Chp. 4 - Threads, SMP, and Microkernels

4.1 - Processos e Lightweight Process

4.2 - Multiprocessamento Simétrico

4.3 - Projeto Orientado a Microkernel

4.4 - Gerenciamento de SMP e Threads no Windows NT

4.5 - Gerenciamento de SMP e Threads no Solaris

... Chp. 4 - Threads, SMP, and Microkernels

✭ William STALLINGS; **Operating Systems: Internals and Design Principles**, New Jerssey, Prentice-Hall, 1998, ISBN: 0-13-887407-7

✭ Eleri CARDOZO; Maurício MAGALHÃES; Luís F. FAINA; **Sistemas Operacionais**, Dep. de Eng. de Computação e Automação Industrial, Fac. de Engenharia Elétrica e de Computação, UNICAMP, 1996.

4.1 - Processos e Lightweight Process

✰ **conceito de processo:** envolve a coordenação (junção) de duas características:

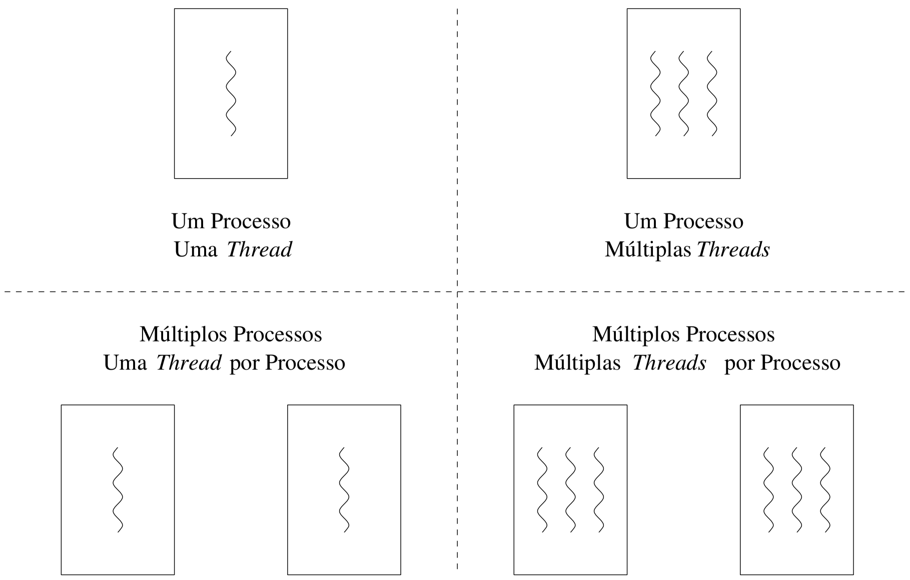
- unidade de alocação de recurso: ao processo é atribuído um espaço de endereçamento virtual que de tempo em tempo utiliza-se da memória principal, bem como do controle de outros recursos (p. ex., entrada/saída, arquivos, etc.);
- unidade escalonável: define um caminho (rota) de execução em um ou mais programas e, assim possui um estado que a descreve.

✰ **thread (lightweight process):** surge da idéia de que em alguns S.O. estas duas características são distintas e portanto podem ser tratadas de forma independente;

- ... *lightweight processes* acomodadas dentro de um processo são caracterizados pelo contador de programa, registradores e pilha daquele processo;
- ... *lightweight processes* compartilham código, dados e recursos (espaço de endereçamento, variáveis globais, arquivos, etc.) — daí seu nome.
- ... as *threads* operam como processos e, portanto, possuem estado e podem criar outras *threads*, alocar recursos, etc.

... 4.1 - Processos e Lightweight Process

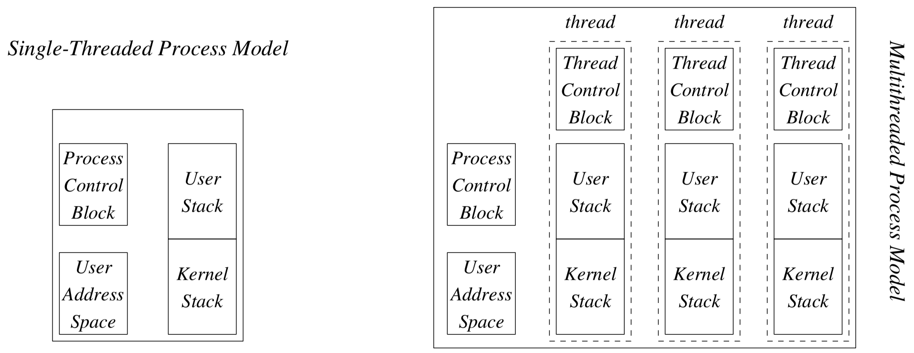
✰ *multithreading* refere-se a habilidade do sist. oper. suportar múltiplas *threads* dentro de um único processo — algumas abordagens comumente suportadas:



... 4.1 - Processos e Lightweight Process

✡ ... dentro de um processo, podemos ter uma ou mais *threads*, cada uma delas apresentando as seguintes características:

- estado de execução (executando, pronta para ser executada, etc.);
- contexto da *thread* quando esta não estiver sendo executada;
- pilha de execução;
- algumas variáveis locais de armazenamento para cada *thread*;
- acesso à memória e recursos do seu processo, compartilhados com todas as outras *threads* pertencentes ao processo.



