



# SISTEMAS OPERACIONAIS

Processos

# Processo (1)

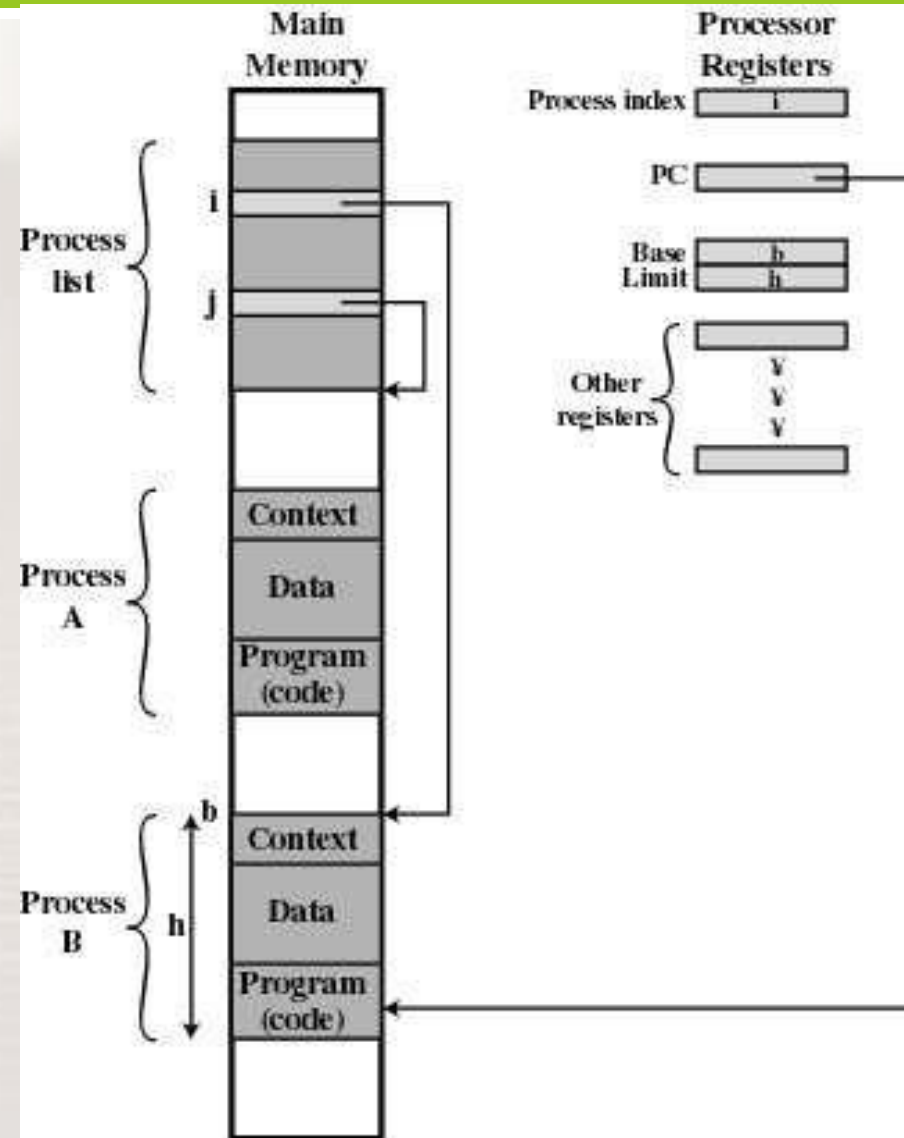
2

- Abstração usada pelo S.O. para designar a execução de um programa.
  - É caracterizado por uma *thread* de execução, um estado corrente e um conjunto associado de recursos do sistema.
- Um processo é um programa individual em execução (uma instância de um programa rodando em um computador).
  - É também referenciado como “tarefa” (*task*) ou mesmo “*job*”.
  - O processo é uma entidade ativa (i.e., é um conceito dinâmico), ao contrário do programa.
  - Cada processo é representado no SO por estruturas de controle (ex: bloco de controle de processo).

# Processo (2)

3

- Uma possível implementação de processos



# Processo (3)

4

- Do ponto de vista da UCP, um processo executa instruções do seu repertório em alguma sequência ditada pelos valores do registrador PC (*program counter*).

