GERENCIAMENTO DE MEMÓRIA

1-

a) O tamanho é 4, pois temos a tabela de página com 16 linhas (0 a 15). Assim, para representarmos 16 valores, precisamos de 4 bits, pois com 4 bits, conseguimos representar de 0 (0000) até 15 (1111). Podemos também deduzir usando exponenciação relativo ao valor 2. Assim, temos que 16 = 24. O valor 4 presente no expoente corresponde a quantidade de bits.

b)Nesse caso, procuramos pelo maior valor presente na tabela de páginas. Na linha 13, vemos a presença do maior valor que é 31. Convertendo ele para binário, temos 11111. Assim, são necessários 5 bits para representar de 0 até 31.

c)

129 = 1000 0001
p = 1000 = linha 8 -> f = 25 = 11001+0001 = 401

57 = 0011 1001
p = 0011 = linha 3 -> f = 1 = 01+1001 = 25

23 = 0001 0111
p = 0001 = linha 1 -> f = 7 = 111+0111 = 119

191 = 1011 1111
p = 1011 = linha 11 -> f = 2 = 10+1111 = 47

93 = 0101 1101
p = 0101 = linha 5 -> f = 18 = 10010+1101 = 301

137 = 1000 1001
p = 1000 = linha 8 -> f = 25 = 11001+1001 = 409

29 = 0001 1101
p = 0001 = linha 1 -> f = 7 = 111+1101 = 125

12 = 0000 1100
p = 0000 = linha 0 -> f = 23 = 10111+1100 = 380

46 = 0010 1110
p = 0010 = linha 2 -> f = 0 = 0+1110 = 14

20 = 0001 0100
p = 0001 = linha 1 -> f = 7 = 111+0100 = 116

150 = 1001 0110
p = 1001 = linha 9 -> f = 14 = 1110+0110 = 230

2-

a)Para ambos, teremos tamanho 2. O motivo para p1 é que a tabela de página de nível 1 apresenta 4 linhas. Assim, temos 22 = 4. O expoente nos informa a quantidade de bits. Nesse caso, 2 bits. O mesmo ocorre para p2, visto que as tabelas de página de nível 2 também apresentam 4 linhas.

b)Sim. Analisando os valores contidos nas tabelas de página de nível 2, temos como maior valor o número 15 (tabela de página #2 (nível 2), linha 0). Assim, para representarmos 15, precisamos de 4 bits, pois 15 em binário é igual a 1 1 1 1.

c)
27 = 00 01 1011 = p1 = 00 = 0 -> linha 0 = pagina 1 ->
01 = 1 -> linha 1 -> pagina 1 - > f = 8 = 1000+1011 = 139

202 = 11 00 1010 = p1 = 11 = 3 -> linha 3 = pagina 2 ->
00 = 0 -> linha 0 -> pagina 2 - > f = 15 = 1111+1010 = 250

190 = 10 11 1110 = p1 = 10 = 2 -> linha 2 = pagina 0 ->
11 = 3 -> linha 3 -> pagina 0 - > f = 2 = 10+1110 = 46

15 = 00 00 1111 = p1 = 00 = 0 -> linha 0 = pagina 1 ->
00 = 0 -> linha 0 -> pagina 1 - > f = 11 = 1011+1111 = 191

116 = 01 11 0100 = p1 = 01 = 1 -> linha 1 = pagina 3 ->
11 = 3 -> linha 3 -> pagina 3 - > f = 5 = 101+0100 = 84

162 = 10 10 0010 = p1 = 10 = 2 -> linha 2 = pagina 0 ->
10 = 2 -> linha 2 -> pagina 2 - > f = 12 = 1100+0010 = 194

29 = 00 01 1101 = p1 = 00 = 0 -> linha 0 = pagina 1 ->
01 = 1 -> linha 1 -> pagina 1 - > f = 8 = 1000+1101 = 141

