

# Sistemas Operacionais de Redes

## Gerência de Processos



# Sistemas Operacionais de Redes

## Gerência de Processos

### Objetivos

- Apresentar os principais conceitos de gerência de processos, demonstrando o funcionamento do algoritmo escalonador

# Sistemas Operacionais de Redes

## Gerência de Processos

### Diferença entre um processo e um programa

- É sutil, porém crucial. O processo constitui-se de certo tipo de atividade. Ele possui um programa, uma entrada, uma saída e um estado. Já o programa é o código. O processo também pode ser visto com um programa em execução e sua dinâmica. É importante ressaltar que:
  - O mesmo programa sendo rodado por dois usuários gera dois processos;
  - Um programa pode gerar vários processos.

# Sistemas Operacionais de Redes

## Gerência de Processos

- Uma REDE DE COMPUTADORES é formada por um conjunto de módulos processadores capazes de trocar informações e compartilhar recursos, interligados por um sistema de comunicação.
- O sistema de comunicação vai se constituir de um arranjo **topológico**, interligando os vários módulos processadores através de enlaces físicos (meios de transmissão) e de um conjunto de regras com o fim de organizar a comunicação (protocolos).

# Sistemas Operacionais de Redes

## Gerência de Processos

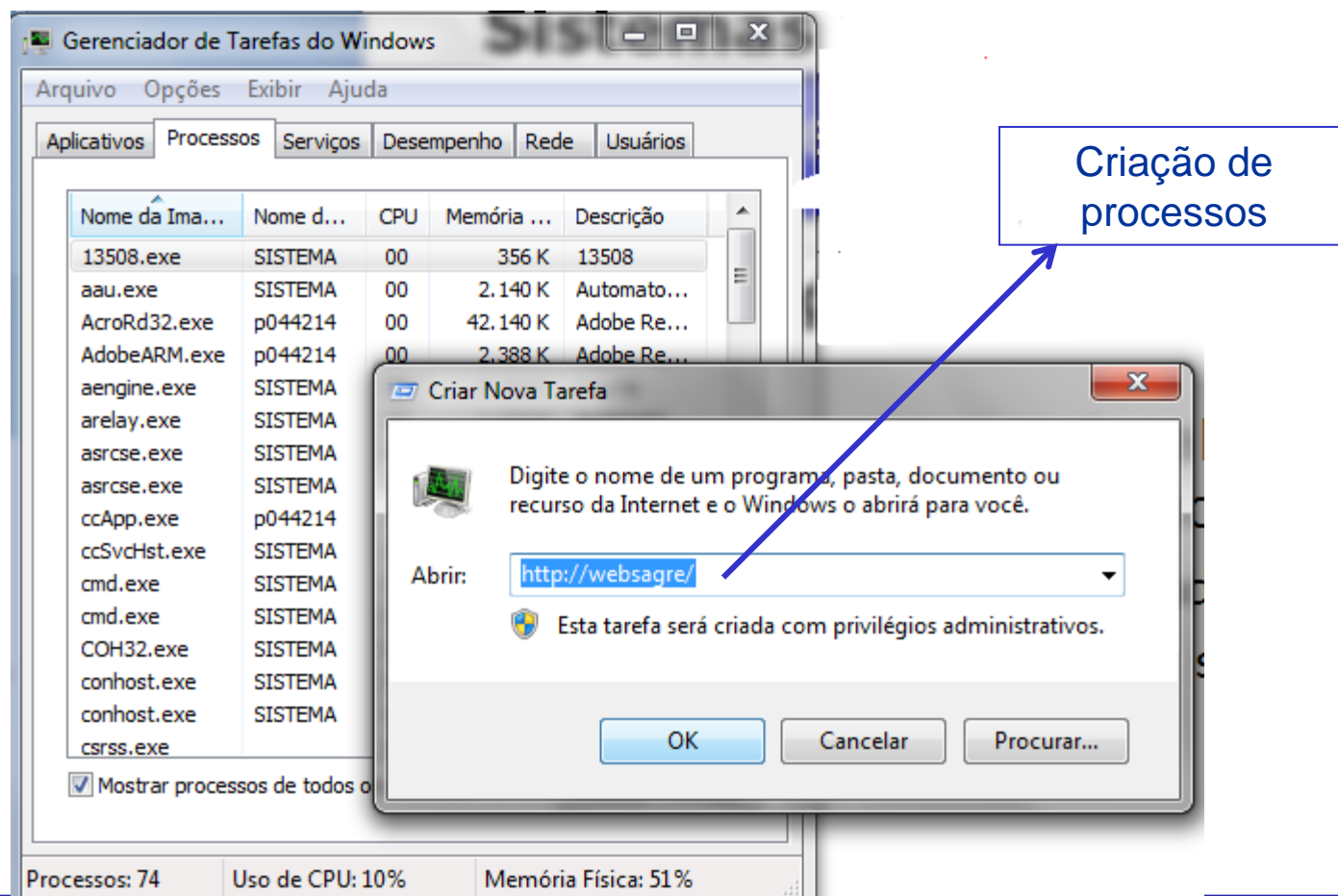
### Hierarquia de Processos

- Todo sistema operacional que suporta o conceito de processo precisa ter um mecanismo de criar e destruir processos.
- No Unix, processos são criados pela chamada de sistema "FORK" ou pelo shell;
- No Windows, processos são criados por meio do desktop ou pelo gerenciador de tarefas;

