

O problema com isso é que o processador com o S/O acaba se tornando ou ocioso ou o gargalo do sistema Operacional. A solução para isso é distribuir o Sistema Operacional pelas CPU's. Por exemplo, se um processo precisa acessar o disco, ele roda a parte do S/O do S/O Operacional que acessa o disco, aumentando a eficiência.

Exclusão mútua e Sincronização

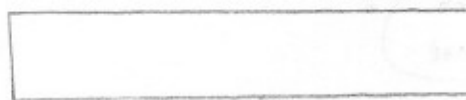
- Multiprogramação / Processos
- Multiprocessamento

Não existe interação explícita entre os processos. Eles sabem:

- 1) compartilhar recursos
- 2) cooperação, mas os processos não conhecem o identificador do outro
- 3) cooperação, onde os processos conhecem o identificador do outro processo

Concorrência / Compartilha por Recursos:

Por exemplo, dois processos independentes tentam imprimir em uma mesma impressora. A impressora recebe informações alternadamente e trabalha com os trabalhos de impressão, o que não é desejável. Portanto, usamos um Spooler de impressão que é um job que gerencia os trabalhos de impressão.



Spooler de impressão

Para um processo trabalhar com trabalhos de impressão ele deve colocar um job no spool. Temos:

```

struct spool {
    job *inicio;
    job *fim;
    t spool;
}

```

```

struct job {
    char *doc;
    job *next;
    t;
}

```

spool -> inicio = spool -> fim = NULL;

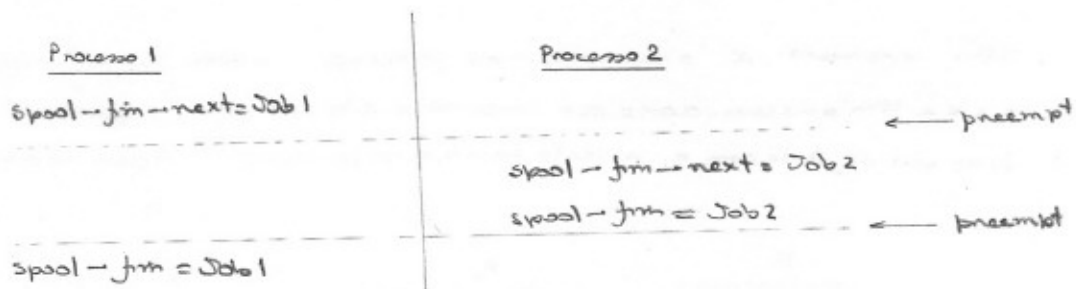
Para adicionar um job no spool devemos executar:

```

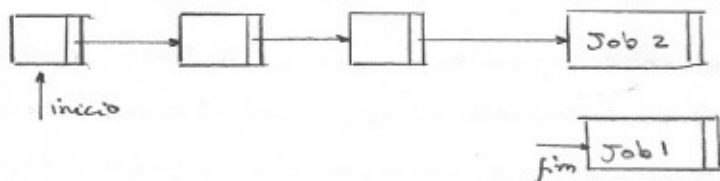
spool -> fim -> next = Job;
spool -> fim = Job;

```

O pior caso é dois processos quiserem ao mesmo tempo colocar um Job no file, e o preempt ocorre no meio das duas comandos. Suponha que existem dois processos e que a execução é como no diagrama abaixo:



As diferenças sequenciais de comandos, temos:



Isso corrige a estrutura do file. Isso sempre pode acontecer como podem ocorrer erros a partir de processos compartilhando pelos mesmos recursos.

Tanto a multiprogramação (paralelismo virtual) como o multiprocessamento (paralelismo real) geram os mesmos problemas.

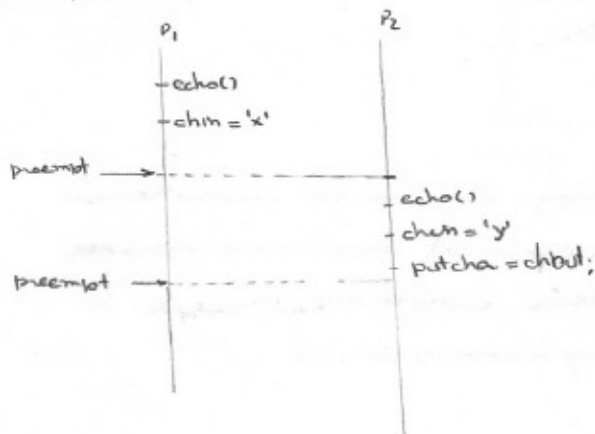
Suponha que dois processos P_1 e P_2 executam o código:

```

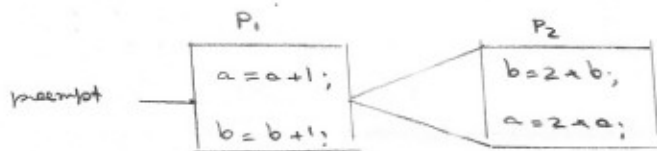
1 void echo {
2     charin.getcha();
3     charout = charin;
4     putcha = charout;
5 }
    
```

onde charin e charout são variáveis globais.

Se ocorre um preempt antes dos pontos 2 e 3, temos o seguinte problema:



Outro exemplo, é o seguinte: dois processos usam as variáveis a e b e têm a necessidade de manter a e b sempre iguais. Assim, tudo que é feito com a , é feito com b e vice-versa. Assim, temos:



Se ocorre um preempt entre os pontos $a = a + 1$, e $b = b + 1$, há uma quebra na coerência dos dados, ou seja, mudamos de $a = b$. Para evitar isso, a solução é realizar partes do código, ou seja, indicar que partes devem ser executadas de forma serial. Isso é o que chamamos de Região Crítica.