... 4.1 - Processos e Lightweight Process

✡ ... a *thread* constitui-se em uma unidade de escalonamento — solução para melhoria do desempenho, por reduzir o tempo para a troca do contexto que, neste caso, é menor que de um processo;

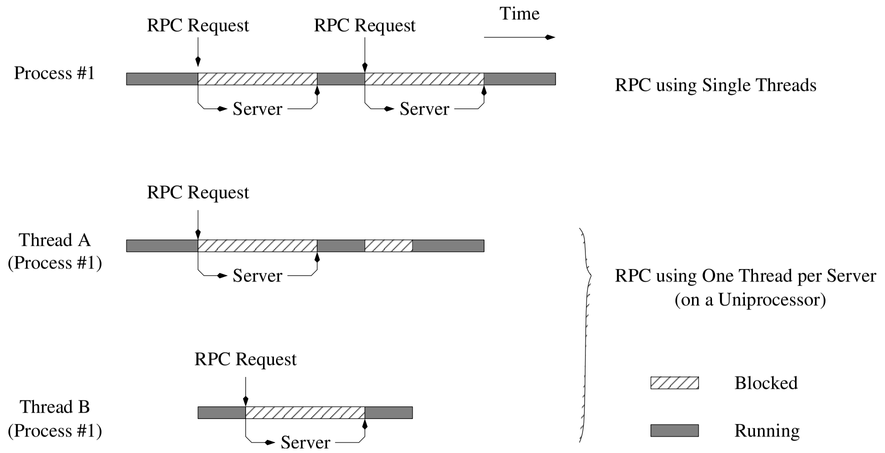
- ... se no entanto, no referirmos ao escalonamento de *threads* pertencentes a processos diferentes, será necessário a troca total de contexto;

- ... um aspecto interessante no conceito de *thread* é a possibilidade de explorarmos a concorrência entre processos através, por exemplo, do compartilhamento de memória.

... 4.1 - Processos e Lightweight Process

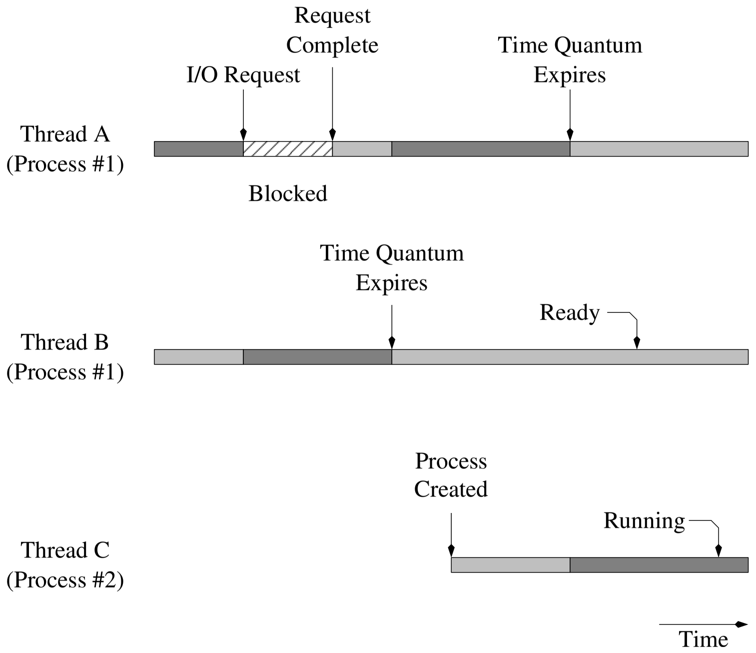
✰ Assim como nos processos, *threads* podem assumir um dos 03 estados: **running**, **ready** e **blocked**; ... não faz sentido associarmos o estado de **suspend** a *threads*, dado que é um conceito no nível de processo.

