

## Aula 5

### Professores:

Felipe M. G. França  
Valmir C. Barbosa

### Conteúdo:

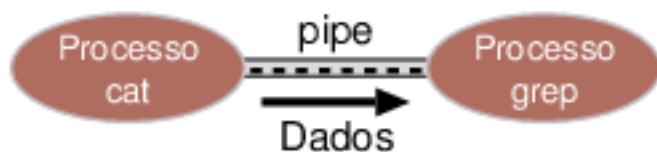
#### Comunicação entre processos

- Introdução
- Problemas inerentes à comunicação
- Propostas para exclusão mútua

# Comunicação entre processos

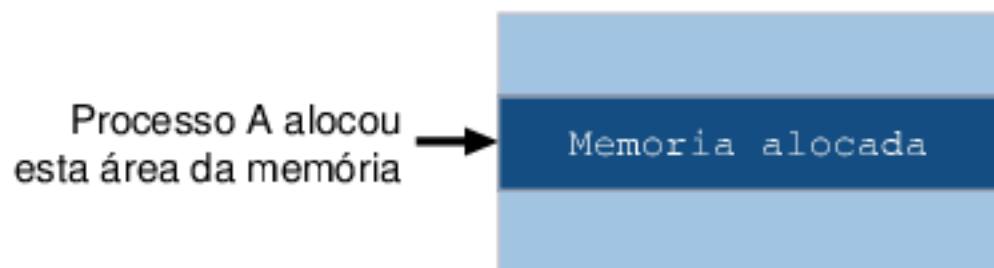
- Os processos que cooperam entre si para executar uma tarefa precisam se comunicar uns com os outros:

Exemplo: cat c1 c2 c3 | grep árvore



O processo cat se comunica, via pipe, com o processo grep, para enviar para este processo os arquivos que foram passados pelo usuário ao processo anterior.

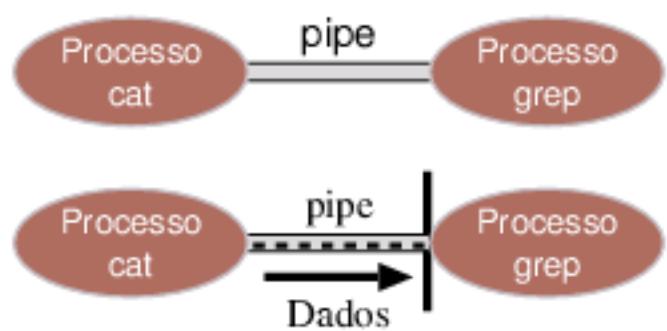
- A comunicação entre os processos deve ser feita de tal modo que seja estruturada e independente de interrupções.
- Devemos evitar que os processos interfiram uns com os outros, um problema chamado de **exclusão mútua**:



O que ocorreria se um outro processo pudesse alocar e usar a mesma área de memória do processo A?

# Comunicação entre processos

- Na comunicação entre processos cooperantes, a **sincronização** destes processos é importante:
  - No exemplo anterior: cat c1 c2 c3 | grep árvore:



O processo grep somente poderá executar quando o processo cat colocar algum dado no pipe.

Se o pipe ficar cheio, o processo cat somente poderá enviar novos dados ao pipe após algum dado ser processado pelo processo grep.

# Condições de corrida

- Ocorre quando os processos cooperantes precisam compartilhar um recurso, que ambos podem ler e gravar:

Diretório do spooler

4	abc
5	prog.c
6	prog.n
7	

Um exemplo de um recurso compartilhado, um diretório de spooler, usado por um daemon de impressão. Este diretório contém várias entradas, numeradas por 1, 2, ..., sendo que cada uma pode armazenar o nome de um arquivo. O daemon percorre cada uma das entradas seqüencialmente, voltando para a primeira entrada após a impressão do arquivo da última entrada.

Variáveis de controle

`out = 4`

`in = 7`



# Condições de corrida

- ➡ Ocorre quando os processos cooperantes precisam compartilhar um recurso, que ambos podem ler e gravar:

Diretório do spooler

4	abc
5	prog.c
6	prog.n
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	
98	
99	
100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	
129	
130	
131	
132	
133	
134	
135	
136	
137	
138	
139	
140	
141	
142	
143	
144	
145	
146	
147	
148	
149	
150	
151	
152	
153	
154	
155	
156	
157	
158	
159	
160	
161	
162	
163	
164	
165	
166	
167	
168	
169	
170	
171	
172	
173	
174	
175	
176	
177	
178	
179	
180	
181	
182	
183	
184	
185	
186	
187	
188	
189	
190	
191	
192	
193	
194	
195	
196	
197	
198	
199	
200	
201	
202	
203	
204	
205	
206	
207	
208	
209	
210	
211	
212	
213	
214	
215	
216	
217	
218	
219	
220	
221	
222	
223	
224	
225	
226	
227	
228	
229	
230	
231	
232	
233	
234	
235	
236	
237	
238	
239	
240	
241	
242	
243	
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	
256	
257	
258	
259	
260	
261	
262	
263	
264	
265	
266	
267	
268	
269	
270	
271	
272	
273	
274	
275	
276	
277	
278	
279	
280	
281	
282	
283	
284	
285	
286	
287	
288	
289	
290	
291	
292	
293	
294	
295	
296	
297	
298	
299	
300	
301	
302	
303	
304	
305	
306	
307	
308	
309	
310	
311	
312	
313	
314	
315	
316	
317	
318	
319	
320	
321	
322	
323	
324	
325	
326	
327	
328	
329	
330	
331	
332	
333	
334	
335	
336	
337	
338	
339	
340	
341	
342	
343	
344	
345	
346	
347	
348	
349	
350	
351	
352	
353	
354	
355	
356	
357	
358	
359	
360	
361	
362	
363	
364	
365	
366	
367	
368	
369	
370	
371	
372	
373	
374	
375	
376	
377	
378	
379	
380	
381	
382	
383	
384	
385	
386	
387	
388	
389	
390	
391	
392	
393	
394	
395	
396	
397	
398	
399	
400	
401	
402	
403	
404	
405	
406	
407	
408	
409	
410	
411	
412	
413	
414	
415	
416	
417	
418	
419	
420	
421	
422	
423	
424	
425	
426	
427	
428	
429	
430	
431	
432	
433	
434	
435	
436	
437	
438	
439	
440	
441	
442	
443	
444	
445	
446	
447	
448	
449	
450	
451	
452	
453	
454	
455	
456	
457	
458	
459	
460	
461	
462	
463	
464	
465	
466	
467	
468	
469	
470	
471	
472	
473	
474	
475	
476	
477	
478	
479	
480	
481	
482	
483	
484	
485	
486	
487	
488	
489	
490	
491	
492	
493	
494	
495	
496	
497	
498	
499	
500	
501	
502	
503	
504	
505	
506	
507	
508	
509	
510	
511	
512	
513	
514	
515	
516	
517	
518	
519	
520	
521	
522	
523	
524	
525	
526	
527	
528	
529	
530	
531	
532	
533	
534	
535	
536	
537	
538	
539	
540	
541	
542	
543	
544	
545	
546	
547	
548	
549	
550	
551	
552	
553	
554	
555	
556	
557	
558	
559	
560	
561	
562	
563	
564	
565	
566	
567	
568	
569	
570	
571	
572	
573	
574	
575	
576	
577	
578	
579	
580	
581	
582	
583	
584	
585	
586	
587	
588	
589	
590	
591	
592	
593	
594	
595	
596	
597	
598	
599	
600	
601	
602	
603	
604	
605	
606	
607	
608	
609	
610	
611	
612	
613	
614	
615	
616	
617	
618	
619	
620	
621	
622	
623	
624	
625	
626	
627	
628	
629	
630	
631	
632	
633	
634	
635	
636	
637	
638	
639	
640	
641	
642	
643	
644	
645	
646	
647	
648	
649	
650	
651	
652	
653	
654	
655	
656	
657	
658	
659	
660	
661	
662	
663	
664	
665	
666	
667	
668	
669	
670	
671	
672	
673	
674	
675	
676	
677	
678	
679	
680	
681	
682	
683	
684	
685	
686	
687	
688	
689	
690	
691	
692	
693	
694	
695	
696	

# Condições de corrida

- Ocorre quando os processos cooperantes precisam compartilhar um recurso, que ambos podem ler e gravar:

Diretório do spooler

4	abc
5	prog.c
6	prog.n
7	cederj.txt
8	:

O processo B decidiu imediatamente imprimir o arquivo cederj.txt. O processo então lê o valor da variável `in`, que é igual a 7, coloca o nome do arquivo na Entrada 7, e atualiza a variável `in` para apontar para a Entrada 8.

Variáveis de controle

`out = 4`

`in = 8`



# Condições de corrida

- ➡ Ocorre quando os processos cooperantes precisam compartilhar um recurso, que ambos podem ler e gravar:

Diretório do spooler

4	abc
5	prog.c
6	prog.n
7	prova.txt
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	
66	
67	
68	
69	
70	
71	
72	
73	
74	
75	
76	
77	
78	
79	
80	
81	
82	
83	
84	
85	
86	
87	
88	
89	
90	
91	
92	
93	
94	
95	
96	
97	
98	
99	
100	
101	
102	
103	
104	
105	
106	
107	
108	
109	
110	
111	
112	
113	
114	
115	
116	
117	
118	
119	
120	
121	
122	
123	
124	
125	
126	
127	
128	
129	
130	
131	
132	
133	
134	
135	
136	
137	
138	
139	
140	
141	
142	
143	
144	
145	
146	
147	
148	
149	
150	
151	
152	
153	
154	
155	
156	
157	
158	
159	
160	
161	
162	
163	
164	
165	
166	
167	
168	
169	
170	
171	
172	
173	
174	
175	
176	
177	
178	
179	
180	
181	
182	
183	
184	
185	
186	
187	
188	
189	
190	
191	
192	
193	
194	
195	
196	
197	
198	
199	
200	
201	
202	
203	
204	
205	
206	
207	
208	
209	
210	
211	
212	
213	
214	
215	
216	
217	
218	
219	
220	
221	
222	
223	
224	
225	
226	
227	
228	
229	
230	
231	
232	
233	
234	
235	
236	
237	
238	
239	
240	
241	
242	
243	
244	
245	
246	
247	
248	
249	
250	
251	
252	
253	
254	
255	
256	
257	
258	
259	
260	
261	
262	
263	
264	
265	
266	
267	
268	
269	
270	
271	
272	
273	
274	
275	
276	
277	
278	
279	
280	
281	
282	
283	
284	
285	
286	
287	
288	
289	
290	
291	
292	
293	
294	
295	
296	
297	
298	
299	
300	
301	
302	
303	
304	
305	
306	
307	
308	
309	
310	
311	
312	
313	
314	
315	
316	
317	
318	
319	
320	
321	
322	
323	
324	
325	
326	
327	
328	
329	
330	
331	
332	
333	
334	
335	
336	
337	
338	
339	
340	
341	
342	
343	
344	
345	
346	
347	
348	
349	
350	
351	
352	
353	
354	
355	
356	
357	
358	
359	
360	
361	
362	
363	
364	
365	
366	
367	
368	
369	
370	
371	
372	
373	
374	
375	
376	
377	
378	
379	
380	
381	
382	
383	
384	
385	
386	
387	
388	
389	
390	
391	
392	
393	
394	
395	
396	
397	
398	
399	
400	
401	
402	
403	
404	
405	
406	
407	
408	
409	
410	
411	
412	
413	
414	
415	
416	
417	
418	
419	
420	
421	
422	
423	
424	
425	
426	
427	
428	
429	
430	
431	
432	
433	
434	
435	
436	
437	
438	
439	
440	
441	
442	
443	
444	
445	
446	
447	
448	
449	
450	
451	
452	
453	
454	
455	
456	
457	
458	
459	
460	
461	
462	
463	
464	
465	
466	
467	
468	
469	
470	
471	
472	
473	
474	
475	
476	
477	
478	
479	
480	
481	
482	
483	
484	
485	
486	
487	
488	
489	
490	
491	
492	
493	
494	
495	
496	
497	
498	
499	
500	
501	
502	
503	
504	
505	
506	
507	
508	
509	
510	
511	
512	
513	
514	
515	
516	
517	
518	
519	
520	
521	
522	
523	
524	
525	
526	
527	
528	
529	
530	
531	
532	
533	
534	
535	
536	
537	
538	
539	
540	
541	
542	
543	
544	
545	
546	
547	
548	
549	
550	
551	
552	
553	
554	
555	
556	
557	
558	
559	
560	
561	
562	
563	
564	
565	
566	
567	
568	
569	
570	
571	
572	
573	
574	
575	
576	
577	
578	
579	
580	
581	
582	
583	
584	
585	
586	
587	
588	
589	
590	
591	
592	
593	
594	
595	
596	
597	
598	
599	
600	
601	
602	
603	
604	
605	
606	
607	
608	
609	
610	
611	
612	
613	
614	
615	
616	
617	
618	
619	
620	
621	
622	
623	
624	
625	
626	
627	
628	
629	
630	
631	
632	
633	
634	
635	
636	
637	
638	
639	
640	
641	
642	
643	
644	
645	
646	
647	
648	
649	
650	
651	
652	
653	
654	
655	
656	
657	
658	
659	
660	
661	
662	
663	
664	
665	
666	
667	
668	
669	
670	
671	
672	
673	
674	
675	
676	
677	
678	
679	
680	
681	
682	
683	
684	
685	
686	
687	
688	
689	
690	
691	
692	
693	
694	
695	

