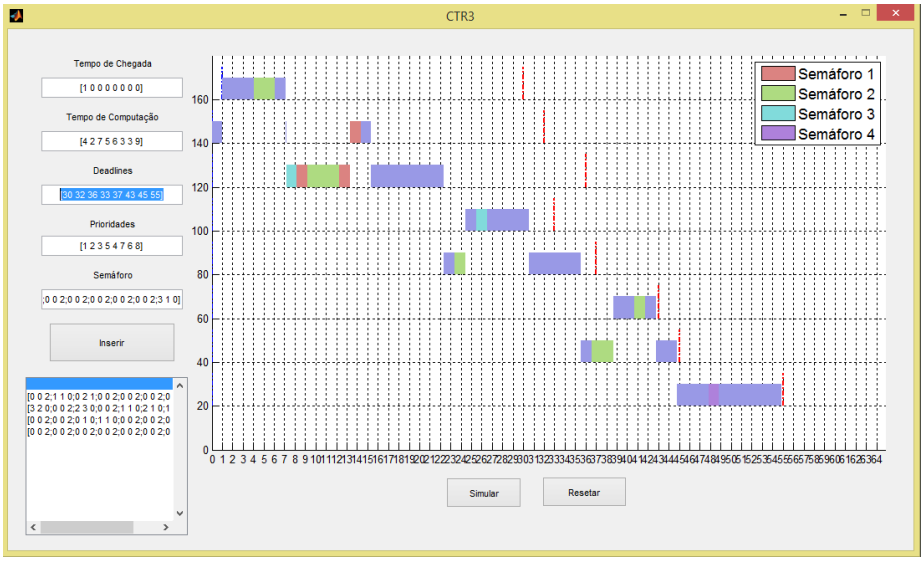
Tabela 8: *Semáforos 1.*

Tarefas	Sa_1	Sa_C1	Status 1	Sa_2	Sa_C2	Status 2
τ_1	0	0	2	3	2	0
τ_2	1	1	0	0	0	2
τ_3	0	2	1	2	3	0
τ_4	0	0	2	0	0	2
τ_5	0	0	2	1	1	0
τ_6	0	0	2	2	1	0
τ_7	0	0	2	1	2	0
τ_8	0	0	2	0	0	2

Tabela 9: *Semáforos 2.*

Tarefas	Sa_3	Sa_C3	Status 3	Sa_4	Sa_C4	Status 4
τ_1	0	0	2	0	0	2
τ_2	0	0	2	0	0	2
τ_3	0	1	0	0	0	2
τ_4	1	1	0	0	0	2
τ_5	0	0	2	0	0	2
τ_6	0	0	2	0	0	2
τ_7	0	0	2	0	0	2
τ_8	0	0	2	3	1	0

