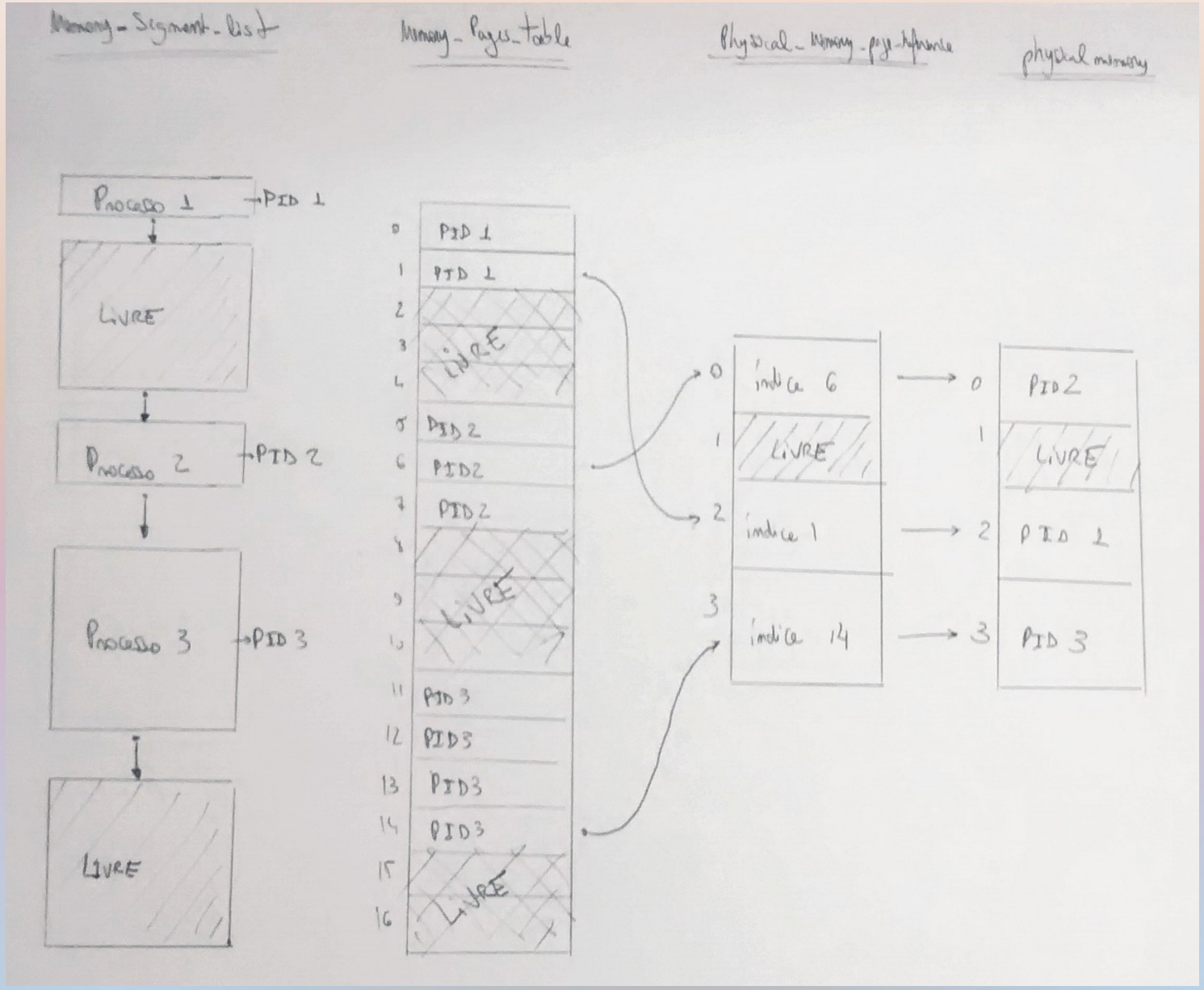
EP2 SO

Carlos Augusto Motta n° USP 7991228
Danilo Aleixo – n° USP 7972370

Estruturas de Dados



Estruturas de Dados

- **Memory_segment_list**: É uma lista encadeada de segmentos da memoria virtual, onde para cada elemento da lista ligada, temos um segmento que ou esta livre, ou esta com um processo no segmento. É a estrutura que os algoritmos de gerenciamento de espaço livre vão usar para encontrar o próximo segmento livre. Cada célula possui as seguintes informações: posição da página inicial para este segmento; o tamanho do segmento; o PID do processo que esta representado no segmento; e a próxima célula.

- **Memory_pages_table:** É um array com todas as paginas que estarão na memoria virtual, então para cada segmento que existe um processo em memory_segment_list suas respectivas páginas estarão representadas nessa estrutura.
Cada célula deste array possui as seguintes informações: PID do processo representado nesta página; se esta página está referenciada dentro da memória física, e seu índice; e se foi usada recentemente.

- **Physical_memory_page_reference:** Para a memória física temos dois arrays, o physical_memory_page_reference tem para cada quadro de página na memória física o índice da página no array memory_pages_table.

- **Physical_memory:** enquanto que no physical_memory temos o PID do processo que esta sendo executado na posição de memória. Obs: se não tiver nenhum processo então recebe “-1” que é representado por 255, em binário 1111 1111.

Manipulando o tempo

- Para manipular o tempo usamos uma hash chamada “time_events_list”
- Esse hash tem como chave o t0 de cada evento que irá ocorrer no sistema e como valor tem um objeto chamado TimeEvent, que tem como informações: to → tempo inicial da entrada do evento; mode → qual será o evento que irá ocorrer;
