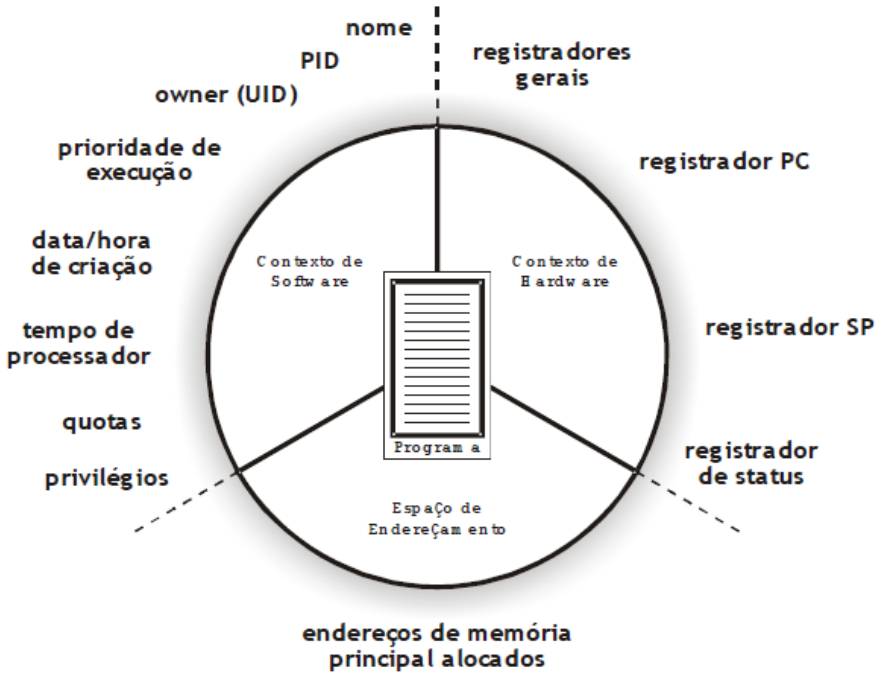
MATERIAL DE APOIO RESUMO 5	
DISCIPLINA: Sistemas Operacionais	**PROFESSOR: Lincon M. Peretto**
BIBLIOGRAFIA: MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de Sistemas Operacionais**. Capítulo 4. 4ª ed. São Paulo: LTC, 2007.	

Processos

Processo: Tudo que compete por recurso de máquina, ou seja, utiliza-se dos recursos disponíveis como memória principal, processador. Nos sistemas multiprogramáveis o SO é responsável pela gerência dos processos e recursos, para que isso funcione cada programa deve estar associado a um processo. Através dos processos é possível utilizar os recursos como, por exemplo, dispositivos de E/S.

***Estrutura do processo*:** Nos sistemas multiusuários, cada processo é associado a um usuário, portanto ao executar um programa temos a impressão de que todos os recursos estão disponíveis para execução deste programa em particular, porém isso é só impressão e os recursos do sistema continuam compartilhados, sendo utilizados de acordo com o possível. Este fenômeno recebe o nome de pseudo-paralelismo. O processo é formado por 4 componentes: Contexto de Hardware, Contexto de Software, Espaço de endereçamento e Bloco de Controle do Processo.



***Contexto de Hardware*:** Armazena a informação de hardware para que o processo possa voltar a ser executado, ou seja, armazena informação dos status dos registradores.