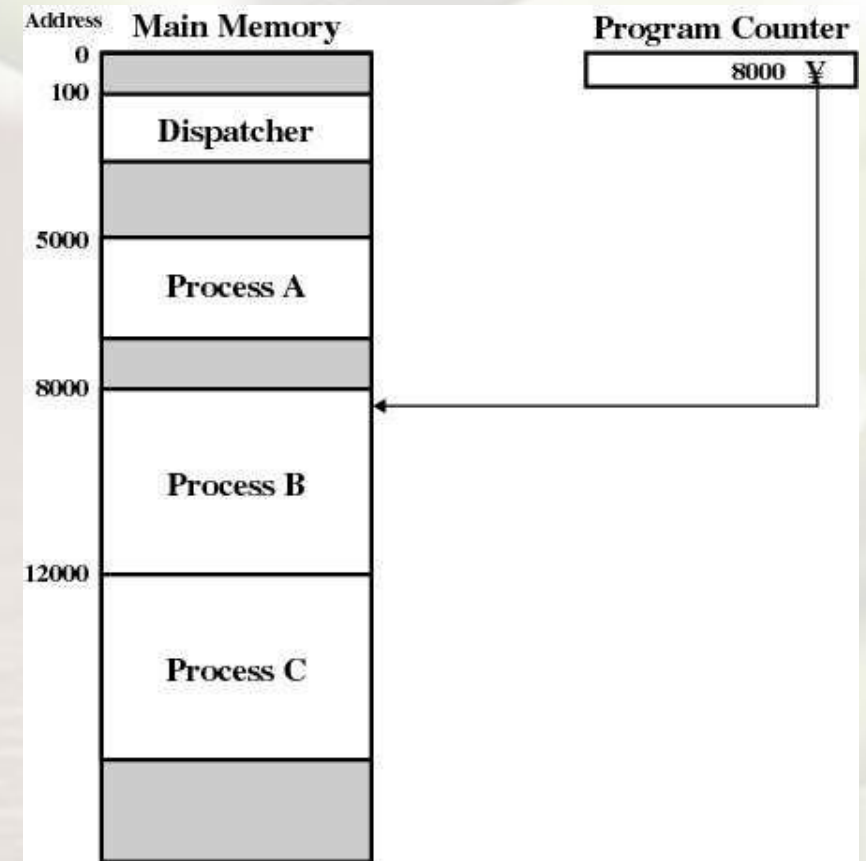


Figure 3.1 Snapshot of Example Execution (Figure 3.3) at Instruction Cycle 13

# Processo (4)

5

- O comportamento de um processo pode ser caracterizado pela sequência de instruções executadas (trace).

5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

(a) Trace of Process A

(b) Trace of Process B

(c) Trace of Process C

5000 = Starting address of program of Process A  
8000 = Starting address of program of Process B  
12000 = Starting address of program of Process C

**Figure 3.2** Traces of Processes of Figure 3.1

# Processo (5)

- A multiprogramação pressupõe a existência de vários processos disputando o processador.
- Necessidade de algoritmos de escalonamento de processos.

overhead

1	5000	27	12004
2	5001	28	12005
3	5002	-----Time out	
4	5003	29	100
5	5004	30	101
6	5005	31	102
-----Time out		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	-----Time out	
16	8003	41	100
-----I/O request		42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		-----Time out	

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;

first and third columns count instruction cycles;

second and fourth columns show address of instruction being executed

**Figure 3.3 Combined Trace of Processes of Figure 3.1**

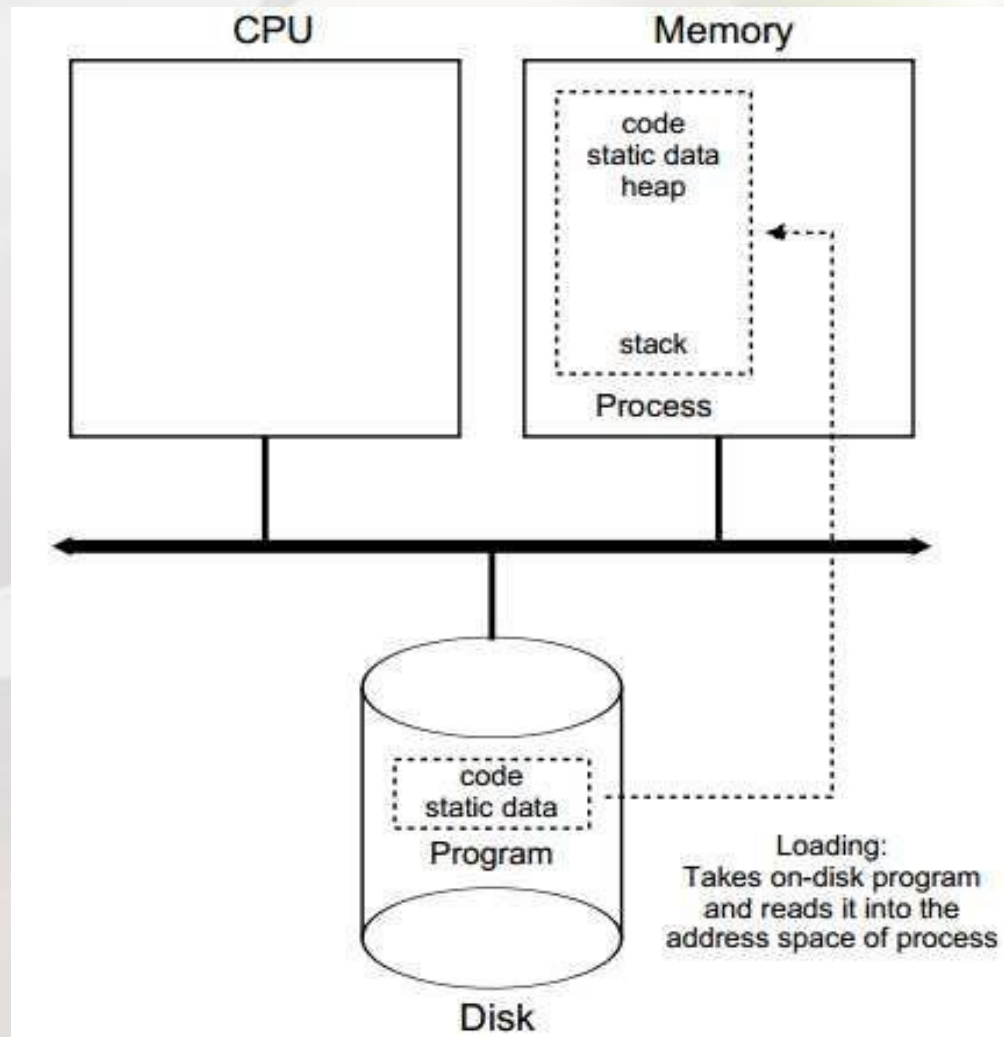


# Overhead

7

- Execução do escalonamento
  - Tarefa de alternar a CPU entre dois processos
  - O tempo depende muito do hardware
    - Velocidade da memória, no. de registradores, instruções especiais de carga de registradores.
    - 1 a 1000 microseg.
- Troca de contexto