process_name → nome do processo;
number_of_bytes → numero de bytes do processo;
PID → pid do processo; memory_position → posição de memória acessada.
- Temos três tipos de evento:
  - Adicionar Processo
  - Remover Processo
  - Acessar Memória
- E para cada tipo de evento temos uma interação diferente
- Assim podemos ordenar o hash (time_events_list) pelas chaves, e portanto para cada segundo que se passa no sistema, vamos supor que estamos no tempo Tn, podemos acessamos a hash com a chave Tn e simplesmente executar os eventos.
- Para algoritmos que precisamos saber se a página na memória esta sendo usada, basta que voltemos no passado, por exemplo acessar a chave (Tn – 3) e atualizar as páginas dizendo se foram ou não acessadas.

Manipulando o tempo

- Para manipular o tempo usamos uma hash chamada “time_events_list”
- Esse hash tem como chave o t0 de cada evento que irá ocorrer no sistema e como valor tem um objeto chamado TimeEvent, que tem como informações: to → tempo inicial da entrada do evento; mode → qual será o evento que irá ocorrer;
process_name → nome do processo;
number_of_bytes → numero de bytes do processo;
PID → pid do processo; memory_position → posição de memória acessada.
- Temos três tipos de evento:
  - Adicionar Processo
  - Remover Processo
  - Acessar Memória
- E para cada tipo de evento temos uma interação diferente
- Assim podemos ordenar o hash (time_events_list) pelas chaves, e portanto para cada segundo que se passa no sistema, vamos supor que estamos no tempo Tn, podemos acessamos a hash com a chave Tn e simplesmente executar os eventos.
- Para algoritmos que precisamos saber se a página na memória esta sendo usada, basta que voltemos no passado, por exemplo acessar a chave (Tn – 3) e atualizar as páginas dizendo se foram ou não acessadas.

