Sistema Operacional

Processo e Threads

Introdução a Processos

Todos os computadores modernos são capazes de fazer várias coisas ao mesmo tempo. Enquanto executa um programa do usuário, um computador pode também ler os dados de um disco, mostrar um texto na tela etc.

Em um sistema multiprogramado, a CPU salta de programa para programa, executando cada um deles por dezenas ou centenas de milissegundos.

Enquanto em cada momento, o processador pode executar apenas um processo, em um segundo ele pode executar vários processos, passando para o usuário a impressão de paralelismo.

Ter controle sobre os vários processos que estão em execução é uma tarefa difícil, e a principal tarefa do sistema operacional é gerenciar todos os processos que estão em execução, como também, os recursos que cada processo utiliza.

O Modelo de processo

Todos os processos que executam no computador são organizados em vários processos **seqüenciais**. Um processo é apenas um programa em execução acompanhado dos valores atuais do contador de programa, dos registradores, e das variáveis.

Conceitualmente, cada processo tem sua própria CPU virtual, mas, na realidade, a CPU troca , a todo o momento, um processo para outro. Esses conceito de *pseudo* paralelismo é conhecido como multiprogramação.

A Figura 1 ilustra um computador multiprogramado com 4 processos na memória.

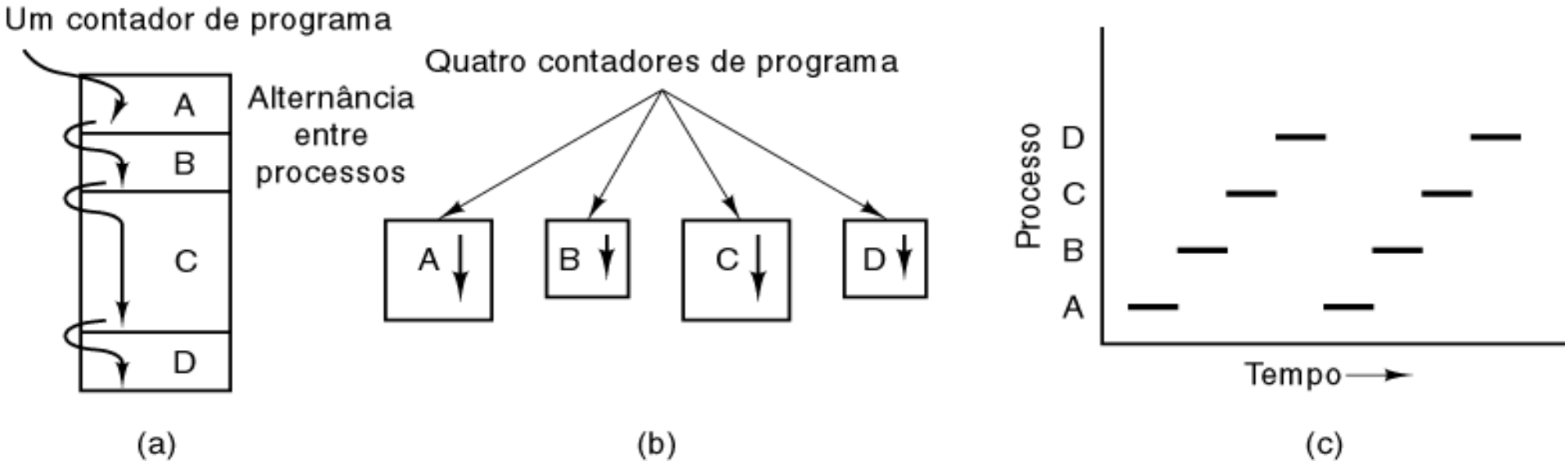


Figura 1 – Um computador multiprogramado

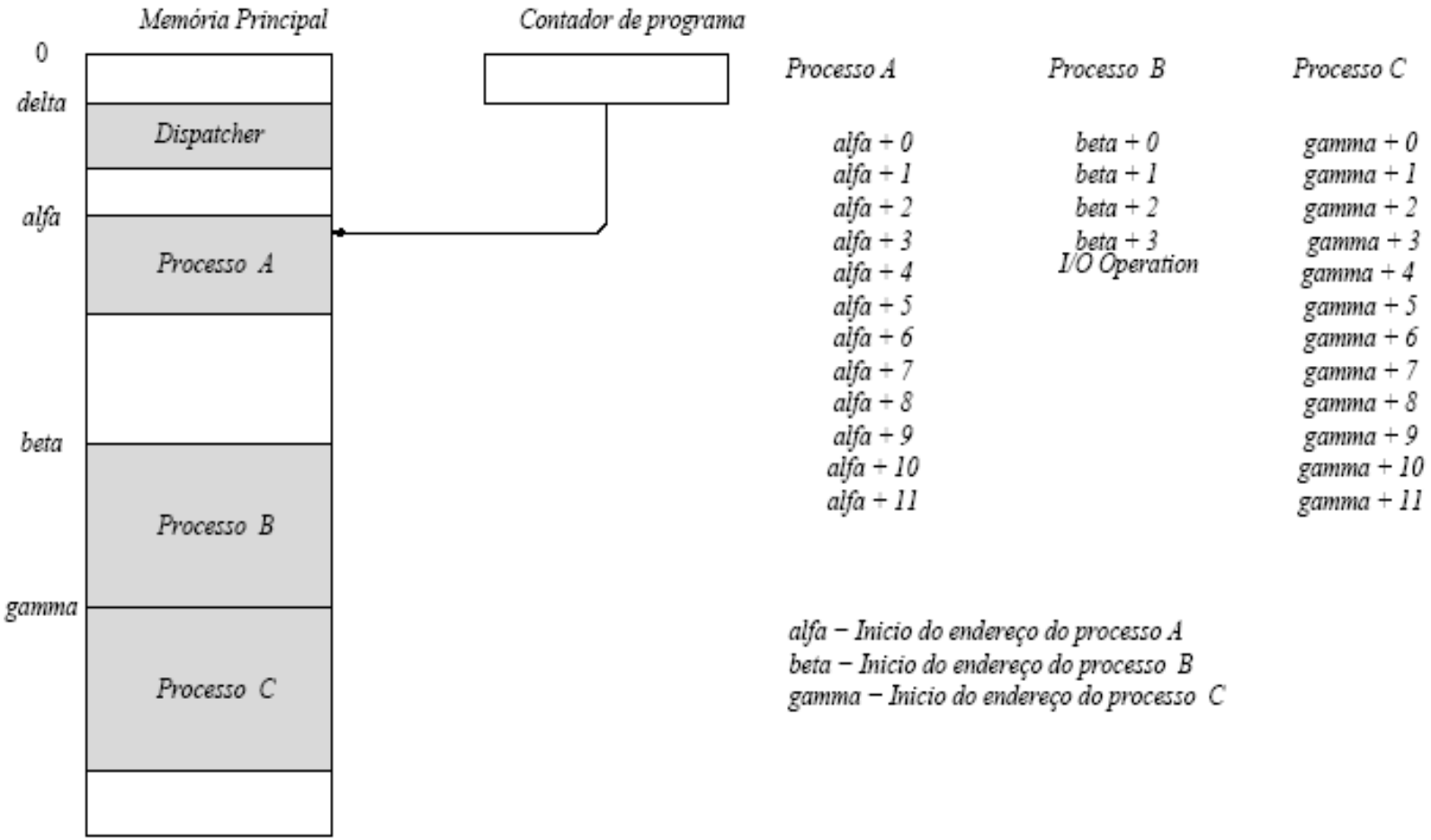
A figura 1-a mostra quatro processos na memória. A figura 1-b ilustra que cada processo possui seu contador de programas, e a figura 1-c mostra a seqüência de execução dos processos.