# Criação de processo





# Criação de processo

9

```
#include <stdio.h>
void main (int argc, char *argv[])
{
    int pid;
    pid = fork();
    if (pid < 0){
        fprintf stderr, "Fork falhou");
        exit (-1);
    }
    else if (pid == 0) {
        printf ("processo filho");
    } else {
        printf ("processo pai");
        wait (NULL);
        printf ("filho concluiu");
        exit(0);
    }
}
```

# Chamadas de Sistema (SVCs)

## Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

## File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

## Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

## Miscellaneous

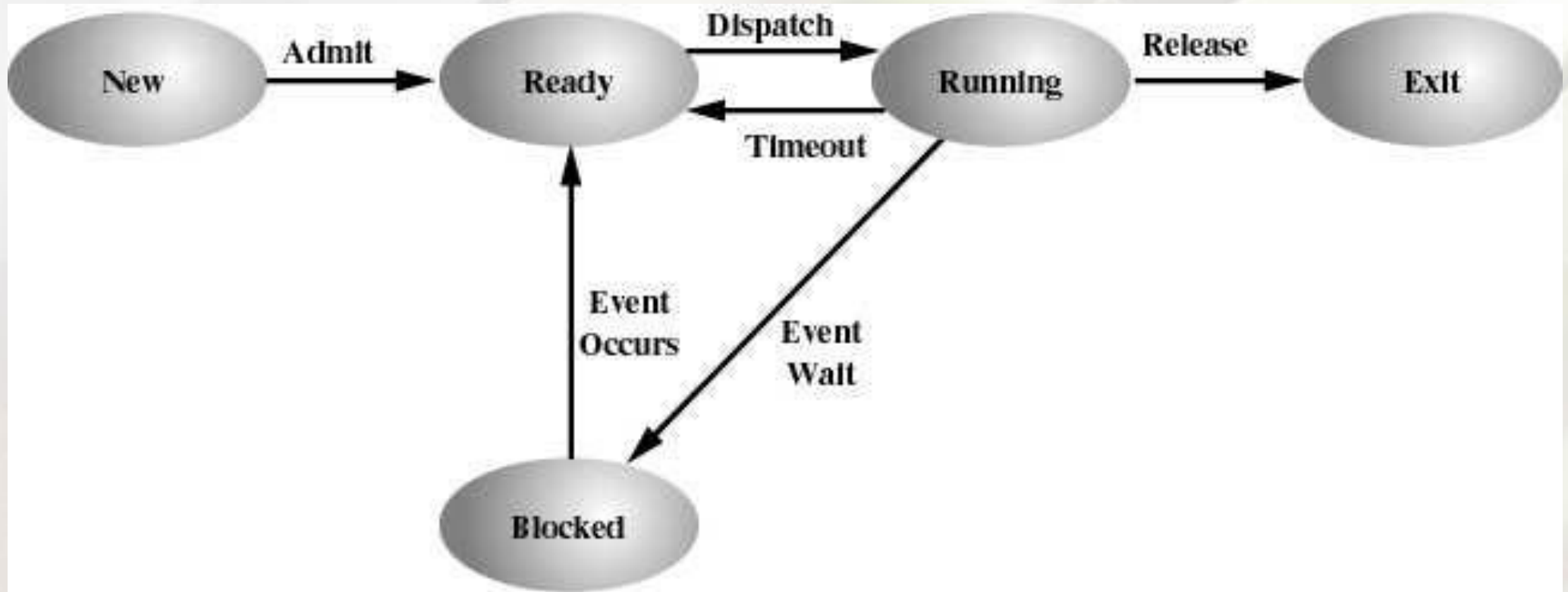
Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970

# Estados de um Processo

11

- Durante a sua execução, um processo passa por diversos estados, refletindo o seu comportamento dinâmico, isso é, a sua evolução no tempo.
- Exemplos de estados:
  - New: recém criado.
  - Ready: pronto para execução.
  - Running: em execução.
  - Blocked: esperando por um evento.
  - Exit: processo terminado.
- Apenas um único processo pode estar no estado “*running*” num dado processador, num dado instante.