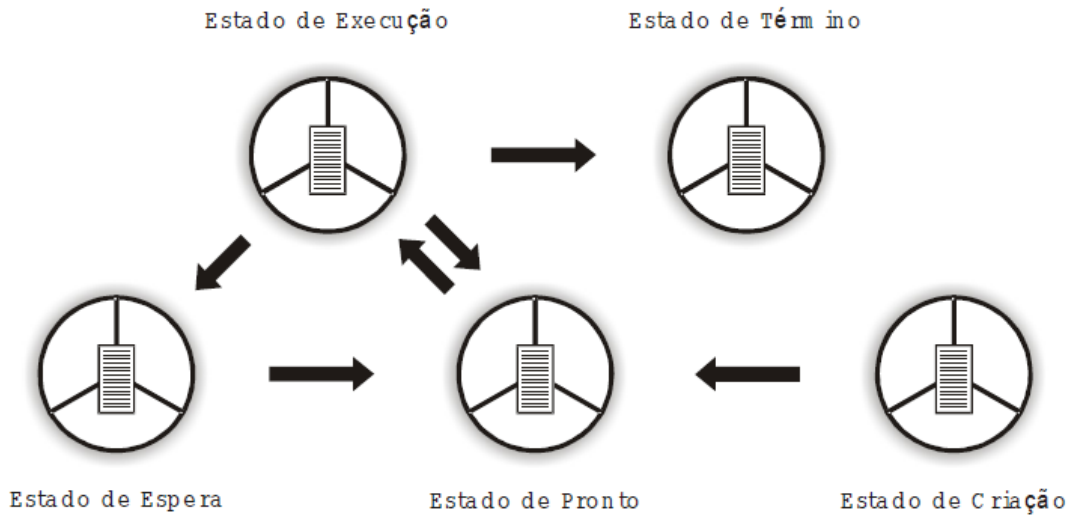
Contexto de software: são as características de cada processo como, por exemplo: limites de uso dos recursos, número máximo de arquivos abertos, prioridade na execução, PID (número único de identificação do processo), quotas, número máximo de arquivos abertos, tamanho que ele pode utilizar da memória principal.

***Espaço de endereçamento*:** é a área de memória pertencente ao processo onde serão gravadas as instruções e dados dos programas. Este espaço deve ser protegido para que outros processos não tenham direito de acesso sobre os processos armazenados.

***Bloco de controle do Processo (PCB)*:** O processo é implementado pelo SO através de uma estrutura PCB armazenado na memória principal em uma área exclusiva do SO. Nela fica tudo que o processo precisa saber:

- Prioridade
- Localização e tamanho da memória ocupado
- Identificação dos arquivos abertos no momento
- Tempo de processador utilizado
- Estados do processo

***Estados do Processo*:** Execução, Pronto e Espera.



***Execução**:* É quando o processo está em execução pelo SO, ou seja, o processo está ocupando o processador.

***Pronto**:* O processo está pronto quando está aguardando o SO para execução, o que ocorre seguindo os critérios de escalonamento.

***Espera**:* Um processo em espera aguarda algum comando ou a liberação de algum recurso para entrar em execução, normalmente uma interrupção.

Mudanças de estado: um processo muda de estado durante a sua execução. Isso ocorre através de eventos gerados pelo próprio processo ou quando solicitados pelo SO. Ex: Interrupção, Chamada de Sistema, Escalonamento, Término do tempo de execução.

Interrupção**:** Durante a execução de um programa podem ocorrer situações (externas ao processo) onde será necessário interromper a execução e voltar após uma determinada solicitação, este é o principio dos sistemas multiprogramáveis. A esta atividade de interromper uma tarefa e retornar posteriormente damos o nome de interrupção.

Chamada de Sistema: Consiste na requisição de um serviço ao sistema operacional por parte da aplicação. Garantindo assim que a aplicação não terá acesso direto ao núcleo do sistema evitando danos que possam comprometer a utilização do SO.

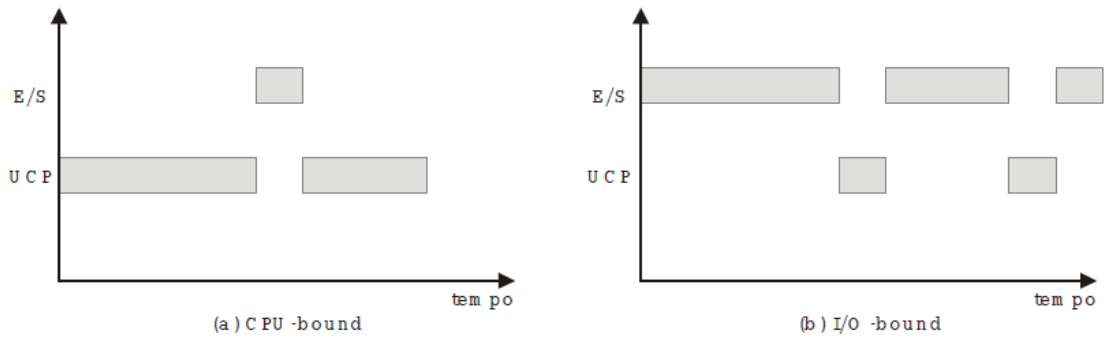
Criação e Eliminação de Processos: Processos são criados e eliminados por diferentes motivos. A criação ocorre a partir do momento que o PCB é criado e o espaço de endereçamento de memória é alocado e eliminado quando o PCB é eliminado e os recursos desalocados.