Algoritmos de gerência de espaço livre

Vamos usar a seguinte entrada para testar os algoritmos de gerência de espaço livre:

```
160 320
1 proc1   3   64
2 proc2   5   80
3 proc3  10  96
4 proc6   7   48
5 proc4  10  16
6 proc5   8   32
7 proc5  10  32
```

First Fit

O algoritmo First Fit implentamos de uma maneira que percorremos a lista encadeada de segmentos de memória, partindo do começo da mesma e escolhendo a primeira posição livre.
E a saída da entrada que estamos usando é a seguinte: (A lista ligada tem como informações da célula: posição inicial do segmento, posição final e PID do processo no segmento)

SAÍDA:
[0, 19, livre] -> nil

#<TimeEvent:0x00000001fbb8b8 @t0=1, @mode=:add_process, @process_name="proc1", @number_of_bytes=64, @pid=1>
[0, 3, 1] -> [4, 19, livre] -> nil

#<TimeEvent:0x00000001fbb7c8 @t0=2, @mode=:add_process, @process_name="proc2", @number_of_bytes=80, @pid=2>
[0, 3, 1] -> [4, 8, 2] -> [9, 19, livre] -> nil

#<TimeEvent:0x00000001fbb840 @t0=3, @mode=:remove_process, @pid=1>
[0, 3, livre] -> [4, 8, 2] -> [9, 19, livre] -> nil

#<TimeEvent:0x00000001fbb700 @t0=3, @mode=:add_process, @process_name="proc3", @number_of_bytes=96, @pid=3>
[0, 3, livre] -> [4, 8, 2] -> [9, 14, 3] -> [15, 19, livre] -> nil

#<TimeEvent:0x00000001fbb610 @t0=4, @mode=:add_process, @process_name="proc6", @number_of_bytes=48, @pid=4>
[0, 2, 4] -> [3, 3, livre] -> [4, 8, 2] -> [9, 14, 3] -> [15, 19, livre] -> nil **PEGA O PRIMEIRO ESPAÇO QUE CABE O ELEMENTO NESTE CASO DE [0,2]**

#<TimeEvent:0x00000001fbb750 @t0=5, @mode=:remove_process, @pid=2>
[0, 2, 4] -> [3, 8, livre] -> [9, 14, 3] -> [15, 19, livre] -> nil

#<TimeEvent:0x00000001fbb548 @t0=5, @mode=:add_process, @process_name="proc4", @number_of_bytes=16, @pid=5>
[0, 2, 4] -> [3, 3, 5] -> [4, 8, livre] -> [9, 14, 3] -> [15, 19, livre] -> nil **PEGA O PRIMEIRO ESPAÇO QUE CABE O ELEMENTO NESTE CASO DE [3,3]**

First Fit

#<TimeEvent:0x00000001fbb480 @t0=6, @mode=:add_process, @process_name="proc5", @number_of_bytes=32, @pid=6>
[0, 2, 4] -> [3, 3, 5] -> [4, 5, 6] -> [6, 8, livre] -> [9, 14, 3] -> [15, 19, livre] -> nil **PEGA O PRIMEIRO ESPAÇO QUE CABE O ELEMENTO NESTE CASO DE [4,5]**

#<TimeEvent:0x00000001fbb598 @t0=7, @mode=:remove_process, @pid=4>
[0, 2, livre] -> [3, 3, 5] -> [4, 5, 6] -> [6, 8, livre] -> [9, 14, 3] -> [15, 19, livre] -> nil

#<TimeEvent:0x00000001fbb3b8 @t0=7, @mode=:add_process, @process_name="proc5", @number_of_bytes=32, @pid=7>
[0, 1, 7] -> [2, 2, livre] -> [3, 3, 5] -> [4, 5, 6] -> [6, 8, livre] -> [9, 14, 3] -> [15, 19, livre] -> nil **PEGA O PRIMEIRO ESPAÇO QUE CABE O ELEMENTO NESTE CASO DE [0,1]**

#<TimeEvent:0x00000001fbb408 @t0=8, @mode=:remove_process, @pid=6>
[0, 1, 7] -> [2, 2, livre] -> [3, 3, 5] -> [4, 8, livre] -> [9, 14, 3] -> [15, 19, livre] -> nil

