
	<p align="center">&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	
--	--	--

### Resolução da lista de exercícios

Bruno Rodrigues Caputo (brunorcx@hotmail.com)

DCC403-Sistemas Operacionais 2019.1-Turma 01

Universidade Federal de Roraima

DCC -Departamento de Ciência da Computação - Bloco V

Campus Universitário do Paricarana - Aeroporto

69310-000 Boa vista, RR

**ATENÇÃO:** Descrever as soluções com o máximo de detalhes possível, no caso de programas, inclusive a forma como os testes foram feitos. Todos os artefatos (relatório, código fonte de programas, e outros) gerados para este trabalho devem ser adicionados em um repositório no site [github.com](https://github.com), com o seguinte formato nome\_labos\_rr\_2019. Para as questões que requisitarem a escrita/implementação de programas deve ser apresentado: o modo de compilar/executar o programa; a linha de comando para executar o programa; e um exemplo de entrada/saída do programa.

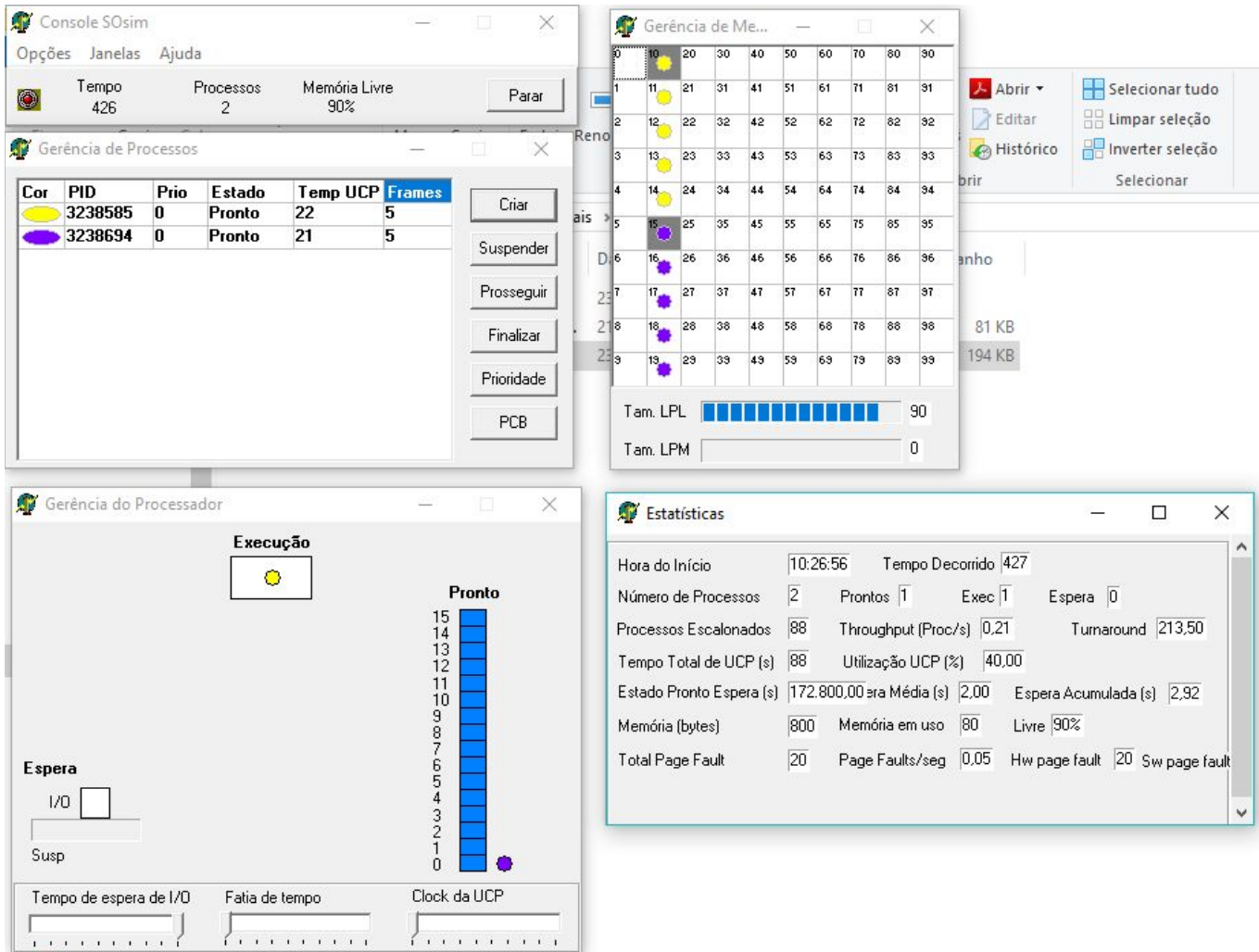
[Questão-1] Utilizando o simulador SOSim (disponível em <http://www.training.com.br/sosim>) apresente os resultados do simulador e uma análise para cada item abaixo.