Na realidade, existe apenas um contador de programa, com isso, quando um processo vai executar, o contador de programa do sistema é carregado com o endereço do processo que irá executar.

Quando acaba o tempo da CPU alocado a um processo, o contador de programa físico é salvo, no contador de programa lógico do processo na memória.

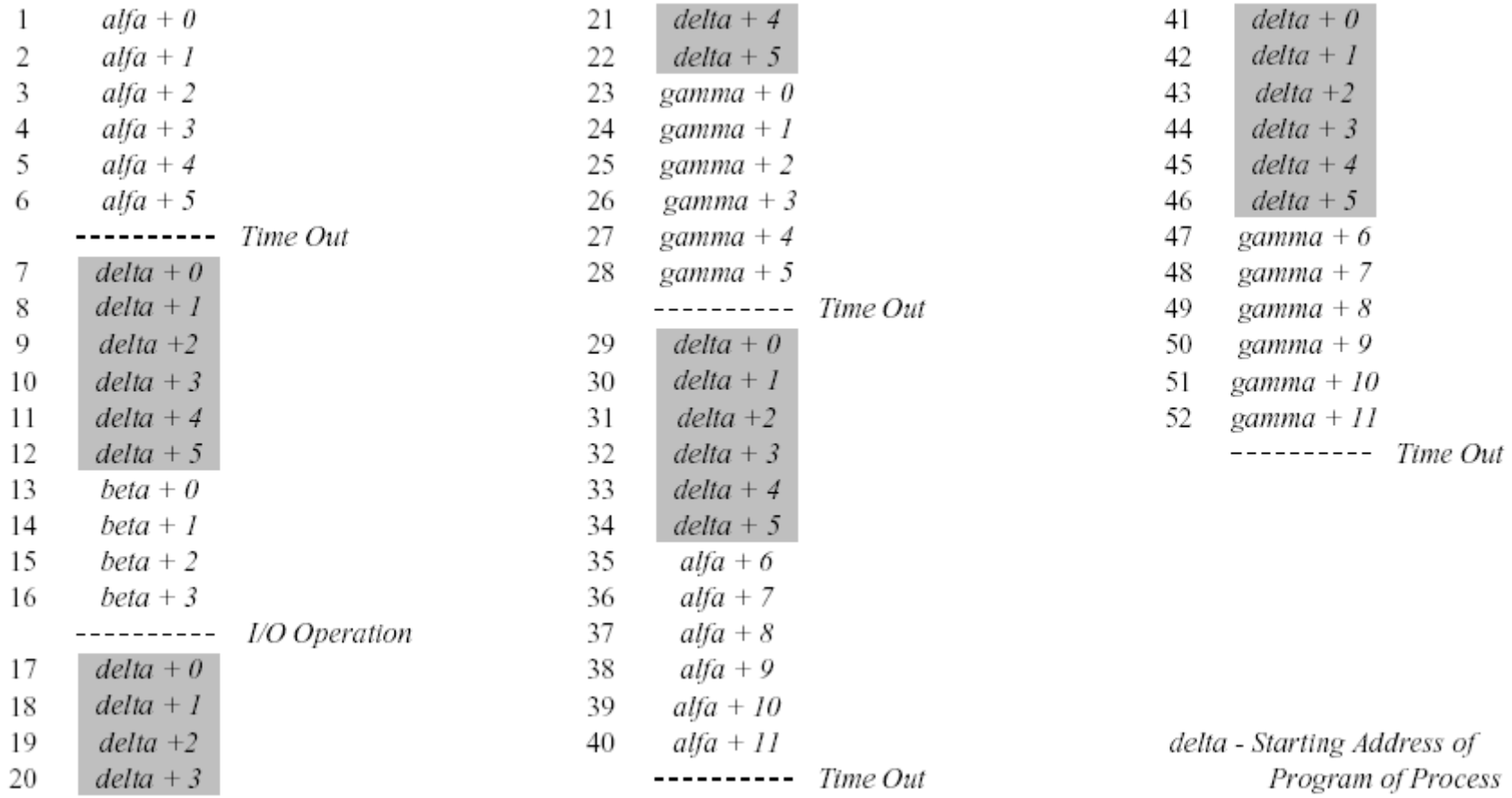
Na figura 1-c vemos que, por um intervalo de tempo suficientemente longo, todos os processos estão avançando, mas, a cada instante, apenas um único processo está realmente executando.