# Sistemas Operacionais de Redes

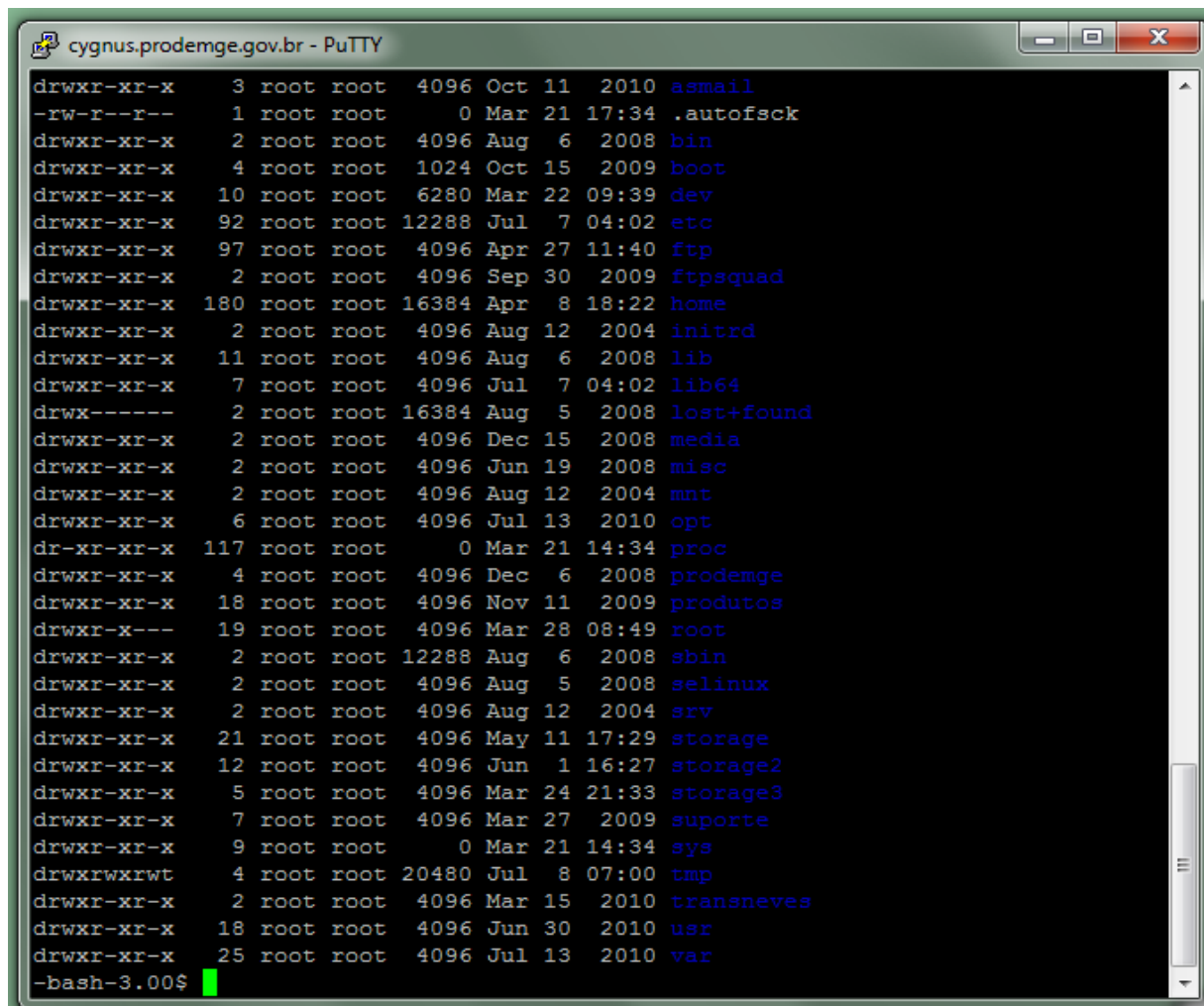
## Gerência de Processos

### Gerenciador de tarefas windows



# Sistemas Operacionais de Redes

## Gerência de Processos



```
cygnus.prodemge.gov.br - PuTTY
drwxr-xr-x 3 root root 4096 Oct 11 2010 asmail
-rw-r--r-- 1 root root 0 Mar 21 17:34 .autofsck
drwxr-xr-x 2 root root 4096 Aug 6 2008 bin
drwxr-xr-x 4 root root 1024 Oct 15 2009 boot
drwxr-xr-x 10 root root 6280 Mar 22 09:39 dev
drwxr-xr-x 92 root root 12288 Jul 7 04:02 etc
drwxr-xr-x 97 root root 4096 Apr 27 11:40 ftp
drwxr-xr-x 2 root root 4096 Sep 30 2009 ftpsquad
drwxr-xr-x 180 root root 16384 Apr 8 18:22 home
drwxr-xr-x 2 root root 4096 Aug 12 2004 initrd
drwxr-xr-x 11 root root 4096 Aug 6 2008 lib
drwxr-xr-x 7 root root 4096 Jul 7 04:02 lib64
drwx----- 2 root root 16384 Aug 5 2008 lost+found
drwxr-xr-x 2 root root 4096 Dec 15 2008 media
drwxr-xr-x 2 root root 4096 Jun 19 2008 misc
drwxr-xr-x 2 root root 4096 Aug 12 2004 mnt
drwxr-xr-x 6 root root 4096 Jul 13 2010 opt
dr-xr-xr-x 117 root root 0 Mar 21 14:34 proc
drwxr-xr-x 4 root root 4096 Dec 6 2008 prodemge
drwxr-xr-x 18 root root 4096 Nov 11 2009 produtos
drwxr-x--- 19 root root 4096 Mar 28 08:49 root
drwxr-xr-x 2 root root 12288 Aug 6 2008/sbin
drwxr-xr-x 2 root root 4096 Aug 5 2008 selinux
drwxr-xr-x 2 root root 4096 Aug 12 2004 srv
drwxr-xr-x 21 root root 4096 May 11 17:29 storage
drwxr-xr-x 12 root root 4096 Jun 1 16:27 storage2
drwxr-xr-x 5 root root 4096 Mar 24 21:33 storage3
drwxr-xr-x 7 root root 4096 Mar 27 2009 suporte
drwxr-xr-x 9 root root 0 Mar 21 14:34 sys
drwxrwxrwt 4 root root 20480 Jul 8 07:00 tmp
drwxr-xr-x 2 root root 4096 Mar 15 2010 transneves
drwxr-xr-x 18 root root 4096 Jun 30 2010 usr
drwxr-xr-x 25 root root 4096 Jul 13 2010 var
-bash-3.00$
```