Processos Foreground (Primeiro Plano): É aquele que permite a comunicação direta entre o usuário e o processo, neste caso aguarda a entrada através do teclado, mouse, monitor etc.

Processos Background (Segundo Plano): Não há iteração do usuário no processo, neste caso os canais de E/S não estarão relacionados a nenhum dispositivo de E/S interativos, mas sim a arquivos de E/S.

Processos CPU-Bound: São processos que passam a maior parte do tempo em execução, ou seja, ocupando o processador. Ex: aplicações cientificas que usam muitos cálculos**.**

I/O – Bound: São processos que passam a maior parte do tempo em espera. Ex: aplicações comerciais.



Formas de Criação de Processos: Logon Interativo, Linguagem de Comandos ou Rotina do Sistema Operacional.

Logon Interativo: O usuário por intermédio de um terminal fornece ao sistema um nome de identificação (username ou logon) e uma senha (password). O SO autentica estas informações verificando se estão corretas, em caso positivo, um processo foregroung é criado, possibilitando ao usuário interagir com o sistema utilizando uma linguagem de comandos. O arquivo de usuários é um arquivo do sistema operacional onde são armazenados todos os usuários autorizados a ter acesso ao sistema. O processo também pode eliminado interativamente quando o usuário realiza o procedimento de saída do sistema, também chamado logout ou logoff, digitando um comando para encerramento da sessão.

Linguagem de Comandos: Um usuário pode a partir do seu processo, criar novos processos por intermédio de comandos da linguagem de comandos. O principal objetivo para que um usuário crie diversos processos é a possibilidade de execução de programas concorrentemente.

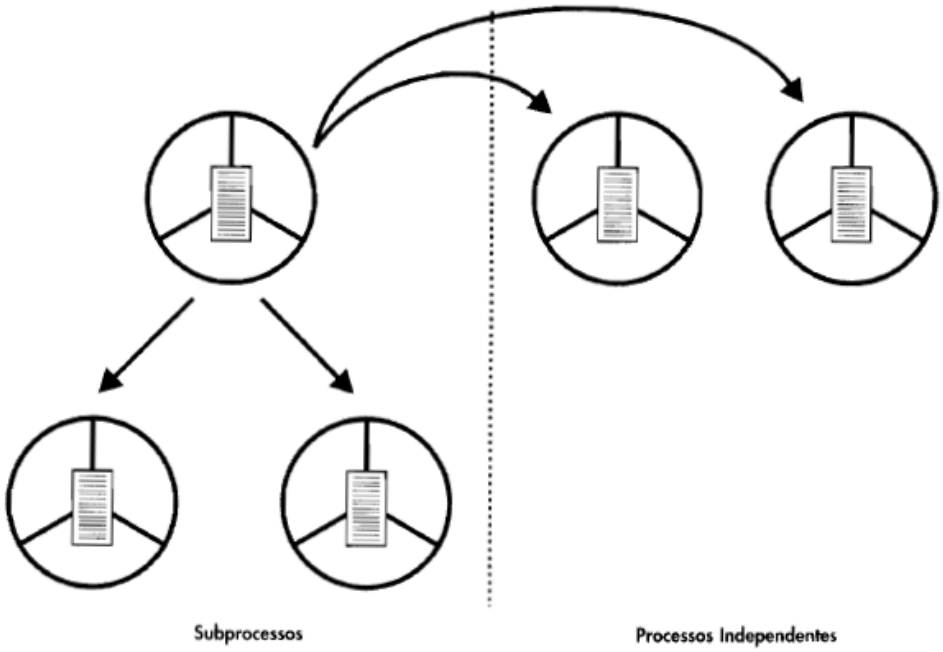
Rotina do Sistema Operacional: um processo pode ser criado a partir de qualquer programa executável com o uso de rotinas do SO. A criação deste processo possibilita a execução de outros programas concorrentemente ao programa chamador. A rotina depende do SO e possui diversos parâmetros de criação. Abaixo exemplo de criação do processo Notepad.exe (Bloco de Notas) no Windows utilizando a linguagem de programação Delphi:

```
procedure TForm1.CriaProcesso(Sender: TObject);
var
    status: boolean;
    si: STARTUPINFO;
    pi: PROCESS_INFORMATION;
    Handle: THandle;
    NomeExe: PChar;
begin
    NomeExe := PChar('\WINNT\NOTEPAD.EXE');
    FillChar(si, SizeOf(si), 0);
    si.cb := SizeOf(si);
    status := CreateProcess(NomeExe, nil, nil, nil, TRUE,
                            NORMAL_PRIORITY_CLASS, nil, nil, si, pi);
    if (not status) then MessageBox (Handle, 'Erro na criação do
                                      processo', nil, MB_OK);
   end;
```

