MAC0422 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 21 de Setembro de 2020

Roteiro

Semáforos

Condição de corrida

Escalonamento de processos

Semáforos

Resolvendo o problema da seção crítica com semáforos

- n processos repetidamente executam a seção crítica
- a seção crítica requer acesso exclusivo para algum recurso compartilhado
- na seção não crítica são manipuladas variáveis locais

Protocolo de acesso à seção crítica – com semáforos


```
sem mutex=1;

/* Cada thread roda: */
Thread() {
   while (true) {
      P(mutex);
      secao critica;
      V(mutex);
      secao nao critica;
   }
}
```

Sintaxe

- Precisamos de uma forma para definir expressões como atômicas
  - Usaremos < e >
- O código: <x++;> é transformado em uma ação atômica

Em algoritmos


```
P(s): <while (s==0) skip; s = s - 1;>
V(s): <s = s + 1;>
```

- Considere `s = 1`
  - Se dois processos tentarem rodar `P(s)` ao mesmo tempo
    - Apenas um vai conseguir
  - Se um processo tentar rodar `P(s)` e outro tentar rodar `V(s)`
    - Os dois vão conseguir (em ordem imprevisível)
    - `s` vai terminar igual a `1`

Regra importante ao implementar as operações

- Processos esperando em `P` são acordados na ordem em que eles executaram `P`

  Uma quantidade adequada de execuções de `V` por outros processos garantirá que processos esperando em `P` eventualmente prosseguirão

Em C, no Linux

- Semáforos genéricos: `semaphore.h`
- Tipo `sem_t`
- Funções `sem_init`, `sem_wait`, `sem_post` e `sem_destroy`

Em C, no Linux

- Semáforos binários: `pthread.h`
- **Obs.:** `apt-get install glibc-doc`
- Tipo `pthread_mutex_t`
- Funções `pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock` e `pthread_mutex_destroy`

Relembrando

- n processos repetidamente executam uma seção de código crítica e uma seção de código não crítica

```
Thread SC[i=1 to n] {
  while (true) {
     protocolo de entrada;
     secao critica;
     protocolo de saida;
     secao nao critica
  }
}
```

- Considerando que um processo que entra na seção crítica, sairá alguma hora
- Pode ser usado para resolver problemas de condição de corrida

Propriedades de uma solução para o problema

- Os protocolos precisam respeitar essas 4 propriedades:
  1. Exclusão mútua
  2. Ausência de deadlock (livelock)
  3. Ausência de espera desnecessária
  4. Entrada garantida

Condição de corrida

Quando acontece e o que é?

- Principalmente com processos que compartilham áreas para leitura e escrita (um arquivo por exemplo)
- O problema acontece quando a comutação da execução dos processos segue alguma ordem inesperada, ou seja, o programa só funcionará direito a depender da ordem com que os processos sejam escalonados pelo SO, o que não é desejável
- O termo "condição de corrida" vem do fato que os dois processos em execução remetem a uma corrida em que a depender da ordem de chegada o programa pode "perder" (Obs.: pode acontecer com threads também, como no exemplo dos produtores-consumidores que será mostrado algumas aulas adiante)

Exemplo de um sistema de impressão

- Quando um processo quer imprimir um arquivo ele coloca o nome deste arquivo em um diretório especial no sistema de arquivos do computador onde a impressora está conectada
- Um processo está em execução o tempo todo verificando o conteúdo deste diretório especial. Se há um arquivo lá, imprime e apaga o arquivo colocado para ser impresso
- Suponha que dentro do diretório há um conjunto de slots e dentro de cada um é possível escrever o nome de um arquivo a ser impresso
- Suponha que há uma variável global `out` apontando para o próximo arquivo a ser impresso e uma variável global `in` apontando para o próximo slot livre no diretório

Exemplo de um sistema de impressão

- Considere que os slots 0 a 3 estão vazios
- Considere que os slots 4 a 6 estão cheios (contém nomes de arquivos a serem impressos)
- Praticamente ao mesmo tempo dois processos querem enfileirar um arquivo para imprimir
- Pode acontecer uma condição de corrida nesse caso se:
  - Primeiro processo lê a variável `in` e armazena localmente o valor 7 como o próximo slot vazio. Nesse instante o SO interrompe a execução do primeiro processo e começa a executar o segundo
  - Segundo processo lê a variável `in` e armazena localmente o valor 7 como o próximo slot vazio. Ele continua sua execução, armazena o nome do arquivo a ser impresso dentro do slot 7 e atualiza `in` para 8.