Bash shell do  
Linux

# Sistemas Operacionais de Redes

## Gerência de Processos

### Estados do Processo

- Cada processo é uma entidade independente (possui seu próprio fluxo de controle) e possui um estado. Os possíveis estados de um processo, segundo Tanenbaum (1995) são **rodando, pronto e esperando**;
- Os processos também interagem entre si (saída de um serve de entrada para outro), mostrando a possibilidade da intercomunicação entre processos.
  - Ex: `cat Arq1, Arq2, Arq3 | grep empresa`



# Sistemas Operacionais de Redes

## Gerência de Processos

- Neste caso, se o processo "grep" estiver pronto, antes do processo "cat" terminar, o que ocorrerá?
  - O processo "grep" irá ficar no estado de bloqueado até que o processo "cat" termine a sua execução, pois a entrada de um é exatamente a saída do outro.
- O bloqueio de processos pode ocorrer também por força do sistema operacional, caso o S.O decida passar o processador para outro processo por um intervalo de tempo. A isto damos o nome de processamento "preemptivo".
- O que é processamento Preemptivo e Cooperativo?

# Sistemas Operacionais de Redes

## Gerência de Processos

### Diagrama de Estados do Processo

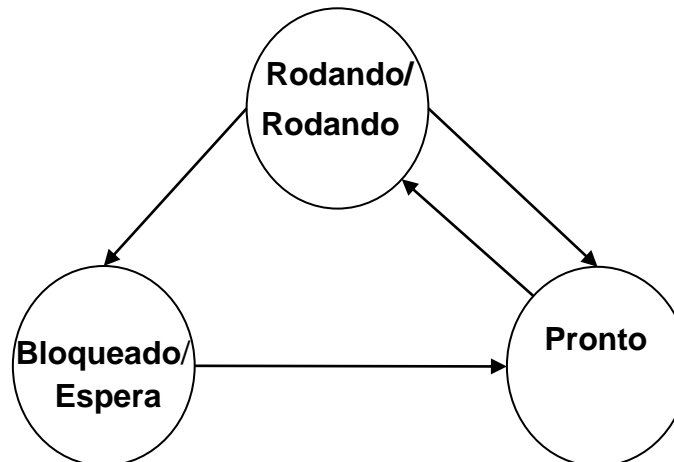


# Sistemas Operacionais de Redes

## Gerência de Processos

### Possíveis Estados de um processo

- **Execução/rodando:** usando o processador neste instante.
- **Pronto:** em condições de rodar, mas bloqueado temporariamente para dar vez a outro processo;
- **Espera/Bloqueado:** impedido de rodar até que ocorra um evento externo ao processo. (Caso do Exemplo) não pode rodar nem se o processador estiver disponível.



# Sistemas Operacionais de Redes

## Gerência de Processos

### Exemplo de processos

- Programas que executam comandos de usuários;
- Partes do S.O (trata de tarefa como manipulação de arquivos ou gerência de transferência de informações do disco para fita);
- Normalmente os processos do S.O ficam "bloqueados" esperando ser "invocados" através de interrupções ou chamadas de sistemas.

Ex: quando um comando é digitado num terminal, o processo (interpretador de comandos) sai do estado de bloqueado e passa para o estado de rodando.

# Sistemas Operacionais de Redes

## Gerência de Processos

### Implementação de Processos

- Segundo Tanenbaum (1995, p.23) o sistema operacional mantém uma tabela de processos com uma entrada por processo.
- Esta entrada contém informações sobre o estado do processo, sobre a memória alocada, os valores de seu contador de programa e de seu apontador de pilha, o estado de seus arquivos abertos, sua contabilidade no uso de recursos, sua prioridade, e tudo o mais que for necessário guardar quando o processo passar do estado *rodando* para o *estado pronto*, de maneira que seja possível reiniciar seu processamento mais tarde, como se nada tivesse acontecido.
- As informações contidas na tabela de processos variam de sistema operacional para sistema operacional, mas sempre algumas dizem respeito à gerência de processo, outras à gerência de memória, e outras ao sistema de arquivos.

# Sistemas Operacionais de Redes

## Gerência de Processos

Figura 1 – Alguns campos da tabela de processo do UNIX

Fonte: Tanenbaum (1995, p.23)

Valor dos registradores	Ponteiro para o segmento de texto	Máscara UMASK
Valor do contador de programa	Ponteiro para o segmento de dados	Diretório raiz
Valor da palavra de estado (PSW)	Ponteiro para o segmento bss	Diretório de trabalho
Valor do apontador da pilha	Estado da saída	Descritores de
Estado do processo	Estado do sinal	arquivo
Instante de início de processo	Identificação do processo	Identificação efetiva
Tempo de processador utilizado	Processo-pai	use
Tempo de processador do processo-filho	Grupo de processos	Identificação efetiva
Tempo de ocorrência do próximo alarme	Identificação real do usuário	grp
Ponteiros para as filas de mensagens	Identificação efetiva do usuário	Parâmetros de
Bits de sinal pendentes	Identificação real do grupo de usuário	chamadas
Diversos bits de flag	Identificação efetiva do grupo de usuários	Diversos bits de flag
	Mapas de bits para sinais	
	Diversos bits de flag	