# Modelo de 5 Estados (1)



**Figure 3.5 Five-State Process Model**

# Modelo de 5 Estados (2)

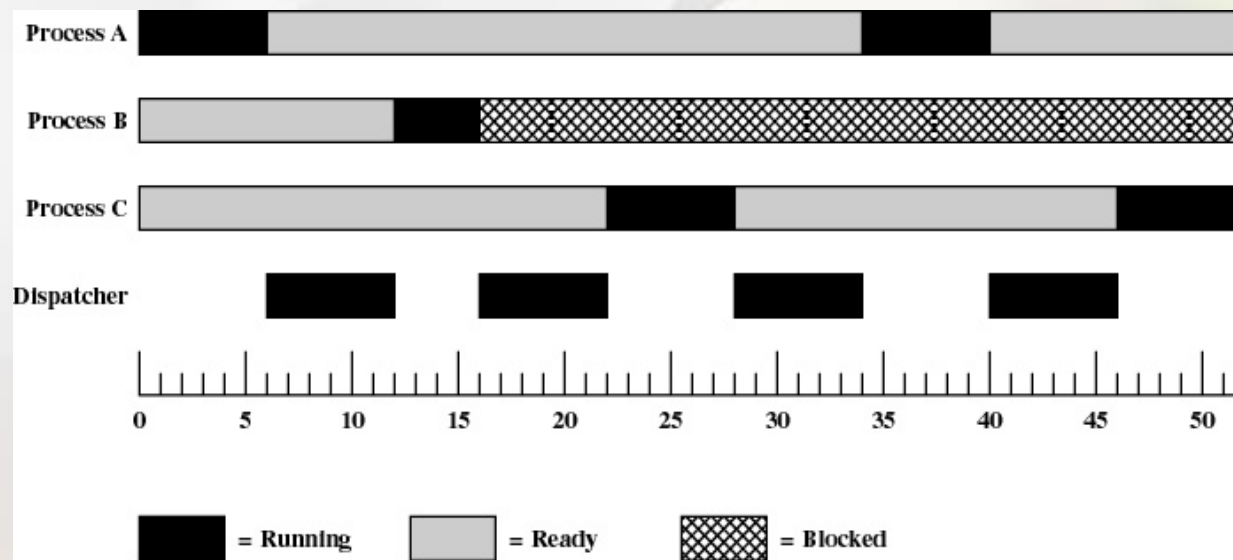


Figure 3.6 Process States for Trace of Figure 3.3

1	5000	27	12004
2	5001	28	12005
3	5002	-----Time out	
4	5003	29	100
5	5004	30	101
6	5005	31	102
-----Time out		32	103
7	100	33	104
8	101	34	105
9	102	35	5006
10	103	36	5007
11	104	37	5008
12	105	38	5009
13	8000	39	5010
14	8001	40	5011
15	8002	-----Time out	
16	8003	41	100
-----I/O request		42	101
17	100	43	102
18	101	44	103
19	102	45	104
20	103	46	105
21	104	47	12006
22	105	48	12007
23	12000	49	12008
24	12001	50	12009
25	12002	51	12010
26	12003	52	12011
		-----Time out	

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;

first and third columns count instruction cycles;

second and fourth columns show address of instruction being executed

Figure 3.3 Combined Trace of Processes of Figure 3.1

# Transições de Estados (1)

14

## □ **New:**

- Um novo processo é criado para executar o programa.
  - Novo *batch job*.
  - *Logon* interativo (usuário se conecta ao sistema).
  - S.O. cria processo para prover um serviço (ex: impressão).
  - Processo cria um outro processo ("*process spawning*").

## □ **New -> Ready:**

- No estado **New**, recursos foram alocados pelo S.O. mas não existe um compromisso de que o processo será executado.
  - Número de processos já existentes;
  - Quantidade de memória virtual requerida, etc.
- Manter um bom desempenho do sistema é o fator limitante da criação de novos processos.

# Transições de Estados (2)

15

## □ **Ready -> Running:**

- Definido pela política de escalonamento de processos adotada pelo S.O.

## □ **Running -> Exit:**

- Processo terminou as suas atividades ou foi abortado.
  - Término normal;
  - Término do processo pai (em alguns sistemas)
  - Excedeu o limite de tempo;
  - Memória não disponível;
  - Erro aritmético ou de proteção;
  - Execução de instrução inválida ou de instrução privilegiada no modo usuário;
  - Intervenção do S.O. (ex: ocorrência de deadlock);



# Transições de Estados (3)

16

- **Running -> Ready:**
  - Tempo máximo de execução sem interrupção foi atingida;
  - Processo é "preemptado" pelo S.O.
- **Running -> Blocked:**
  - Processo requisitou alguma coisa pela qual deve esperar
- **Blocked -> Ready:**
  - Evento pelo qual o processo espera aconteceu.
- **Ready -> Exit:**
- **Bloqued -> Exit:**
  - Processo pai termina um processo filho.
  - Processo pai é terminado, e os processos filhos associados são também finalizados.

# Troca de Contexto

17

- *Contexto* de um processo são todas as informações necessárias para que o S.O. possa restaurar a execução do processo a partir do ponto interrompido.
- A troca de contexto ocorre sempre que um novo processo é selecionado para execução (isso é, quando a UCP é chaveada para um outro processo).
- O tempo de troca de contexto é puro *overhead* e é dependente de suporte de hardware (ex: salvamento automático do PC).