12 = 00 00 1100 = p1 = 00 = 0 -> linha 0 = pagina 1 ->
00 = 0 -> linha 0 -> pagina 1 - > f = 11 = 1011+1100 = 188

47 = 00 10 1111 = p1 = 00 = 0 -> linha 0 = pagina 1 ->
2 = 2 -> linha 2 -> pagina 1 - > f = 4 = 100+1111 = 79

5 = 00 00 0101 = p1 = 00 = 0 -> linha 0 = pagina 1 ->
00 = 0 -> linha 0 -> pagina 1 - > f = 11 = 1011+0101 = 181

132 = 10 00 0100 = p1 = 10 = 2 -> linha 2 = pagina 0 ->
00 = 0 -> linha 0 -> pagina 0 - > f = 9 = 1001+0100 = 152

3-

a)Como o sistema operacional suporta apenas 4 processos, é suficiente 2 bits, pois, 22 = 4. Como cada processo possui 16 páginas, é suficiente 4 bits, pois 24 = 16. Por fim, como cada página apresenta 8 endereço, é suficiente 3 bits, pois 23 = 8.

b)Como na tabela de página invertida as linhas da tabela representam f, temos 64 valores diferentes para f. Assim, é suficiente 6 bits, pois 26 = 64.

c)

431 = 11 0101 111
110101 = 53 está na linha -> 32 = 100000+111 = 263

510 = 11 1111 110
111111 = 63 está na linha -> 59 = 111011+110 = 478

152 = 01 0011 000
010011 = 19 está na linha -> 57 = 111001+000 = 456

235 = 01 1101 011
011101 = 29 está na linha -> 10 = 1010+011 = 83

315 = 10 0111 011
100111 = 39 está na linha -> 9 = 1001+011 = 75

92 = 00 1011 100
001011 = 11 está na linha -> 53 = 110101+100 = 428

2 = 00 0000 010
000000 = 0 está na linha -> 16 = 10000+010 = 130

51 = 00 0110 011

000110 = 6 está na linha -> 50 = 110010+011 = 403

389 = 11 0000 101
110000 = 48 está na linha -> 39 = 100111+101 = 317

DEADLOCK

1- As tabelas a seguir apresentam as matrizes alocação, máximo e o vetor disponível para um conjunto de processos/recursos em um dado sistema operacional. Para cada um dos cenários, verifique se o sistema está ou não em deadlock. Em caso de não deadlock, apresente uma sequência de execução acompanhada do valor do vetor disponível após a execução de cada processo. Em caso de deadlock, justifique sua resposta, apresentando a matriz necessária.

A)

Disponível		
A	B	C
1	2	1

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	2	2	3	5	4	3
P_1	3	1	0	7	2	2
P_2	1	2	0	3	3	1
P_3	0	1	1	2	4	2
P_4	4	1	0	4	2	0

B)

Disponível		
A	B	C
1	1	2

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	1	2	1	4	3	1
P_1	2	3	1	5	3	2
P_2	1	3	1	2	4	6
P_3	1	0	0	3	4	1
P_4	1	2	2	5	3	4

C)

Disponível		
A	B	C
0	1	3

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	1	5	0	4	4	2
P_1	1	0	3	5	0	5
P_2	1	1	0	2	2	1
P_3	1	0	2	3	0	4
P_4	1	1	1	5	4	5

A) Sem deadlock, todos os processos foram executados.

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	2	2	3	5	4	3	3	2	0	1	2	1	P4
P1	3	1	0	7	2	2	4	1	2	5	3	1	P2
P2	1	2	0	3	3	1	2	1	1	6	5	1	P3
P3	0	1	1	2	4	2	2	3	1	6	6	2	P1

P4	4	1	0	4	2	0	0	1	0	9	7	2	P0
										11	9	5	

B) Existe DeadLock logo no início, nenhum processo pode ser executado.