# Condições de corrida

- Ocorre quando os processos cooperantes precisam compartilhar um recurso, que ambos podem ler e gravar:

Diretório do spooler

4	abc
5	prog.c
6	prog.n
7	prova.txt
8	:

Como o diretório de spooler não está inconsistente, pois a variável `in` está corretamente apontando para a próxima entrada livre, o daemon de impressão não vai notar o que ocorreu, e o arquivo do processo B nunca será impresso.

Variáveis de controle

`out = 4`

`in = 8`



## Condições de corrida

- ➡ No caso anterior, vimos um exemplo do que é chamado de uma **condição de corrida**:
  - Ocorre quando dois ou mais processos acessam uma recurso compartilhado (em geral, na memória ou em algum arquivo).
  - O resultado das execuções dos processos dependem da ordem em que os processos executaram no processador.
  - Somente algumas ordens de execuções do processos não geram os resultados corretos.
- ➡ A detecção da existência de uma condição de corrida em um conjunto de processos cooperantes é complicada.
- ➡ Para solucionar o problema da condição de corrida, deveremos usar o conceito **exclusão mútua** através do uso de **seções críticas**.

# Seções críticas



Devemos usar o conceito de **exclusão mútua**:

- Impede que dois processos acessem o recurso compartilhado ao mesmo tempo.
- Um processo somente pode usar o recurso compartilhado após o outro processo acabar de usá-lo.

Diretório do spooler	
	:
	:
4	abc
5	prog.c
6	prog.n
7	cederj.txt
	:
	:

Variáveis de controle	
	out = 4
	in = 8

O processo B acessa o recurso compartilhado, e coloca o seu arquivo na Entrada 7 do diretório.

Diretório do spooler	
	:
4	abc
5	prog.c
6	prog.n
7	cederj.txt
8	prova.txt
	:
	:

Variáveis de controle	
	out = 4
	in = 9

O processo A acessa o recurso compartilhado, e coloca o seu arquivo na Entrada 8 do diretório, somente após o processo B usar o recurso.

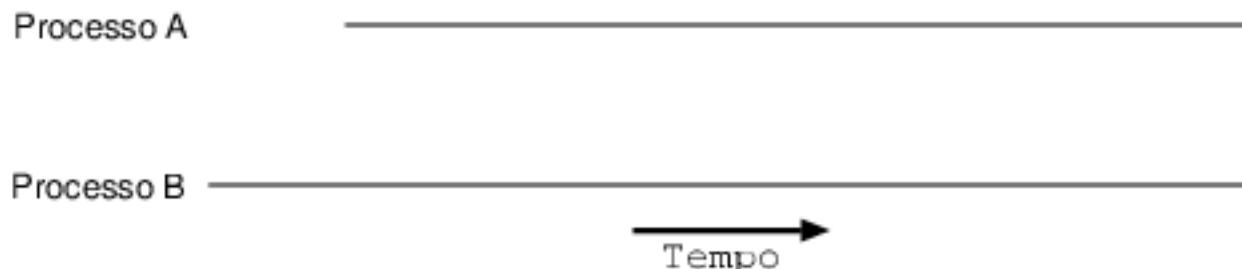
Exemplo anterior executado corretamente, sem a ocorrência de uma condição de corrida, devido à garantia de exclusão mútua.

## Seções críticas

- ➡ Podemos também ver o problema da condição de corrida a partir da seguinte visão abstrata:
  - Ou o processo está executando um código que não acessa nenhum dos recursos compartilhados com outros processos.
  - Ou o processo está executando um código que lê e/ou altera um dos recursos compartilhados com outros processos.
- ➡ O código do processo que acessa um recurso compartilhado com outros processos é chamado de **seção crítica** ou de **região crítica**.
- ➡ No exemplo anterior, as seções críticas dos processos A e B são aquelas que acessam o diretório, e as variáveis de controle.
- ➡ As condições de corrida serão evitadas somente se cada processo executar a sua seção crítica em um intervalo de tempo diferente.