#<TimeEvent:0x00000001fbb688 @t0=10, @mode=:remove_process, @pid=3>
[0, 1, 7] -> [2, 2, livre] -> [3, 3, 5] -> [4, 19, livre] -> nil

#<TimeEvent:0x00000001fbb4f8 @t0=10, @mode=:remove_process, @pid=5>
[0, 1, 7] -> [2, 19, livre] -> nil

#<TimeEvent:0x00000001fbb368 @t0=10, @mode=:remove_process, @pid=7>
[1, 19, livre] -> nil

FIM DA SAÍDA

Essa entrada que usamos prova que o algortimo First Fit foi implementado corretamente

Next Fit

O algoritmo Next Fit irá percorrer a lista encadeada de segmentos de memória e escolher a primeira posição livre, porém partindo sempre da última posição que um elemento foi adicionado.
E a saída da entrada que estamos usando é a seguinte: (A lista ligada tem como informações da célula: posição inicial do segmento, posição final e PID do processo no segmento)

SAÍDA:
[0, 19, livre] -> nil

#<TimeEvent:0x00000000bef4d8 @t0=1, @mode=:add_process, @process_name="proc1", @number_of_bytes=64, @pid=1>
[0, 3, 1] -> [4, 19, livre] -> nil

#<TimeEvent:0x00000000bef3e8 @t0=2, @mode=:add_process, @process_name="proc2", @number_of_bytes=80, @pid=2>
[0, 3, 1] -> [4, 8, 2] -> [9, 19, livre] -> nil

#<TimeEvent:0x00000000bef460 @t0=3, @mode=:remove_process, @pid=1>
[0, 3, livre] -> [4, 8, 2] -> [9, 19, livre] -> nil

#<TimeEvent:0x00000000bef320 @t0=3, @mode=:add_process, @process_name="proc3", @number_of_bytes=96, @pid=3>
[0, 3, livre] -> [4, 8, 2] -> [9, 14, 3] -> [15, 19, livre] -> nil **APÓS ADICIONAR proc3 O NEXT FIT DEVE OLHAR PARA O PRÓXIMO ESPAÇO LIVRE À SUA FRENTE**

#<TimeEvent:0x00000000bef230 @t0=4, @mode=:add_process, @process_name="proc6", @number_of_bytes=48, @pid=4>
[0, 3, livre] -> [4, 8, 2] -> [9, 14, 3] -> [15, 17, 4] -> [18, 19, livre] -> nil **proc6 É ADICIONADO CORRETAMENTE, DEPOIS DA POSIÇÃO DE proc3**
#<TimeEvent:0x00000000bef370 @t0=5, @mode=:remove_process, @pid=2>
[0, 8, livre] -> [9, 14, 3] -> [15, 17, 4] -> [18, 19, livre] -> nil

#<TimeEvent:0x00000000bef168 @t0=5, @mode=:add_process, @process_name="proc4", @number_of_bytes=16, @pid=5>
[0, 8, livre] -> [9, 14, 3] -> [15, 17, 4] -> [18, 18, 5] -> [19, 19, livre] -> nil **O ÚLTIMO PROCESSO ADICIONADO QUE FOI ADICIONADO ESTAVA NA POSIÇÃO [15,17], PORTANTO A PRÓXIMA LIVRE É [18,19]**

Next Fit

#<TimeEvent:0x00000000bef0a0 @t0=6, @mode=:add_process, @process_name="proc5", @number_of_bytes=32, @pid=6>
[0, 1, 6] -> [2, 8, livre] -> [9, 14, 3] -> [15, 17, 4] -> [18, 18, 5] -> [19, 19, livre] -> nil **NOVAMENTE ADICIONAMOS A PARTIR DA ÚLTIMA POSIÇÃO USADA**

#<TimeEvent:0x00000000bef1b8 @t0=7, @mode=:remove_process, @pid=4>
[0, 1, 6] -> [2, 8, livre] -> [9, 14, 3] -> [15, 17, livre] -> [18, 18, 5] -> [19, 19, livre] -> nil