#### (PRÁTICA - A)

##### Simulação:

- Reinicialize o simulador.
- Ative a janela de Estatísticas em Console SOSim / Janelas / Estatísticas.
- Crie dois novos processos: janela Gerência de Processos / Criar – janela Criação de Processos / Criar.

**Questão teórica para responder com a ajuda do simulador.** Observe que em alguns momentos existem processos no estado de pronto porém nenhum em estado de execução. Explique o porquê dessa situação.



**R-** Criamos dois processos que serão executados pela CPU com a mesma prioridade, O processo com PID 3238585 e o processo com PID 3238694. Em alguns momentos pode ocorrer de dois processos terminarem a sua execução e entrarem no estado de pronto, a CPU irá então, escolher qual processador executar. Devemos considerar também que a CPU deve aguardar um processo finalizar para então executar o próximo, o que pode ocasionar momentos em que ambos estejam prontos.

## (PRÁTICA - B) Simulação:

• Execute o simulador SOsim e configure-o para trabalhar com Escalonamento Circular com Prioridades Estáticas: janela Console SOsim / Opções / Parâmetros do Sistema na guia Processador.

### Análise Prática:

• Crie um processo CPU-bound com prioridade 4 e um outro I/O-bound com prioridade 3: janela Gerência de Processos / Criar – janela Criação de Processos / Criar.

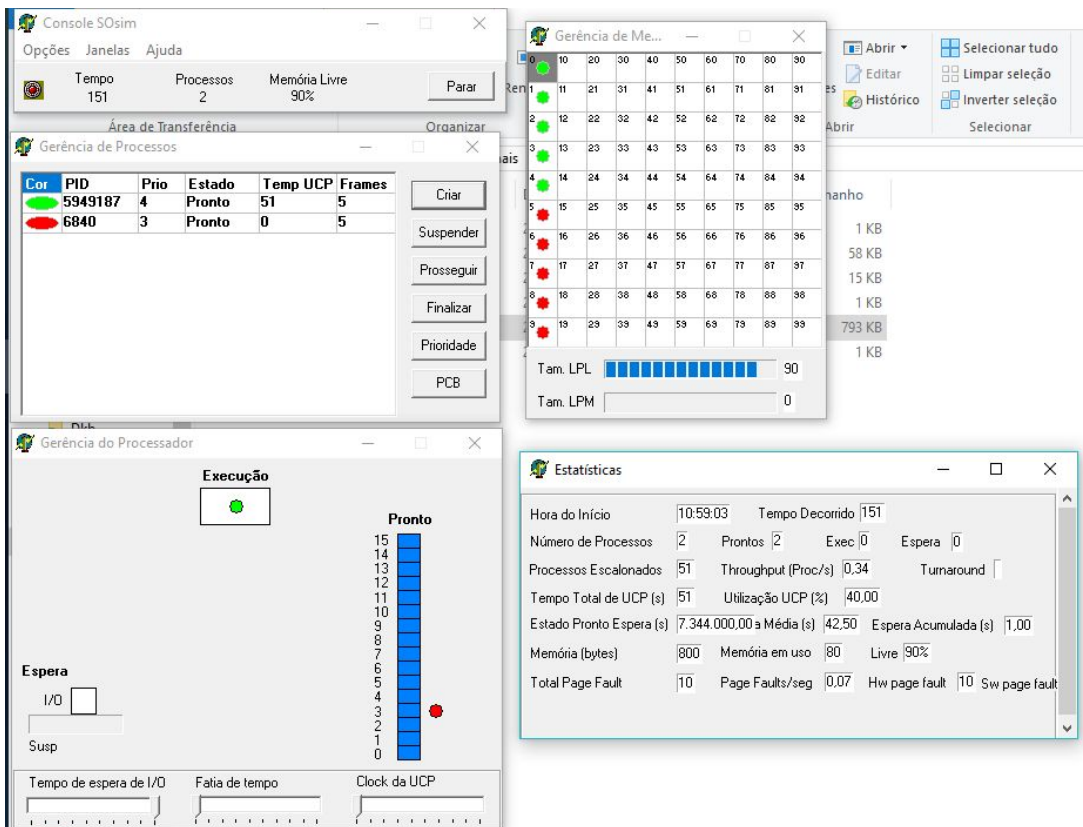
• Na janela Gerência de Processos, observe o escalonamento dos dois processos.