✰ Outro aspecto refere-se a discussão dos benefícios advindos do uso de *threads* que não bloqueiam o processo como um todo, para tanto considere o exemplo de RPC:



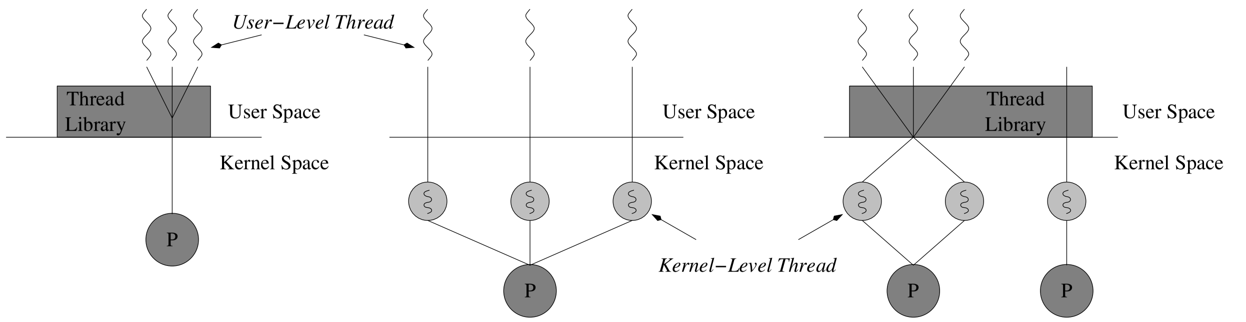
... 4.1 - Processos e Lightweight Process

✰ Ainda em um Sistema Monoprocessado, a multiprogramação possibilita a sobreposição de múltiplas *threads* dentro de múltiplos processos:



... 4.1 - Processos e Lightweight Process

✡ Há 02 grandes categorias de implementação de *threads: user-level threads* e *kernel-level threads* ou também referenciada como *lightweight processes*.



✡ **user-level thread:** quando somente *threads* no nível do usuário estão presentes, todo o seu gerenciamento é feito pela aplicação e, assim, o *kernel* não toma conhecimento da existência das *threads*;

- ... qualquer aplicação pode ser programada para acomodar múltiplas *threads*, para tanto, basta utilizar uma biblioteca de *threads*.

... 4.1 - Processos e Lightweight Process

✡ Vantagens no uso da abordagem ULTs sobre KLTs:

❶ chaveamento de *threads* não requer privilégios do modo *kernel* pois todas as informações (estruturas de dados) para o gerenciamento das *threads* estão contidas no espaço de endereçamento de um único processo;
❷ escalonamento dependente da aplicação, o que possibilita variá-lo conforme as necessidades contempladas pela aplicação;
❸ pode ser executada em qualquer sistema operacional, não exigindo nenhuma mudança no *kernel* de forma a suportar tal funcionalidade.

✡ Desvantagens no uso da abordagem ULTs sobre KLTs:

❶ na maioria dos sistemas operacionais, muitas das chamadas são bloqueantes o que leva ao bloqueio de todo o processo, não só da *thread* que executou a chamada;
❷ aplicações *multithreaded* na abordagem de *threads* no nível do usuário não tiram vantagem de sistemas com suporte ao multiprocessamento.

... 4.1 - Processos e Lightweight Process

✰ **kernel-level thread:** quando somente *threads* no nível do *kernel* estão presentes, nenhum código de gerenciamento está presente na aplicação, ou seja, encontramos somente um API de *threads*;

- ... nesta abordagem, uma aplicação com suporte a múltiplas *threads* deixa a cargo do sistema operacional toda a manutenção e gerenciamento das informações relacionadas às *threads* e aos processos.

✰ Vantagens no uso da abordagem KLTs sobre ULTs:

❶ *kernel* pode simultaneamente escalonar múltiplas *threads* para o mesmo processo sobre uma plataforma multiprocessada;
❷ se um *thread* em um processo é bloqueada, o *kernel* escalona uma outra *thread* para o mesmo processo, ou seja, o processo não é inteiramente bloqueado;
❸ as próprias rotinas do *kernel* podem utilizar-se da abordagem *multithreaded*.

... 4.1 - Processos e Lightweight Process

✰ Principal desvantagens no uso da abordagem KLTs sobre ULTs:

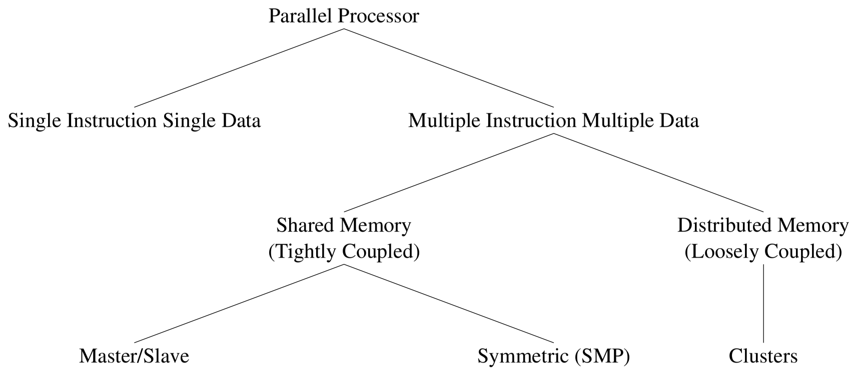
- a transferência do controle de uma *thread* para outra dentro do mesmo processo requer a mudança do modo de execução (usuário para *kernel*);