#<TimeEvent:0x00000000beefd8 @t0=7, @mode=:add_process, @process_name="proc5", @number_of_bytes=32, @pid=7>
[0, 1, 6] -> [2, 3, 7] -> [4, 8, livre] -> [9, 14, 3] -> [15, 17, livre] -> [18, 18, 5] -> [19, 19, livre] -> nil **COMO NÃO HÁ MAIS ESPAÇO LIVRE NO FINAL, A PRÓXIMA POSIÇÃO ACESSÍVEL ESTÁ NO COMEÇO DA LISTA**

#<TimeEvent:0x00000000bef028 @t0=8, @mode=:remove_process, @pid=6>
[0, 1, livre] -> [2, 3, 7] -> [4, 8, livre] -> [9, 14, 3] -> [15, 17, livre] -> [18, 18, 5] -> [19, 19, livre] -> nil

#<TimeEvent:0x00000000bef2a8 @t0=10, @mode=:remove_process, @pid=3>
[0, 1, livre] -> [2, 3, 7] -> [4, 17, livre] -> [18, 18, 5] -> [19, 19, livre] -> nil

#<TimeEvent:0x00000000bef118 @t0=10, @mode=:remove_process, @pid=5>
[0, 1, livre] -> [2, 3, 7] -> [4, 19, livre] -> nil

#<TimeEvent:0x00000000beef88 @t0=10, @mode=:remove_process, @pid=7>
[0, 19, livre] -> nil

FIM DA SAÍDA

Essa entrada que usamos prova que o algortimo Next Fit foi implementado corretamente

Algoritmos de substituição de página

Vamos usar a seguinte entrada para testar os algoritmos de substituição de página:

```
48 320
1 process1 5 200  1 1  16 1  32 1  1 4  16 4
1 process2 6 40    1 4
```

Not Recently Used Page

O algoritmo Not Recently Used Page irá percorrer a memória e buscar por páginas que não Foram acessadas recentemente.
Obs: O primeiro vetor indica os PIDs na memória física e o segundo indica o índice da Página na tabela de páginas;

Início:
Estado da memória física:
[255, 255, 255]
[-1, -1, -1]

#<TimeEvent:0x000000010daa88 @t0=1, @mode=:add_process, @process_name="process1", @number_of_bytes=200, @pid=1>
Estado da memória física:
[255, 255, 255]
[-1, -1, -1]

#<TimeEvent:0x000000010da9c0 @t0=1, @mode=:memory_access, @pid=1, @memory_position=1>
"Vou usar o quadro 0 porque estava livre :)"
Estado da memória física:
[1, 255, 255]
[0, -1, -1]

#<TimeEvent:0x000000010da970 @t0=1, @mode=:memory_access, @pid=1, @memory_position=16>
"Vou usar o quadro 1 porque estava livre :)"
Estado da memória física:
[1, 1, 255]
[0, 1, -1]

#<TimeEvent:0x000000010da920 @t0=1, @mode=:memory_access, @pid=1, @memory_position=32>
"Vou usar o quadro 2 porque estava livre :)"
Estado da memória física:
[1, 1, 1]
[0, 1, 2]

AGORA A MEMÓRIA FÍSICA ESTÁ CHEIA

Not Recently Used Page

#<TimeEvent:0x000000010da808 @t0=1, @mode=:add_process, @process_name="process2", @number_of_bytes=40, @pid=2>
Estado da memória física:
[1, 1, 1]
[0, 1, 2]
← AOS 3s TODAS AS PÁGINAS TEM SEU BIT R RESETADO
#<TimeEvent:0x000000010da8a8 @t0=4, @mode=:memory_access, @pid=1, @memory_position=1>
Estado da memória física:
[1, 1, 1]
[0, 1, 2] **← ACESSO À PÁGINA COM ÍNDICE 0 E CONSEQUENTEMENTE BIT R SETADO AGORA**