• Analise o problema do starvation.

Questões teóricas para responder com a ajuda do simulador

• Por que o problema do starvation pode ocorrer?

• Cite duas ações que o administrador do sistema pode realizar quando é identificada a situação de starvation em um processo?



	<p align="center">&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	
--	--	--

**R-** Como temos prioridades diferentes neste caso, e ainda o processo com maior prioridade é *CPU bound* entramos no problema de *Starvation*. O segundo processo fica aguardando informações de entrada e saída e não é escolhido para ser executado pela *CPU*. Como mostrado na imagem acima temos o escalonamento de 51 vezes do processo com PID 5949187 e 0 para o processo restante.

Para resolver este problema podemos estabelecer um tempo mínimo para que a prioridade deste processo aumente gradativamente enquanto este se encontra na lista de processos prontos.

### (PRÁTICA – C)

#### Simulação:

- Execute o simulador SOsim e configure-o para trabalhar com Escalonamento Circular: janela Console SOsim / Opções / Parâmetros do Sistema na guia Processador.
- Configure a política de busca de páginas sob demanda: janela Console SOsim / Opções / Parâmetros do Sistema na guia Memória.
- Re-inicie o simulador SOsim para que a nova parametrização passe a ser válida.

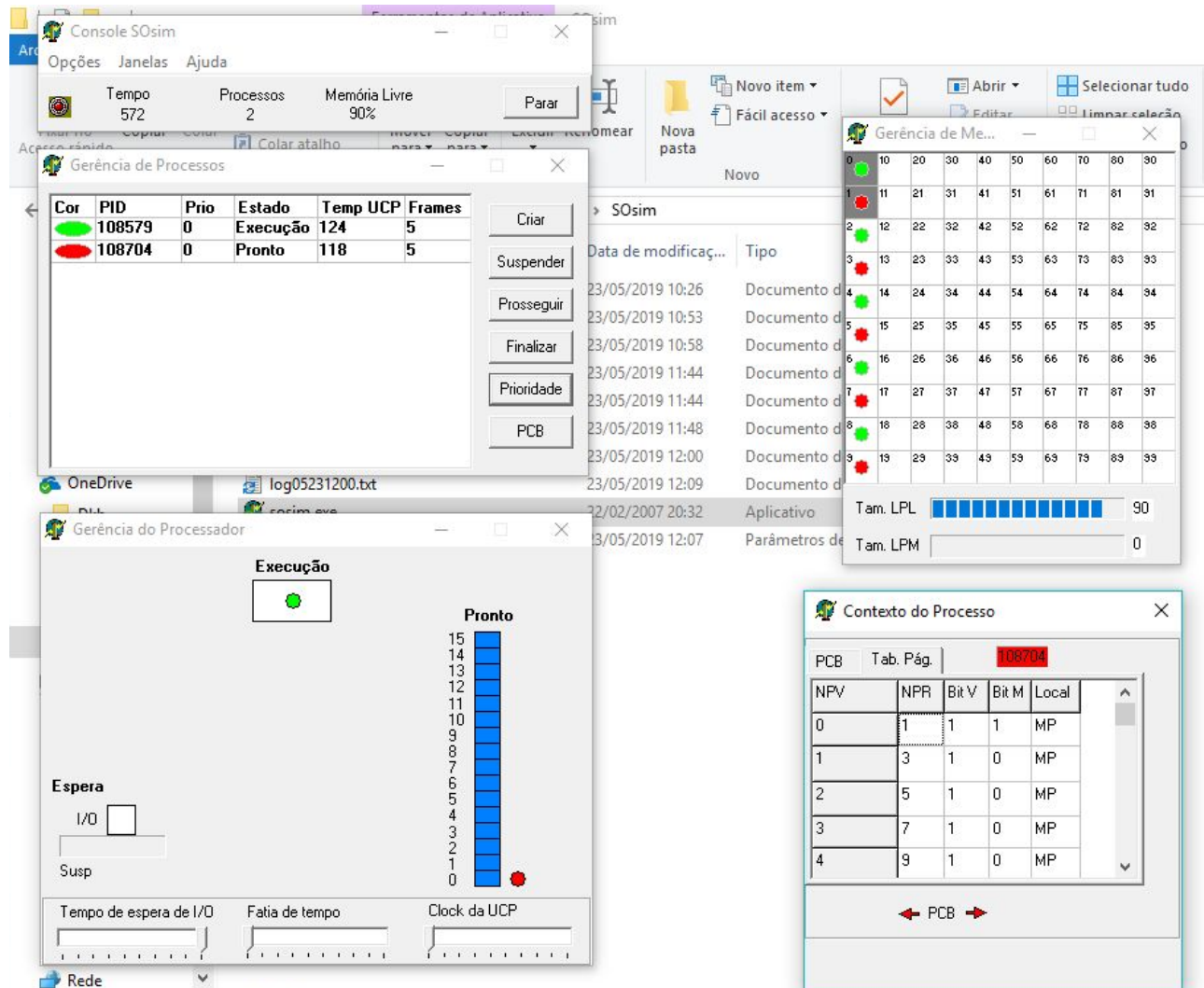
#### Análise Prática:

- Crie dois processos CPU-bound: janela Gerência de Processos / Criar – janela Criação de Processos / Criar.
- Ative a janela Contexto do Processo para visualizar a tabela de páginas do processo criado: Gerência de Processos / PCB na guia Tab. de Pag.
