SO

1º Teste de 2008/09

Lamport – bakery algorithm

```
1   int senha[N];           // Inicializado a 0
2   int escolha[N];         // Inicializado a FALSE
3   Fechar (int i) {
4           int j;
5           escolha[i] = TRUE;
6           senha [i] = 1 + maxn(senha);
7           escolha[i] = FALSE;
8           for (j=0; j<N; j++) {
9                   if (j==i) continue;
10                  while (escolha[j]) ;
11                  while (senha [j] && (senha [j] < senha [i])||
12                                  ((senha [i] == senha [j]) && (j < i)));
13          }
14  }
15  Abrir (int i) {senha [i] = 0;}
```

Situação Inicial

O estado do algoritmo no momento inicial está descrito nas tabelas senha e escolha.

Processos	senha	escolha
1	0	FALSE
2	1	FALSE
3	0	FALSE
4	2	FALSE
5	0	FALSE

Considere que em cada *timeslice* o processo consegue executar totalmente as funções Fechar e Abrir. Com base nestas informações responda às questões seguintes preenchendo as tabelas

Etapa 1 e 2

. [0,5 valor] O Processo 3 executa fechar e na instrução 6 calcula `maxn` e **perde o processador em seguida**

Processos	senha	escolha
1	0	FALSE
2	1	FALSE
3	0	TRUE
4	2	FALSE
5	0	FALSE

2. [0,5 valor] O Processo 4 retoma a execução na instrução 8 e executa-se até terminar o seu *timeslice*

Processos	senha	escolha
1	0	FALSE
2	1	FALSE
3	0	TRUE
4	2	FALSE
5	0	FALSE

Etapa 3 e 4

3. [0,5 valor] O Processo 5 inicia a execução de fechar e executa-se até terminar o seu *timeslice*.

Processos	senha	escolha
1	0	FALSE
2	1	FALSE
3	0	TRUE
4	2	FALSE
5	3	FALSE

4. [0,5 valor] O Processo 2 executa Abrir e termina o seu *timeslice*

Processos	senha	escolha
1	0	FALSE
2	0	FALSE
3	0	TRUE
4	2	FALSE
5	3	FALSE

Etapa 5 e 6

5. [0,5 valor] O Processo 3 continua até terminar o seu *timeslice*

Processos	senha	escolha
1	0	FALSE
2	0	FALSE
3	3	FALSE
4	2	FALSE
5	3	FALSE

6. [0,5 valor] O Processo 4 executa Abrir e termina o seu *timeslice*

Processos	senha	Escolha
1	0	FALSE
2	0	FALSE
3	3	FALSE
4	0	FALSE
5	3	FALSE

Indique qual a ordem de execução dos processos 3 e 5. Justifique

```
8       for (j=0; j<N; j++) {
9               if (j==i) continue;
10              while (escolha[j]) ;
11              while (senha [j] && (senha [j] < senha [i])||
12                      ((senha [i] == senha [j]) && (j < i)));
13              }
14      }
```

- Tanto o processo 5 como o processo 3 escolhem 3, porque ambos viram o mesmo estado da tabela senha

- Se o processo 5 se executar primeiro
  - No ciclo de comparação **as senhas são iguais e 3 <5** portanto **fica em ciclo**
- Se o processo 3 se executar primeiro
  - No ciclo de comparação 3 tem as senhas são iguais mas 5<3 torna a condição falsa e **não fica em ciclo**
- Conclusão executa-se sempre 3 primeiro e depois 5

Monitor

```
#define LIVRE 0
#define OCUPADO 1
#define N 5
int EGarfos[N] = {LIVRE, LIVRE, LIVRE, LIVRE, LIVRE};
int filosofos_bloqueados =0;

Requisitar (int id)
{
        monitor_enter ();
        while  ((Egarfos (id) == OCUPADO) || (Egarfos ((id+1)%N) == OCUPADO))
        {
                filosofos_bloqueados ++;
                monitor_wait();
        }
        Egarfos (id) = OCUPADO;
        Egarfos ((id+1)%N) = OCUPADO;
        monitor_exit();
}

Libertar (int id)
{
        monitor_enter ();
        Egarfos (id) = LIVRE;
        Egarfos ((id+1)%N) = LIVRE;
        while (filosofos_bloqueados ) {
                filosofos_bloqueados --;
                monitor_signal();
        }
        monitor_exit();
}
```