## Seções críticas

- Para que a cooperação entre os processos seja eficiente, além de correta, deveremos garantir as seguintes condições:
- Dois ou mais processos não podem executar simultaneamente as suas seções críticas.
  - Não podemos fazer nenhuma suposição sobre a velocidade de execução ou do número de processadores.
  - Um processo que não está executando a sua seção crítica não pode bloquear um outro processo.
  - Um processo deve sempre ser capaz de executar a sua seção crítica em um intervalo de tempo finito.



O usuário iniciou a execução de dois processos que compartilham um recurso. Vamos supor que o computador possui dois processadores, o que permite os processos A e B executarem em paralelo.

## Seções críticas

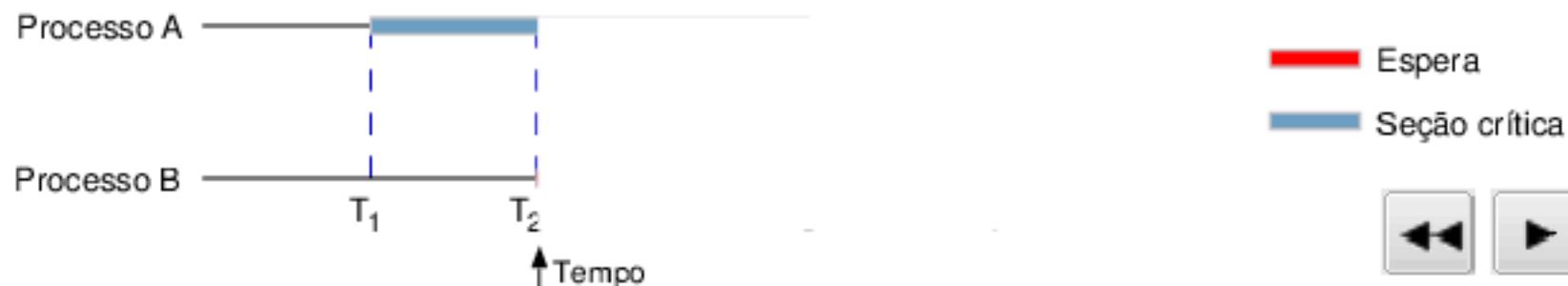
- Para que a cooperação entre os processos seja eficiente, além de correta, deveremos garantir as seguintes condições:
- Dois ou mais processos não podem executar simultaneamente as suas seções críticas.
  - Não podemos fazer nenhuma suposição sobre a velocidade de execução ou do número de processadores.
  - Um processo que não está executando a sua seção crítica não pode bloquear um outro processo.
  - Um processo deve sempre ser capaz de executar a sua seção crítica em um intervalo de tempo finito.



Os processos A e B continuam as suas execuções em paralelo. Após algum tempo (no tempo  $T_1$ ), o processo A entra na sua seção crítica, para acessar o recurso compartilhado.

## Seções críticas

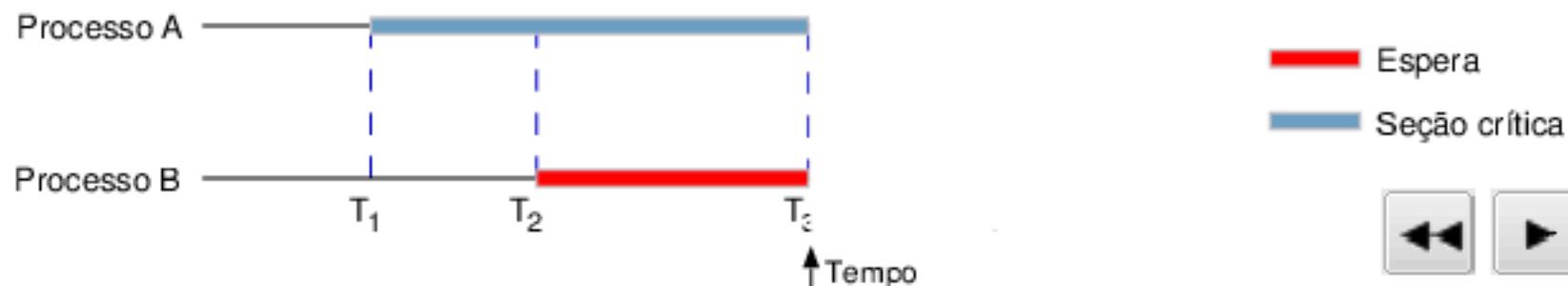
- Para que a cooperação entre os processos seja eficiente, além de correta, deveremos garantir as seguintes condições:
- Dois ou mais processos não podem executar simultaneamente as suas seções críticas.
  - Não podemos fazer nenhuma suposição sobre a velocidade de execução ou do número de processadores.
  - Um processo que não está executando a sua seção crítica não pode bloquear um outro processo.
  - Um processo deve sempre ser capaz de executar a sua seção crítica em um intervalo de tempo finito.



Os processos A e B continuam as suas execuções, até B decidir que também vai usar a sua seção crítica (no tempo T2). Como A ainda está na sua seção crítica, B não poderá entrar na sua seção crítica.

