

Sistemas Operacionais

Processos

1

Processos

● Definição:

- Conceito mais abrangente que o de *programa em execução*;
- Programa:
 - Entidade estática e permanente;
 - Um programa é um conjunto de instruções imutáveis.
- Processo:
 - Entidade dinâmica e efêmera;
 - O processo é um elemento ativo, que altera seu estado;
 - Identificado por um número único (PID).

2

Processos

- Um sistema multiprogramável simula um ambiente de monoprogramação para cada usuário;
- A cada troca, é necessário que o sistema preserve todas as informações da tarefa que foi interrompida;
- A estrutura responsável pela manutenção de todas as informações necessárias à execução de um programa, como conteúdo de registradores e espaço de memória, chama-se PROCESSO.

3

Processos

- **Conceito:**
 - O conceito **processo** pode ser definido como sendo o *ambiente onde um programa é executado*;
 - simular um ambiente de monoprogramação.
 - A maioria dos processos de um sistema executam programas do usuário. Entretanto alguns podem realizar tarefas do sistema.
 - Processos Daemon.

4

Processos

- Processos são...
- Criados:
 - Momento da execução;
 - Chamadas de sistema;
 - Associados a uma sessão de trabalho:
 - ex.: *login* + *senha* → *shell* (processo)

5

Processos

- Destruídos:
 - Situações Normais:
 - término da execução;
 - por outros processos.
 - Situações Anormais:
 - exceder tempo limite;
 - Falta de memória;
 - Erros de proteção:
 - Gravação em arquivo RO, acesso a end. de memória.
 - Erros aritméticos:
 - Divisão por zero, underflow, overflow.

6

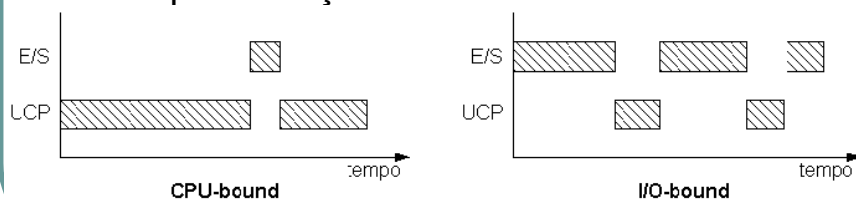
Processos

- Ciclos de um processo
 - Ciclo de processador:
 - tempo que ocupa a CPU.
 - Ciclo de E/S:
 - tempo em espera pela conclusão de um evento.
 - *Primeiro ciclo é sempre de processador*
 - Trocas de ciclos por:
 - Chamada de Sistema (CPU → E/S)
 - Interrupção (CPU → E/S ou E/S → CPU)
 - Ocorrência de evento (E/S → CPU)

7

Processos

- CPU Bound
 - ciclos de processador >> ciclos de E/S
- I/O Bound
 - Ciclos de E/S >> ciclos de processador
- Sem quantificação exata.



8

Processos

- O S.O. materializa o processo através de uma estrutura chamada de *Tabela de Processos* ou *Bloco de Controle de Processo* (BCP).
- É definido um ambiente com as informações necessárias à execução de um programa:
 - **Contexto de Hardware:**
Conteúdo de Registradores e área de memória;
 - **Contexto de Software:**
Número máximo de arquivos abertos e buffers de E/S.
- Informações variam de SO p/ SO:
 - Basicamente são informações pertinentes à gerência de:
 - *processo, memória e arquivos.*