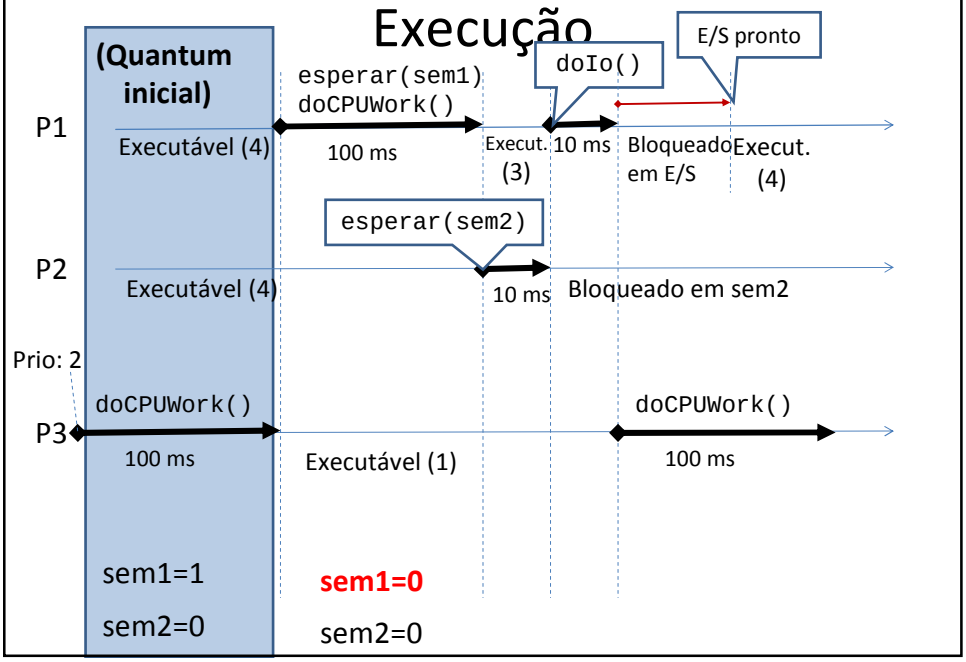
Escalonamento com prioridades dinâmicas

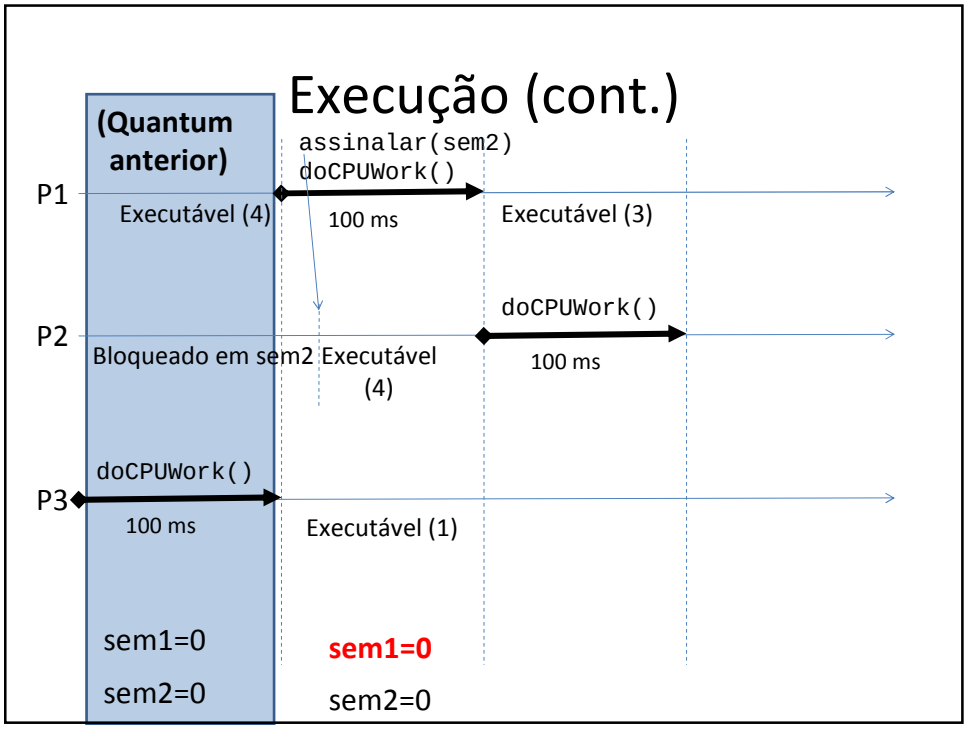
- Quantum fixo = 100ms, sem preempção
- 4 níveis de prioridade
  - CPU durante todo o quantum desce 1 nível
  - Bloqueio sobe 2 níveis
- 3 Processos:

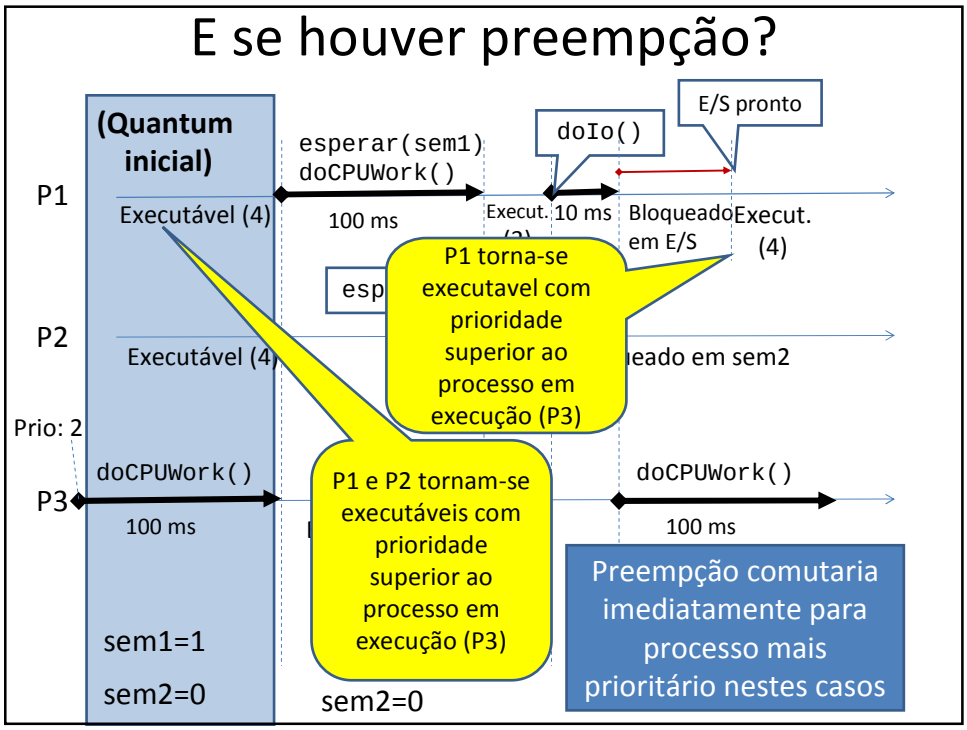
```
mainP1( ) {
 while (TRUE) {
  esperar(sem1);
  doCPUwork();
  doIO();
  assinalar(sem2);
  doCPUwork();
 }
}
```

```
mainP2( ) {
 while (TRUE) {
  esperar(sem2);
  doCPUwork();
  doIO();
  assinalar(sem1);
  doCPUwork();
 }
}
```

```
mainP3( ) {
 while (TRUE) {
  doCPUwork();
 }
}
```




Execução (cont.)
(Quantum anterior)
assinalar(sem2)
doCPUWork()
P1
Executável (4)
100 ms
Executável (3)
doCPUWork()
P2
Bloqueado em sem2
Executável (4)
100 ms
doCPUWork()
P3
100 ms
Executável (1)
sem1=0
sem2=0
sem1=0
sem2=0


E se houver preempção?
(Quantum inicial)
esperar(sem1)
doCPUWork()
doIo()
E/S pronto
P1
Executável (4)
100 ms
Execut.
10 ms
Bloqueado em E/S
Execut. (4)
P1 torna-se executavel com prioridade superior ao processo em execução (P3)
P2
Executável (4)
Prio: 2
doCPUWork()
P3
100 ms
P1 e P2 tornam-se executáveis com prioridade superior ao processo em execução (P3)
doCPUWork()
100 ms
Preempção comutaria imediatamente para processo mais prioritário nestes casos
sem1=1
sem2=0
sem2=0