Processos Independentes, Subprocessos e Threads: São maneiras diferentes de se implementar a concorrência dentro de uma aplicação. Busca-se subdividir o código em partes para trabalharem de forma cooperativa. No caso de acesso a banco de dados onde as solicitações ocorrem em grande quantidade este processo se torna muito importante.

Processos Independentes: Maneira mais simples de implementar a concorrência, neste caso não existe vínculo do processo criado com o seu criador, exigindo a alocação de um PCB, contextos de hardware e software e espaço de endereçamento próprios.

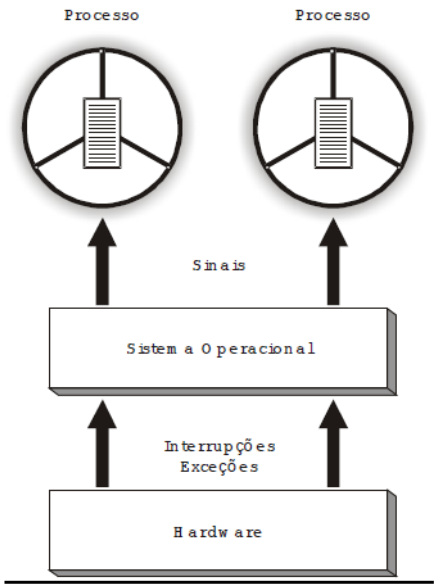
Subprocessos: são processos criados dentro de uma estrutura hierárquica. Neste modo o processo criador é denominado processo-pai, enquanto o novo processo é chamado de subprocesso ou processo-filho. Uma característica desta criação é dependência existente entre o processo criador e o subprocesso. Neste modelo, caso o processo-pai deixe de existir os subprocessos são automaticamente eliminados, além de cada um possuir seu próprio PCB.



Processos do Sistema Operacional: Os processos também podem ser implementados na arquitetura do sistema operacional disponibilizando serviços para processos das aplicações e do próprio sistema operacional. Quando processos são utilizados para implementação de serviços do sistema, estamos retirando código do seu núcleo, tornando-o menor e mais estável. No caso de um ou mais serviços não desejados, basta não ativar os processos responsáveis, liberando mais memória para o processo dos usuários. Exemplos de serviços:

- Auditoria e segurança;
- Serviços de rede;
- Contabilização de uso de recursos;
- Contabilização de erros;
- Gerência de impressão;
- Interfaces de comando (Shell).

Sinais: É um mecanismo que permite notificar o processo através de eventos gerados pelo SO ou por outros processos, o uso dos sinais é importante para gerência de processos além de possibilitar a comunicação e sincronização entre os processos. A maior parte dos sinais é gerada pelo SO ou hardware. Ex: falta de espaço em disco, a sequência de caracteres Ctrl+C para interromper a execução de programa.

