Victor Barbosa Rocha

**Prática A:**

**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.**

Porque esse é o tempo em que o próximo processo na fila de execução do escalonamento circular vai checar a disponibilidade de recursos para entrar no modo de execução.

**Prática B:**

**Questões teóricas para responder com a ajuda do simulador**

**• Por que o problema do starvation pode ocorrer?**

Porque na fila de prioridades, na medida que o processo é executado e retornar ao estado pronto de volta para fila, ele já retorna como o próximo da fila, isso acontecendo em sequência impossibilita com que o próximo processo tenha chance de usar a CPU.

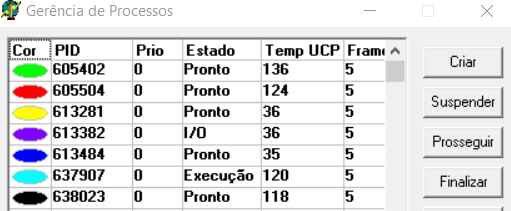
**• Cite duas ações que o administrador do sistema pode realizar quando é identificada a**

**situação de starvation em um processo?**

Abandonar a fila de prioridades por uma fila de tempo de espera.

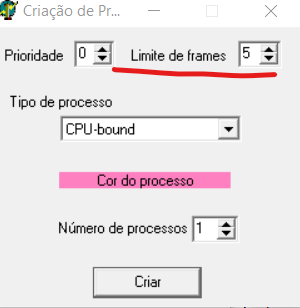
Utilizar semáforos para sincronizar processos para entrarem em repouso a medida que utilizam os recursos e despertar os processos para que estes também utilizem a CPU.

**Prática C:**



**Qual o espaço de endereçamento real máximo de um processo?**

5 frames.



**Qual o espaço de endereçamento real mínimo de um processo?**

1 frame. Visto que na prática com somente um espaço na memória, o processo já sinaliza estar em execução.

**Qual o tamanho da página virtual**?

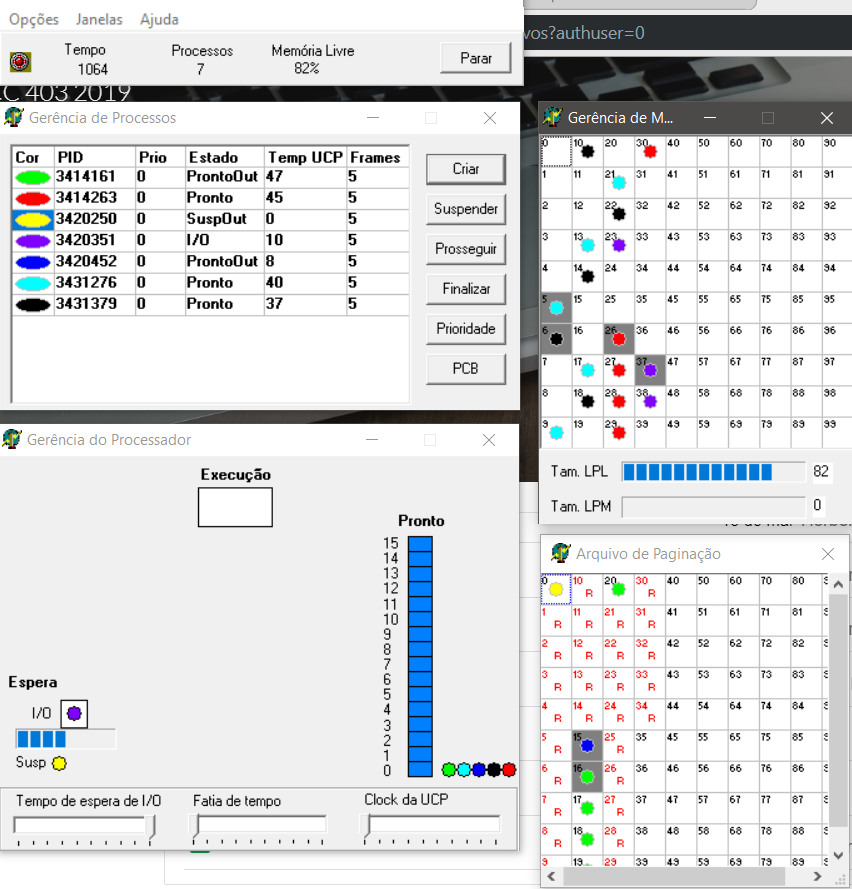
5 frames.