Com a alternância da CPU entre os processos, a taxa na qual o processo realiza sua computação não será uniforme.



Fígura 2 – Um computador multiprogramado

A Figura 2 mostra os passos da execução dos processos da Figura 1, mas no ponto de vista do processador com 6 ciclos de instruções.



Fígura 3: Execução dos processos no ponto de vista do processador.

Criação de processos

Os sistemas operacionais precisam assegurar de algum modo, a existência de todos os processos necessários.

Em sistemas muito simples, por exemplo, o controlador de um forno micro ondas, é possível ter todos os processos que serão necessários presentes quando o sistema é ligado.

Porém, em sistemas de propósito geral, é necessário, de algum modo de criar e terminar processos durante a execução do sistema, quando for preciso.

Há quatro eventos principais que fazem com que processos sejam criados:

1 – Início do sistema;

2 – Execução de uma chamada de sistema de criação de processo por um processo em execução;

3 – Uma requisição do usuário para criar um novo processo;

4 – Início de um Job em lote

Quando o sistema operacional é carregado, em geral, criam-se vários processos. Alguns deles são processos em primeiro plano, ou seja, interagem com o usuários.

Outros são processos em segundo plano, que não estão associados a usuário em particular, mas que apresentam alguma função específica.

Por exemplo, um programa em segundo plano pode ser designado a aceitar mensagens eletrônicas, ficando a maior parte do dia ocioso, mas surgindo a qualquer momento quando uma mensagem chega.

Processos que ficam em segundo plano com a finalidade de tratar alguma atividade como, por exemplo, receber mensagens eletrônicas, páginas da internet, impressão são chamados de ***deamons***.

É comum aos grandes sistemas lançarem mão de dezenas desses processos. No Unix, o programa ***ps*** pode ser usado para relacionar os processos que estão executando.

No Windows, digitando-se uma vez CTRL-ALT DEL, mostra-se o que está em execução.

Além dos processos criados durante inicialização dos sistema operacional, novos processos podem ser criados depois disso. Muitas vezes, um processo em execução emitirá chamadas ao sistema para criar um ou mais novos processos.

Criar novos processo é particularmente útil quando a tarefa a ser executada pode facilmente ser formulada com base em vários processos relacionados, mas interagindo de maneira independente.

Por exemplo, se uma grande quantidade de dados estiver sendo trazida via rede para que seja subseqüentemente processada, poderá ser conveniente criar um processo para trazer esses dados e armazena-los em um local compartilhado na memória, enquanto outro processo remove os dados e os processa.

Em um sistema multiprocessador, permitir que cada processo execute em uma CPU diferente também torna o trabalho mais rápido.

Em sistemas interativos, os usuários podem iniciar um programa digitando um comando ou clicando (duas vezes) um ícone. Cada uma dessas ações inicia um novo processo e executa nele o programa selecionado.

A última situação na qual processos são criados, aplica-se somente a sistemas em lote encontrados em grandes computadores de grande porte.

Nesses sistemas, usuários podem submeter ***jobs*** em lote para o sistema, que, quando julgar que tem recursos para executar o ***job***, o sistema operacional criará um novo processo e executará nele o próximo ***job*** da fila de entrada

Tecnicamente, em todos esses casos, um novo processo é criado por um processo existente executando uma chamada ao sistema de criação de processo.

O que o processo faz é executar uma chamada ao sistema para criar um novo processo, e assim indica, qual programa executar nele.

Término de processos

Depois de criado, um processo começa e executar e faz o seu trabalho. Contudo, nada é para sempre, nem mesmos os processos. Mais cedo ou mais tarde o novo processo terminará, normalmente em razão das seguintes condições:

1 - Saída normal (voluntária);

2 – Saída por erro (voluntária);

3 – Erro fatal (involuntário);

4 – Cancelamento por um outro processo (involuntário);

Na maioria das vezes, os processos terminam porque fizeram o seu trabalho. Por exemplo, quando um compilador acaba de compilar um programa, ele executa uma chamada de sistema indicando para o sistema operacional que ele terminou.

O segundo motivo para o término é que o processo descobre um erro fatal. Por exemplo, se um usuário digitar:

gcc prog.c

para compilar o programa ***prog.c*** e esse arquivo não existe, o compilador simplesmente emite uma chamada de saída ao sistema. Processos interativos com base na tela geralmente não fecham quando é passado um parâmetro errado, Em vez disso, pergunta ao usuário se ele quer tentar novamente.

A terceira razão para o término é um erro causado pelo processo, muitas vezes causados pelo programa. Entre os vários exemplos, estão a ***execução ilegal de instrução***, a referência à memória inexistente ou a divisão por zero.

Em alguns sistemas, (por exemplo, o Unix), alguns processos podem sinalizar para o sistema operacional que eles mesmo deseja tratar certos erros. Neste caso, o processo não é finalizado, mas sim interrompido pelo sistema operacional.

A quarta razão pela qual um processo pode terminar se dá quando um processo executa uma chamada ao sistema dizendo ao sistema operacional para cancelar alguns processos. No Unix, essa chamada é a ***Kill***. A função correspondente no ***Win32*** é a ***TerminateProcess***.

Hierarquia de processos

Em alguns sistemas, quando um processo cria outro processo, o processo pai e o processo filho continuam, de certa maneira associados. O próprio processo filho pode criar mais processos, formando uma hierarquia de processos.

No Unix, um processo, todos os seus filhos e descendentes formam um grupo de processo. Quando um usuário envia um sinal do teclado, o sinal é entregue para todos os membros do grupo de processo associado com o teclado.

Individualmente, cada processo pode capturar o sinal, ignorá-lo ou tomar uma ação predefinida, por exemplo, ser cancelado pelo sinal. Exemplos de sinais enviados pelo teclado são: ***CTRL + C***, ***CTRL + ALT + DEL*** etc.

Um outro exemplo de hierarquia de processo pode ser observado quando o ***Unix*** inicia. Um processo especial, chamado, ***init***, está presente na imagem da carga do sistema. Quando começa a executar, ele lê um arquivo dizendo quantos terminais existem. Então, ele se bifurca em um novo processo para executar cada terminal.