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	1	2	1	4	3	1	3	1	0	1	1	2	
P1	2	3	1	5	3	2	3	0	1				
P2	1	3	1	2	4	6	1	1	5				
P3	1	0	0	3	4	1	2	4	1				
P4	1	2	2	5	3	4	4	1	2				

C) Existe DeadLock logo no início, nenhum processo pode ser executado.

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	1	5	0	4	4	2	3	1	2	0	1	3	
P1	1	0	3	5	0	5	4	0	2				
P2	1	1	0	2	2	1	1	1	1				
P3	1	0	2	3	0	4	2	0	2				
P4	1	1	1	5	4	5	4	3	4				

D)

Disponível		
A	B	C
2	2	3

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	2	2	3	2	2	3
P_1	3	1	0	5	1	2
P_2	1	2	0	3	3	1
P_3	2	1	1	2	3	2
P_4	4	1	0	4	2	0

E)

Disponível		
A	B	C
0	4	2

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	4	2	1	4	4	2
P_1	2	3	1	6	3	3
P_2	2	3	1	2	4	6
P_3	1	0	0	2	3	1
P_4	1	2	2	5	3	4

F)

Disponível		
A	B	C
3	1	0

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	0	4	0	3	4	2
P_1	1	0	3	2	2	5
P_2	1	1	0	3	1	1
P_3	1	0	2	1	0	4
P_4	1	1	1	4	2	5

D) Sem deadlock, todos os processos foram executados.

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	2	2	3	2	2	3	0	0	0	2	2	3	P0
P1	3	1	0	5	1	2	2	0	2	4	4	6	P1
P2	1	2	0	3	3	1	2	1	1	7	5	6	P2
P3	2	1	1	2	3	2	0	2	1	8	7	6	P3
P4	4	1	0	4	2	0	0	1	0	10	8	7	P4
										14	9	7	

E) Sem deadlock, todos os processos foram executados.

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	4	2	1	4	4	2	0	2	1	0	4	2	P0
P1	2	3	1	6	3	3	4	0	2	4	6	3	P1
P2	2	3	1	2	4	6	0	1	5	6	9	4	P3
P3	1	0	0	2	3	1	1	3	1	7	9	4	P4
P4	1	2	2	5	3	4	4	1	2	8	11	6	P2

	Alocação			Máximo			Necessário			Disponível			
										10	14	7	

F) Existe DeadLock logo no início, nenhum processo pode ser executado.

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	0	4	0	3	4	2	3	0	2	3	1	0	
P1	1	0	3	2	2	5	1	2	2				
P2	1	1	0	3	1	1	2	0	1				
P3	1	0	2	1	0	4	0	0	2				
P4	1	1	1	4	2	5	3	1	4				

G)

	Disponível		
	A	B	C
	2	5	0

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	2	2	3	4	4	3
P_1	4	1	0	7	1	0
P_2	1	2	3	3	3	3
P_3	2	1	1	2	4	2
P_4	4	1	1	4	2	0

H)

	Disponível		
	A	B	C
	1	1	1

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	3	5	1	4	6	1
P_1	2	3	1	5	3	2
P_2	1	3	5	2	4	6
P_3	2	3	0	3	4	1
P_4	1	2	2	5	3	4

I)

	Disponível		
	A	B	C
	0	0	3

	Alocação			Máximo		
	A	B	C	A	B	C
P_0	1	4	0	3	4	2
P_1	1	2	3	2	0	5
P_2	3	2	0	3	2	1
P_3	1	0	3	1	0	4
P_4	3	5	3	5	5	5

G) Erro no processo 4, alocação maior que o máximo (P4-C).