- Na janela Gerência de Memória observe a alocação dos frames na memória principal.
- Na janela Contexto do Processo observe as alterações nas tabelas de páginas dos dois processos navegando com as setas inferiores.

#### Questões teóricas para responder com a ajuda do simulador:

- Qual o espaço de endereçamento real máximo de um processo?
- Qual o espaço de endereçamento real mínimo de um processo?

- Qual o tamanho da página virtual?





The screenshot displays the SOsim operating system simulation interface. Key components include:

- Console SOsim:** Shows system status: Tempo 572, Processos 2, Memória Livre 90%.
- Gerência de Processos:** A table listing processes:
 

Cor	PID	Prio	Estado	Temp UCP	Frames
Verde	108579	0	Execução	124	5
Vermelho	108704	0	Pronto	118	5
- Gerência do Processador:** Shows a bar chart for process execution and a list of ready processes (Pronto) numbered 1 to 15.
- Gerência de Memória:** A grid showing memory allocation for 100 pages (0-99). It includes sliders for 'Tam. LPL' (set to 90) and 'Tam. LPM' (set to 0).
- Contexto do Processo:** A detailed view of process 108704, showing its PCB (Process Control Block) with fields: NPV, NPR, Bit V, Bit M, and Local. The table below shows the mapping of virtual pages to physical memory frames:
 

NPV	NPR	Bit V	Bit M	Local
0	1	1	1	MP
1	3	1	0	MP
2	5	1	0	MP
3	7	1	0	MP
4	9	1	0	MP

**R-** Cada processo pode alocar no máximo cinco páginas da memória principal, representado pelos frames dos dois processos, já quanto ao mínimo temos que cada processo pode alocar 1 frame(página) da memória principal e por fim o tamanho da página possui oito endereços, a memória como um todo contém 100 páginas.