## Seções críticas

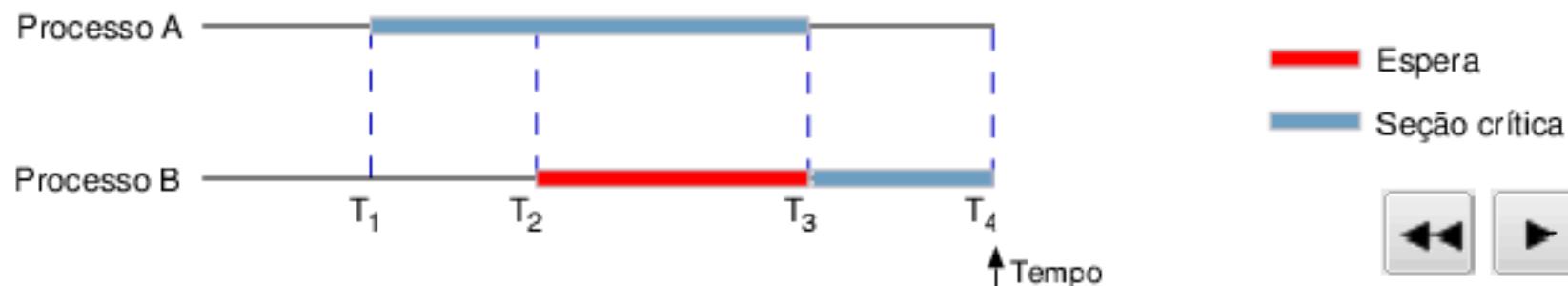
- Para que a cooperação entre os processos seja eficiente, além de correta, deveremos garantir as seguintes condições:
- Dois ou mais processos não podem executar simultaneamente as suas seções críticas.
  - Não podemos fazer nenhuma suposição sobre a velocidade de execução ou do número de processadores.
  - Um processo que não está executando a sua seção crítica não pode bloquear um outro processo.
  - Um processo deve sempre ser capaz de executar a sua seção crítica em um intervalo de tempo finito.



O processo A ainda estava executando a seção crítica, e o processo B continuava esperando, até A saiu seção crítica (no tempo  $T_3$ ). Agora, B pode finalmente entrar na sua seção crítica, e acessar o recurso.

## Seções críticas

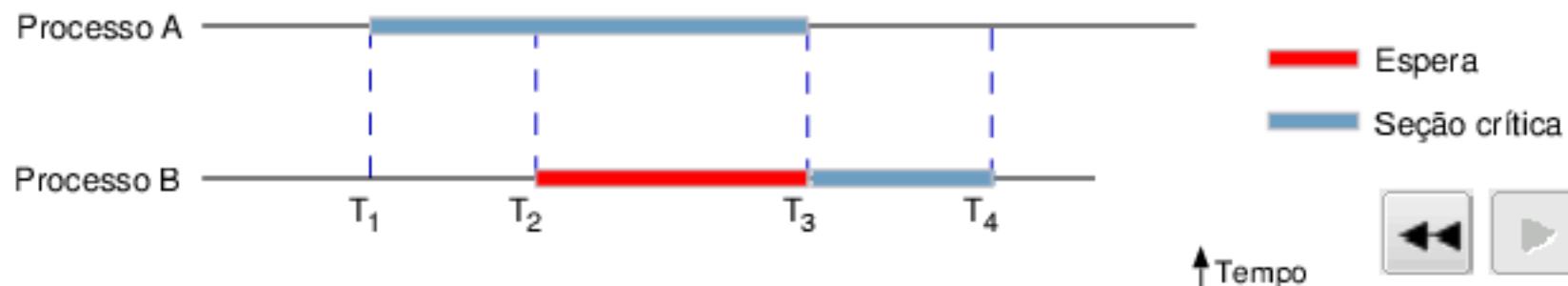
- Para que a cooperação entre os processos seja eficiente, além de correta, deveremos garantir as seguintes condições:
- Dois ou mais processos não podem executar simultaneamente as suas seções críticas.
  - Não podemos fazer nenhuma suposição sobre a velocidade de execução ou do número de processadores.
  - Um processo que não está executando a sua seção crítica não pode bloquear um outro processo.
  - Um processo deve sempre ser capaz de executar a sua seção crítica em um intervalo de tempo finito.



O processo B ainda estava executando a seção crítica, e o processo A executava algum outro código, até que o B saiu da sua seção crítica (no tempo T4). Agora, A e B não estão em suas seções críticas.

## Seções críticas

- Para que a cooperação entre os processos seja eficiente, além de correta, deveremos garantir as seguintes condições:
- Dois ou mais processos não podem executar simultaneamente as suas seções críticas.
  - Não podemos fazer nenhuma suposição sobre a velocidade de execução ou do número de processadores.
  - Um processo que não está executando a sua seção crítica não pode bloquear um outro processo.
  - Um processo deve sempre ser capaz de executar a sua seção crítica em um intervalo de tempo finito.



Os processos A e B continuam a executar os códigos não relacionados às suas seções críticas por algum tempo, até que ambos terminam a sua execução, sendo que B termina antes de A.

## Exclusão mútua

- ➡ Para que somente um processo possa entrar na sua seção crítica, deveremos usar as primitivas de exclusão mútua.
- ➡ Um meio de obtermos exclusão mútua é através do uso das propostas baseadas em **espera ocupada**:
  - O processo fica esperando até que possa finalmente acessar a sua seção crítica.
  - A grande desvantagem deste método (que foi usado no exemplo anterior) pode ser o desperdício de tempo do processador.
  - Vamos estudar seguintes propostas:
    - Desabilitando as interrupções.
    - A instrução TSL.

## Desabilitando as interrupções

- ➡ O processo desabilita as interrupções ao entrar na seção crítica, e as reabilita ao sair da seção crítica.
- ➡ Como as interrupções estão desabilitadas, o escalonador não será capaz de parar a execução do processo.
- ➡ A proposta não é adequada, pois se o processo não reabilitar as interrupções, o sistema irá parar de funcionar.
- ➡ A proposta não funciona se existe mais de um processador, pois o processo somente afeta o processador que o está executando.
- ➡ A conclusão é que esta proposta não é adequada para a exclusão mútua, devendo somente ser usada dentro do núcleo do sistema.