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	2	2	3	4	4	3	2	2	0	2	5	0	
P1	4	1	0	7	1	0	3	0	0				
P2	1	2	3	3	3	3	2	1	0				

	Alocação			Máximo			Necessário			Disponível			
P3	2	1	1	2	4	2	0	3	1				
P4	4	1	1	4	2	0	0	0	-1				

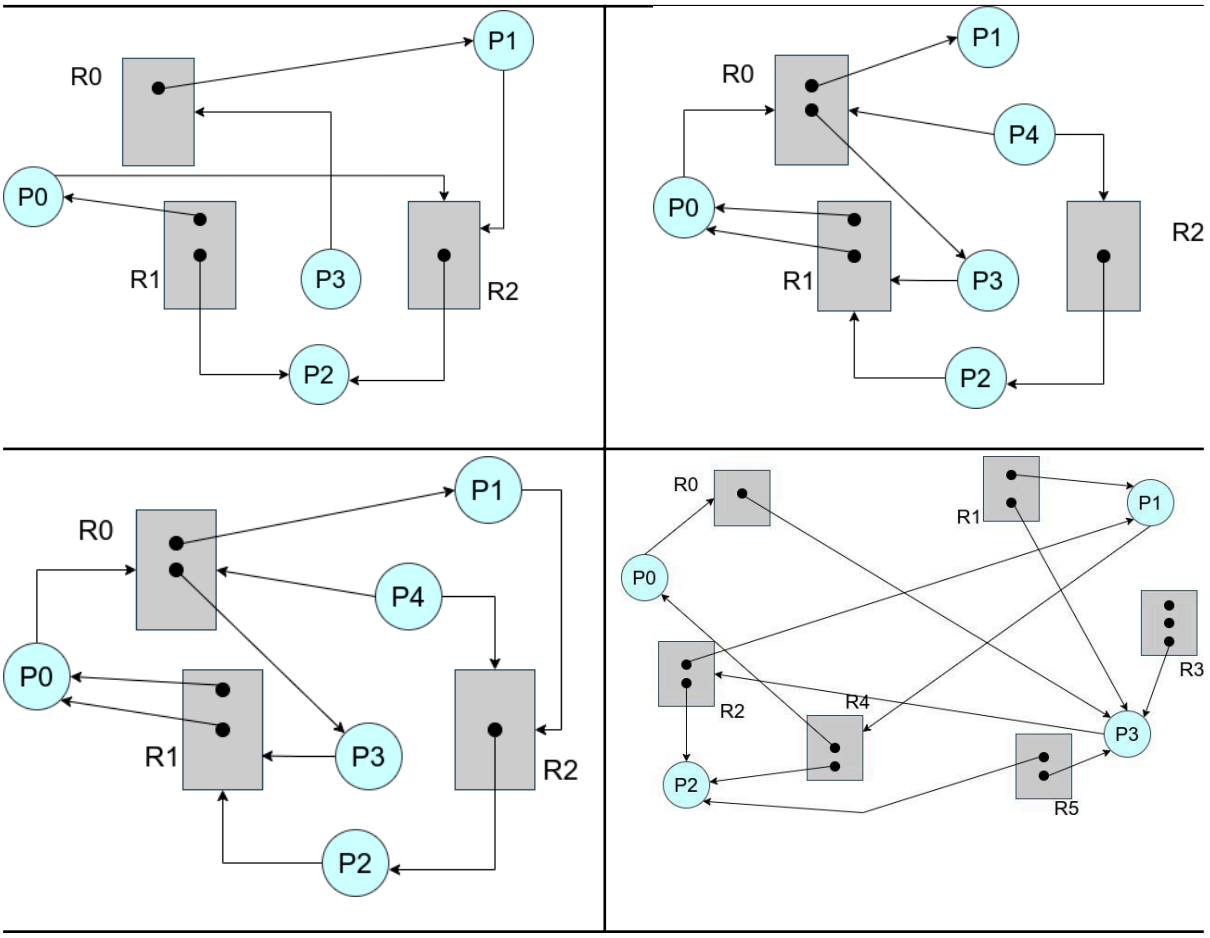
H) Sem deadlock, todos os processos foram executados.