# Quando Chavear um Processo

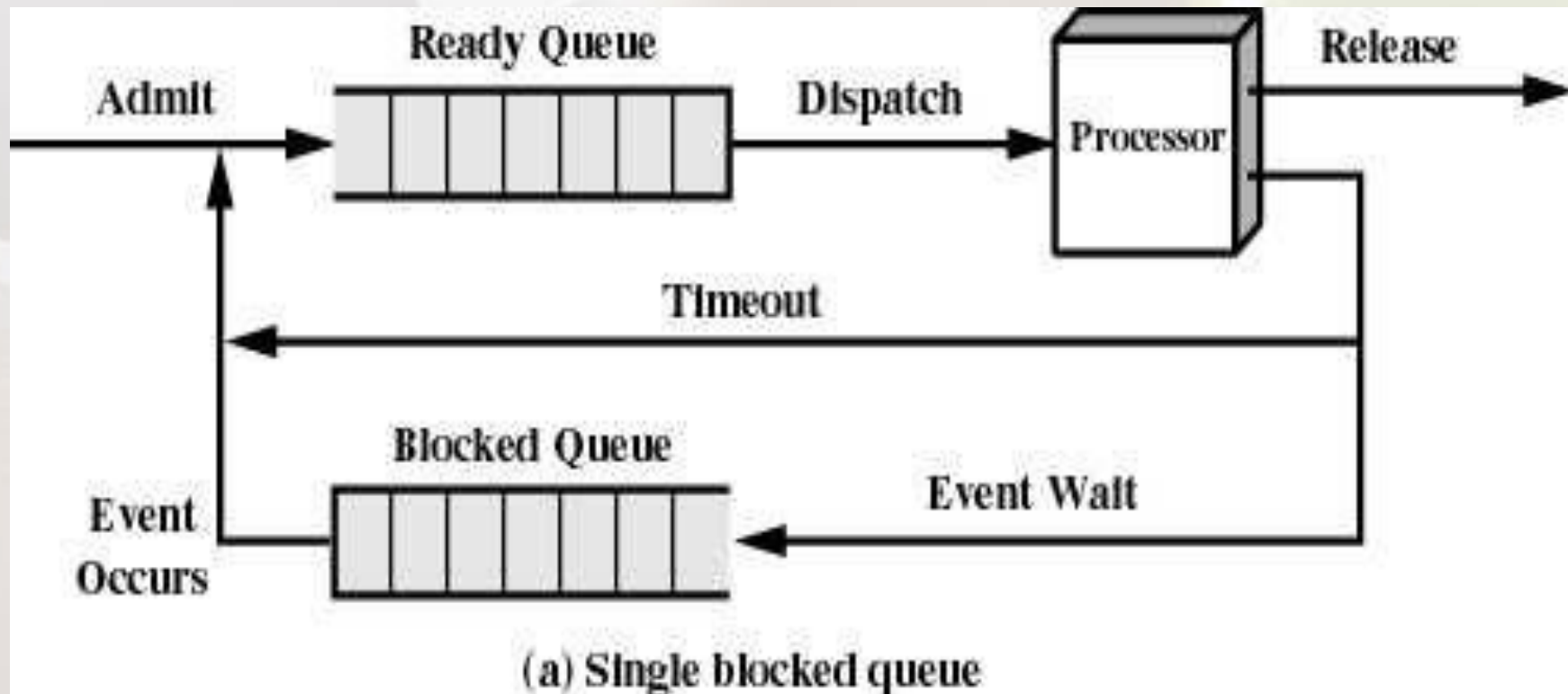
18

- Interrupção do relógio
  - Fatia de tempo de posse da UCP expirou;
- Interrupção de E/S
- Falta de memória
  - Endereço de memória está na memória virtual (disco); logo deve ser trazido para a memória principal.
- Trap
  - Ocorrência de erro.
  - Pode fazer com que o processo seja movido para o estado Exit.

# Filas do Sistema (1)

19

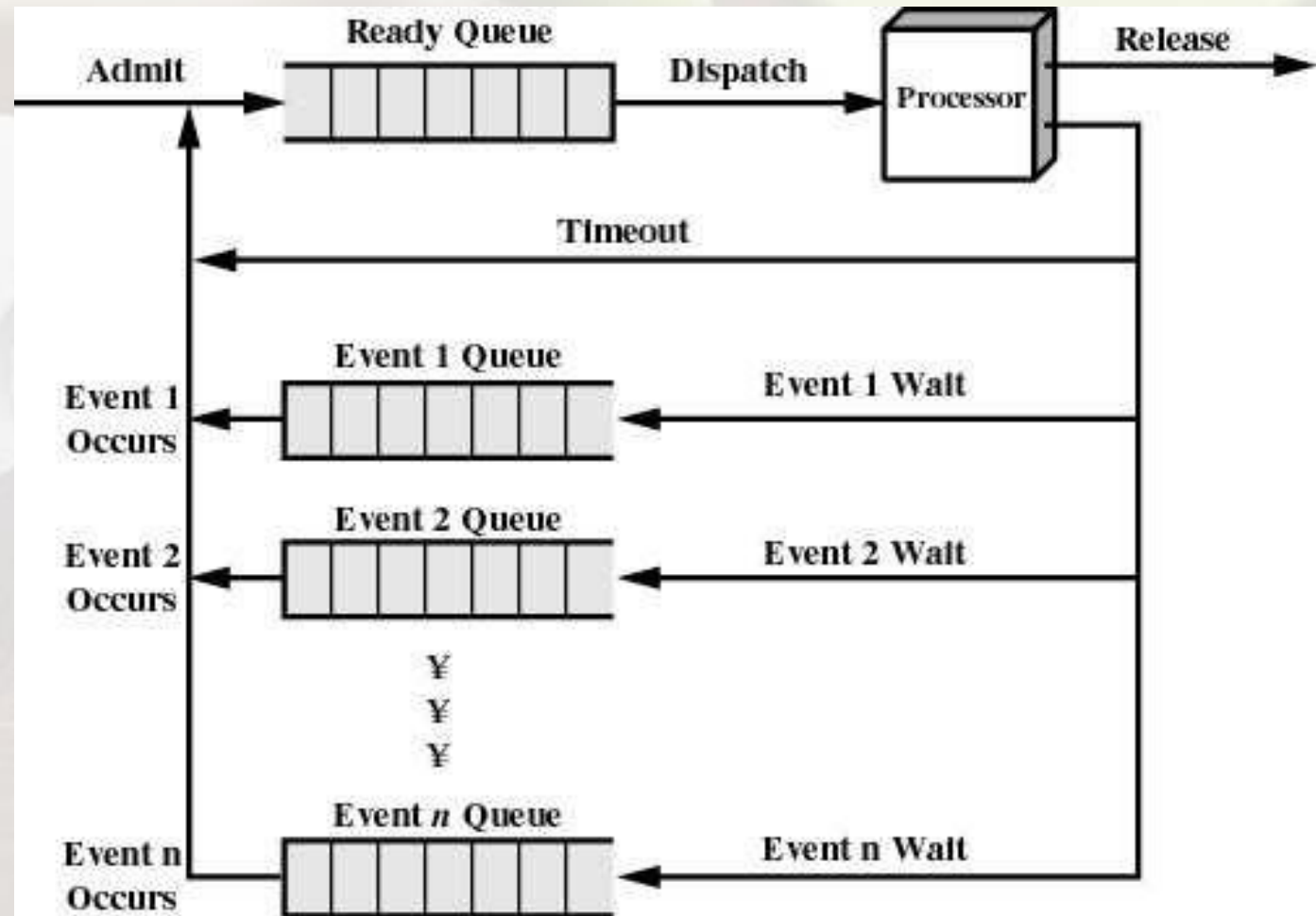
- Um processo sempre faz parte de alguma fila.
- Eventos realizam a transição de uma fila para outra.
- Fila de prontos e uma ou mais filas de bloqueados.