Exemplo de um sistema de impressão

- Primeiro processo é selecionado pelo SO e volta a executar. Nesse ponto, ele põe o nome do arquivo a ser impresso dentro do slot 7 sobrescrevendo o valor que o Segundo processo tinha colocado e atualiza `in` para 8.
- Nesse caso o segundo processo não teria o arquivo impresso

Escalonamento de processos

Escalonador de processos

- Parte do SO responsável por escolher qual o próximo processo que será executado
- Necessário quando o computador passou a executar múltiplos processos (mesmo em casos onde não há múltiplos processadores)
- Tornou-se um problema sério principalmente com computadores pessoais e com a necessidade de interatividade (tem que rodar vários processos dando a impressão para o usuário que apenas o processo dele está executando)
- O escalonador precisa ser bem projetado para evitar que ele consuma muito recurso
- Ser justo com o escalonamento não é trivial. Depende do propósito do computador e dos processos

Justiça

- Difícil ter um algoritmo de escalonamento justo para todos os tipos de computadores

  Em clusters e grades (alto desempenho) o objetivo costuma ser vazão (número de *jobs* por unidade de tempo)

  Em computadores pessoais o objetivo costuma ser maior interatividade (um pouco de cada um dos processos por unidade de tempo)

  Em servidores o objetivo costuma ser atender requisições de usuários o mais rápido possível em detrimento de outros processos

Justiça

- Difícil ter um algoritmo de escalonamento justo para todos os tipos de processos
  - *CPU-bound* ou *CPU-intensive*
  - *IO-bound* ou *IO-intensive*

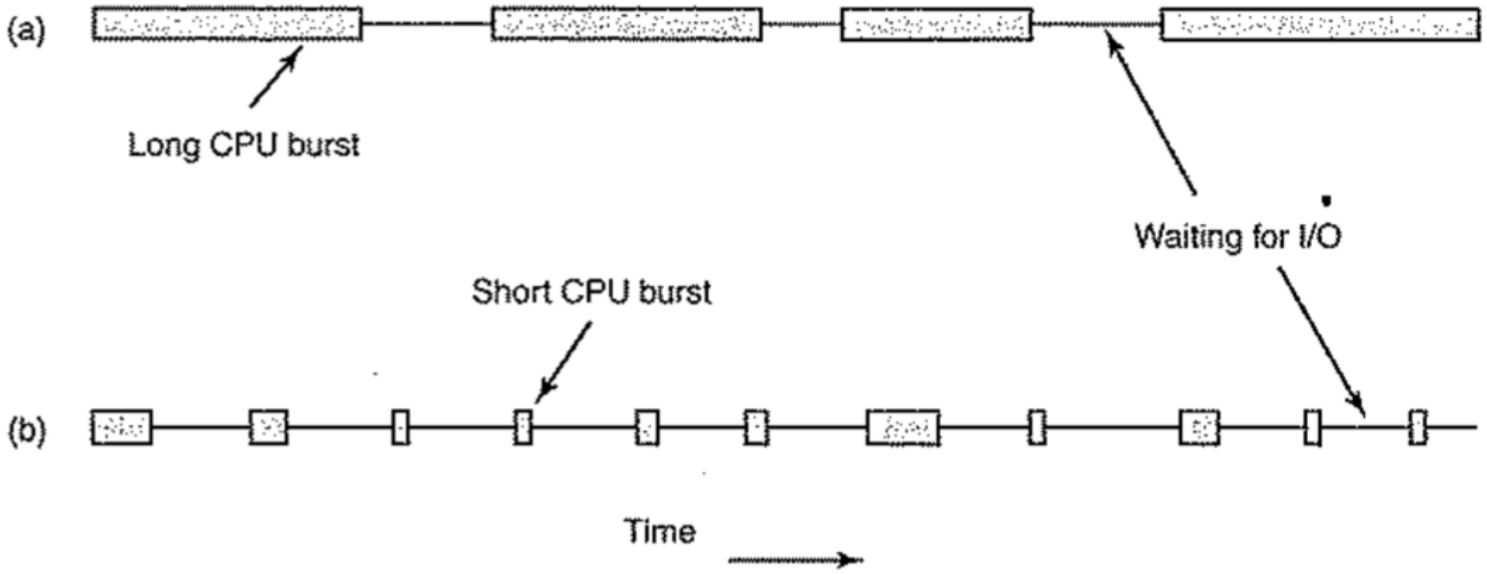

Figure 2-38. Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process.

Quando escalonar?

- Quando um processo é criado: rodar o pai ou o filho?
- Quando um processo é finalizado: se não há nenhum na fila de prontos?
- Quando um processo é bloqueado num semáforo: como saber qual vai permitir ele rodar?
- Quando uma operação de E/S é finalizada: para tudo para atender quem estava esperando essa operação?
- A cada pulso de clock? **Preemptivo** vs **Não preemptivo**

  Para isso funcionar é necessário uma interrupção de clock ao fim de cada pulso para dar controle da CPU de volta ao escalonador

  Sem interrupção de clock: só pode ter escalonador não preemptivo

FCFS (Em lote)

- First-Come First-Served
- Mais simples algoritmo de escalonamento
- Ordena os processos prontos em uma fila por ordem de chegada e executa nessa ordem
- Cada processo roda até o final ou até ele ser bloqueado (por semáforo ou E/S por exemplo). Não há preempção