## A instrução TSL

- ➡ Comum no hardware dos computadores projetados para possuírem vários processadores (multiprocessadores).
- ➡ A instrução, cujo formato geral é TSL RX, **LOCK**, onde RX é um registrador e **LOCK** uma posição da memória:
  - Lê o conteúdo da posição **LOCK**, e o coloca no registrador RX.
  - Depois grava, na posição **LOCK**, um valor diferente de zero.
- ➡ A execução da instrução é **atômica**, ou seja, **indivisível**:
  - O processador bloqueia o barramento da memória até executar as duas operações.
  - Nenhum outro processador então poderá acessar a posição **LOCK**, até que o barramento seja desbloqueado.
- ➡ Como a operação de ler a memória e armazenar um valor não nulo é atômica, a exclusão mútua é garantida.

# A instrução TSL

- A proposta é baseada no uso dos seguintes procedimentos em linguagem assembler:
- O recurso compartilhado é protegido por uma variável **LOCK**.
  - O processo deverá chamar **ENTER\_REGION** antes de usar o recurso, e deverá chamar **LEAVE\_REGION** após usá-lo.



```
ENTER_REGION:  
TSL REGISTER, LOCK  
CMP REGISTER, #0  
JNE ENTER_REGION  
RET
```

Procedimento em assembler usado para adquirir o acesso ao recurso compartilhado.

```
LEAVE_REGION:  
MOVE LOCK, #0  
RET
```

Procedimento em assembler usado para liberar o acesso ao recurso compartilhado.

## Exclusão mútua

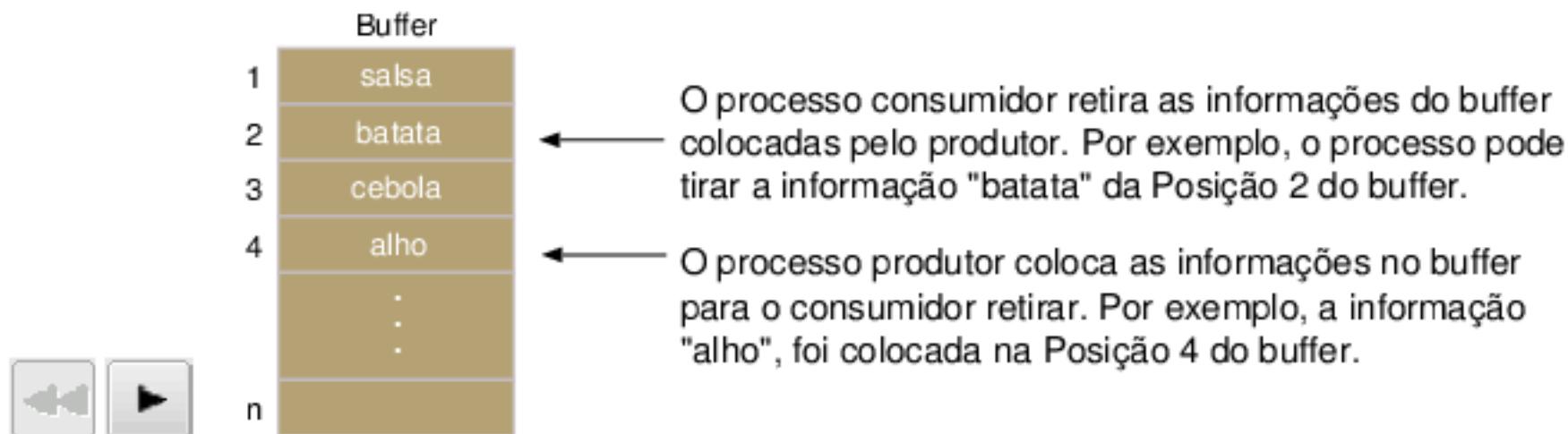
- ➡ Vamos agora estudar a seguinte proposta alternativa, que irá **bloquear o processo**, em vez de utilizar espera ocupada, se este não puder acessar a sua seção crítica:
  - Semáforos.
  
- ➡ Antes de estudarmos os semáforos, vamos descrever o problema do produtor e consumidor, usado para ilustrar nosso estudo desta proposta.

# Problema do produtor e consumidor



Neste problema dois processos, o produtor e o consumidor, compartilham um buffer:

- O processo produtor deposita novas informações no buffer, a não ser que este esteja cheio.
- O processo consumidor retira as informações do buffer, a não ser que este esteja vazio.



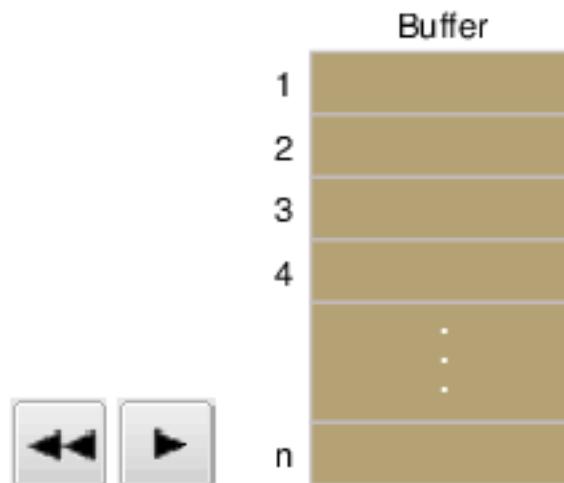
O problema pode ser facilmente estendido para  $m$  produtores e  $n$  consumidores.

# Problema do produtor e consumidor



Neste problema dois processos, o produtor e o consumidor, compartilham um buffer:

- O processo produtor deposita novas informações no buffer, a não ser que este esteja cheio.
- O processo consumidor retira as informações do buffer, a não ser que este esteja vazio.



Se o processo consumidor tentar retirar informações de um buffer vazio, o processo deverá ser bloqueado até que o processo produtor coloque as informações no buffer.