3.4.2 Resultado da Simulação



4 Anexo

```
function varargout = CTR3(varargin)
% CTR3 MATLAB code for CTR3.fig
%      CTR3, by itself, creates a new CTR3 or raises the existing
%      singleton*.
%
%      H = CTR3 returns the handle to a new CTR3 or the handle to
%      the existing singleton*.
%
%      CTR3('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CTR3.M with the given input arguments.
%
%      CTR3('Property','Value',...) creates a new CTR3 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before CTR3_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to CTR3_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help CTR3

% Last Modified by GUIDE v2.5 27-Nov-2014 00:06:00

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @CTR3_OpeningFcn, ...
                   'gui_OutputFcn',  @CTR3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before CTR3 is made visible.
function CTR3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to CTR3 (see VARARGIN)

% Choose default command line output for CTR3
handles.output = hObject;
% Numero de semaforos
handles.ns = 0;
% Inicializacao da variavel dos semaforos
handles.Soriginal = [];

set(gca,'YTick',[0:20:100])
%set(gca,'YColor','w')
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes CTR3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = CTR3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% tic;
% i=3000;
% while (i>0)
%     hold on
%     a=toc;
%     tic;
%     i=i-1;
%     if(mod(i,100)==0)
%         plot(i,i,'*')
%         drawnow
%     end
% end


%Array com o tempo de chegada das tarefas
chegada = str2num(get(handles.edit1, 'String'));
%chegada=[3 0 4];

%Array com tempo de computacao das tarefas
tempComp = str2num(get(handles.edit2, 'String'));
%tempComp=[3 2 2];

%Array com deadline das tarefas
deadline = str2num(get(handles.edit3, 'String'));
%deadline=[2 5 4];

%Array com as prioridades das tarefas
%Quanto menor o valor, maior a prioridade
prioridade = str2num(get(handles.edit4, 'String'));
%prioridade=[1 2 3];

%Zonas criticas guardadas por semaforos
%Elementos de uma linha (relativa a uma tarefa):
%1- Onde comeca em relacao a computacao da tarefa
%2- Duracao
%3- Ativa ou nao (0 -> inativa, 1-> ativa, 2->finalizada ou inexistente)
% Soriginal=[0 0 2;1 2 0;0 1 0]
%
% Soriginal(:,:,2)=[1 3 0;2 2 0;0 0 2]


%%%%% Fim dos valores do problema %%%%%%%%%%%%

%Cria copia de Soriginal para ser modificada durante execucao
S = handles.Soriginal;
%S=Soriginal;

%Array com tempo de computacao restante das tarefas
compRestante=tempComp;

%Array com tempo de computacao gasto das tarefas
compFeita=zeros(length(tempComp));

% %Tempo total contando com zonas criticas
tam = size(S);

%compTotal=tempComp;
% for i=1:tam(3)
%     compTotal=compTotal+S(:,2,i)';
%     %Maior prioridade bloqueada por S
%     Sbloq(i)=102;
% end
if length(tam)< 3;
```

```
150     tam(3) = 1;
151 end
152 %Achando Ceilings (menor prioridade)
153 for i=1:tam(3)
154     %Menor prioridade bloqueada por cada sinal
155     prioridadeBloqueada(i)=101;
156 
157     %Escolha das cores no plot de cada secao critica
158     [cores(i,1),cores(i,2),cores(i,3)]=HSVtoRGB(1+(i-1)*(358/(tam(3))),.41,.86);
159 
160     for j=1:tam(1)
161         if S(j,2,i)>0
162             C(i)=j;
163             break;
164         end
165     end
166 end
167 
168 % if exist('C') == 0
169 %     C = [];
170 % end
171 %Adicionando Ceiling quando nao ha semaforos ativos
172 C(length(C)+1)=100;
173 
174 %Tempo decorrido desde o comeco da simulacao
175 tempo=0;
176 
177 %Array com prioridades das tarefas
178 for i=1:length(chegada)
179     prioridadeAtiva(i)=100;
180 end
181 
182 %Tarefas que ja chegaram
183 ativos=zeros(length(chegada));
184 
185 %Largura total do plot
186 largura=max((sum(sum(S(:,2,:)))+sum(tempComp))*1.2,(max(deadline)+2));
187 
188 %Comeca a contar o relogio do simulador
189 tic;
190 
191 %Funcoes do plot
192 cla
193 hold on
194 grid on
195 set(gca,'xtick',[0:1:largura])
196 
197 xlim([0,largura])
198 ylim([0,20*(1+length(chegada))])
199 set(gca,'YTick',[0:20:20*length(chegada)])
200 
201 for i=1:length(chegada)
202     plot([chegada(i),chegada(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i)
      +15],'-.b','LineWidth',2)
203     plot([deadline(i),deadline(i)],[20*(length(chegada)+1-i),20*(length(chegada)+1-i
      )+15],'-.r','LineWidth',2)
204 end
205 
206 drawnow
207 
208 %Semaforo de maior ceiling
209 SmaiorCeiling=length(C);
210 
211 %%%% Loop Principal %%%%
212 
213 while (true)
214     %Para contador e guarda o tempo marcado
215     paraContador=toc;
216 
217     %Recomeca o contador
218     tic;
219 
220     %Acrescenta o tempo de iteracao ao tempo total
221     tempo=tempo+paraContador;
222 
223     %Checa processos atuais para saber quais sao ativos
224     for i=1:length(chegada)
225         if ~ativos(i) && chegada(i)<=tempo
226             ativos(i)=1;
227             prioridadeAtiva(i)=prioridade(i);
228         end
229     end
230 
231     %Decide quem vai ser a tarefa processada (menor prioridade
232     % entre tarefas ativas)
233     [minimo,proximoProcesso]=min(prioridadeAtiva);
234 
235     %Checa para ver se alguma zona critica e ativada
236     for i=1:tam(3)
237         if S(proximoProcesso,3,i)==0 & compFeita(proximoProcesso) >= S(
      proximoProcesso,1,i)
238             %Checa maior Ceiling (menor prioridade) entre semaforos ativos
239             for j=1:tam(3)
240                 if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
241                     SmaiorCeiling=j;
242                 end
243             end
244 
245             %Checa para ver se e possivel entrar na zona critica (nao ha
```

```
246             %bloqueio
247             if prioridadeAtiva(proximoProcesso)<C(SmaiorCeiling) | any(S(
    proximoProcesso,3,:)==1)
248                 S(proximoProcesso,3,i)=1;
