T4 - Gerência de E/S Deadlocks

Vinicius Gasparini

SOP - BCC - UDESC - 2020.2

Exercício 1

A fila de requisições geradas é dada pelos valores da Figura 1.

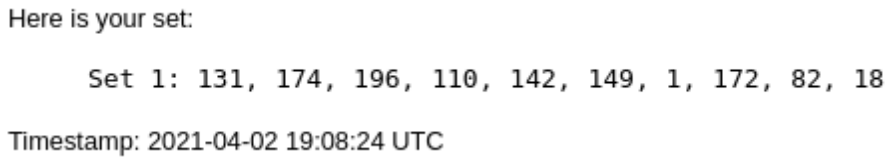


Figura 1: Conjunto de valores sorteados.

Considerando que

- O disco possui 200 cilindros
- O tempo de deslocamento entre trilhas adjacentes é de 0.8 ms
- A velocidade de rotação é de 6000 rmp
- O disco possui setores de 512 bytes
- O disco possui 128 setores por trilha
- Em cada requisição são lidos 8 KB de setores consecutivos
  - 8 KB = 8192 bytes $\rightarrow \frac{8192}{512} = 16$ setores

O tempo médio de acesso de cada requisição é dado pela expressão:

$$t_{acesso} = t_{seek} + t_{lat} + t_{transf}$$

Sendo,

$$t_{seek} = (\textit{distância entre cilindros}) \times 0.8\ ms$$

$$t_{lat} = \frac{1}{2r} = \frac{1}{2 \times \frac{6000}{60}} = 5\ ms$$

$$t_{transf} = \frac{16 \times 512}{\frac{6000}{60} \times (512 \times 128)} = 1.25\ ms$$
$$t_{acesso} = t_{seek} + 6.25\ ms$$

Algoritmo FCFS

Este algoritmo utiliza como critério de ordenação a ordem de chegada das requisições, isto é, pode ser simulado através de uma fila.

Sequência de execução: `100 131 174 196 110 142 149 1 172 82 18`

- **100** $\rightarrow$ 131: $dist(100, 131) \times 0.8 + 6.25 = 31.05\ ms$
- 131 $\rightarrow$ 174: $dist(131, 174) \times 0.8 + 6.25 = 40.65\ ms$
- 174 $\rightarrow$ 196: $dist(174, 196) \times 0.8 + 6.25 = 23.85\ ms$
- 196 $\rightarrow$ 110: $dist(196, 110) \times 0.8 + 6.25 = 75.05\ ms$
- 110 $\rightarrow$ 142: $dist(110, 142) \times 0.8 + 6.25 = 31.85\ ms$
- 142 $\rightarrow$ 149: $dist(142, 149) \times 0.8 + 6.25 = 11.85\ ms$
- 149 $\rightarrow$ 1: $dist(149, 1) \times 0.8 + 6.25 = 124.65\ ms$
- 1 $\rightarrow$ 172: $dist(1, 172) \times 0.8 + 6.25 = 143.05\ ms$
- 172 $\rightarrow$ 82: $dist(172, 82) \times 0.8 + 6.25 = 78.25\ ms$
- 82 $\rightarrow$ 18: $dist(82, 18) \times 0.8 + 6.25 = 57.45\ ms$

Tempo total de leitura para o conjunto de requisições: **617.7 *ms***

Algoritmo SSF

Este algoritmo utiliza como critério de ordenação a distância entre consultas adjacentes. Uma vez as consultadas ordenadas, podemos utilizar uma fila para simular sua execução.

Sequência de execução: `100 110 131 142 149 172 174 196 82 18 1`

- 100 $\rightarrow$ 110: $dist(100, 110) \times 0.8 + 6.25 = 14.25\ ms$
- 110 $\rightarrow$ 131: $dist(110, 131) \times 0.8 + 6.25 = 23.05\ ms$
- 131 $\rightarrow$ 142: $dist(131, 142) \times 0.8 + 6.25 = 15.05\ ms$
- 142 $\rightarrow$ 149: $dist(142, 149) \times 0.8 + 6.25 = 11.85\ ms$
- 149 $\rightarrow$ 172: $dist(149, 172) \times 0.8 + 6.25 = 24.65\ ms$
- 172 $\rightarrow$ 174: $dist(172, 174) \times 0.8 + 6.25 = 7.85\ ms$
- 174 $\rightarrow$ 196: $dist(174, 196) \times 0.8 + 6.25 = 23.85\ ms$

- 196 → 82: $dist(196, 82) \times 0.8 + 6.25 = 97.45\ ms$
- 82 → 18: $dist(82, 18) \times 0.8 + 6.25 = 57.45\ ms$
- 18 → 1: $dist(18, 1) \times 0.8 + 6.25 = 19.85\ ms$

Tempo total de leitura para o conjunto de requisições: **295.3 *ms***

Algoritmo Elevador

Este algoritmo utiliza como critério de ordenação o sentido para o qual o braço está se movimento no momento, sendo assim, as primeiras requisições a serem atendidas são aquelas que estiverem em ordem crescente caso o braço venha se movimentando da esquerda→direita, e ordem decrescente caso o braço esteja em movimentação da direita→esquerda. Quando o braço chega ao fim/inicio do seu curso, ele altera seu sentido de movimentação até que todas as requisições sejam atendidas. Em teoria, isso ocorrerá no máximo 2 vezes.