	<p align="center">&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	
--	--	--

## (PRÁTICA – D)

### Simulação:

- Execute o simulador SOsim e configure-o para trabalhar com Escalonamento Circular: janela Console SOsim / Opções / Parâmetros do Sistema na guia Processador.
- Configure a política de busca de páginas sob demanda: janela Console SOsim / Opções / Parâmetros do Sistema na guia Memória.
- Configure a memória livre para possuir sempre 20% de frames livres: janela Console SOsim / Opções / Parâmetros do Sistema na guia Memória.
- Re-inicie o simulador SOsim para que a nova parametrização passe a ser válida.

### Análise Prática:

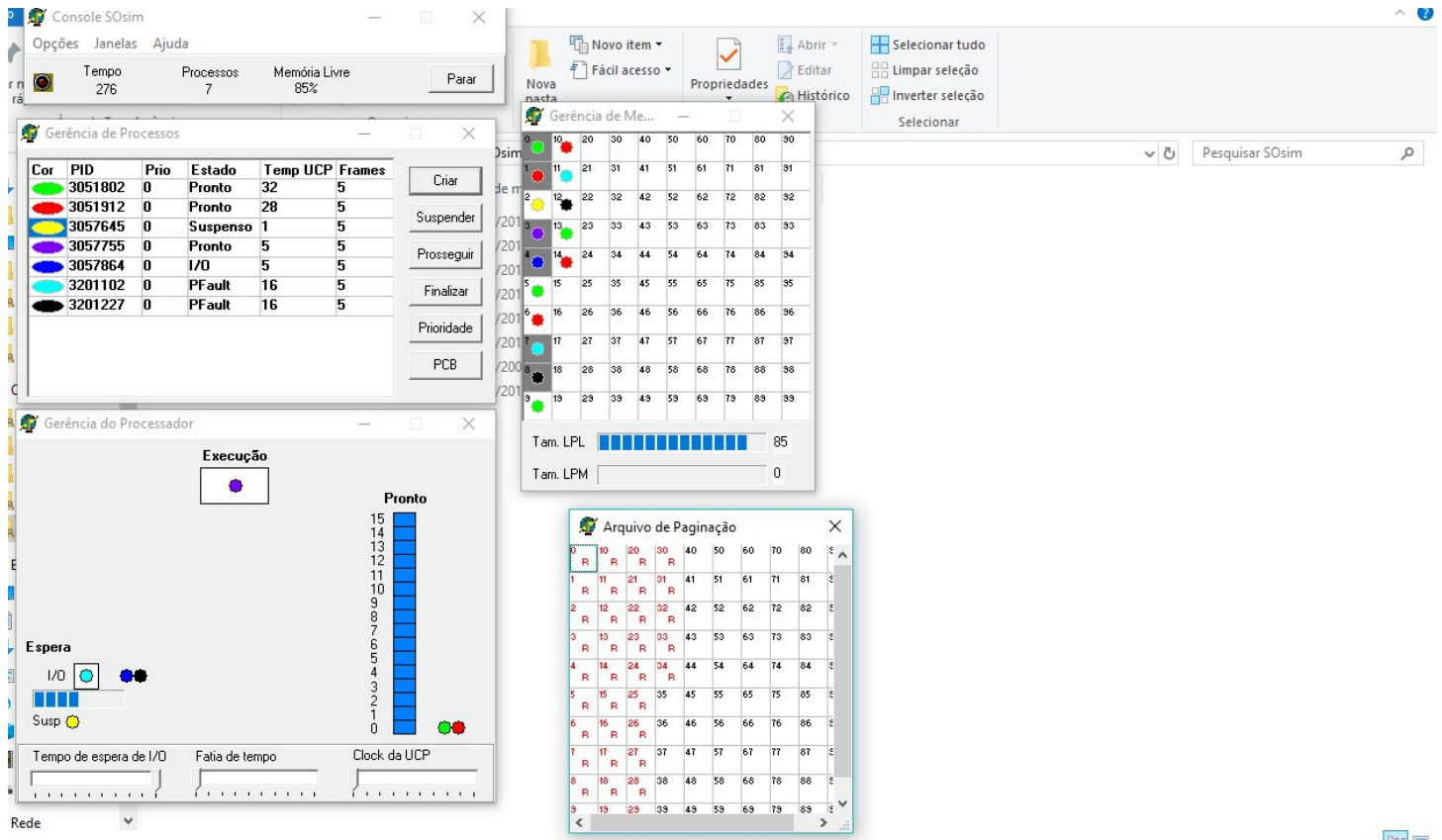
- Criar dois processos CPU-bound e três I/O-bound com limite de cinco frames para cada processo: janela Gerência de Processos / Criar.
- Suspenda um dos processos I/O-bound: janela Gerência de Processos / Suspend.
- Ative a janela Arquivo de Paginação para visualizar o arquivo de paginação do sistema: Console SOsim / Janelas / Arquivo de Paginação
- Crie mais dois processos CPU-bound: janela Gerência de Processos / Criar.
- Observe os estados dos processos outswapped.