# Filas do Sistema (2)

20

- ❑ Múltiplas filas de bloqueados



(b) Multiple blocked queues

# Processos Suspensos (1)

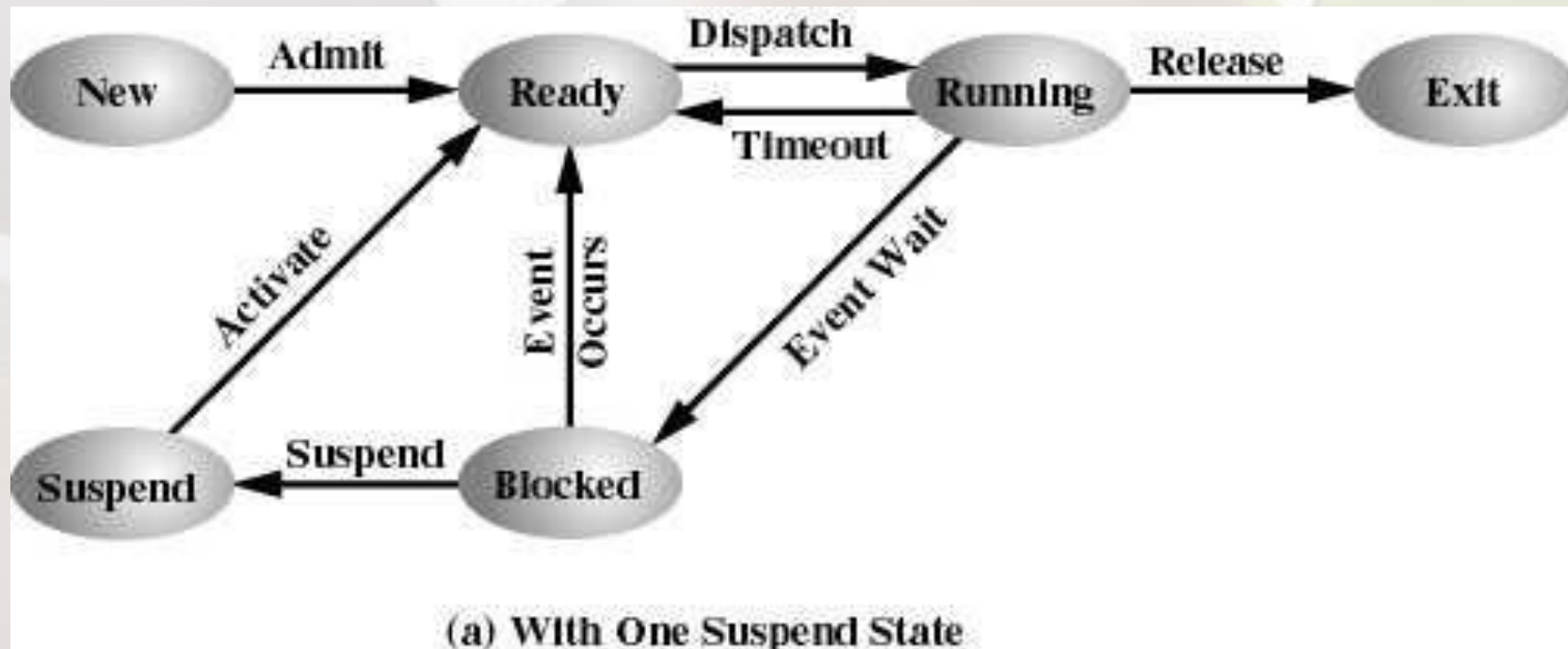
21

- Processador é tão mais rápido que os dispositivos de E/S que **todos** os processos em memória poderiam ficar em situação de espera.
  - Mesmo com multiprogramação, o processador poderia ficar a maior parte do tempo ocioso!
- Aumento de memória para acomodar mais processos:
  - Aumento do custo;
  - Disponibilidade de mais memória geralmente resulta em processos maiores e não em maior número de processos.
- *Swapping*: procedimento que consiste em mover todo ou parte de um processo da memória para o disco.

# Processos Suspensos (2)

22

- Quando **nenhum** dos processos na memória principal está no estado **Ready** o sistema operacional manda um dos processos **bloqueados para o disco**, e o coloca numa **fila de processos suspensos**.