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	3	5	1	4	6	1	1	1	0	1	1	1	P0
P1	2	3	1	5	3	2	3	0	1	4	6	2	P1
P2	1	3	5	2	4	6	1	1	1	6	9	3	P2
P3	2	3	0	3	4	1	1	1	1	7	12	8	P3
P4	1	2	2	5	3	4	4	1	2	9	15	8	P4
										10	17	10	

I) Erro no processo 1, alocação maior que o máximo (P1-B). Necessário

	Alocação			Máximo			Necessário			Disponível			
	A	B	C	A	B	C	A	B	C	A	B	C	EXECUTOU
P0	1	4	0	3	4	2	2	0	2	0	0	3	
P1	1	2	3	2	0	5	1	-2	2				
P2	3	2	0	3	2	1							
P3	1	0	3	1	0	4							
P4	3	5	3	5	5	5							

2. Analise os seguintes grafos de alocação de recursos. Há presença de deadlock? Justifique sua resposta:



A)Não possui deadlock.
P3 está bloqueado pois não possui R0 que está sendo usado por P1. P1 está bloqueado pois não possui R2 que está sendo usado por P2. P0 está bloqueado pois não possui R2 que está sendo usado por P2.
P2 não está bloqueado, pois possui todos os recursos que precisa R2 e R1. P2 executa -> libera R2 e R2
P2 executa -> Desbloqueia P1 e P0 P1
executa -> Desbloqueia P3

B) Não tem deadlock

P0 está bloqueado pois não possui R0 que está sendo usado por P1 e P3 P2 está bloqueado pois não possui R1 que está sendo usado por P0
P3 está bloqueado pois não possui R1 que está sendo usado por P0 P4 está bloqueado pois não possui R2 que está sendo usado por P2 P1 não está bloqueado pois possui todos os recursos que precisa R0 P1 executa -> Libera R0 e Desbloqueia P0

P0 executa -> Libera R1
P0 executa -> Desbloqueia P3 e P2
P2 executa -> Libera R2 e Desbloqueia P4

C) Tem deadlock.
P0 está bloqueado pois não possui R0 que está sendo usado por P3 e P1 P1 está bloqueado pois não possui R2 que está sendo usado por P2
P2 está bloqueado pois não possui R1 que está sendo usado por P0 P3 está bloqueado pois não possui R1 que está sendo usado por P0 P4 está bloqueado pois não possui R2 que está sendo usado por P2

D)P0 está bloqueado, pois não possui R0 que está em uso por P3.
P3 está bloqueado, pois não possui R2 que está em uso por P2 e P1.
P1 está bloqueado, pois não possui R4 que está em uso por P0 e P2.
P2 não está bloqueado. Ele possui todos os recursos que precisa: R2, R4 e R5.
Quando P2 finaliza, ele libera R4 e R2. P1 e P3 desbloqueiam.
Quando P3 finaliza, ele libera R0. P0 desbloqueia.
Não há deadlock.

SINCRONIZAÇÃO

1) Operações com semáforos

A seguir, apresentamos uma sequência de operações do semáforo no início e no final das tarefas A, B, C. Considere que cada tarefa executa em um núcleo de processador dedicado. E considere que cada ação (P(Sx), V(Sx) ou .) possui tempo igual a 1T.

	Task A	Task B	Task C
1	P(SA)	P(SB)	P(SC)
2	P(SA)	.	P(SC)
3	P(SA)	.	P(SC)
4	.	.	.
5	.	.	.
6	.	V(SC)	V(SB)
7	V(SB)	V(SA)	V(SB)
8	END	.	V(SA)
9		END	END

Determine para os 6 casos a,b,c,d,e,f apresentados na tabela abaixo, se e em qual sequência as tarefas são executadas, usando as inicializações das variáveis do semáforo dadas na tabela.

Semáforos	a)	b)	c)	d)	e)	f)	g)
SA	2	3	2	0	3	1	1
SB	0	0	1	0	1	0	1
SC	2	2	1	3	3	3	1

A) Deadlock, nenhuma task finaliza, TA e TC bloqueadas no T3 (tempo 3) e TB bloqueado em T1 (tempo 1).