Sequência de execução: `100 110 131 142 149 172 174 196 82 18 1`

- 100 → 110: $dist(100, 110) \times 0.8 + 6.25 = 14.25\ ms$
- 110 → 131: $dist(110, 131) \times 0.8 + 6.25 = 23.05\ ms$
- 131 → 142: $dist(131, 142) \times 0.8 + 6.25 = 15.05\ ms$
- 142 → 149: $dist(142, 149) \times 0.8 + 6.25 = 11.85\ ms$
- 149 → 172: $dist(149, 172) \times 0.8 + 6.25 = 24.65\ ms$
- 172 → 174: $dist(172, 174) \times 0.8 + 6.25 = 7.85\ ms$
- 174 → 196: $dist(174, 196) \times 0.8 + 6.25 = 23.85\ ms$
- 196 → 82: $dist(196, 82) \times 0.8 + 6.25 = 97.45\ ms$
- 82 → 18: $dist(82, 18) \times 0.8 + 6.25 = 57.45\ ms$
- 18 → 1: $dist(18, 1) \times 0.8 + 6.25 = 19.85\ ms$

Tempo total de leitura para o conjunto de requisições: **295.3 *ms***

Para a resolução deste exercício, foi desenvolvido o código `q1.py` em anexo.

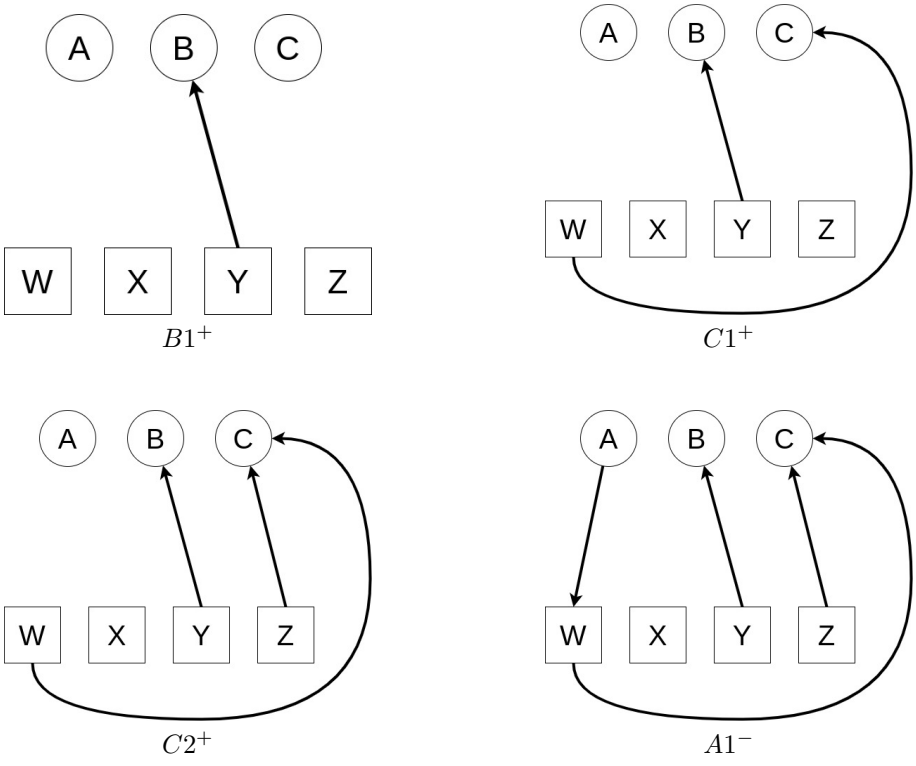
Exercício 2

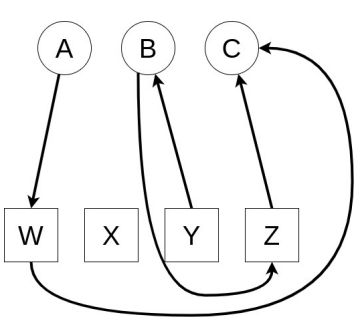
a)

Processo	Operação
$B1^+$	B requisita Y
$C1^+$	C requisita W
$C2^+$	C requisita Z
$A1^-$	A requisita W
$B2^-$	B requisita Z
$C3^-$	C requisita Y
Deadlock	

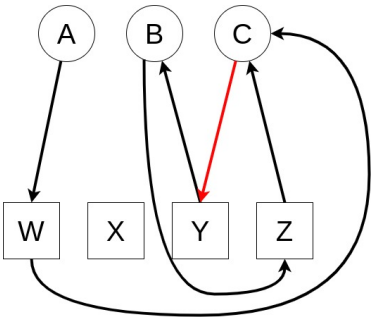
Tabela 1: Sequência de execução provocando *deadlock*

b)





$B2^{+}$



$C3^{-}$

c) Uma proposta de solução para a situação apresentada é alterar as seguintes operações do **Processo C**

- C2 requisita Z → C2 requisita Y
- C3 requisita Y → C3 requisita Z

Processo	Operação
$B1^{+}$	B requisita Y
$C1^{+}$	C requisita W
$C2^{-}$	C requisita Y
$A1^{-}$	A requisita W
$B2^{+}$	B requisita Z
$C3^{-}$	C requisita Z
B3	B usa Y+Z
B4	B libera Y
$C2^{+}$	C requisita Y
B5	B libera Z
$C3^{+}$	C requisita Z
C4	C usa W+Z+Y
C5	C libera Y
C6	C libera Z
C7	C libera W
$A1^{+}$	A requisita W
$A2^{+}$	A requisita X
$A3^{+}$	A requisita Z
A4	A usa W+X+Z
A5	A libera W
A6	A libera X
A7	A libera Z
Concluído	

Tabela 2: Execução bem sucedida

Exercício 3

Implementação realizada nos arquivos `ex3.c`, `requests.c`, compilada através do arquivo `Makefile`. Teste com a execução do enunciado através do comando `./ex3 < in`.