Hierarquia de processos

Por outro lado, o windows não apresenta nenhum conceito de hierarquia de processos. Todos os processos são iguais. Algo parecido com a hierarquia de processo acontece somente quando o processo é criado.

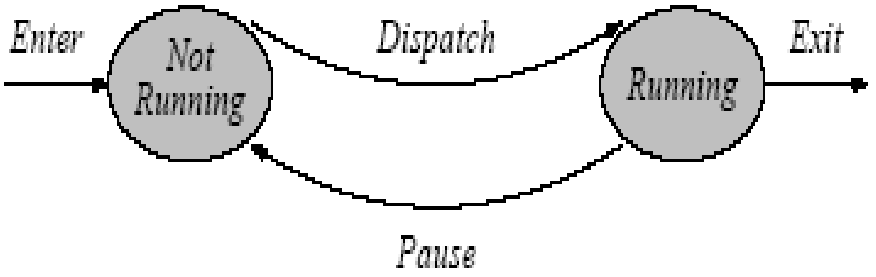
Ao pai é dado um identificador especial (chamado ***Handle***), que ele pode usar para controlar o filho. Contudo, ele,é livre para passar este identificador para outros processos, invalidando assim a hierarquia.

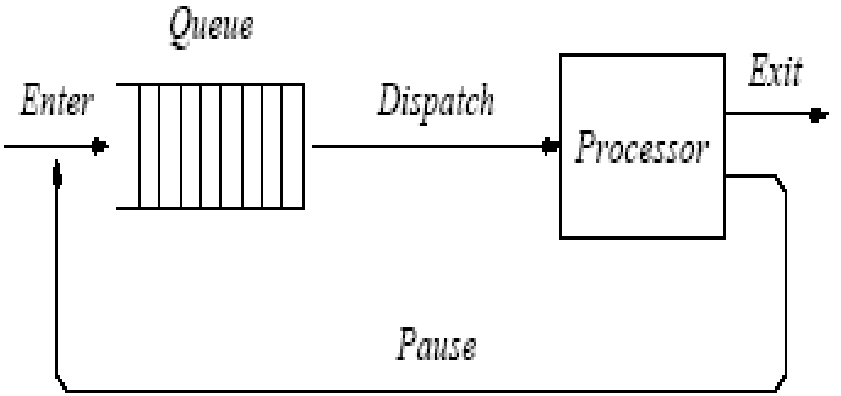
Os processos no Unix não podem deserdar seus filhos.

Estados de processos

Como já citado, a principal responsabilidade do sistema operacional é o controle da execução dos processos, que inclui por sua vez, determinar o padrão com o qual os processos são executados, bem como a alocação de recursos requisitados.

O modelo mais simples de execução é aquele que contempla o processo em pelo menos dois estados: o processo está sendo executado ou não pelo processador.





Estados de processos

Assim, o processo pode estar em dois estados: ***running*** (executando) ou **not running** (não executando);

Processos que não estão executando devem ser mantidos em uma fila, esperando a sua vez de ser executado.

Quando um processo novo é criado, o sistema operacional reserva um espaço em memória para a estrutura de dados que serão usadas para gerenciar o processo, além de alocar espaço em memória para o processo.

Como apresentado no modelo de 2 estados, a fila de processo que não estão executando contém processos prontos para serem executados, mas pode ocorrer que um processo seja bloqueado por outro processo.

Neste caso, os processos estariam todos na fila de prontos para executar ?

Imagine a seguinte situação: Três processos interagindo entre si, sendo que um depende da resposta do processamento do outro para continuar a sua execução.

Caso, o processo tenha que esperar pela resposta do outro processo para continuar seu processamento, ele deverá ser bloqueado, até que se tenha alguma resposta do processamento do outro processo.

Em um outro caso, o sistema operacional pode decidir bloquear um processo que está pronto para executar por algum tempo. Não há vários processadores para dar atendimento exclusivo para todos os processos.

A figura 4 a seguir mostra o estado destes processos.

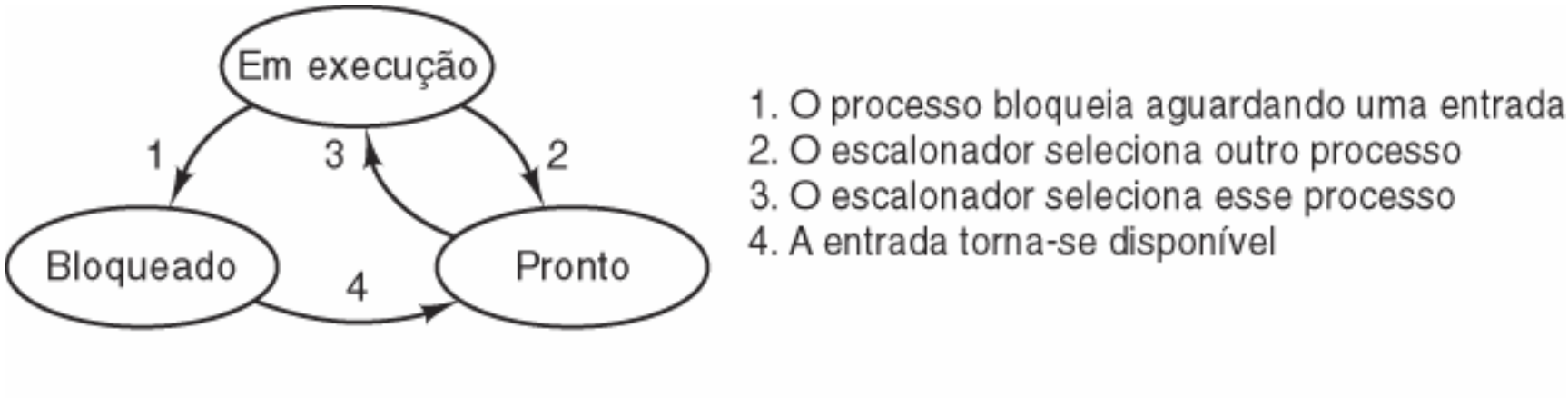


Figura 4. Um processo pode estar em execução, bloqueado ou pronto para executar

O diagrama mostra os seguintes estados:

1 – Em execução (realmente usando a CPU naquele instante);

2 – Pronto (executável; temporariamente parado para dar lugar para outro processo);

3 – Bloqueado (incapaz de executar enquanto um evento externo não ocorra);

Logicamente, os dois primeiros estados são similares. Em ambos os casos, o processo vai executar, só que no segundo não há, temporariamente, CPU disponível para ele.