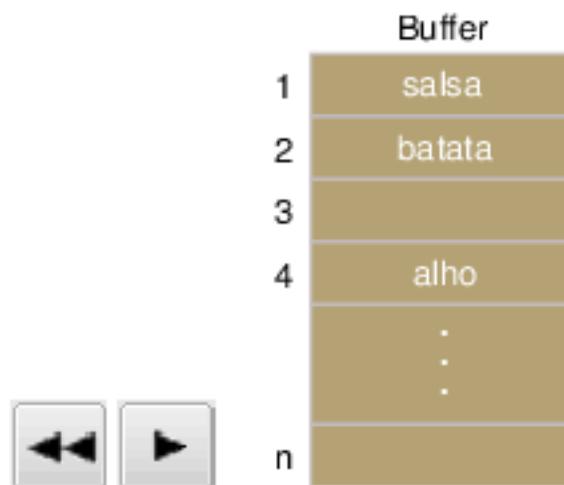
O problema pode ser facilmente estendido para  $m$  produtores e  $n$  consumidores.

# Problema do produtor e consumidor



Neste problema dois processos, o produtor e o consumidor, compartilham um buffer:

- O processo produtor deposita novas informações no buffer, a não ser que este esteja cheio.
- O processo consumidor retira as informações do buffer, a não ser que este esteja vazio.



O processo produtor gerou algumas informações e as colocou no buffer. Se o processo consumidor estiver bloqueado, este processo poderá ser desbloqueado, e retirar, por exemplo, a informação da Entrada 3.



O problema pode ser facilmente estendido para  $m$  produtores e  $n$  consumidores.

# Problema do produtor e consumidor



Neste problema dois processos, o produtor e o consumidor, compartilham um buffer:

- O processo produtor deposita novas informações no buffer, a não ser que este esteja cheio.
- O processo consumidor retira as informações do buffer, a não ser que este esteja vazio.

Buffer	
1	salsa
2	batata
3	couve
4	alho
	:
	:
n	cebola



Depois de o processo consumidor retirar a informação do buffer, o processo produtor criou três informações. Como existem somente duas entradas livres (a 3 e a n), o processo produtor é bloqueado ao inserir a terceira informação.



O problema pode ser facilmente estendido para  $m$  produtores e  $n$  consumidores.

# Problema do produtor e consumidor



Neste problema dois processos, o produtor e o consumidor, compartilham um buffer:

- O processo produtor deposita novas informações no buffer, a não ser que este esteja cheio.
- O processo consumidor retira as informações do buffer, a não ser que este esteja vazio.



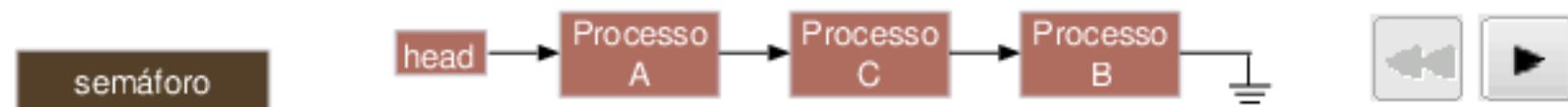
Depois de o processo consumidor retirar pelo menos uma informação do buffer, o processo produtor, que foi bloqueado, poderá finalmente executar, e colocará a terceira informação na Entrada 4 do buffer.



O problema pode ser facilmente estendido para  $m$  produtores e  $n$  consumidores.

# Semáforos

- Esta proposta usa uma variável inteira chamada de **semáforo**, e uma fila associada a esta variável:
  - O valor do semáforo indica quantas vezes ele pode ser decrementado sem que o processo seja bloqueado.
  - A fila contém os processos que foram bloqueados ao tentar decrementar o semáforo.
- São definidas duas operações sobre um semáforo: **P** e **V**:

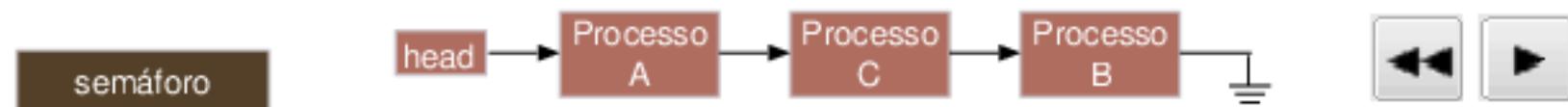


Fila associada ao semáforo, com os processos bloqueados.

Podemos executar duas operações sobre um semáforo: a **P** e a **V**.

# Semáforos

- Esta proposta usa uma variável inteira chamada de **semáforo**, e uma fila associada a esta variável:
  - O valor do semáforo indica quantas vezes ele pode ser decrementado sem que o processo seja bloqueado.
  - A fila contém os processos que foram bloqueados ao tentar decrementar o semáforo.
- São definidas duas operações sobre um semáforo: **P** e **V**:



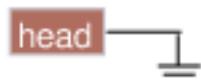
Fila associada ao semáforo, com os processos bloqueados.

A operação **P** inicialmente verifica o valor do semáforo. Se o valor for maior do que 0, então este valor será decrementado. Caso contrário, o processo que executou **P** será bloqueado, e colocado na fila.

# Semáforos

- Esta proposta usa uma variável inteira chamada de **semáforo**, e uma fila associada a esta variável:
  - O valor do semáforo indica quantas vezes ele pode ser decrementado sem que o processo seja bloqueado.
  - A fila contém os processos que foram bloqueados ao tentar decrementar o semáforo.
- São definidas duas operações sobre um semáforo: **P** e **V**:

semáforo = 0

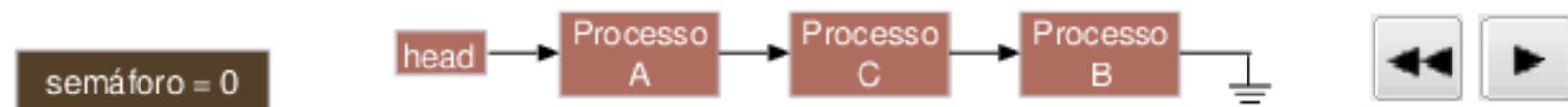


Fila associada ao semáforo, com os processos bloqueados.