Semáforos	a)
SA	2 1 0
SB	0
SC	2 1 0

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T3
TA	P(SA)	P(SA)	X										
TB	X	X	X										
TC	P(SC)	P(SC)	X										

B) Não existe Deadlock, Todas as tasks finalizam. TA em T8(tempo 8), TB em T16(tempo 16) e TC em T20 (tempo 20). Com isso concluímos que qualquer teste que tenha no mínimo SA, SB e SC como 3,0 e 2 respectivamente, não haverá deadlock.

Semáforos	b)
SA	3 2 1 2
SB	0 1 0
SC	2 1 0 1 0

Task	T1	T2	T3	T4	T5	T6	T7	T8	**...T12**	T13	T14	T15	T16	T17
TA	P(SA)	P(SA)	P(SA)	-	-	-	V(SB)	END						
TB	X	X	X	X	X	X	X	P(SB)	-	V(SC)	V(SA)	-	END	
TC	P(SC)	P(SC)	X	X	X	X	X	X	X	X	P(SC)	-	-	V(SB)

C) Deadlock pois tarefa C não executa. TA finaliza em T3, TB finaliza em T9.

Semáforos	c)
SA	2 1 0 1 0
SB	1 0 1
SC	1 0 1 0

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T3
TA	P(SA)	P(SA)	X	X	X	X	X	P(SA)	-	-	-	V(SB)	END
TB	P(SB)	-	-	-	-	V(SC)	V(SA)	-	END				
TC	P(SC)	X	X	X	X	X	P(SC)	X	X	X	X	X	

D) Existe Deadlock, a tarefa TA não executa e fica bloqueada em T15. TB finaliza em T15 e TC finaliza em T9

Semáforos	d)
SA	0 1 0 1
SB	0 1 0 1
SC	3 2 1 0 1

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15
TA	X	X	X	X	X	X	X	X	P(SA)	X	X	X	X	P(SA)	X
TB	X	X	X	X	X	X	P(SB)	-	-	-	-	V(SC)	V(SA)	-	END
TC	P(SC)	P(SC)	P(SC)	-	-	V(SB)	V(SB)	V(SA)	END						

E) Não tem deadlock, todas as task são executadas sem problemas. TA finaliza em T8. TB e TC finalizam em T9.

Semáforos	e)
SA	3 2 1 0
SB	1 0 1 2
SC	3 2 1 0 1

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
TA	P(SA)	P(SA)	P(SA)	-	-	-	V(SB)	END						
TB	P(SB)	-	-	-	-	V(SC)	V(SA)	-	END					
TC	P(SC)	P(SC)	P(SC)	-	-	V(SB)	V(SB)	V(SA)	END					

F) Não tem deadlock, todas as task são executadas sem problemas. TA finaliza em T20. TB finaliza em T15. TC finaliza em T9

Semáforos	f)
SA	1 0 1 0 1
SB	0 1 0 1
SC	3 2 1 0 1

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16
TA	P(SA)	X	X	X	X	X	X	X	P(SA)	X	X	X	X	P(SA)	-	-
TB	X	X	X	X	X	X	P(SB)	-	-	-	-	V(SC)	V(SA)	-	END	
TC	P(SC)	P(SC)	P(SC)	-	-	V(SB)	V(SB)	V(SA)	END							

T17	T18	T19	T20
-	-	V(SB)	END

G) Possui deadlock. Apenas de TB finaliza, em T9. TA fica bloqueado em T9 e TC em T8.