O terceiro estado é diferente dos dois primeiros, pois o processo não pode executar, mesmo que a CPU não tenha nada a fazer.

Existe também, dois estados que pode ser representado em um diagrama, pois um processo antes de ser executado, ele deve existir, ou seja criado, e depois de executado (por inteiro), o programa é encerrado.

A figura 5 mostra este diagrama:

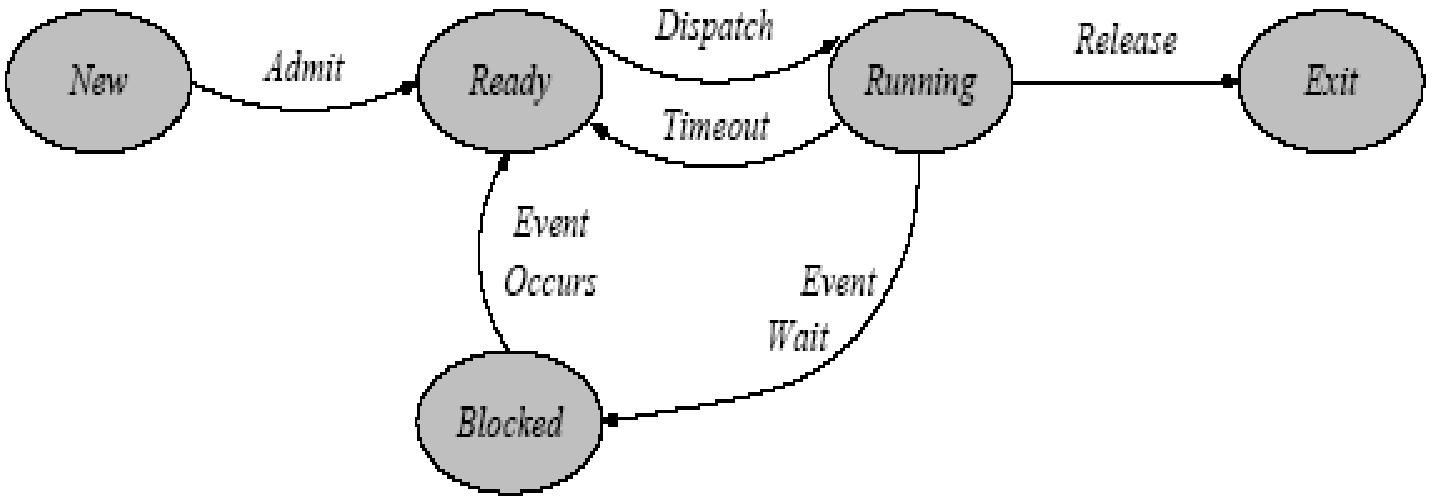


Figura 5: os estados "novo" e "sair"

O estado "novo" corresponde a um estado que acabou de ser definido.

O estado "sair" corresponde a um estado intermediário pelo qual o processo passa antes que a estrutura de dados usada para seu gerenciamento seja liberada.

Considerando novamente o exemplo dos processos A, B, C

```
1   alfa + 0
2   alfa + 1
3   alfa + 2
4   alfa + 3
5   alfa + 4
6   alfa + 5
    ---------- Time Out
7   delta + 0
8   delta + 1
9   delta +2
10  delta + 3
11  delta + 4
12  delta + 5
13  beta + 0
14  beta + 1
15  beta + 2
16  beta + 3
    ---------- I/O Operation
17  delta + 0
18  delta + 1
19  delta +2
20  delta + 3
```

```
21  delta + 4
22  delta + 5
23  gamma + 0
24  gamma + 1
25  gamma + 2
26  gamma + 3
27  gamma + 4
28  gamma + 5
    ---------- Time Out
29  delta + 0
30  delta + 1
31  delta +2
32  delta + 3
33  delta + 4
34  delta + 5
35  alfa + 6
36  alfa + 7
37  alfa + 8
38  alfa + 9
39  alfa + 10
40  alfa + 11
    ---------- Time Out
```

```
41  delta + 0
42  delta + 1
43  delta +2
44  delta + 3
45  delta + 4
46  delta + 5
47  gamma + 6
48  gamma + 7
49  gamma + 8
50  gamma + 9
51  gamma + 10
52  gamma + 11
    ---------- Time Out

delta - Starting Address of
        Program of Process
```

Considerando o exemplo dos processos A, B, C, abaixo segue uma tabela mostrando os estados de cada processo.

Time	Processo A	Processo B	Processo C
1-6	***Running***	*Ready*	*Ready*
7-12	*Ready*	*Ready*	*Ready*
13-16	*Ready*	***Running***	*Ready*
17-22	*Ready*	*Blocked*	*Ready*
23-28	*Ready*	*Blocked*	***Running***
29-34	*Ready*	*Blocked*	*Ready*
35-40	***Running***	*Blocked*	*Ready*
41-46	*Ready*	*Blocked*	*Ready*
47-52	*Ready*	*Blocked*	***Running***

As possíveis transições de um processo são as seguintes:

Null --> Novo: Um novo processo é criado para executar um programa.

Novo --> Pronto: O sistema operacional irá mover o processo do estado de novo para o estado de **pronto.**

Pronto --> Executando : O sistema operacional escolhe um dos processos na fila de **pronto** para ser executado.

Executando --> Sair : O processo indica para o sistema operacional que ele terminou seu trabalho, e o sistema operacional o retira do sistema.

Executando --> Pronto : O processo não terminou o seu trabalho, e necessita de mais tempo da CPU.