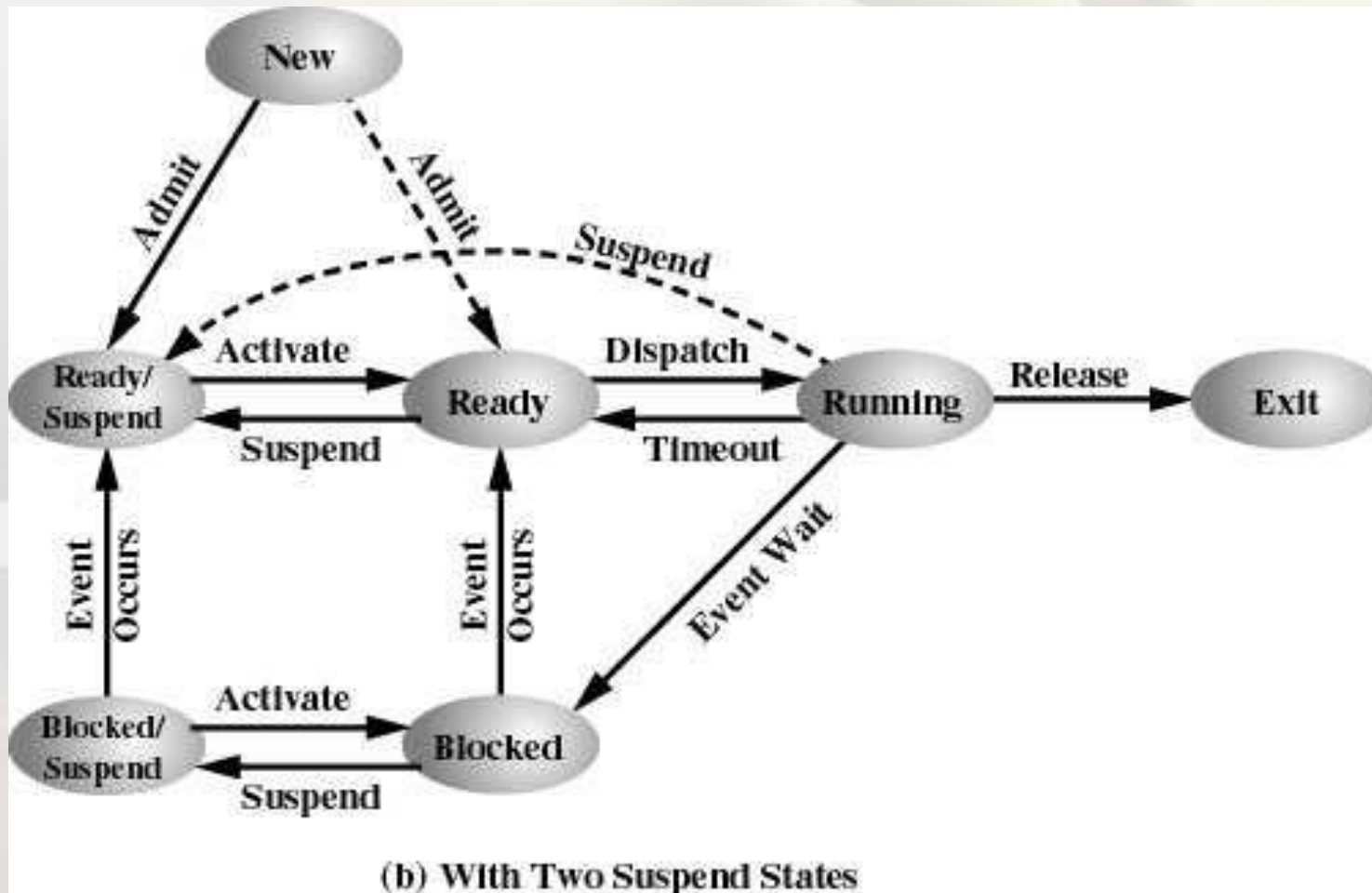


# Processos Suspensos (3)

23

- O S.O traz então do disco algum outro processo da fila de suspensos ou atende a uma solicitação de criação de um novo processo.
- O swap é uma operação de E/S e, como tal, existe a possibilidade de tornar o problema ainda pior, caso o sistema de E/S não seja eficiente.
- Modelo mais elaborado: dois novos estados são então
  - *Blocked, suspend*: o processo está em memória secundária e aguardando por um evento.
  - *Ready, suspend*: o processo está em memória secundária mas está disponível para execução, tão logo ele seja carregado na memória principal.
  - *Blocked*: processo está na MP e aguardando por um evento]