#<TimeEvent:0x000000010da858 @t0=4, @mode=:memory_access, @pid=1, @memory_position=16>
Estado da memória física:
[1, 1, 1]
[0, 1, 2] **← ACESSO À PÁGINA COM ÍNDICE 1 E CONSEQUENTEMENTE BIT R SETADO AGORA**

#<TimeEvent:0x000000010da740 @t0=4, @mode=:memory_access, @pid=2, @memory_position=1>
Estado da memória física:
[1, 1, 2]
[0, 1, 13] **COMO O ÍNDICE 0 E 1 DA TABELA DE PÁGINAS ESTÃO COM BIT R SETADOS, A PRÓXIMA POSIÇÃO DE MEMÓRIA DISPONÍVEL É A DE ÍNDICE 2**
#<TimeEvent:0x000000010daa10 @t0=5, @mode=:remove_process, @pid=1>
...
#<TimeEvent:0x000000010da790 @t0=6, @mode=:remove_process, @pid=2>
Estado da memória física:
[1, 1, 2]
[0, 1, 13]

FIM

Essa entrada que usamos prova que o algortimo NRUP foi implementado corretamente

First In First Out

O algoritmo First In First Out irá substituir as páginas de acordo com sua ordem de entrada Na memória física, então a primeira que entrou é a primeira a sair.
Obs: O primeiro vetor indica os PIDs na memória física e o segundo indica o índice da Página na tabela de páginas;

Início:
Estado da memória física:
[255, 255, 255]
[-1, -1, -1]

#<TimeEvent:0x00000001a33008 @t0=1, @mode=:add_process, @process_name="process1", @number_of_bytes=200, @pid=1>
Estado da memória física:
[255, 255, 255]
[-1, -1, -1]

#<TimeEvent:0x00000001a32f40 @t0=1, @mode=:memory_access, @pid=1, @memory_position=1>
Estado da memória física:
[1, 255, 255]
[0, -1, -1] **PRIMEIRO A SER INSERIDO NA FILA, INDICE PAG 0**

#<TimeEvent:0x00000001a32ef0 @t0=1, @mode=:memory_access, @pid=1, @memory_position=16>
Estado da memória física:
[1, 1, 255]
[0, 1, -1] **INSERE INDICE PAG 1 NA FILA**

#<TimeEvent:0x00000001a32ea0 @t0=1, @mode=:memory_access, @pid=1, @memory_position=32>
Estado da memória física:
[1, 1, 1]
[0, 1, 2] **AGORA A MEMÓRIA FÍSICA ESTÁ CHEIA E INSERE INDICE PAG 2 NA FILA**

First In First Out

#<TimeEvent:0x0000000144bd20 @t0=1, @mode=:add_process, @process_name="process2", @number_of_bytes=40, @pid=2>
[1, 1, 1]
[0, 1, 2]

#<TimeEvent:0x000000014524e0 @t0=4, @mode=:memory_access, @pid=1, @memory_position=1>
[1, 1, 1]
[0, 1, 2]

#<TimeEvent:0x0000000144bd70 @t0=4, @mode=:memory_access, @pid=1, @memory_position=16>
[1, 1, 1]
[0, 1, 2]

#<TimeEvent:0x0000000144bc58 @t0=4, @mode=:memory_access, @pid=2, @memory_position=1>
[2, 1, 1]
[13, 1, 2] **RETIRA O PRIMEIRO QUE FOI COLOCADO NA FILA (INDICE PAG 0) E COLOCA O NOVO INDICE PAG 13**

#<TimeEvent:0x00000001451298 @t0=5, @mode=:remove_process, @pid=1>
[2, 1, 1]
[13, 1, 2]

#<TimeEvent:0x0000000144bca8 @t0=6, @mode=:remove_process, @pid=2>
[2, 1, 1]
[13, 1, 2]

FIM

Essa entrada que usamos prova que o algortimo FIFO foi implementado corretamente