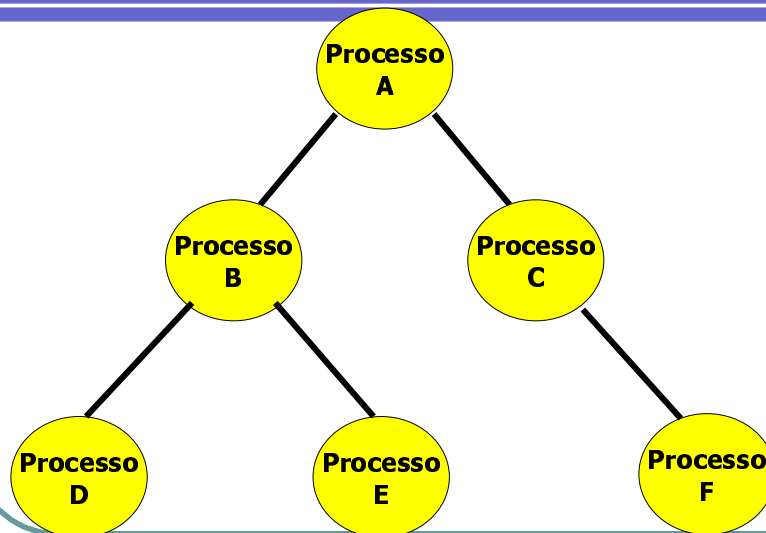
9

Processos

- Rotinas de sistema:
 - Processos e subprocessos.
(hierarquia de processos)
 - Processo criador é processo pai;
 - Processo criado é processo filho.
 - Representação através de uma árvore
 - Evolução dinâmica com o tempo.

10

Hierarquia de Processos



11

Processos

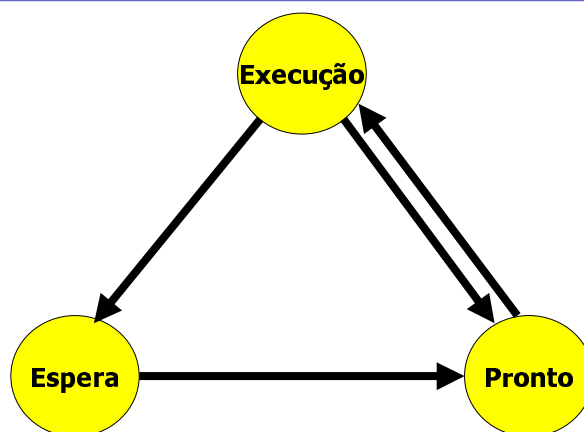
- **Mudança de Estado do Processo**
 - Um processo muda de estado diversas vezes
 - Eventos voluntários
 - Eventos involuntários
- **Estados de um processo:**
 - Execução (running);
 - Pronto (ready);
 - Espera (wait).
 - Bloqueado.

12

Processos

- Pronto → Execução: **escalonamento**;
- Execução → Espera: **evento voluntário**;
- Espera → Pronto: **não existe mudança do estado de espera para execução**;
- Execução → Pronto: **eventos involuntários**.

13



14

Processos

- Listas de Processos

- Vários processos no estado de pronto ou espera:
 - Lista de processos no estado de pronto;
 - Lista de processos no estado de espera.

15

Processos

- O SO possui dois estados de execução
 - Modo usuário:
 - certas instruções não podem ser executadas.
 - acessa apenas dados e código do próprio processo.
 - Modo Supervisor (protegido):
 - possibilita a execução de todas as instruções do processador;
 - modo de execução do SO.

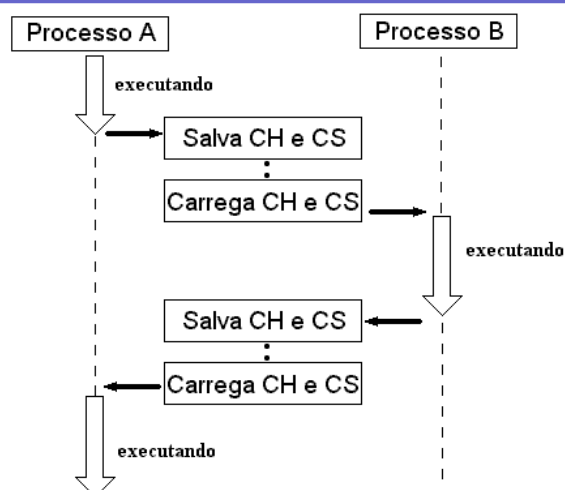
16

Processos

- Mudança de contexto:
 - troca de um processo por outro na CPU;
 - revezamento na utilização do processador;
 - Interrupção e posterior restauração transparentes.

17

Mudança de contexto



18

Escalonamento

- *Listas de Processos*
- O S.O. é responsável por determinar a ordem, pela qual os processos em estado de pronto devem ganhar a CPU.