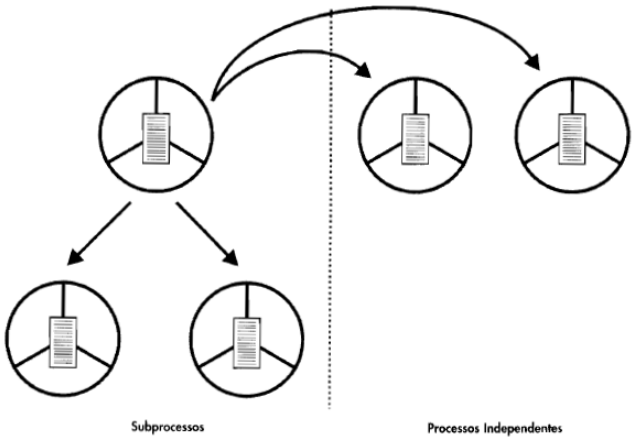


Threads: Os modelos anteriores apresentam problemas que estão relacionados ao consumo de recursos já que para cada novo subprocesso é necessário alocar novos recursos e concorrência já que a comunicação e sincronização são consideradas pouco eficientes já que cada processo possui seu próprio PCB. As threads foram introduzidas na tentativa de reduzir o tempo gasto na criação, eliminação e troca de contextos nas aplicações concorrentes, bem como economizar recursos do sistema. Threads compartilham o processador da mesma maneira que um processo, porém compartilham alguns recursos tornando a comunicação mais simples e rápida.

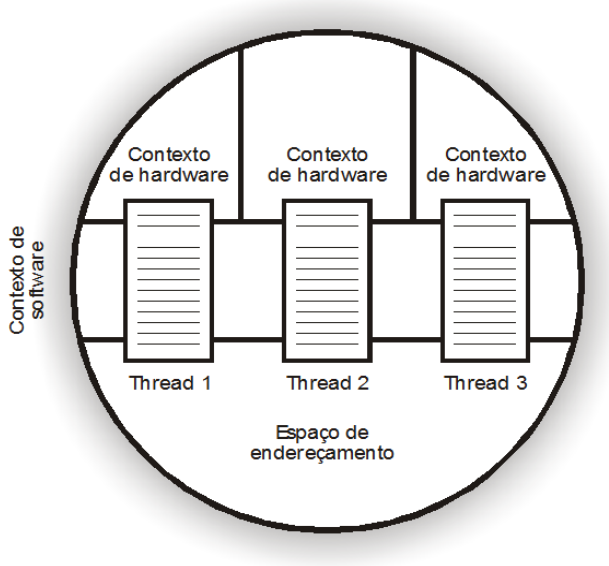
Até o final da década de 1970, sistemas operacionais como Tops-10 (DEC), MVS (IBM) e Unix (BellLabs), suportavam apenas processos com uma única thread (monothread). Em 1979, durante o desenvolvimento do SO Toth, foi introduzido o conceito de processos lightweight (peso leve), onde o espaço de endereçamento de um processo era compartilhado por vários programas. Apesar de revolucionária, a ideia não foi utilizada comercialmente e somente em meados de 1980, com o desenvolvimento do SO Mach, ficou clara a separação entre o conceito de processo e thread.

A partir do conceito de multithread é possível projetar e implementar aplicações concorrentes de forma eficiente, pois um processo pode ter partes diferentes do seu código sendo executadas concorrentemente, com um menor overhead do que utilizando múltiplos processos.

Ambiente monothread: Um processo suporta apenas um programa com isso é demandado uma maior utilização dos recursos computacionais, outro problema ocorre na utilização de endereçamentos de memória já que cada processo utiliza um espaço específico. O problema neste tipo de implementação é que o uso de processos no desenvolvimento de aplicações concorrentes demanda consumo de diversos recursos do sistema, ou seja, sempre que um novo processo é criado, o sistema deve alocar recursos para cada processo, como observado anteriormente na utilização de sub-processos ou processos independentes. O MS-DOS e as primeiras versões do Windows são exemplos de sistemas monothreads.



Ambiente multithread: Não existe a idéia de um programa associado ao processo, mas sim, a threads compartilhando o contexto de software e endereçamento de memória. De forma simples, podemos definir uma thread como sub-rotinas de um programa, ou seja, ela pode ser executada paralelamente ao programa. As threads possuem as mesmas formas de utilização do processador e também a mesma ideia de funcionalidade e estado dos processos.



Thread é uma forma de o processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas simultaneamente. Ela permite que o usuário do programa, utilize uma funcionalidade do ambiente enquanto outras threads realizam outros cálculos e operações.