Semáforos	f)
SA	1 0 1
SB	1 0
SC	1 0 1

Task	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
TA	P(SA)	X	X	X	X	X	X	P(SA)	X					
TB	P(SB)	-	-	-	-	V(SC)	V(SA)	-	END					
TC	P(SC)	X	X	X	X	X	P(SC)	X						

2)

Codigo #A

O comportamento do código é controlado pelo objeto lock, para sincronizar o acesso a função wish entre as duas threads. O lock garante que apenas uma thread possa ser executada protegida por ele de cada vez, quando a thread adquire o lock, nenhuma outra pode adquiri-lo ate que a atual o libere, isso é feito através do acquire() e release().

l=Lock(): Cria um objeto de bloqueio.

l.acquire(): Adquire o bloqueio, bloqueando o acesso para outras threads.

l.release(): Libera o bloqueio.

Hi Sireesh
Your age is 15
Hi Nitya
Your age is 20
Hi Sireesh
Your age is 15
Hi Nitya
Your age is 20
Hi Sireesh
Your age is 15
Hi Nitya
Your age is 20

Código #B

O comportamento do código B é controlado pelo uso do objeto semaphore para sincronizar o acesso a função wish entre 4 threads. O Semaphore é inicializado com um valor de 2, o que significa que até duas threads podem acessar a função wish ao mesmo tempo.

s=Semaphore(2): Cria um semáforo com um contador inicial de 2.

s.acquire(): Adquire uma unidade do semáforo. Se o contador for maior que zero, a thread prossegue; caso contrário, ela fica bloqueada até que uma unidade esteja disponível.

s.release(): Libera uma unidade do semáforo, incrementando o contador.

Hi Sireesh
Hi Nitya
Hi Sireesh
Hi Nitya
Hi Nitya
Hi Sireesh
Hi Shiva

Hi Ajay
Hi Ajay
Hi Shiva
Hi Ajay
Hi Shiva

Código #C
O comportamento do código C é controlado pelo uso do objeto lock para sincronizar o acesso à variável global g entre várias threads. A funções add_one e add_two são definidas para adicionar 1 ou 2 a variável global g, respectivamente. Essas funções são projetadas para serem executadas por threads criadas dinamicamente dentro de um loop. O Lock é adquirido no início de cada função e liberado após a modificação da variável global g, garantindo que apenas uma thread possa modificar g por vez.

3)
I- O código é uma simulação de transferências entre duas contas bancárias (conta1 e conta2) usando threads. Cada thread representa uma transferência de um valor de 1 real da conta 1 para a conta 2.

II- O resultado varia, sendo que o saldo da conta 1 fica com uma média de 98 ao final e o saldo da conta 2 fica com uma média de 2 ao final da execução.

III- O resultado nem sempre é o mesmo ao executar o código 10 vezes, pois nao há uma ordem de execução das threads,ou seja, a variação pode ocorrer quando várias threads tentam acessar e modificar as variáveis de saldo das contas simultaneamente.

IV- Correção: Uso do Lock no método run() resolve e faz com o que a conta 1 zere e a conta dois fique com 100.

```
import threading
from threading import Semaphore

s = Semaphore(1)


class ContaBancaria:
def __init__(self, nome, saldo):
self.nome = nome
self.saldo = saldo
```

```
def __str__(self):
return self.nome


conta1 = ContaBancaria("conta1", 100)
conta2 = ContaBancaria("conta2", 0)


class ThreadTransferenciaEntreContas(threading.Thread):
def __init__(self, origem, destino, valor):
threading.Thread.__init__(self)
self.origem = origem
self.destino = destino
self.valor = valor

def run(self):
s.acquire()
origem_saldo_inicial = self.origem.saldo
origem_saldo_inicial -= self.valor
# time.sleep(0.001)
self.origem.saldo = origem_saldo_inicial
destino_saldo_inicial = self.destino.saldo
destino_saldo_inicial += self.valor
# time.sleep(0.001)
self.destino.saldo = destino_saldo_inicial
s.release()


if __name__ == "__main__":
threads = []
for i in range(100):
threads.append(ThreadTransferenciaEntreContas(conta1, conta2, 1))
for thread in threads:
thread.start()
for thread in threads:
thread.join()
print("Saldo da", conta1, ":", conta1.saldo)
print("Saldo da", conta2, ":", conta2.saldo)
```