# Sistemas Operacionais de Redes

## Gerência de Processos

Gerenciador de Tarefas do Windows

Arquivo Opções Exibir Ajuda

Aplicativos Processos Serviços Desempenho Rede Usuários

Nome da Imagem	PID	Nome de Usuário	Identificação de Sessão	CPU	Tempo de CPU	Reserva de Memória Paginada	Falhas de Página	Prior. Básica	Thres
Tempo Ocioso...	0	SISTEMA	0	86	169:13:18	0 K	0	N/A	
System	4	SISTEMA	0	01	00:29:50	0 K	43.729	Normal	
smss.exe	228	SISTEMA	0	00	00:00:00	8 K	320	Normal	
aengine.exe	260	SISTEMA	0	01	01:03:57	156 K	2.570.517	Normal	
csrss.exe	332	0	0	00	00:07:42	151 K	69.830	Alta	
arelay.exe	340	SISTEMA	0	00	00:00:00	47 K	9.335	Normal	
csrss.exe	392	1	0	00	00:00:32	381 K	211.461	Alta	
wininit.exe	400	SISTEMA	0	00	00:00:00	71 K	1.437	Alta	
LMS.exe	440	SISTEMA	0	00	00:00:00	52 K	957	Normal	
services.exe	452	SISTEMA	0	00	00:06:51	95 K	119.510	Normal	
winlogon.exe	476	SISTEMA	1	00	00:00:00	88 K	3.736	Alta	
lsass.exe	504	SISTEMA	0	00	00:12:35	75 K	12.619	Normal	
lsn.exe	516	SISTEMA	0	00	00:00:03	24 K	4.288	Normal	
svchost.exe	620	SISTEMA	0	00	00:01:02	60 K	51.522	Normal	
nvsvs.exe	684	SISTEMA	0	00	00:00:00	56 K	941	Normal	
explorer.exe	696	p044214	1	00	00:01:52	405 K	2.086.492	Normal	
gbpsv.exe	708	SISTEMA	0	00	00:00:00	30 K	754	Normal	
svchost.exe	772	SERVIÇO DE REDE	0	00	00:00:12	67 K	22.744	Normal	
SCHTASK.EXE	780	p044214	1	00	00:00:08	113 K	1.567	Normal	
PeIService.exe	844	SISTEMA	0	00	00:00:00	29 K	1.319	Normal	
svchost.exe	880	SERVIÇO LOCAL	0	00	00:02:13	91 K	75.854	Normal	
svchost.exe	912	SISTEMA	0	00	00:15:43	137 K	881.864	Normal	
svchost.exe	940	SISTEMA	0	00	00:01:19	191 K	1.720.415	Normal	
ccApp.exe	1084	p044214	1	00	00:00:31	122 K	3.578.306	Normal	
svchost.exe	1120	SERVIÇO LOCAL	0	00	00:00:04	66 K	20.273	Normal	
nvsvs.exe	1148	SISTEMA	1	00	00:00:00	148 K	2.967	Normal	
PWMDBSVC.exe	1208	SISTEMA	0	00	00:00:00	55 K	10.672	Normal	
conhost.exe	1216	SISTEMA	1	00	00:00:00	62 K	932	Normal	
cmd.exe	1224	SISTEMA	1	00	00:00:00	28 K	601	Normal	
Smc.exe	1284	SISTEMA	0	01	00:26:59	183 K	4.099.859	Normal	
PeIElvDm.exe	1312	SISTEMA	1	00	00:00:00	107 K	1.145	Normal	
svchost.exe	1352	SERVIÇO DE REDE	0	00	00:01:27	86 K	166.647	Normal	
ccSvcHst.exe	1460	SISTEMA	0	00	00:10:21	109 K	4.844.661	Normal	
jusched.exe	1472	p044214	1	00	00:00:00	116 K	1.809	Normal	
spoolsv.exe	1676	SISTEMA	0	00	00:00:19	138 K	118.336	Normal	
taskhost.exe	1696	p044214	1	00	00:00:01	103 K	5.131	Normal	

# Sistemas Operacionais de Redes

## Gerência de Processos

Como ocorreria uma interrupção de processo gerada por uma operação de E/S?

Existe um processo de usuário rodando quando acontece uma interrupção de disco. Então:

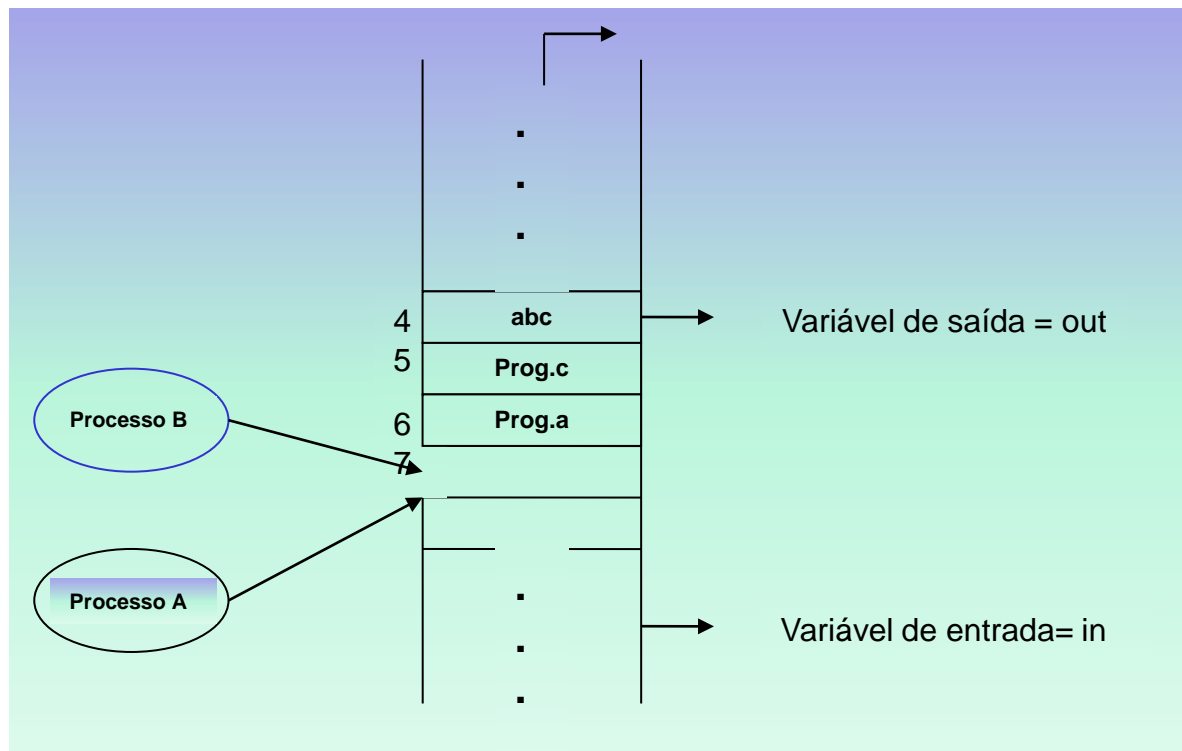
- Os valores dos PC (programa *counter*), da *psw* (*program status word*) e alguns outros registradores são levados para a pilha pelo hardware responsável pela instrução;
- Este hardware carrega no "PC" o valor contido no endereço especificado pelo vetor de interrupção de disco; O hardware só faz isso, o resto é por conta do software;
- Em seguida a **rotina de serviço de interrupção** salva o valor de todos os registradores na entrada da tabela de processos, correspondente ao processo corrente;
- O número do processo corrente e um ponteiro para sua entrada na tabela de processos são mantidos como variáveis globais do sistema, para possibilitar acesso rápido;
- Então as informações colocadas na pilha pela interrupção são removidas e o apontador de pilha é ajustado para apontar para uma pilha temporária, utilizada pelo processo que serve a interrupção;
- Ações como o salvamento de registradores e a colocação de valores no apontador de pilha não podem ser expressas em C (pequeno código de linguagem de máquina, que chama um procedimento em C, quando termina). Quando esta rotina termina, ela chama um programa "C" para tratar a interrupção;
- O programa "C" identifica os processos, que requisitaram o trabalho de disco que conseqüentemente gerou a interrupção. Este processo provavelmente estará "bloqueado" e deve ter seu estado mudado para "rodando" pelo escalonador, que vai escolher qual processo rodará.



# Sistemas Operacionais de Redes

## Gerência de Processos

**Condição de corrida:** Problema de sobreposição de arquivo a serem impressos no diretório de "Spool" do Unix.



# Sistemas Operacionais de Redes

## Gerência de Processos

- A questão da condição de corrida é: Como resolver este problema? Ou seja, como evitar condições de corrida?
- A solução para tal problema necessariamente passa pela exclusão mútua de execução.
- **Exclusão mútua de execução** é uma forma de impedir que outros processos usem ou acessem uma variável ou um arquivo compartilhado quando um determinado processo já o estiver usando, ou seja, impedir que dois programas rodem suas **regiões críticas** ao mesmo tempo.
- **Região critica** é a parte do programa, cujo processamento pode levar à ocorrência de condição de corrida.

# Sistemas Operacionais de Redes

## Gerência de Processos

### Exclusão mútua com espera ocupada

- Quando um programa estiver acessando a “memória compartilhada” dentro de sua região crítica, nenhum outro processo poderá entrar em sua seção crítica de acesso à memória (seção correspondente).
- **Formas de implementar a exclusão mútua (pg. 26 e 26)???**
  - Inibição de interrupções (O programa habilita interrupção de hardware);
  - Variáveis de tratamento (única variável setada para 1=recurso ocupado e 0=recurso liberado);
  - Estrita alternância (algoritmo que testa variável em loop);
  - Solução de Peterson(usa função que vez de variável ( enter region(), leave region()));
  - Instrução TSL(Test and set locked) implementada em hardware.

# Sistemas Operacionais de Redes

## Gerência de Processos

### **Bloqueio e desbloqueio de processos: Primitivas SLEEP/WAKEUP**

- problema da prioridade invertida
  - Imagine que um processo está rodando sua região crítica, então ele é escalonado em determinado momento assumindo o estado de pronto. Entretanto o escalonador sempre dá maior prioridade a outro processo, que está em espera ocupada por querer usar o recurso que o primeiro processo está usando, por estar rondando sua região crítica.
  - Diante deste fato, o primeiro processo nunca terá a chance de sair da sua região crítica e o segundo processo, que tem maior prioridade, também nunca terá a chance de sair do loop de espera ocupada em que se encontra.

# Sistemas Operacionais de Redes

## Gerência de Processos

- Surge então a idéia das primitivas SLEEP E WAKEUP.
- Essas duas primitivas fazem parte de uma solução que tem como objetivo evitar o problema da prioridade invertida e funcionam da seguinte forma:
  - **SLEEP** – é uma chamada de sistema que bloqueia o processo que a chamou, ou seja, suspende tal processo, até que outro o acorde;
  - **WAKEUP** – possui um parâmetro, o processo a ser acordado. Usa esta técnica em vez da espera ocupada.