O benefício do uso de threads advém do fato do processo poder ser dividido em mais de uma linha de tarefas, enquanto uma linha está esperando determinado dispositivo de I/O ou qualquer outro recurso do sistema, o processo como um todo não fica parado, pois quando uma thread entra no estado de espera, outra thread aguarda na fila de aptos para executar.

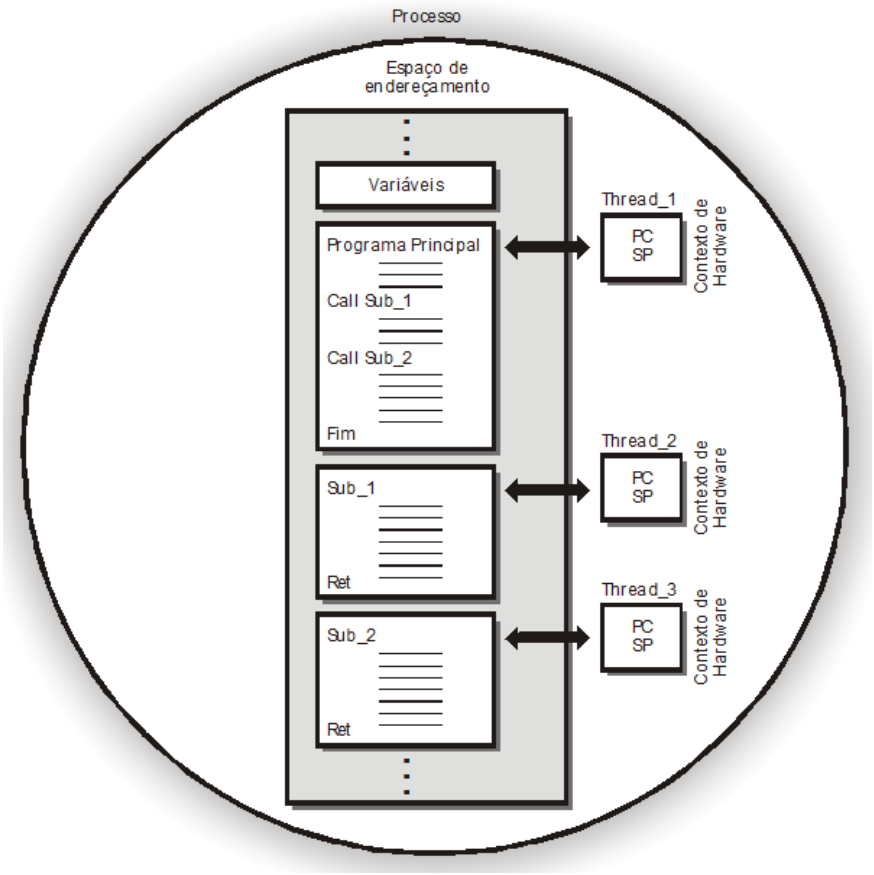
Comportamentos de uma thread:

- Pode criar outra da mesma forma que um processo;
- Pode esperar outra para se juntar;
- Pode voluntariamente desistir, por não ser mais preciso;
- Pode se duplicar.

Características de uma thread:

- Também chamado de processo leve, pois como compartilham o mesmo contexto de software, são mais rápidas;
- Criar um processo pode ser caro em termos de tempo, memória, e sincronização entre processos;
- Threads podem ser criadas sem a replicação do processo inteiro;
- Como partilham o mesmo espaço de endereçamento, a comunicação entre elas é mais rápida.
- O tempo gasto para troca de threads é menor, em parte porque não há necessidade de troca de espaço de endereçamento.

No exemplo abaixo, existe um programa principal que realiza a chamada de duas sub-rotinas assíncronas (Sub_1 e Sub_2). Inicialmente, o processo é criado apenas com a Thread_1 para a execução do programa principal. Quando o programa principal chama as rotinas Sub_1 e Sub_2, são criadas as Thread_2 e Thread_3, respectivamente, e executados independentemente do programa principal. Neste processo, as três threads são executadas concorrentemente.


Processo
Espaço de endereçamento
Variáveis
Programa Principal
Call Sub_1
Call Sub_2
Fim
Sub_1
Ret
Sub_2
Ret
Thread_1
PC
SP
Contexto de Hardware
Thread_2
PC
SP
Contexto de Hardware
Thread_3
PC
SP
Contexto de Hardware