### Questão teórica para responder com a ajuda do simulador:



- Quais os critérios utilizados pelo simulador para selecionar o processo a ser transferido para o arquivo de paginação (swap out)?



- Quando o processo deve ser transferido novamente para a memória principal (swap in)?



R-Temos 7 processos, os arquivos de paginação tem como alocação de páginas fixas, assim temos 35 páginas, sendo cada processo tendo o limite máximo de 5 páginas. Um processo será transferido para memória principal quando ele se encontrar no topo da fila de espera, no caso o processo com PID 3201102, o limite máximo for alcançado e ele não conseguir ser executado pela CPU, enquanto outros processos são executados. Os processos CPU bound acabam sendo colocados na memória principal primeiramente, neste caso o tempo limite já foi excedido para os dois últimos processos que são CPU bound, eles então irão ser colocados na memória principal após aguardar na fila de entrada e saída.

	<p align="center">&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	
--	--	--

**[Questão-2] Com relação ao problema de Deadlock. Pesquisa e descreva o algoritmo do banqueiro (criado por Dijkstra) que pode ser utilizado para evitar impasses. Sempre que recursos são solicitados, o algoritmo avalia se atender à solicitação levará a um estado inseguro e se isso ocorrer, ela não é atendida. Adicionalmente, escreva o algoritmo do banqueiro em C/C++ e apresente alguns exemplos de sua execução.**



**R-** O algoritmo do banqueiro tenta evitar dinamicamente o *deadlock*, através de alocação de recursos conforme a necessidade, fazendo um escalonamento cuidadoso, contudo, teremos um alto custo para determinar e ter conhecimento de quais recursos deverão ser usados. É utilizado a noção de estados seguros e inseguros, sendo respectivamente, estados que não provocam *deadlocks* e possuem garantia de que os processos terminarão, e em segundo, estados que podem provocar *deadlocks*. Podemos exemplificar da seguinte maneira conforme a tabela abaixo:

Processos	Possui	Máximo
<b>A</b>	<b>3</b>	<b>9</b>
<b>B</b>	<b>2</b>	<b>4</b>
<b>C</b>	<b>2</b>	<b>7</b>

**Livre: 3**

Neste exemplo temos um estado seguro, como temos 3 recursos livres o processo B é um forte candidato a ser o próximo a receber os recursos livres, uma vez que precisa de 2 recursos para atingir a sua capacidade máxima, após finalizar poderá liberá-los. Assim, ficamos com 5 recursos livres, podemos alocá-los para o processo C agora atingindo o seu máximo e por último após a finalização de C executaremos o processo A.



 <p>UFRR</p>	<p>&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	 <p>Departamento de CIÊNCIA DA COMPUTAÇÃO</p>
--	---	---

Para compilar o programa basta digitar o seguinte comando:

```
gcc -pthread -o banqueiro sistema.c Banqueiro.c Processos.c Dados.c
```

Para executar o programa basta digitar o seguinte comando:



```
./banqueiro
```

Exemplo de Saída, neste exemplo 3 é número de processos, 9 4 7 os recursos disponíveis para os processos inicialmente.

```
./banqueiro -n 3 -a 9 4 7
Quantidade de recurso INICIALMENTE disponível:
9 4 7
Numero de Clientes:      3
Numero de recursos:      3
Quantidade total de recursos de cada processo:
P0: 4 2 5
P1: 8 2 2
P2: 4 3 5

=====
= Iniciando Simulação! =
=====

P0: Recurso alocado!
    Quantidade de recursos requeridos:
    3 1 1
    Quantidade de recursos disponivel no sistema:
    9 4 7
    Quantidade de recursos do sistema após a alocação:
    6 3 6
    Quantidade de recursos do processo após a alocação:
    3 1 1
    Quantidade de recursos necessarios para finalizar do
processo:
```