**Prática D:**

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

[Sobre a simulação: Seguindo os passos do laboratório, deixando o mínimo possível de frames disponível em 20% e criando 4 processos CPU bound e 3 I/O bound, teria espaço suficiente para alocar tudo sem necessidade de swapping. Visto que temos 100 frames livres para alocar 7 processos que no máximo iriam ocupar 5 frames cada, 35 no total e 80 disponíveis. Fiz uma alteração para observar o swapping. Coloquei 7 processos CPU bound e 3 I/O bound e diminuí o número mínimo de memória livre para 80 frames, ou seja, poderia utilizar 20 frames para alocar todos os processos, esses que iriam ocupar 50 frames no máximo. Agora foi possível observar swapping.]

As páginas dos processos na memória principal aparentam realizar swap out em conjunto, o que pode mostrar que elas trabalham utilizando um Working Set de frames. Quando elas voltam para a memória, também voltam em conjunto. No manual do simulador é dito que a política de substituição de páginas é FIFO, ou seja, as páginas substituídas devem levar em consideração o seu tempo de chegada na memória principal. Por elas trabalharem com uma política de substituição de páginas local, a escolha por quais processos serão substituídos é feita entre as páginas de cada processo. Em outras palavras, é FIFO entre páginas de um mesmo processo.



Outras observações:

O processo suspenso é o primeiro a sofrer swap out.

Os processos geralmente sofrem swap out em conjunto, e swap in também.

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

Segue a fila do buffer. Processos que estão fora e são chamados na fila ficam no estado **ProntoOut,** a partir dali eles são transferidos para a memória, executados e depois dão espaço para outros também ProntoOut.



**[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.**

O algoritmo do banqueiro visa evitar deadlocks simulando a alocação de recursos de antemão para ver se o quanto o processo vai alocar, ou seja, o quanto ele precisa é menor ou igual ao quanto há disponível.

Na implementação desse algoritmo temos:

* n -> indicando o número de processos.
* m -> indicando os tipos de recursos, exemplo, recursos = A B C, m = 3;
* disponibilidade[m] -> o quanto há de disponível de cada recurso.
* max[i, j] = k -> diz que o processo P[i] usa no MÁXIMO k do recurso R[j].
* alocado[i, j] = k -> diz que o processo P[i] está no momento com k alocado de recurso R[j]
* precisa[i, j] = k -> diz que o processo P[i] precisa de k do recurso do tipo R[j]. Precisa é achado por max - alocado.

O algoritmo vai ver se o precisa[i, j] <= disponibilidade. Se sim, é seguro continuar, atualizo meu disponibilidade somando com o alocado[i]. Se não, o processo não prossegue podem continuar depois quando tiver disponibilidade suficiente.

**[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.**

O algoritmo do barbeiro, ou problema do barbeiro dorminhoco, é uma analogia a comunicação entre processos onde temos uma barbearia hipotética, UM barbeiro, UMA cadeira onde ele atende os clientes (e onde dorme ocioso), e N cadeiras para os clientes esperarem cada ele esteja ocupado com outro cliente.

A sequência de eventos é a seguinte:

* Sem clientes, ele cochila na esperança de acordar quando um chegar.
* O cliente chega é acorda ele.
* Se outros chegarem e ele estiver ocupado, podem esperar nas N cadeiras na sala de espera.
* Se não tiver onde sentar, ele não tem como esperar, vão embora.

Nessa implementação vamos usar 3 semáforos:

1º um para o cliente contando quantos estão esperando na sala de espera;

2° um mutex para alternar o status de ocupado e desocupado do barbeiro;

3º outro para a sincronizarmos nossa área de exclusão mútua;