### Fragilidades deste problema

- Apesar de simples pode ocorrer “condição de corrida”;
- Pode levar à condição de corrida igual à situação do diretório de Spool;
- Usa uma variável “*count*” para controlar o número de itens no buffer;
- Tanto o processo “consumidor” quanto o “produtor” devem testar variável antes de colocar ou retirar um valor do buffer.

# Sistemas Operacionais de Redes

## Gerência de Processos

### O Problema do Produtor X Consumidor

- Imagine que haja dois processos, um que produz resultados/saídas e outro que os consome. Imagine que haja um buffer limitado para armazenar o que o processo produtor produz, de forma que o processo consumidor precisa agir e retirar algo deste buffer para que o produtor continue a produzir. Então o seu funcionamento se consiste em:
  - O produtor produz e coloca informação no buffer e o consumidor a tira, mas o buffer é limitado e o produtor só poderá colocar mais informação no buffer se o consumidor já tiver tirado alguma informação e houver espaço no buffer e vice e versa.

# Sistemas Operacionais de Redes

## Gerência de Processos

<pre> 1 #include "prototypes.h" 2 #define N 100 3 Int count=0; 4 Void producer (void) { 5     Int item; 6     While (TRUE) { 7         Producer_item(&amp;item) ; 8         IF (count==N) 9             Sleep( ); 10        Enter_item(item); 11        Count++; 12        IF (count==1) 13            wakeup(consumer); 14    } 15} </pre>	<pre> 1 #include "prototypes.h" 2 #define N 100 3 Int count=0; 4 Void Consumer (void) { 5     Int item; 6     While (TRUE) { 7         IF (count==0) 8             Sleep(); 9         remove_item(&amp;item); 10        count-- 11        IF (count==N - 1) 12            wakeup(producer); 13        consume_item(item); 14    } 15} </pre>
<p>a) Se count for igual a 1 (linha 12) significa que o buffer não está cheio, mas já tem informação a ser consumida e precisa chamar o consumidor.</p>	<p>a) o buffer só irá encher se o escalonador mudar de processo quando o count -1(linha 10) for ser processado;</p> <p>b) se a linha 11 for verdadeira, significa que o buffer já está vazio e é preciso acordar o processo produtor.</p>

# Sistemas Operacionais de Redes

## Gerência de Processos

### Problema e falha da solução SLEEP/WAKEUP

- A condição de corrida pode ocorrer por causa do acesso irrestrito à variável "count";
- Vamos ver como isso acontece: imagine que o buffer está vazio e o processo consumidor acabou de ler a variável "count" e o seu valor é igual a 0 (zero). Neste momento houve um escalonamento e o processo consumidor assumiu o estado de pronto e o processo "produtor" começa a rodar, colocando um item no *buffer* e incrementando a variável *count*, cujo valor passa para 1(um).
- Considerando que o valor de *count*, para o processo consumidor é 0 (zero), o processo produtor assume que o processo consumidor está dormindo (linha 12) e envia uma primitiva "wakeup" para acordá-la.



# Sistemas Operacionais de Redes

## Gerência de Processos

### Problema e falha da solução SLEEP/WAKEUP

- Infelizmente o processo consumidor não está no estado de “esperando” e sim “pronto”. **Esta primitiva irá se perder**, pois somente processos no estado de “esperando” podem se acordados;
- Quando o processo “consumidor” sair do estado de pronto e voltar a rodar, ele verificará que o valor de “count” é 0 (zero) (linha 7) e irá para o estado de “esperando”. Neste caso o processo consumidor irá produzir até encher o buffer e entrará em estado de “esperando”.
- **Conclui-se que mais cedo ou mais tarde os dois processos estarão no estado de “esperando”.**

# Sistemas Operacionais de Redes

## Gerência de Processos

### Semáforos

- O problema da condição de corrida só foi totalmente resolvido após o surgimento dos semáforos no mundo dos Sistemas Operacionais. Os semáforos são variáveis do tipo binárias, que armazenam número de sinais para o futuro. Estes sinais podem assumir o valor 0 ou um valor positivo.
- Basicamente usa duas operações em seu funcionamento: D o w n e U P.
  - A operação "D o w n" verifica se o valor do semáforo é maior que 0 (zero). Se sim, então seu valor é decrementado e o processo continua sua execução; senão o processo que executou a operação "D o w n" assume o estado de esperando (dormindo);
  - A operação "UP" incrementa o valor do semáforo e acorda um eventual processo que esteja dormindo naquele semáforo para evitar a "condição de corrida"

Obs : Os procedimentos: verificar semáforo, o processo executar a operação "down" e a colocação do processo para dormir (estado de esperando) são partes de uma **AÇÃO ATÔMICA** (indivisível, que é essencial para evitar a condição de corrida)

# Sistemas Operacionais de Redes

## Gerência de Processos

- Para explicitar como funcionam os semáforos iremos abordar o problema do **Produtor X Consumidor usando semáforos**.
  - OBS: No caso dos semáforos o problema de perda de sinal observado na solução Sleep e Wakeup não ocorre devido à atomicidade. Com a utilização dos semáforos o Sistema Operacional não tem autonomia para escalonar processo durante as ações descritas acima. Não uma variável “count” de acesso irrestrito.
- Para se resolver o problema do produtor X consumidor serão usados 03 (três) semáforos;
  - Full – serve para contar o número de posições que já foram preenchidas no buffer;
  - Empty – serve para contar o número de posições que ainda estão vazias no buffer;
  - Mutex – Vale inicialmente 1 (um). (garante que só um processo entrará em sua região crítica). São os semáforos binários.
- **Premissas básicas:** Se cada processo executar um “Down” sobre o semáforo binário antes de entrar em sua região crítica e um “UP” logo que sair, a “exclusão mútua destas regiões críticas estará garantida.