19

Escalonamento de Processos

- Escalonador \Rightarrow parte do S.O. responsável pelo escalonamento de processos
- Algoritmo de escalonamento \Rightarrow algoritmo utilizado para a escolha do próximo processo a ser ativado

20

Escalonamento de Processos

- Critérios para um bom algoritmo de escalonamento
 - 1. Justiça: cada processo deve receber uma parcela justa de tempo da UCP;
 - 2. Eficiência: UCP deve estar 100% do tempo ocupada;
 - 3. Tempo de execução em batch: mínimo tempo esperando saídas;
 - 4. Taxa de execução (throughput): máximo número de trabalhos executados por hora;
 - 5. Tempo de resposta: mínimo de tempo para usuários interativos

21

Escalonamento de Processos

- Conflitos: por exemplo (3 e 5)
- Dificuldades: cada processo é único e não se pode (ou é difícil) prever quanto tempo de UCP será necessário
- Interrupção de Relógio: S.O. ganha o controle (execução) e decide manter ou trocar o processo em execução

22

Escalonamento de Processos

- Escalonamento com preempção (*preemptive scheduling*)
 - estratégia utilizada que permite um processo que é logicamente executável ser temporariamente suspenso
- Escalonamento com execução global (*run to completion*)
 - uma vez iniciada a execução de um processo, este é permitido utilizar a UCP até completar toda a sua tarefa

23

Técnicas de Escalonamento

- O escalonador de processos deve considerar as características dos processos:
 - CPU-Bound; I/O Bound; tempo real; tempo compartilhado, etc.
 - Preempção e não-preempção.

24

Técnicas de Escalonamento

- Ordem de Chegada (FIFO)
 - Simples de implementar: fila;
 - O primeiro a chegar será o primeiro a ser selecionado para execução.

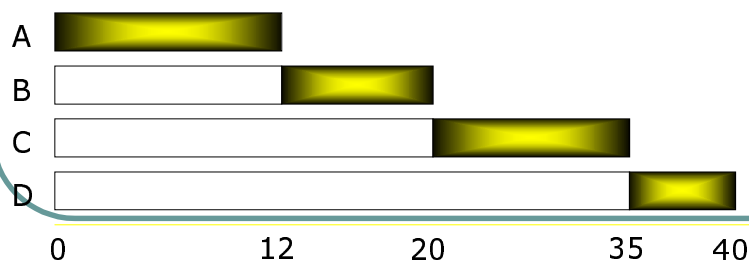
25

FIFO

- Exemplo:

Processo Duração (u.t.)

A	12
B	8
C	15
D	5



26

Técnicas de Escalonamento

- Escalonamento SJF
 - Não preemptivo
 - Seleciona o processo que necessite menos tempo de CPU

27

Shortest-Job-First

Exemplo:

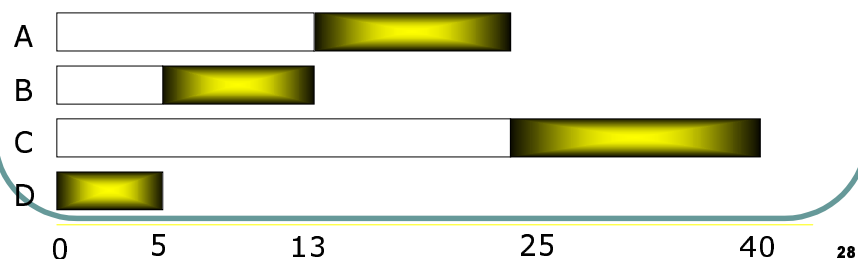
Processo Duração (u.t.)

A 12

B 8

C 15

D 5



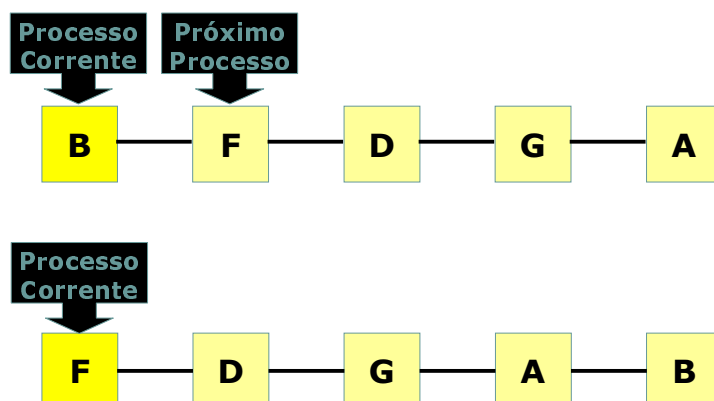
28

Round Robin

- Simples, antigo, justo e amplamente utilizado;
- Nenhum processo monopoliza a CPU;
- Existe um tempo limite para a utilização do processador (quantum);
- Trata da mesma forma todos os processos.
- Cada processo pode ser executado por um tempo pré-determinado \Rightarrow QUANTUM
- Preemptivo: um novo processo é colocado em execução

29

Escalonamento Circular



30

Escalonamento de Processos

- Eficiência \Rightarrow tamanho do quantum

Δt : Quantum

x: tempo para chaveamento de processos

Exemplo 1: $\Delta t = 20$ mseg

x = 5 mseg \Rightarrow 20% de tempo
de UCP é perdido

Exemplo 2: $\Delta t = 500$ mseg

x = 5 mseg \Rightarrow 1% de tempo
de UCP é perdido

- Qual é o novo problema?