Suponha que o valor do semáforo é 1. Se um processo executar a operação **P**, o valor do semáforo passará a ser 0, e o processo não será bloqueado.

# Semáforos

- Esta proposta usa uma variável inteira chamada de **semáforo**, e uma fila associada a esta variável:
  - O valor do semáforo indica quantas vezes ele pode ser decrementado sem que o processo seja bloqueado.
  - A fila contém os processos que foram bloqueados ao tentar decrementar o semáforo.
- São definidas duas operações sobre um semáforo: **P** e **V**:

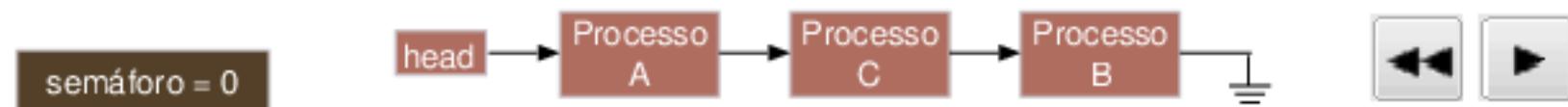


Fila associada ao semáforo, com os processos bloqueados.

Suponha que o processo A executou a operação **P**, e que o valor do semáforo é 0. Então A será bloqueado, e colocado na fila. Se, logo em seguida, dois processos, C e B, executarem a operação, então estes processos serão bloqueados, e colocados na fila.

# Semáforos

- Esta proposta usa uma variável inteira chamada de **semáforo**, e uma fila associada a esta variável:
  - O valor do semáforo indica quantas vezes ele pode ser decrementado sem que o processo seja bloqueado.
  - A fila contém os processos que foram bloqueados ao tentar decrementar o semáforo.
- São definidas duas operações sobre um semáforo: **P** e **V**:



Fila associada ao semáforo, com os processos bloqueados.

A operação **V** também verifica o valor do semáforo. Se o valor for 0, e a fila não estiver vazia, a operação escolhe um dos processos da fila, e o coloca no estado pronto. Caso contrário, a operação incrementa o valor do semáforo.

# Semáforos

- Esta proposta usa uma variável inteira chamada de **semáforo**, e uma fila associada a esta variável:
  - O valor do semáforo indica quantas vezes ele pode ser decrementado sem que o processo seja bloqueado.
  - A fila contém os processos que foram bloqueados ao tentar decrementar o semáforo.
- São definidas duas operações sobre um semáforo: **P** e **V**:



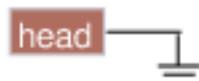
Fila associada ao semáforo, com os processos bloqueados.

Suponha que um processo executou a operação **V** sobre o semáforo. Como o valor do semáforo é 0 e a fila não está vazia, um processo, digamos o C, é escolhido e colocado no estado pronto. Se a operação for executada mais duas vezes, os processos A e B serão escolhidos.

# Semáforos

- Esta proposta usa uma variável inteira chamada de **semáforo**, e uma fila associada a esta variável:
  - O valor do semáforo indica quantas vezes ele pode ser decrementado sem que o processo seja bloqueado.
  - A fila contém os processos que foram bloqueados ao tentar decrementar o semáforo.
- São definidas duas operações sobre um semáforo: **P** e **V**:

semáforo = 1



Fila associada ao semáforo, com os processos bloqueados.

Se um outro processo executar uma operação **V** sobre o semáforo, o valor do semáforo será agora incrementado, pois o valor do semáforo é 0, e a fila está vazia.

# Semáforos

→ Vamos ver como os semáforos podem ser usados para resolver o problema do consumidor e do produtor:

```
procedure consumer;
var
  item: integer;
begin
  while (true) do
    begin
      P(full);
      P(mutex);
      item := remove_item;
      V(mutex);
      V(empty);
      consume_item(item);
    end;
end;
```

```
procedure producer;
var
  item: integer;
begin
  while (true) do
    begin
      item := produce_item;
      P(empty);
      P(mutex);
      insert_item(item);
      V(mutex);
      V(full);
    end;
end;
```

**empty**: semáforo que conta o número de entradas vazias no buffer. Inicializado com o tamanho do buffer.  
**full**: semáforo que conta o número de itens inseridos no buffer. Inicializado com o valor 0.  
**mutex**: semáforo usado para garantir o acesso exclusivo ao buffer. inicializado com 1.

Variáveis compartilhadas

O semáforo **full** garante que o processo consumidor será suspenso caso o buffer esteja vazio. O semáforo **empty** garante que o processo produtor seja suspenso se o buffer estiver cheio. Logo, estes semáforos garantem a correta sincronização entre os processos consumidor e produtor.

# Semáforos

- Para que a proposta com semáforos funcione corretamente, a execução das operações deve ser **atômica**:
  - Em geral os semáforos são implementados no núcleo, e as operações **P** e **V** são na verdade chamadas ao sistema.
  - Como são operações que usam poucas instruções, o sistema operacional pode desabilitar as interrupções ao executá-las.
  - Se o hardware tem múltiplos processadores, devemos usar a instrução TSL ao acessar o semáforo.
- Um semáforo que é inicializado com o valor 1, e que é usado para exclusão mútua, é chamado de um **semáforo binário**.
- Os semáforos com valores maiores do que 1 em geral são usados para a sincronização dos processos cooperantes.

# Semáforos

- O uso de semáforos simplifica o tratamento de interrupções:
  - Associamos um semáforo a cada um dos dispositivos, com o valor inicial igual a 0.
  - Quando o driver do dispositivo inicializa, executa uma operação **P** sobre o semáforo do dispositivo, e é bloqueado.
  - Ao dispositivo gerar uma interrupção, executamos a operação **V** sobre o seu semáforo, que irá desbloquear o driver.