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources	Most UNIX Implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:M	Combines attributes of M:1 and 1:M cases.	TRIX

4.2 - Symmetric Multiprocessing - SMPs

✡ A medida que a tecnologia dos computadores evolui e o custo do *hardware* cai, projetistas têm se deparado com mais e mais situações onde o paralelismo pode ser utilizado — melhorando a performance e até mesmo a confiabilidade;

- ... duas são as abordagens comuns que provêem suporte ao paralelismo: *symmetric multiprocessing* e *clusters* de processadores.
- *symmetric multiprocessing* - ainda bem aceita, a taxonomia introduzida por Flynn enfatiza sistemas de processamento paralelo subdividindo-os em quatro categorias;



... 4.2 - Symmetric Multiprocessing - SMPs

✡ **clusters:** caracterizados por processadores que possuem memória dedicada e comunicação suportada por rede ou caminhos pré-definidos;

- ... se os processadores compartilham uma memória, então cada processador acessa programas e dados armazenados na memória compartilhada e os processadores comunicam um com outro através da memória — **sistemas multiprocessados com memória compartilhada**.

✡ Uma classificação geral de multiprocessadores com memória compartilhada é baseada em como os processos são atribuídos aos processadores:

- ✯ **master/slave** - o *kernel* do S.O. sempre é executado em um processador em particular tendo os demais a tarefa de executar os processos e utilitários;
- ✯ **symmetric multiprocessor** - o *kernel* pode ser executado em qualquer processador e, tipicamente, cada processador escalona seu próprio conjunto de processos e *threads* — esta abordagem torna o S.O. mais complexo.

4.3 - Projeto Orientado a Microkernel

✰ **microkernel:** é um pequeno núcleo de sistema operacional que através de um *framework* possibilita futuras extensões (o termo é discutível e tem sido abordado de maneiras diferentes por diferentes sistemas operacionais).

✰ Por ainda se tratar de um conceito **impreciso**, várias são as perguntas:

- quão pequeno o *kernel* deve ser para ser qualificado como *microkernel* ?
- como projetar *drivers* de dispositivos para se obter a melhor performance enquanto se abstrai as funções do *hardware*?
- onde executar operações que não sejam do *kernel* ? ... modo *kernel* ou usuário ?
- onde manter o código do sub-sistema ?

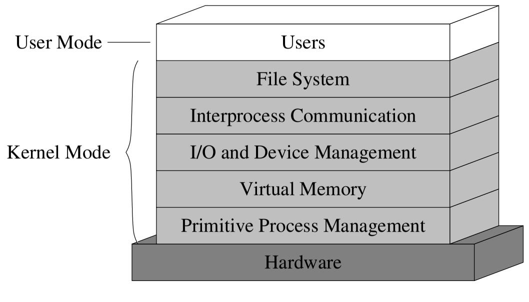
✰ A abordagem *microkernel* se popularizou no Sistema Operacional Mach, que em teoria ofereceria alto grau de flexibilidade e modularidade;

- ... outro exemplo é o Windows NT onde não só a modularidade justifica o seu uso mas também a portabilidade são tidos como benefícios chaves.

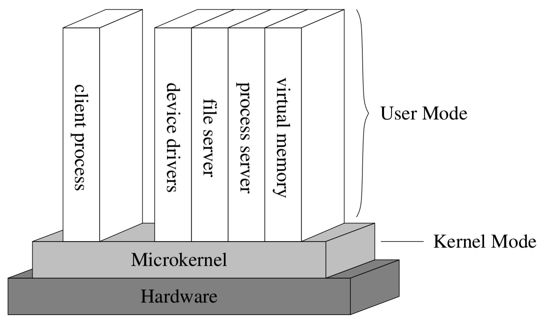
... 4.3 - Projeto Orientado a Microkernel

✰ *layered operating systems* - funções são organizadas hierarquicamente e interações somente se dão entre camadas adjacentes sendo que a maioria das camadas ou suas funcionalidades são executadas no modo *kernel*.

✰ *microkernel* - somente as funções absolutamente necessárias estão disponibilizadas no *kernel* do sistema operacional, assim, muitas dos serviços e aplicações essenciais são construídos sobre o *kernel* e executados no modo usuário.