31

Escalonamento por Prioridades

- Considera que alguns processos devem ser tratados de maneira diferente que outros;

Ex.: I/O-bound X CPU-bound

- Postergação indefinida:
 - Técnica de envelhecimento;
 - Processos que envelhecem sem executar, têm sua prioridade aumentada.

32

Escalonamento por Múltiplas Filas

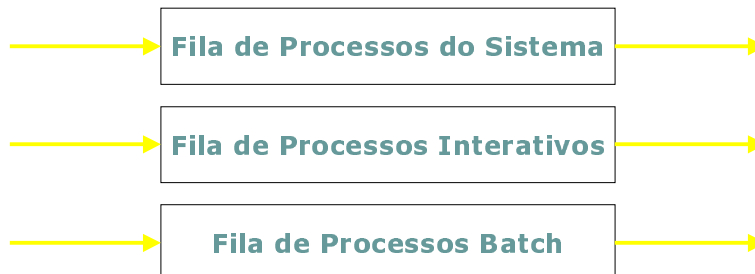
- Diversos processos do Sistema possuem características diferentes;
- Existe uma fila de processador para cada processo
 - FIFO, SJF, aleatória.
- Uso de prioridades para selecionar qual fila utilizará o processador.

33

Escalonamento por Múltiplas Filas

● Exemplo:

↑ Maior Prioridade



↓ Menor Prioridade

34

Outras Abordagens

- Escalonamento Garantido;
- Política Vs Mecanismo;
- Escalonamento de dois níveis.

35

Comunicação entre Processos

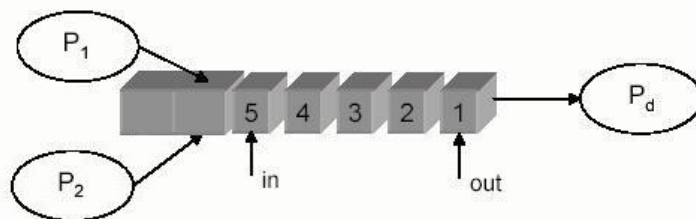
- A programação concorrente implica em um compartilhamento de recursos
 - Variáveis compartilhadas são recursos essenciais para a programação concorrente
- Acessos a recursos compartilhados devem ser feitos de forma a manter um estado coerente e correto do sistema
- Ex.: relação produtor-consumidor é uma situação bastante comum em SO.

36

Variáveis Compartilhadas

- Servidor de impressão:

- Processos usuários produzem “impressões”;
- Impressões são organizadas em uma fila a partir da qual um processo (consumidor) os lê e envia para a impressora.



37

Variáveis Compartilhadas

- Considerar a seguinte situação:

- P1 lê a variável *in* (5)
 - Interrupção de relógio para P1
- P2 lê a variável *in* (5) e atualiza para 6
- P1 executa novamente e atualiza *in* para 6 e escreve na posição já usada por P2

- Consequência:

- O diretório de impressão está consistente e o processo de impressão não percebe o erro
 - O processo P2 **NÃO** obtém a saída esperada

38

Condições de Corrida

- **RACE CONDITION** (disputa)
 - Situações onde dois ou mais processos estão acessando dados compartilhados e o resultado final do processamento depende da ordem em que os acessos são feitos.
- **Seção ou Região Crítica:**
 - Segmento de código no qual um processo realiza a alteração de um recurso compartilhado.
- **Solução**
 - diversos algoritmos e primitivas são propostas para resolver o problema

39

Condições de Corrida

- **Solução Completa**
 - DOIS ou mais processos NÃO podem estar SIMULTANEAMENTE dentro de suas S.C.;
 - NENHUMA hipótese sobre velocidade relativa entre processos ou número de UCPs pode ser feita;
 - NENHUM processo parado fora da S.C. pode bloquear qualquer outro processo;
 - NENHUM processo pode esperar indefinidamente para ser autorizado a entrar em sua S.C.