# Sistemas Operacionais de Redes

## Gerência de Processos

### Solução do problema do Consumdor X **Produtos** com uso de Semáforos

```
# Include "prontotypes.h";
# Define n 100;
typedef int semaphore;
Semaphore mutex =1; /*controla acesso a região crítica
Semaphore empty = n; /* conta as posições vazias do Buffer
Semaphore full = 0; /* conta as posições ocupadas de Buffer
```

#### Void producer (void)

```
{
int Item;
While(true)
{
    Producer_item(&item);
    Down(&empty);
    Down(&mutex); // entra na região crítica
    Enter_ item(item); // coloca item no buffer
    Up(&mutex); //deixa região crítica, incrementa mutex
    Up(&full); // incrementa cont. de posições ocupadas.
}
}
```

```
# Include "prontotypes.h";
# Define n 100;
typedef int semaphore;
Semaphore mutex =1; /*controla acesso a região crítica
Semaphore empty = n; /* conta as posições vazias do Buffer
Semaphore full = 0; /* conta as posições ocupadas de Buffer
```

#### Void consumer (void)

```
{
Int item;
While (true)
{
    // loop
    Down (&full); // decrem. o cont. de pos. ocupadas
    Down (&mutex); // entra na região crítica
    Remove_item(&item); //retira item do buffer
    Up(&mutex) ; // deixa a região critica
    Up(&empty) ; // increm. contador de pos. vazias
    Cosumer_item(item); //faz algo com o item retirado.
}
}
```

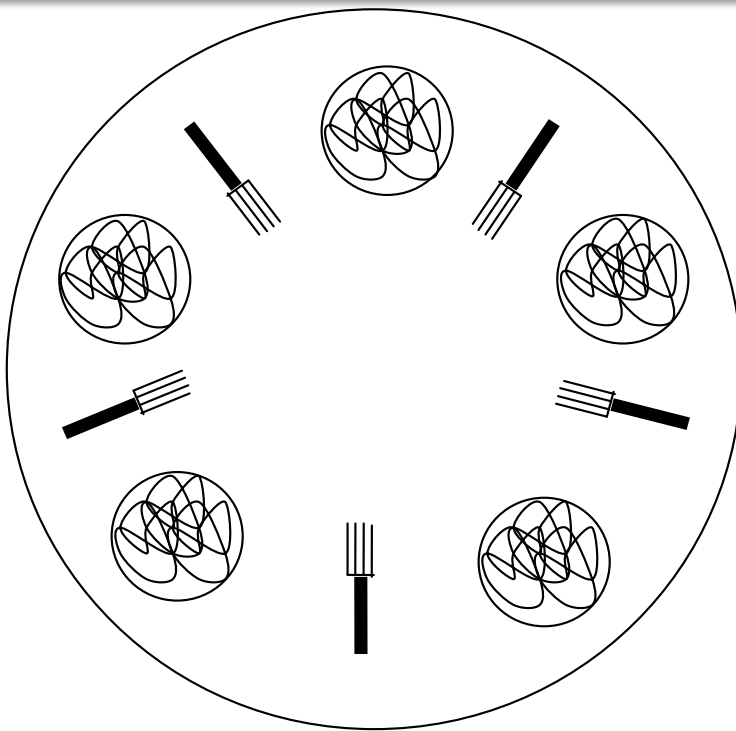
# Sistemas Operacionais de Redes

## Gerência de Processos

- **Problemas clássicos de comunicação entre processos**
  - O problema do filósofo Glutão;
  - O problema do barbeiro dorminhoco;
  - O problema dos leitores e escritores.
  
- **Problema do filósofo Glutão**
  - Cinco filósofos estão sentados em torno da mesa redonda.
  - Cada filósofo tem um prato de macarrão na sua frente.
  - O macarrão é muito escorregadio e preciso de dois garfos.
  - Entre cada prato existe um garfo.

# Sistemas Operacionais de Redes

## Gerência de Processos



### Premissas básicas:

- a) O filósofo somente come e pensa (neste caso);
  - a) Quando o filósofo tem fome ele tenta pegar o garfo da direita e o da esquerda, um por vez, em qualquer ordem;
  - c) Se tiver sucesso, pega-se os garfos come-se e solta-os.
- Como fazer um Algoritmo que simule a ação de cada filósofo sem provocar um Deadlock.

### Situação do “Deadlock”

- Todos os filósofos resolvem comer ao mesmo tempo;
- Todos pegam o garfo da esquerda;
- Todos morrem de fome;

# Sistemas Operacionais de Redes

## Gerência de Processos

### Escalonamento de processos

- É a parte do sistema operacional que decide entre dois processos que estão em estado de pronto, qual vai rodar. Usa um algoritmo chamado Algoritmo escalonador.

### Algoritmo de escalonamento.

- Como o escalonador decide quem deve rodar? Existem vários critérios que podemos pensar:
  - Justiça - garante chances iguais a todos os processos do S.O;
  - Eficiência - manter o processador ocupado 100% do tempo;
  - Tempo de resposta - minimizar tempo de resposta para usuários interativos;
  - Turnaround - minimizar tempo que usuários batch devem esperar pela saída;
  - Throughput - Maximizar o número de *jobs* processados na unidade de tempo, usualmente este tempo é de uma hora.