249
250             %Tarefa e bloqueada quando tenta acessar zona critica
251             else
252                 %Tarefa que bloqueou pega a prioridade (fica menor que a original
    pra ser
253                 %escolhida antes)
254                 prioridadeAtiva(find((S(:,3,SmaiorCeiling))==1))=proximoProcesso
    -0.1;
255
256                 %Guarda quem teve prioridade bloqueada
257                 prioridadeBloqueada(SmaiorCeiling)=proximoProcesso-0.1;
258             end
259         end
260     end
261
262     %Checa se todos processos ja acabaram
263     if minimo<100
264
265         %Verifica se esta em uma zona critica
266         if ~(any(S(proximoProcesso,3,:)==1))
267
268             %Processa a tarefa
269             compRestante(proximoProcesso)=compRestante(proximoProcesso)-paraContador
    ;
270             if compRestante(proximoProcesso)<0
271                 compRestante(proximoProcesso)=0;
272                 tempo=tempo+compRestante(proximoProcesso);
273                 paraContador=paraContador-compRestante(proximoProcesso);
274             end
275
276             compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;
277
278             %Plot de computacao normal
279             rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-
    proximoProcesso),paraContador,10],'FaceColor',[0.6 0.6 0.9],'LineStyle','None')
    ;
280             drawnow
281
282         %Esta em zona critica
283         else
284             %Verifica qual e a zona critica em que esta. Usa o fato
285             %da zona critica ativa com tempo de comeco maior ser a ativa
286             zonaAtiva=[0 0];
287             for i=1:tam(3)
288                 %zonaAtiva(1) guarda a zona ativa ao fim do loop
289                 if S(proximoProcesso,3,i)==1 & S(proximoProcesso,1,i)>=zonaAtiva(2)
290                     zonaAtiva=[i S(proximoProcesso,1,i)];
291                 end
292             end
293
294             %Computacao e feita na zona critica e no tempo total da tarefa
295             S(proximoProcesso,2,zonaAtiva(1))=S(proximoProcesso,2,zonaAtiva(1))-
    paraContador;
296             compFeita(proximoProcesso)=compFeita(proximoProcesso)+paraContador;
297
298             if S(proximoProcesso,2,zonaAtiva(1))<0
299                 S(proximoProcesso,2,zonaAtiva(1))=0;
300                 tempo=tempo+S(proximoProcesso,2,zonaAtiva(1));
301                 paraContador=paraContador-S(proximoProcesso,2,zonaAtiva(1));
302             end
303
304             %Plot da zona critica
305             rectangle('Position',[tempo-paraContador,20*(length(chegada)+1-
    proximoProcesso),paraContador,10],'FaceColor',cores(zonaAtiva(1),:),'LineStyle'
    ,'None');
306             xlim([0,largura])
307             ylim([0,20*(1+length(chegada))])
308             set(gca,'YTick',[0:20:20*length(chegada)])
309             drawnow
310
311             %Caso tenha acabado a zona critica
312             if S(proximoProcesso,2,zonaAtiva(1))<=0
313                 %Marca fim da zona
314                 S(proximoProcesso,3,zonaAtiva(1))=2;
315
316                 %Reseta a prioridade bloqueada
317                 prioridadeBloqueada(zonaAtiva(1))=101;
318
319                 %Retorna prioridade da tarefa atual a anterior (maior que
320                 %bloqueia entre com seus sinais restantes)
321                 prioridadeAtiva(proximoProcesso)=proximoProcesso;
322                 for i=1:tam(3)
323                 %zonaAtiva(1) guarda a zona ativa ao fim do loop
324                     if S(proximoProcesso,3,i)==1
325                         prioridadeAtiva(proximoProcesso)=min(prioridadeAtiva(
    proximoProcesso),prioridadeBloqueada(i));
326                     end
327                 end
328
329                 %Reseta e checa maior Ceiling (menor prioridade) entre semaforos
    ativos
330                 SmaiorCeiling=length(C);
331
332                 for j=1:tam(3)
333                     if any(S(:,3,j)==1) & C(SmaiorCeiling)>C(j)
```

```
                        SmaiorCeiling=j;
                    end
                end

            end


        end
    end

    %Checa se processo atual terminou
    if compRestante(proximoProcesso)<=0

        %Marca que o processo terminou, alterando prioridade
        prioridadeAtiva(proximoProcesso)=105;

        %Verifica se todos processo ja terminaram, para entao
        % terminar de processar
        if max(compRestante)<=0
            break;
        end
    end
end


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes on mouse press over axes background.
function axes1_ButtonDownFcn(hObject, eventdata, handles,i)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


```

```
% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
w = handles.ns + 1;
initial_temp = cellstr(get(handles.listbox1,'String'))
a = get(handles.edit6, 'String');
temp = [initial_temp; a];
set(handles.listbox1, 'String', temp);
b =  str2num(a);
handles.ns = w;
handles.Soriginal(:,:,w) = str2num(get(handles.edit6, 'String'));
guidata(hObject, handles);


% --- Executes on selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox1 contents as cell
    array
%        contents{get(hObject,'Value')} returns selected item from listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
```

```
525         set(hObject,'BackgroundColor','white');
526 end
527
528
529 % --- Executes on button press in pushbutton3.
530 function pushbutton3_Callback(hObject, eventdata, handles)
531 % hObject    handle to pushbutton3 (see GCBO)
532 % eventdata  reserved - to be defined in a future version of MATLAB
533 % handles    structure with handles and user data (see GUIDATA)
534 handles = rmfield(handles, 'Soriginal');
535 handles.Soriginal = [];
536 set(handles.listbox1, 'String', '');
537 cla
538 guidata(hObject, handles);
```