40

Implementação da Exclusão Mútua

- Interrupções Inibidas
 - Solução mais simples: Processo inibe TODAS as interrupções quando entra na Seção Crítica.
 - Interrupções de relógio não ocorrem:
 - não existe chaveamento de processo.
 - Boa técnica apenas para o SO: não serve como um mecanismo de E.M. para processos usuários.
- Variáveis do tipo Lock (Variáveis de Travamento)
 - Solução por software
 - Variável *lock* com valor
 - 0: S.C. livre
 - 1: S.C. ocupada
 - Problema: mesmo problema do *print spooler*

41

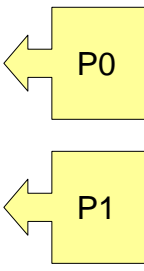
Implementação da Exclusão Mútua

- Estrita Alternância com Espera Ocupada
 - Variável **turn**: indica qual proc. pode entrar na SC

```
while (TRUE)
{while (turn != 0) /*espera*/;
 secao_critica();
 turn = 1;
 secao_ao_critica();}

-----

while (TRUE)
{while (turn != 1) /*espera*/;
 secao_critica();
 turn = 0;
 secao_ao_critica();}
```


 - **Problema:** dependendo da velocidade relativa de P0 e P1, é possível ter-se um processo bloqueado fora da SC sem chance de entrar (violação da regra 3)

42

Implementação da Exclusão Mútua

● Solução de Peterson

```
#define FALSE 0
#define TRUE 1
#define N 2
int turn; /* de quem é a vez */
int interested [N] /* valor inicial FALSE */
enter_region(int process) /* nº do processo: 0 ou 1 */
{
    int other; /* nº do outro processo */
    other = 1 - process; /* o oposto do processo */
    interested[process] = TRUE; /* demonstra o interesse */
    turn = process; /* flag */
    while (turn==process && interested[other]==TRUE) /*loop*/
}
leave_region (int process) /*Processo de quem está saindo*/
{
    interested[process] = FALSE; /*indica a saída da SC*/
}
```

43

Implementação da Exclusão Mútua

● Instrução **TSL** ou **Spin Lock**

- TSL → Test and Set Lock → *swap (reg, flag)*
- Instrução de máquina indivisível
- Operação:
 - Lê o conteúdo de uma posição de memória, escreve em um registro e atribui um valor não nulo à memória.

```
enter_region          leave_region
do {reg=1;            flag=0
    swap (reg, flag)
} while (reg==1)
```

44

Problemas e Soluções

● Problemas

- Busy waiting (espera ocupada)
- Confiar no processo (programador)
- Inversão de prioridades

● Solução

- Bloquear o processo ao invés de executar a espera ocupada;
- Baseado em duas novas primitivas:
 - sleep → bloqueia um processo a espera de um sinal;
 - wakeup → sinaliza um processo.

45

Problema do Produtor/Consumidor

```
produtor() {  
    while(TRUE) {  
        produz_item();  
        if (count==N) sleep();  
        coloca_item();  
        count = count + 1;  
        if (count==1) wakeup(consumidor);  
    }  
}  
consumidor() {  
    while(TRUE) {  
        if (count == 0) ↯ sleep();  
        remove_item();  
        count = count - 1;  
        if (count == N-1) wakeup(produtor);  
        consome_item();  
    }  
}
```

46

Semáforos

- Tipo abstrato de dado composto por um valor inteiro e uma fila de processos.
- Duas operações permitidas:
 - P (testar/down) e V (incrementar/up)

P(S):

S.valor=S.valor-1

Se S.valor<0

Então bloqueia o processo e insere em S.Fila

V(S):

S.valor=S.valor+1

Se S.valor<=0

Então retira processo P de S.fila e acorda P

47

Produtor/Consumidor com Semáforos

```
#define N 100
typedef int semaforo;
semaforo mutex = 1, empty = N, full = 0;
produtor(){int item;
    while (TRUE)
    {
        produce_item(&item);
        down (&empty); down (&mutex);
        enter_item(item);
        up(&mutex); up(&full);}
}
consumer(){
    int item;
    while (TRUE){
        down(&full); down(&mutex);
        remove_item(&item);
        up(&mutex); up(&empty);
        consume_item(&item);}
}
```

48

Monitores

- Mecanismos de alto nível formados por procedimentos, variáveis e estrutura de dados definidos em um módulo.
 - Implementação automática da E.M. entre seus procedimentos;
 - Realizada pelo compilador e não pelo programador.
- Raras linguagens implementam monitores: Concurrent Euclid, Pascal Concorrente.