Layered Kernel



Microkernel

... 4.3 - Projeto Orientado a Microkernel - Benefícios

- **interfaces uniformes:** serviços são providos por meio de envio de mensagens, assim não há distinção para os processos de serviços no nível do *kernel* e do usuário;
- **extensibilidade:** facilita a adição de novos serviços, bem como o suporte a múltiplos serviços na mesma área funcional (modificações não implicam mudanças no *kernel*);
- **flexibilidade:** não somente funcionalidades podem ser alteradas mas, também, aspectos já contemplados podem ser retirados e/ou modificados para tornar a implementação mais eficiente;
- **portabilidade:** aspecto atrativo para sistemas operacionais (todo o código dependente do processador está no *microkernel*);
- **confiabilidade:** quanto maior o *software*, maior é a dificuldade de se conseguir confiabilidade e nesse sentido, um *microkernel* pode ser rigorosamente testado;

... 4.3 - Projeto Orientado a Microkernel - Benefícios

- **suporte a sistemas distribuídos:** se um sist. distribuído é configurado de tal forma que todos os processos e serviços têm identificação única, então temos como efeito uma única imagem do no nível do *microkernel*:
  - um processo pode enviar uma mensagem sem saber em que máquina o serviço requisitado está residindo e/ou localizado.
- **sist. oper. orientados a objetos:** a orientação a objetos pode impor disciplina no projeto do *microkernel*, bem como no desenvolvimento de módulos de extensão no sistema operacional:
  - ... como resultado um grande esforço no projeto de *microkernel* tem sido no uso de técnicas de orientação a objetos;
  - ... uma abordagem nessa linha é o uso de componentes (objetos com interfaces bem definidas que podem ser interconectados para formar um *software* na modalidade de *building blocks*).

4.4 - Gerenciamento de SMP e Threads no Windows

✰ Processo no Windows NT contemplam as necessidades de uma variedade de ambientes de sistemas operacionais, o que exige uma grande flexibilidade de aspectos a serem suportados:

- como os processos são nomeados, representados e protegidos;
- como oferecer suporte a *threads* dentro de processos;
- mecanismos para comunicação entre processos;
- como processos se relacionam com outros processos, etc.

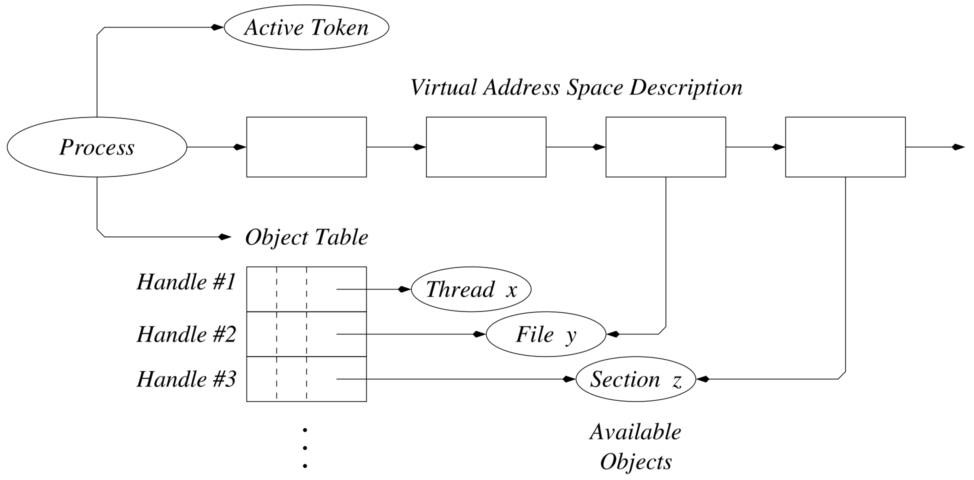
✰ Características importantes dos processos no NT:

- processos são implementados como objetos;
- um processo pode acomodar uma ou mais *threads*;
- ambos processos e *threads* oferecem mecanismos para sincronização.

✰ No Windows NT os serviços e estruturas de processos suportadas pelo *kernel* são relativamente simples e de propósito geral, permitindo assim, que cada sub-sistema emule uma particular funcionalidade e estrutura de processo.

... 4.4 - Gerenciamento de SMP e Threads no Windows

✰ No Windows NT a forma pela qual processos se relacionam com os recursos que ele utiliza e controla pode ser vista no diagrama abaixo:



... 4.4 - Gerenciamento de SMP e Threads no Windows

✰ A estrutura orientada a objetos do Windows NT teve por objetivo facilitar o desenvolvimento de facilidades gerais de processos:

Object Type	Process
Object Body Attributes	Process ID Security Descriptor Base Priority Default Processor Affinity Quota Limits Execution Time I/O Counters VM Operation Counters Exception/Debugging Ports Exit Status
Services	Create Process Open Process Query Process Information Set Process Information Current Process Terminate Process Allocate/Free Virtual Memory Read/Write Virtual Memory Protect Virtual Memory Query Virtual Memory Lock/Unlock Virtual Memory Flush Virtual Memory