Executando --> Bloqueado : O processo é colocado como bloqueado quando ele requisita alguma coisa e necessita esperar por isso.

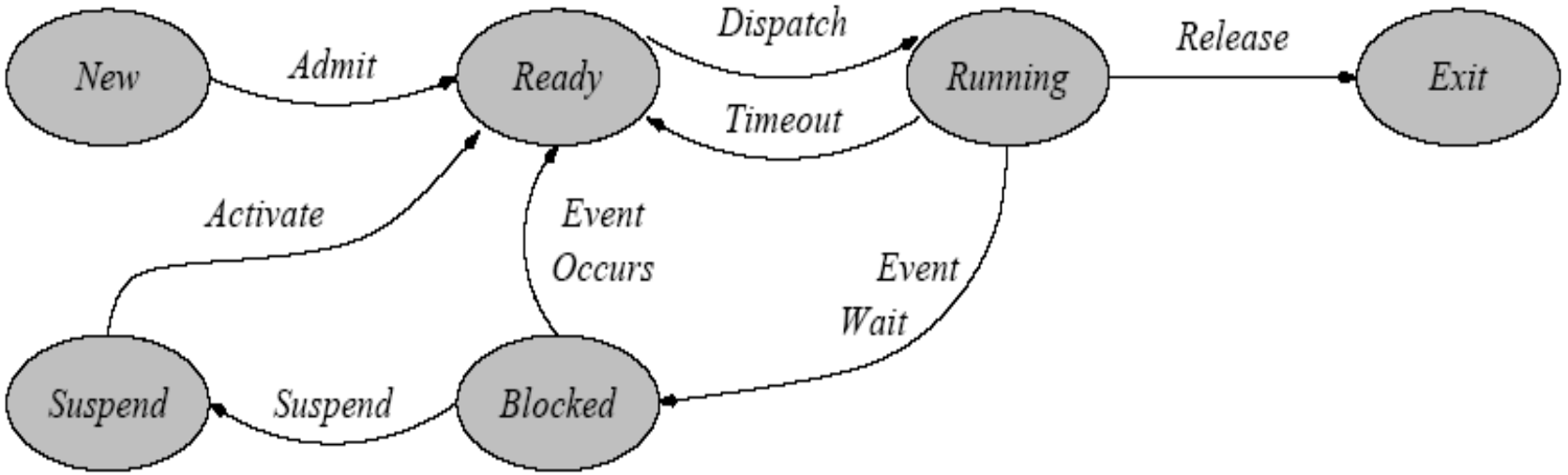
Processos Suspensos

Os três principais estados de um processo (pronto, execução e bloqueado), prove um modelo sistemático para modelar o comportamento dos processos como também para ajudar na implementação de processos nos sistemas operacionais.

Porém, estes estados são suficientes ??

Quando um processo vai executar, ele deve estar na memória principal (*loadable*). O que aconteceria se não houvesse mas espaço disponível na memória principal ????

O estado de swapping



Como pode ser observado, um estado novo foi acrescentado, o estado **Suspend**. Quando um processo está bloqueado, ele pode ser movido pelo sistema operacional para o estado suspenso em disco, efetuando assim um *swapping*

Quando o sistema operacional executa esta operação (*swapping out*), ele tem duas escolhas para fazer na hora de inserir o processo na memória principal:

O estado de swapping

- O sistema pode admitir um processo novo (que acabou de ser criado); ou

- Ele pode carregar um processo previamente suspenso.

Entre as duas ações, o sistema deveria trazer o processo que já está suspenso, pois, ele já foi executado e já está no sistema. !!! Porém, esta abordagem apresenta dificuldades !!!

Um processo, para ir ao estado de suspenso, ele primeiramente estava no estado de bloqueado, com isso, ele não deveria ir para o estado de pronto, pois ele está esperando por algum evento !!!!

Com isso, o aspecto de *design* dos estados dos processos necessitam ser remodelados. Há dois conceitos independentes a serem considerados:

1 - Um processo está bloqueado esperando por um evento !!

2 - Um processo está bloqueado por falta de memória (swapping out).