# Sistemas Operacionais de Redes

## Gerência de Processos

### Escalonamento de processos

- Como assegurar que um processo não vai rodar sozinho, por muito tempo, prejudicando todos os outros (processamento cooperativo)?
- Os computadores modernos possuem um relógio interno que grava, periodicamente, um sinal de interrupção (interrupção de tempo). A cada interrupção de tempo o Sistema Operacional é posto para rodar e decide se o processo que está rodando continua ou se sai do processador, dando lugar a outro processo. Esta estratégia é chamada de **escalonamento preemptivo** ou **processamento preemptivo**



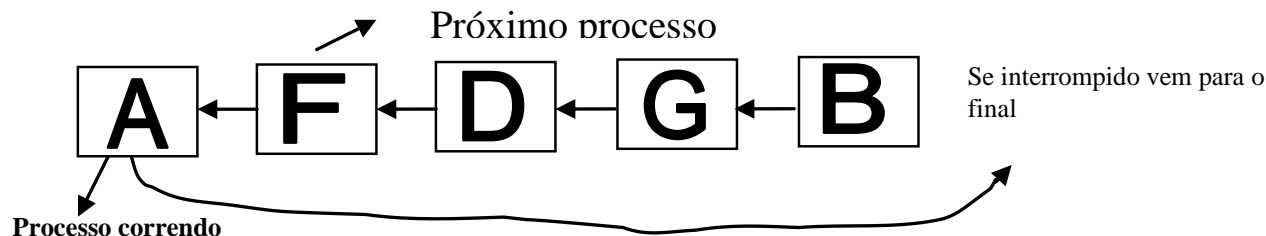
# Sistemas Operacionais de Redes

## Gerência de Processos

### Algoritmos de escalonamento

#### Escalonamento Round Robin - Bach (1986, p. 248)

- O algoritmo escalonador atribui intervalos de tempo (quantum) iguais a cada processo. Se o processo termina antes de espirar seu quantum, o processador é passado para outro imediatamente;
- Existe uma lista de processos, onde o processo que acabou de perder o processador vai para o final dessa fila.



#### Problema:

Se o quantum for muito pequeno ou muito grande. (100 ms é um bom quanta)

# Sistemas Operacionais de Redes

## Gerência de Processos

### Escalonamento com prioridade

- Cada processo é associado a uma determinada prioridade dentre os processos, que estão no estado pronto. O processo com maior prioridade vai rodar primeiro;
- O processador evita a monopolização por parte de um processo com a alta prioridade, decrementando a sua prioridade enquanto roda. Sua prioridade fique mais baixa do que a de todos os processos no estado de pronto.
- As prioridades podem ser atribuídas diretamente pelo sistema operacional;  
Ex: processam que geram E/S têm prioridade neste S.O;
- O processador dá prioridade a esse tipo de processo, pois logo a demanda será passada para o dispositivo de E/S e liberará o processador central para outro processo;
  - Obs.: O Unix pode implementar o aumento de prioridade através do comando "Nice"

# Sistemas Operacionais de Redes

## Gerência de Processos

### Escalonamento com Filas múltiplas

- Tem classes de prioridades e cada fila possui um quanta (time) diferente. Sempre que o processo gasta todo o tempo da 1ª classe, ele vai para a classe anterior, que tem uma prioridade menor.

- Ex: um processo que gasta 100 quanta para rodar.

O Sistema Operacional possui 7 classes de tempo (1s,2s,4s,8s,16s,32s,64s). Neste caso o processo vai para 1ª classe, roda um quanta e cai e assim na classe anterior, roda mais 2 quanta e cai na classe anterior e assim sucessivamente. Da última classe ele só gastará 37 do 64 quanta. Então o processo gastou 7 passos para ser executado. Se fosse no modelo Round Robin gastaria 100 passos.

**Obs1:** Isso favorece aos processos interativos pequenos, pois eles não caíram de prioridade. (São pequenos).

**Obs2:** À medida que os processos mudam de fila eles são escolhidos com menor frequência.

# Sistemas Operacionais de Redes

## Gerência de Processos

- Menor *Job* Primeiro (fila única) seqüencial
  - Estritamente projetado para sistema Bach;
  - O tempo do *job* é conhecido com antecedência, pois o operador já conhece o tempo de cada *job*;
  - O menor *job* da fila roda primeiro;
  - Obs.: Enquanto o *job* não acaba o processador não é liberado (seqüencial).

Obs.: Esta técnica seria ótima para processos interativos, pois eles são pequenos. O problema é determinar o tempo deste tipo de processo.

# Sistemas Operacionais de Redes

## Gerência de Processos

### Escalonamento garantido

- Se baseia em fazer promessas ao usuário a respeito da performance e cumpri-la de alguma forma;
  - Cada usuário terá meio tempo da capacidade de processamento do processador;
  - Calcula o tempo total do processador, que um usuário usou desde que está ativo;
  - Calcula o tempo que o usuário realmente deve merecer, dividindo o tempo decorrido de sua ativação por  $N$  (número de usuários)
    - Ex: Tempo merecido = tempo que o processador está no ar/ $N^o$ . de usuários. Tempo usado pelo usuário = tempo total dele.

# Sistemas Operacionais de Redes

## Gerência de Processos

### Escalonamento de dois níveis

- Até agora assumimos que todos os processos estão no estado pronto na RAM. O que ocorre se não houver espaço para todos os processos na memória RAM? Usa-se o mecanismo de Swap (HD);
- Isso provoca um impacto de tempo na hora de fazer a troca do contexto para escalonar processos;
- Como tratar esse problema? Usando um escalonador de 2 níveis;
- Cria-se um grupo de processos na memória e o escalonador limita-se a escolher processos deste grupo;
- Periodicamente, um escalonador de mais alto nível remove processos que tenham ficado tempo suficiente na RAM e os coloca no HD e carrega para a memória os que estão no HD;
- Novamente o escalonador de mais baixo nível começa a trabalhar.

# Sistemas Operacionais de Redes

## Gerência de Processos

### Escalonamento de dois níveis

- Alguns critérios do escalonador de alto nível (Swap)
  - Quanto tempo se passou desde que o processo está no HD;
  - Quanto tempo de processador o processo gastou;
  - Qual o tamanho do processo;
  - Qual a prioridade do processo.