Object Type	Thread
Object Body Attributes	Thread ID Thread Context Dynamic Priority Base Priority Thread Processor Affinity Thread Execution Time Alert Status Suspension Count Impersonation Token Termination Port Thread Exit Status
Services	Create Thread Open Thread Query Thread Information Set Thread Information Current Thread Terminate Thread Get Context Set Context Suspend Resume Alert Thread Test Thread Alert Register Termination Port

... 4.4 - Gerenciamento de SMP e Threads no Windows

✰ Atributos para objetos Processos no Windows NT:

Process ID	A unique value that identifies the process to the operating system.
Security Descriptor	Describes who created an object, who can gain access to or use the object, and who is denied access to the object.
Base Priority	A baseline execution priority for the process's threads.
Default Processor Affinity	The default set of processors on which the process's threads can run.
Quota Limits	The maximum amount of paged and nonpaged system memory, pagin file space, and processor time a user's processes can use.
Execution Time	The total amount of time all threads in the process have executed.
I/O Counters	Variables that record the number and type of I/O operation that the process's threads have performed.
VM Operating Counters	Variables that record the number and types of virtual memory operations that the process's threads have performed.
Exception/Debugging Ports	Interprocess communication channels to which the process manager sends a message when one of the process's threads causes an exception.
Exit Status	The reason for a process's termination

... 4.4 - Gerenciamento de SMP e Threads no Windows

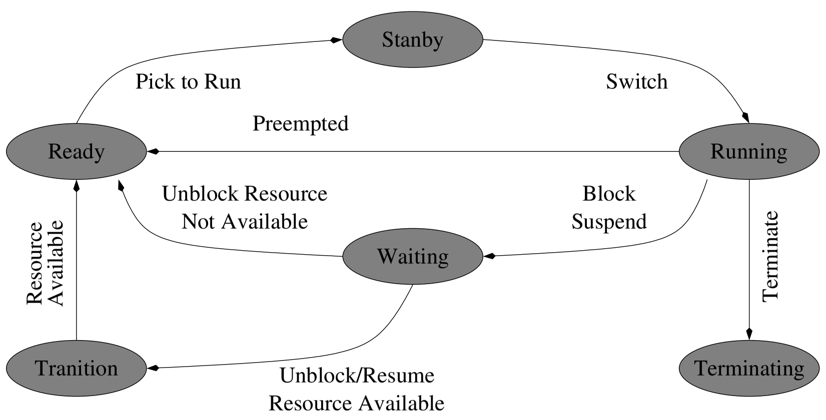
✰ Atributos para objetos Threads no Windows NT:

Thread ID	A unique value that identifies a thread when it calls a server.
Thread Context	The set of register values and other volatile data that defines the execution state of a thread.
Dynamic Priority	The thread's execution priority at any given moment.
Base Priority	The lower limit of the thread's dynamic priority.
Thread Processor Affinity	The set of processors on which the thread can run, which is a subset or all of the processor affinity of the thread's process.
Thread Execution Time	The cumulative amount of time a thread has executed in user mode and in kernel mode.
Alert Status	A flag that inidicates whether the thread should execute an assynchronous procedure call.
Suspension Count	The number of times the thread's execution has been suspended without being resumed.
Impersonation Token	A temporary access token allowing a thread to perform operations on behalf of another process (user by subsystems).
Termination Port	An interprocess communication channel to which the process manager sends a message when the thread terminates (user by subsystems).
Thread Exit Status	The reason for a thread's termination.

... 4.4 - Gerenciamento de SMP e Threads no Windows

✰ **stanby:** uma *thread* foi selecionada para ser na seqüência executada em um processador particular e, assim, aguarda neste estado até o processador ficar disponível;

✰ **waiting:** caracteriza a *thread* que está bloqueada em um evento; voluntariamente espera por uma sincronização; ou ainda foi suspensa pelo ambiente do sub-sistema.



... 4.4 - Gerenciamento de SMP e Threads no Windows

✰ **Support for OS Subsystems:** as facilidades de propósito geral para *threads* e processos tem por objetivo suportar os vários ambientes de sistemas operacionais;

- ... para criar um processo, uma requisição é lançada para o correspondente sub-sistema, que por sua vez encaminha para o NT Executive;
- ... NT Executive cria um objeto processo e retorna o manipulador do objeto para o sub-sistema que encaminhou a requisição, sem no entanto criar *threads*;
- ... quando um processo é criado no Win32 ou OS/2, o novo processo herda muitos dos atributos do processo de instanciação, não obstante, no ambiente NT todo o processo de criação se dá de forma indireta.