# Processos Suspensos (4)



# Máquina de Estados do Unix (1)

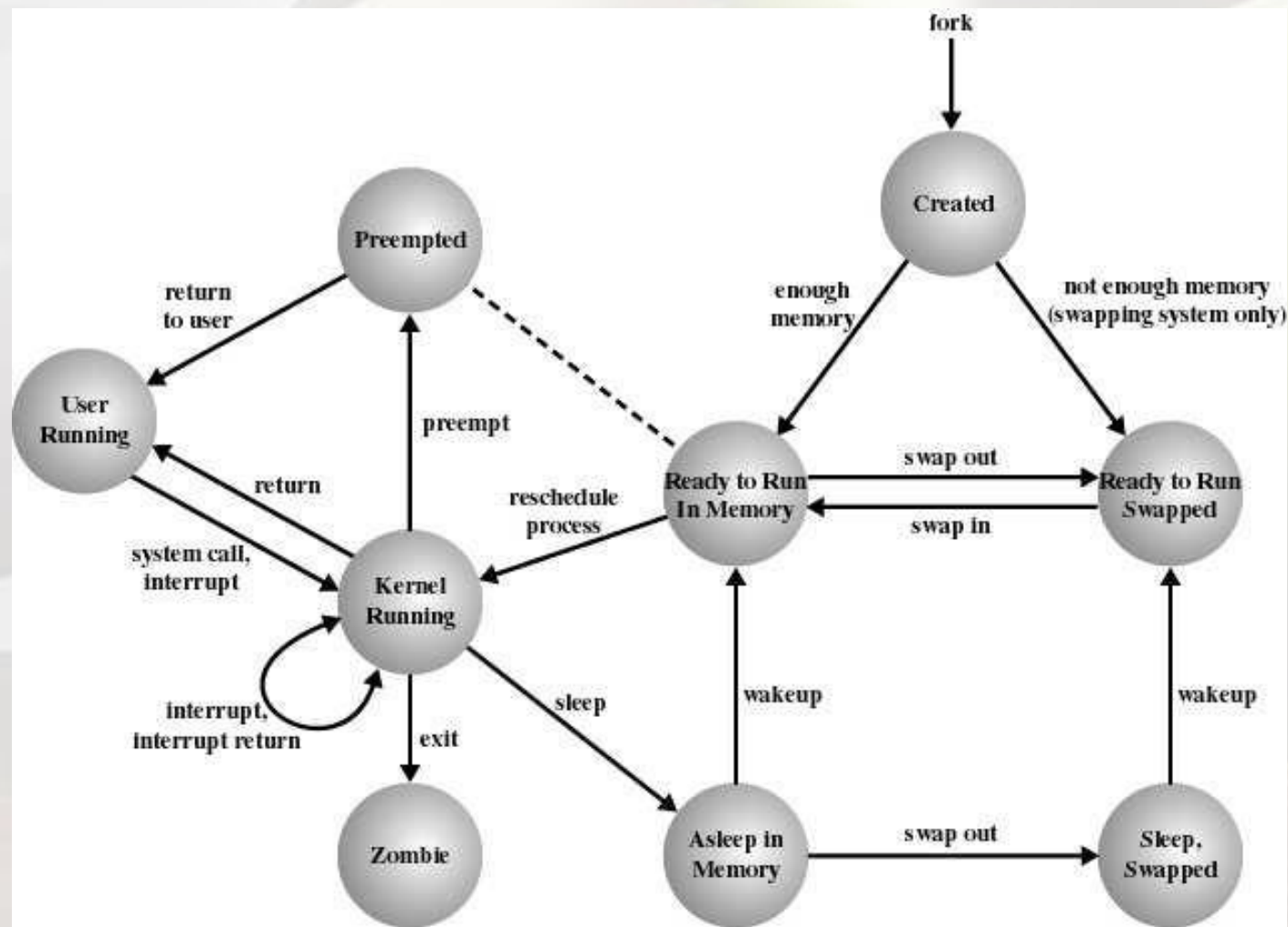


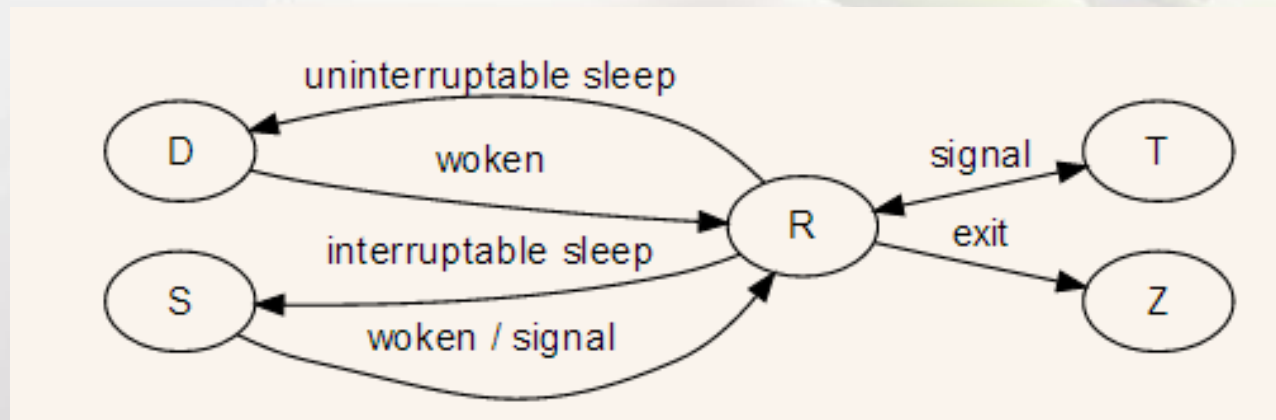
Figure 3.16 UNIX Process State Transition Diagram

# Máquina de Estados do Unix (2)

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

# Máquina de Estados do Linux

27



## PROCESS STATE CODES

R running or runnable (on run queue)

D uninterruptible sleep (usually IO)

S interruptible sleep (waiting for an event to complete)

Z defunct/zombie, terminated but not reaped by its parent

T stopped, either by a job control signal or because it is being traced

# Referências

- Slides adaptados de Roberta Lima Gomes (UFES)
- Bibliografia
  - W. Stallings, "Operating Systems: internals and design principles", 6th Edition, Editora Prentice-Hall, 2009.
    - Capítulo 3 (até seção 3.4 inclusa)
  - Deitel H. M.; Deitel P. J.; Choffnes D. R.; "Sistemas Operacionais", 3ª. Edição, Editora Prentice-Hall, 2005
    - Capítulo 3 (até seção 3.3 inclusa)
  - Silberschatz A. G.; Galvin P. B.; Gagne G.; "Fundamentos de Sistemas Operacionais", 8a. Edição, Editora LTC, 2010.
    - Capítulo 3