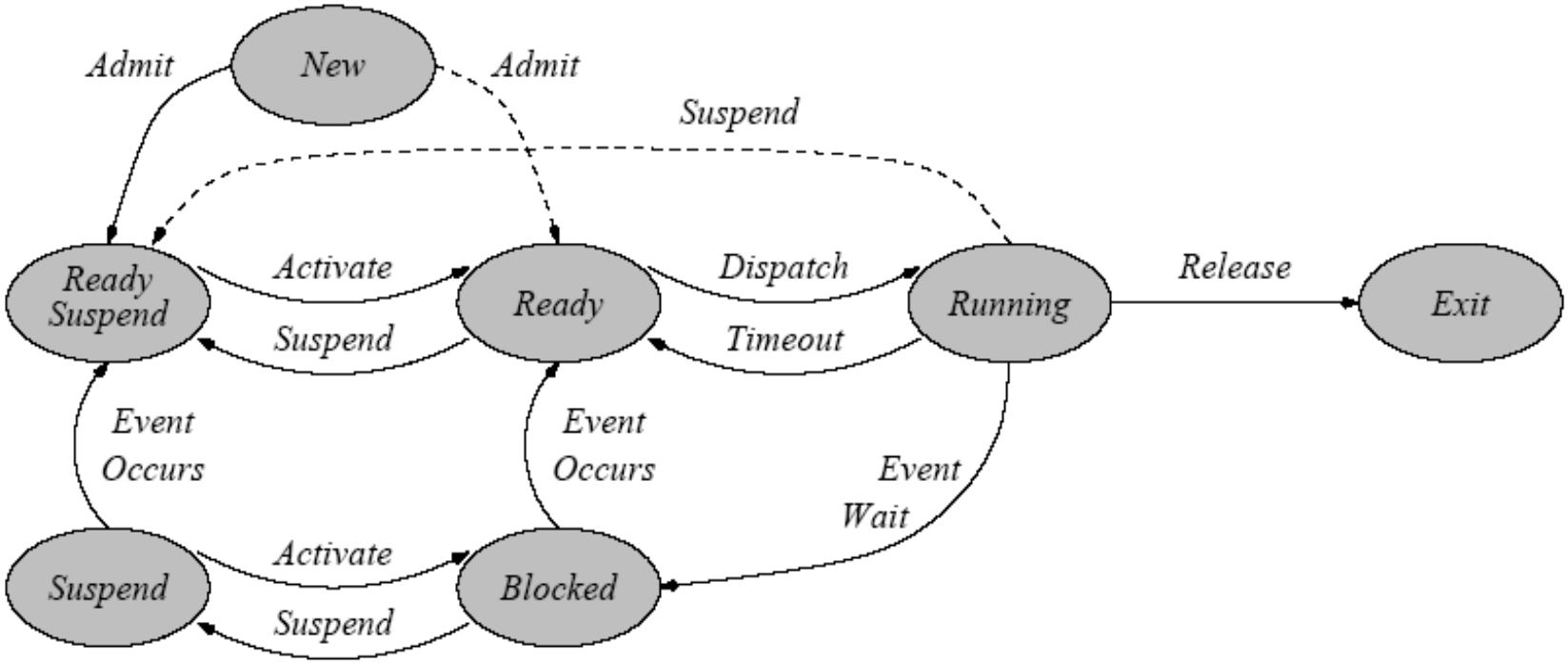
O estado de swapping

Com isso, temos 2 x 2 combinações de estados do processos

- Pronto: O processo está na memória principal e disponível para ser executado;
- Bloqueado: O processo está na memória principal esperando por um evento;
- Bloqueado/Suspenso: O processo está na memória secundária, esperando por um evento;
- Pronto/Suspenso: O processo está na memória secundária, mas está disponível para a execução.

O estado de swapping

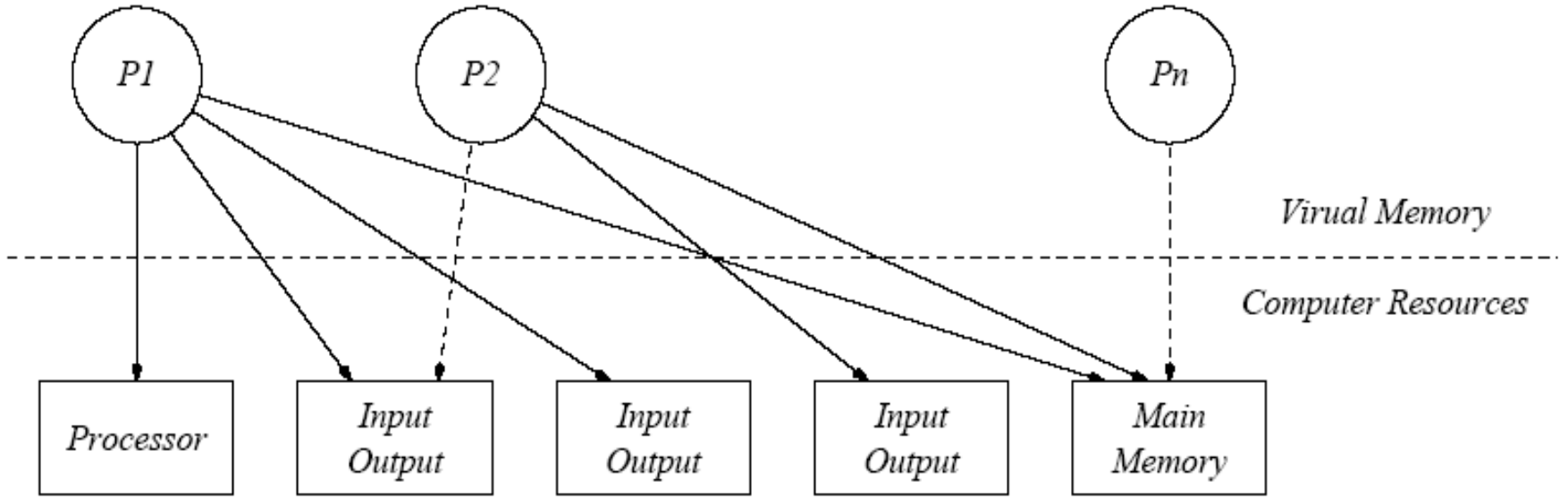
Diagrama de estados dos processos com os estados de bloqueado e suspenso



Descrição dos processos

Os sistemas operacionais controlam todos os eventos do sistema computacional. Ele escalona e envia processos para serem executados, aloca recursos para os processos e responde a requisições feitas por processos de usuários.

Fundamentalmente, nós podemos pensar no sistema operacional como uma entidade que gerencia recursos para os processos. A figura abaixo ilustra este caso.



Descrição dos processos

Para estudarmos o gerenciamento de processos, podemos fazer a seguinte pergunta:

Quais informações o sistema operacional precisa para controlar os processos e gerenciar os recursos para os processos ??

A estrutura de controle do sistema operacional

Se o sistema operacional é o gerenciador dos recursos e dos processos, ele deve ter informações sobre o estado corrente de cada processo e cada recurso.

A abordagem universal para esta questão é: O sistema operacional constrói e mantém tabelas de informações sobre cada entidade gerenciada, por exemplo: memória, dispositivos de I/O, arquivos e processos.

Tabela de memória: São usadas para guardar informações de ambas as memórias, primária e secundária. Uma parte da memória é reservada para o sistema operacional, o restante é disponível para os processos.