... 4.4 - Gerenciamento de SMP e Threads no Windows

✰ **Symmetric Multiprocessing Support:** tanto processos quanto *threads* mesmo aqueles(as) do NT Executive podem ser executados em qualquer processador a menos de alguma restrição;

- ... como padrão, o *microkernel* usa a política *soft-affinity*, ou seja, procura associa uma *thread* ao mesmo processador na qual ela foi executada;
- ... isto ajuda na reutilização dos dados que estejam na *cache* do processador, contribuindo para não degradar a performance do sistema;
- ... no entanto, é possível restringir a execução de *threads* para certos processadores, neste caso, a abordagem é *hard affinity*.

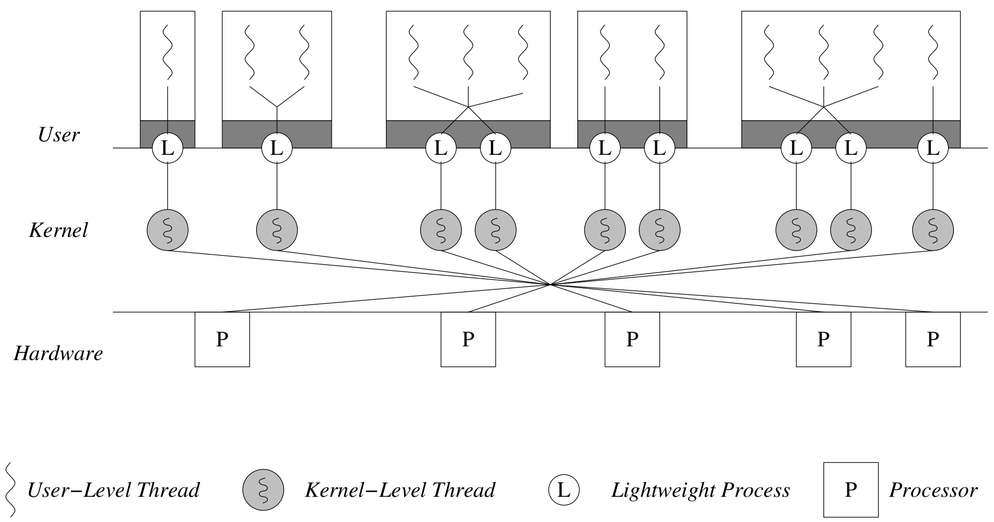
4.5 - Gerenciamento de SMP e Threads no Solaris

✫ Solaris utiliza-se de 04 conceitos relativos a *threads*:

- **processo:** processo normal no UNIX que inclui espaço de endereçamento para dados e código, pilha, e bloco de controle do processo;
- **user-level threads:** implementada através de biblioteca no espaço de endereçamento do usuário, são invisíveis para o sistema operacional;
- **lightweight processes:** podem ser vistos como o mapeamento entre o *threads* no nível do usuário e as *threads* no nível do *kernel*;
- **kernel threads:** constituem os elementos fundamentais que podem ser escalonados e direcionados para serem executados em algum dos processadores.

... 4.5 - Gerenciamento de SMP e Threads no Solaris

✫ A combinação de *threads* no nível do *kernel* e do usuário oferecem ao programador de aplicação a oportunidade de explorar a concorrência de forma muito mais eficiente e apropriada para uma dada aplicação.



... 4.5 - Gerenciamento de SMP e Threads no Solaris

✰ *User-Level Threads* podem ser criadas, destruídas, bloqueadas, ativadas, etc. sem que o *kernel* seja envolvido, portanto, são invisíveis para o sistema operacional;

- ... quando uma *thread* do nível de usuário é conhecida pelo *kernel*, o *kernel* terá que alocar estruturas de dados do *kernel* para cada uma e realizar o chaveamento de *threads* — chaveamento no modo *kernel* é mais caro que no modo usuário;