49

Problemas Clássicos com IPC

- Problema dos Filósofos Famintos
 - Duas possibilidades: *pensar* ou *comer*,
 - para comer, deve-se apanhar o seu garfo e o do vizinho.
 - Ilustrar situações onde há competição por acesso exclusivo a recursos compartilhados.
 - Deadlock: um processo depende de um evento que jamais ocorrerá.
 - Starvation: Rodar indefinidamente sem progresso.
- Problema dos Leitores e Escritores
 - Ilustrar o modelo de acesso a Banco de Dados:
 - dois ou mais processos podem ler a mesma Base;
 - apenas um processo pode atualizar a Base.

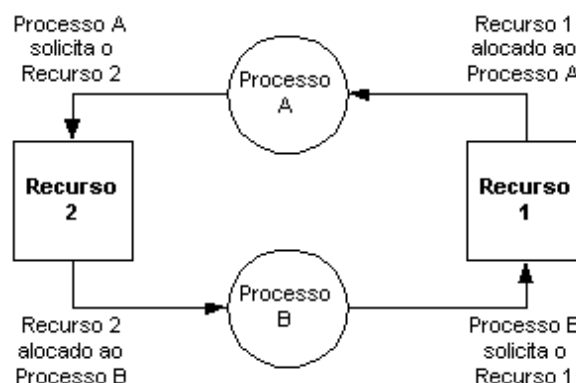
50

Deadlock

- Um processo é dito em **deadlock**, quando está esperando por um evento que nunca ocorrerá.
- Essa situação é consequência, na maioria das vezes, do compartilhamento de recursos do sistema entre vários processos:
 - cada processo tem acesso ao recurso de forma exclusiva.
- **Recursos:**
 - qualquer objeto que os processos podem adquirir:
 - Preemptivos: (pode ser tomado do processo sem prejuízo)
 - Não-Preemptivos: (não pode ser tomado do processo atual)
 - Sequência necessária para usar determinado recurso:
1. Requisitar → 2. Usar → 3. Liberar

51

- Condições necessárias para que ocorra **deadlock**:
 - 1 - Condição de exclusão mútua;
 - 2 - Condição de posse e de espera;
 - 3 - Condição de não-preempção;
 - 4 - Condição de Espera Circular: deve existir uma cadeia circular de dois ou mais processos.



Deadlock

- Estratégias para tratar *deadlocks*

- 1 - Ignorar completamente o problema:

- consideram a frequência em que eles ocorrem.

- 2 - Prevenção de deadlock:

- através da negação de alguma condição.

- 3 - Detecção e correção de deadlock:

- permite que os deadlocks aconteçam e agem após sua ocorrência.

- 4 - Evitar dinamicamente os deadlocks:

- através da alocação cuidadosa de recursos.

53

Deadlock

- Prevenção de deadlock:

- **Princípio de garantir que uma das quatro condições nunca se satisfaçam:**

- 1 - nenhum processo aguarda pela liberação de um recurso (mesmo que já esteja alocado). Ex.: Spool de impressão

Problemas: inconsistência entre processos concorrentes.

- 2 - sempre que um processo inicie sua execução ele requisita todos os recursos de uma vez. *Problemas:* desperdício de recursos (tarefas demoradas) e *starvation*.

- 3 - solução inviável: pode ocasionar vários problemas, até mesmo fazer o processo perder todo o trabalho realizado.

- 4 - processo deve liberar o recurso ao solicitar outro; **ou:** *num. global de recurso* (impossível encontrar a ordenação ideal)

54

Deadlock

- Detecção e correção de deadlock:
 - uso de estrutura de dados capaz de identificar processos/recursos e verificar se há a espera circular;
 - Quebra da espera circular:
 - preempção manual; eliminar processos; **rollback**.
- Evitar dinamicamente deadlock:
 - Definição de estado seguro/inseguro
 - **seguro**: o sistema pode garantir que todos os processos vão terminar;
 - **inseguro**: tal garantia não existe, embora não leve necessariamente a deadlock.
 - Cabe ao sistema decidir se alocará ou não o recurso.

55