 <p>UFRR</p>	<p>&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	 <p>Departamento de CIÊNCIA DA COMPUTAÇÃO</p>
--	---	---

```

1 1 4
P2: Recurso Não alocado!
  Quantidade de recursos requeridos:
2 2 3
  Quantidade de recursos disponivel no sistema:
6 3 6
P2 Morreu!
P2: Recursos Liberados com sucesso
  Quantidade de recursos alocados P2:
0 0 0
  Quantidade de recursos para liberar:
0 0 0
  Quantidade de recursos no sistema após liberar:
6 3 6

=====
= Threads Ativas: 2 =
=====
P1: Recurso alocado!
  Quantidade de recursos requeridos:
0 1 0
  Quantidade de recursos disponivel no sistema:
6 3 6
  Quantidade de recursos do sistema após a alocação:
6 2 6
  Quantidade de recursos do processo após a alocação:
0 1 0
  Quantidade de recursos necessarios para finalizar do
processo:
8 1 2
P0: Recursos Liberados com sucesso
  Quantidade de recursos alocados P0:
3 1 1
  Quantidade de recursos para liberar:
1 0 0



```



<Laboratório S.O>  
por <Bruno Rodrigues Caputo > - 2019.1  
Boa vista, 23/05/2019





```
Quantidade de recursos no sistema após liberar:
7 2 6
P1: Recursos Liberados com sucesso
Quantidade de recursos alocados P1:
0 1 0
Quantidade de recursos para liberar:
0 0 0
Quantidade de recursos no sistema após liberar:
7 2 6
P0: Recurso alocado!
Quantidade de recursos requeridos:
0 1 0
Quantidade de recursos disponivel no sistema:
7 2 6
Quantidade de recursos do sistema após a alocação:
7 1 6
Quantidade de recursos do processo após a alocação:
2 2 1
Quantidade de recursos necessarios para finalizar do
processo:
2 0 4
P1: Recurso alocado!
Quantidade de recursos requeridos:
4 1 1
Quantidade de recursos disponivel no sistema:
7 1 6
Quantidade de recursos do sistema após a alocação:
3 0 5
Quantidade de recursos do processo após a alocação:
4 2 1
Quantidade de recursos necessarios para finalizar do
processo:
4 0 1
P1: Recursos Liberados com sucesso
```

 <p>UFRR</p>	<p>&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	 <p>Departamento de CIÊNCIA DA COMPUTAÇÃO</p>
--	---	---

```

Quantidade de recursos alocados P1:
4 2 1
Quantidade de recursos para liberar:
3 1 0
Quantidade de recursos no sistema após liberar:
6 1 5
P0: Recursos Liberados com sucesso
Quantidade de recursos alocados P0:
2 2 1
Quantidade de recursos para liberar:
0 1 0
Quantidade de recursos no sistema após liberar:
6 2 5
P0: Recurso alocado!
Quantidade de recursos requeridos:
0 1 0
Quantidade de recursos disponivel no sistema:
6 2 5
Quantidade de recursos do sistema após a alocação:
6 1 5
Quantidade de recursos do processo após a alocação:
2 2 1
Quantidade de recursos necessarios para finalizar do
processo:
2 0 4
P1: Recurso Não alocado!
Quantidade de recursos requeridos:
5 1 1
Quantidade de recursos disponivel no sistema:
6 1 5
P1 Morreu!
P1: Recursos Liberados com sucesso
Quantidade de recursos alocados P1:
1 1 1

```

 <p>UFRR</p>	<p>&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	 <p>Departamento de CIÊNCIA DA COMPUTAÇÃO</p>
--	---	---

```

Quantidade de recursos para liberar:
1 1 1
Quantidade de recursos no sistema após liberar:
7 2 6

=====
= Threads Ativas: 1 =
=====
Finalizando processo...
P0: Recursos Liberados com sucesso
    Quantidade de recursos alocados P0:
    2 2 1
    Quantidade de recursos para liberar:
    2 2 1
    Quantidade de recursos no sistema após liberar:
    9 4 7