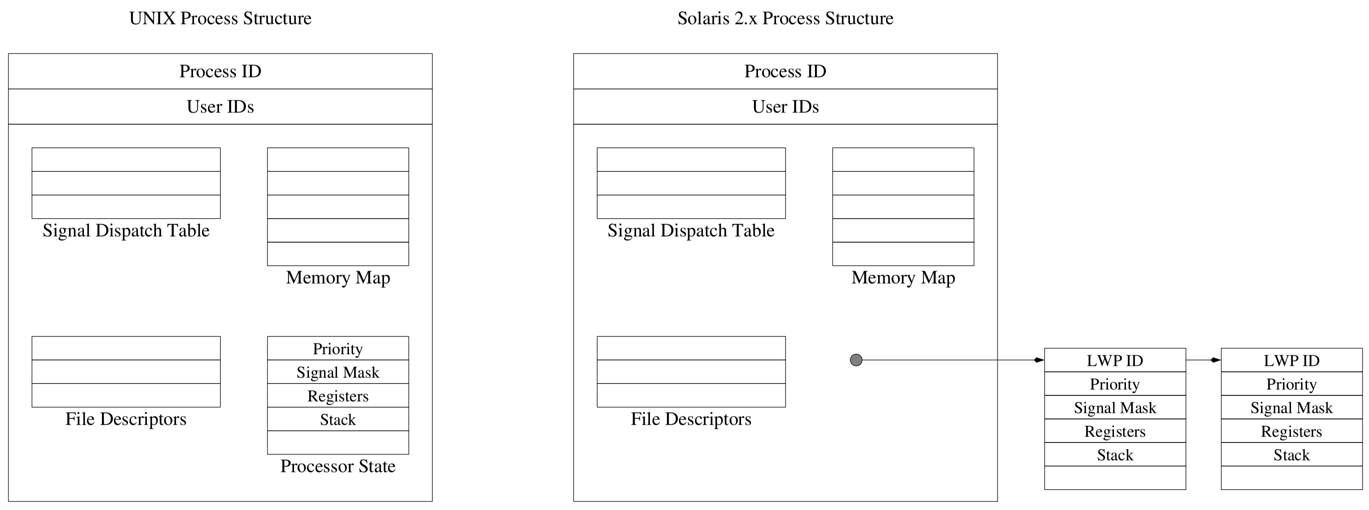
✰ Se uma aplicação envolve *threads* que podem bloquear, tais como aquelas que invocam entrada/saída, então com múltiplos *lightweight processes* que suportem um número igual ou maior de *threads* no nível do usuário é uma abordagem atrativa.

- ... o mapeamento um para um entre *threads* no nível do usuário e *lightweight process* é efetivo para algumas aplicações, por exemplo:

- ... cálculo paralelo de *array* poderia dividir as linhas entre diferentes *threads* e se essa relação for um para um nenhum chaveamento de *thread* seria necessário.

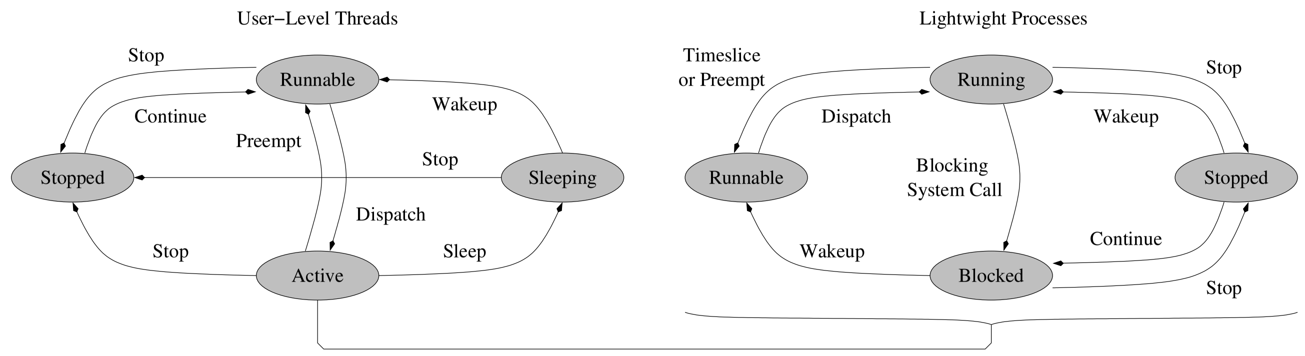
... 4.5 - Gerenciamento de SMP e Threads no Solaris

✰ Estrutura tradicional de processos no UNIX e no Solaris:



... 4.5 - Gerenciamento de SMP e Threads no Solaris

✰ Descrição simplificada dos estados de execução de *threads* ULT e LWP:



- ... lembre-se que a execução de *threads* no nível do usuário ou *user-level threads* é gerenciada pela biblioteca de *threads*;
- ... nenhuma restrição é imposta quanto ao número de *lightweight processes* para suportar as *threads* no nível do usuário, ou seja, *unbound threads*.

... 4.5 - Gerenciamento de SMP e Threads no Solaris

✰ Neste cenário, quando uma *thread* ULT é ativada, a mesma é atribuída a um LWP que executa enquanto a *thread* do *kernel* executa;

✰ Vários são os eventos que conduzem uma *user-level thread* deixar o estado *active*:

- **sincronização:** se uma *thread* invocar uma primitiva de algum mecanismo de concorrência, ela será colocada no estado de *sleep*;
- **suspensão:** qualquer *thread* pode suspender uma dada *thread*, colocando-a no estado de *stopped*, respeitadas as relações de parentesco;
- **preempção:** uma *thread* com mais alta prioridade pode conduzir a *thread* correntemente sendo executada para o estado *runnable*;
- **yielding:** permitir que um outra *thread* de mesma prioridade faça uso do LWP correntemente alocado a *thread*, neste caso a mesma muda de estado (*runnable*).