A tabela de memória deve guardar as seguintes informações:

- A alocação da memória principal para os processos;
- A alocação da memória secundária para os processos;
- Atributos de proteção a blocos segurança para a memória principal e secundária, tal que os processos possam acessar somente certas regiões compartilhada da memória.
- Informações necessária para o gerenciamento da memória virtual;

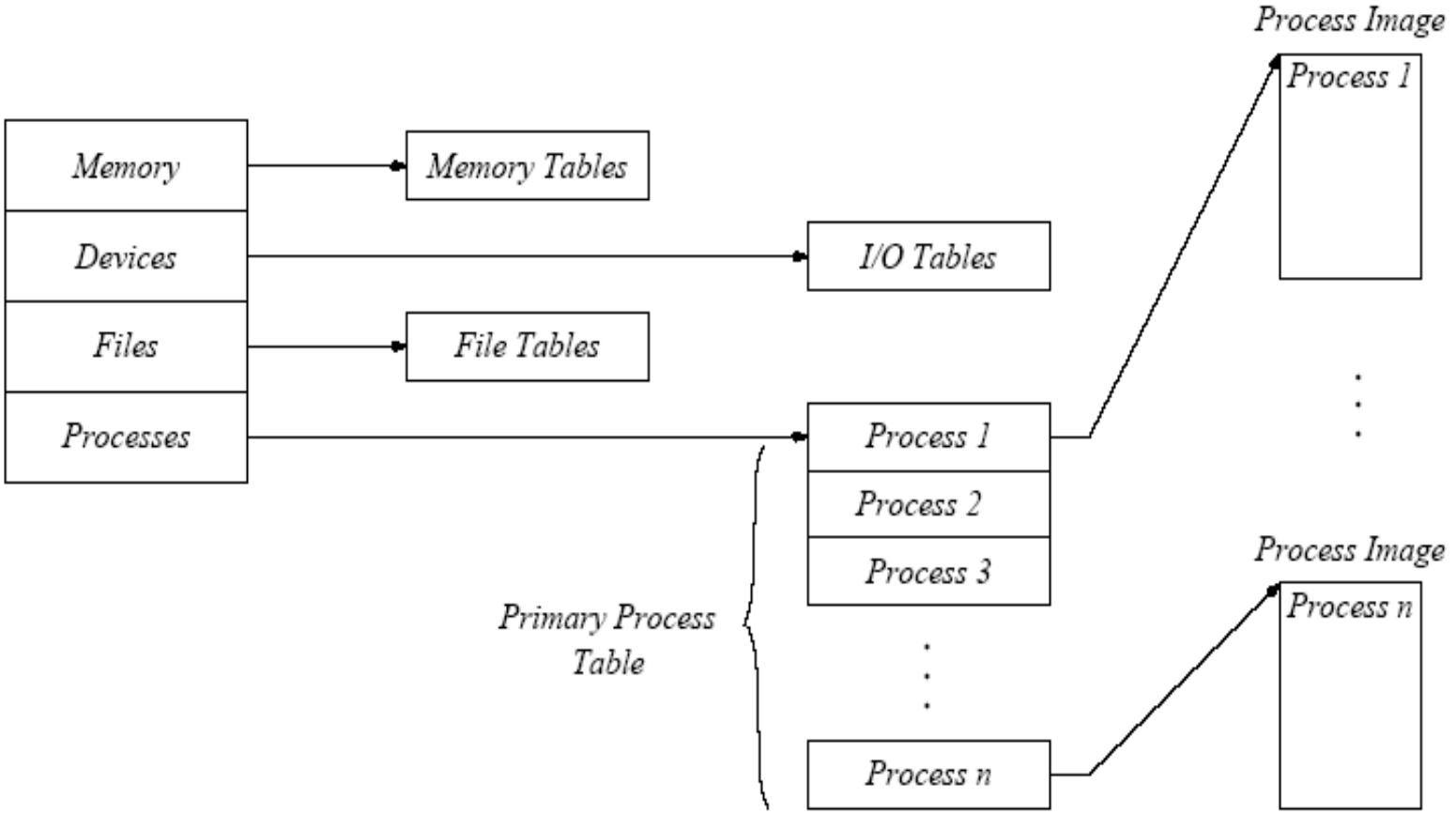
Tabela de I/O: São usadas pelo sistema operacional para gerenciar os dispositivos de I/O e canais do sistema computacional. Em algum determinado tempo, o dispositivo de I/O pode estar disponível para algum processo em particular;

Se alguma operação de *I/O* está em execução, o sistema operacional precisa saber o *status* desta operação e à localização na memória principal que está sendo usado como fonte ou destino desta operação;

Tabela de arquivos: Esta tabela é utilizada para manter informações sobre arquivos existentes, sua localização na memória secundária, seu status corrente e outros atributos, como por exemplo, atributo de leitura, escrita etc.

Tabela de processos: Esta tabela é utilizada para gerenciar todos os processos que estão ativos no sistema. Vamos detalhar o gerenciamento de processo.

A figura abaixo ilustra as tabelas mencionadas:



Para que o sistema operacional possa gerenciar os processo, primeiramente, ele deve conhecer a **localização dos processos**, tal como conhecer os **atributos dos processos** para efetuar o gerenciamento. Isto é conhecido como (*process control block*).

No mínimo, um processo deve possuir um programa ou um grupo de instruções que serão executados; Associado com este programa está a localização dos dados, tal como, as variáveis locais ou variáveis globais, como também as constantes definidas no programa.

Em adicional, a execução típica de um programa envolve uma pilha que é usada para guardar informações de procedimentos e passagem de parâmetros aos procedimentos.

Estas informações, dados, pilhas e atributos, são chamados de **imagem do processo**.

A localização da imagem de um processo dependerá do esquema de gerenciamento da memória usada. Em um simples caso, a imagem de um processo é mantida como sendo contínua, ou em blocos de memória.

Os blocos são mantidos na memória secundária, usualmente o HD. O sistema operacional pode gerenciar um processo, desde que uma parte da imagem do processo esteja na memória principal.

Os atributos dos processos

Em relação aos atributos dos processos, estes podem ser organizados em três categorias:

- a identificação do processo;
- a informação do estado do processador;
- a informação do estado do processo.

Identificação dos processos: Para cada processo é atribuído um número de identificação pelo sistema operacional. Dentre este identificador estão:

- O identificador do processo;
- O identificador do processo que o criou (processo pai);
- O identificador do usuário do processo;

Identificação do estado do processador: Consiste nos conteúdos dos registradores do processo , tal como contador de programa (PC), apontadores da pilha etc.

Identificação do estado dos processos: Estas são as informações necessárias para o sistema operacional controlar e coordenar os vários processos que estão em execução.