```



**[Questão-3] Com relação a problemas clássicos de comunicação entre processos. Escreva o algoritmo do barbeiro (visto em sala de aula), usando threads, em C/C++ e apresente alguns exemplos de sua execução.**

Para compilar o programa basta digitar o seguinte comando:

```
gcc -pthread -o barbeiro barbeiro.c
```

Para executar o programa basta digitar o seguinte comando:



```
./barbeiro
```

 <p>UFRR</p>	<p>&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	 <p>Departamento de CIÊNCIA DA COMPUTAÇÃO</p>
--	---	---

Exemplo de saída: Neste exemplo temos o tempo para cortar o cabelo como 4 segundos; número de clientes como 5; número de cadeiras disponíveis como 3 e por último número de tempo de espera para que cada cliente tente um novo corte de cabelo. O programa começa pegando pid de duas threads, criando threads para as cadeiras, clientes e barbeiro. O barbeiro deve esperar o cliente chegar e por isso é posto em espera com `sem_wait()`, no caso tanto dos clientes como das cadeiras; Ele também fica responsável por liberar uma cadeira após terminar o serviço, por isso, o número de cadeiras deve ser incrementado. Na parte do cliente ele primeiramente irá verificar se o salão está cheio( todas as cadeiras estão ocupadas), se estiver ele então sai, caso contrário ele deve ocupar uma cadeira e esperar o barbeiro cortar o cabelo, por isso decrementar o contador do número de cadeiras, por fim ele receberá o corte enquanto espera ou pode tentar novamente esperando.

```
./barbeiro 4 5 3 2
Main thread beginning
Creating barber thread with id 139690064238336
Creating client thread with id 139690055845632
Client: Thread 539510528 Sitting to wait. Number of chairs left = 2
Barber: Taking a client. Number of chairs available = 3
Creating client thread with id 139690047452928
Barber: Cutting hair for 4 seconds
Client: Thread 539510528 getting a haircut
Client: Thread 539510528 waiting 1 seconds before attempting next haircut
Creating client thread with id 139690039060224
Client: Thread 522725120 Sitting to wait. Number of chairs left = 2
Creating client thread with id 139690030667520
Client: Thread 514332416 Sitting to wait. Number of chairs left = 1
Creating client thread with id 139690022274816
Main thread sleeping for 4 seconds
Client: Thread 505939712 Sitting to wait. Number of chairs left = 0
Client: Thread 531117824 leaving with no haircut
Client: Thread 531117824 waiting 2 seconds before attempting next haircut
Client: Thread 539510528 leaving with no haircut
Client: Thread 539510528 waiting 2 seconds before attempting next haircut
Client: Thread 531117824 leaving with no haircut
Client: Thread 531117824 waiting 2 seconds before attempting next haircut
Client: Thread 539510528 leaving with no haircut
Client: Thread 539510528 waiting 2 seconds before attempting next haircut
Barber: Taking a client. Number of chairs available = 1
Barber: Cutting hair for 3 seconds
```



 <p>UFRR</p>	<p>&lt;Laboratório S.O&gt; por &lt;Bruno Rodrigues Caputo &gt; - 2019.1 Boa vista, 23/05/2019</p>	 <p>Departamento de CIÊNCIA DA COMPUTAÇÃO</p>
--	---	---

```
Client: Thread 522725120 getting a haircut
Client: Thread 522725120 waiting 1 seconds before attempting next haircut
Main thread exiting
```

#### Bibliografia:

Código do algoritmo do banqueiro retirado do sítio

”<https://github.com/matheussn/TrabalhoSO>”

Código do algoritmo do barbeiro retirado do sítio

“<https://github.com/tonymartinez/Sleeping-Barber/blob/master/mybarber.c>”

[Explicação e pesquisa sobre tratamento de \*deadlocks\*, baseadas na aula do professor Jó Ueyama da Univesp\(Universidade Virtual do Estado de São Paulo\).](#)