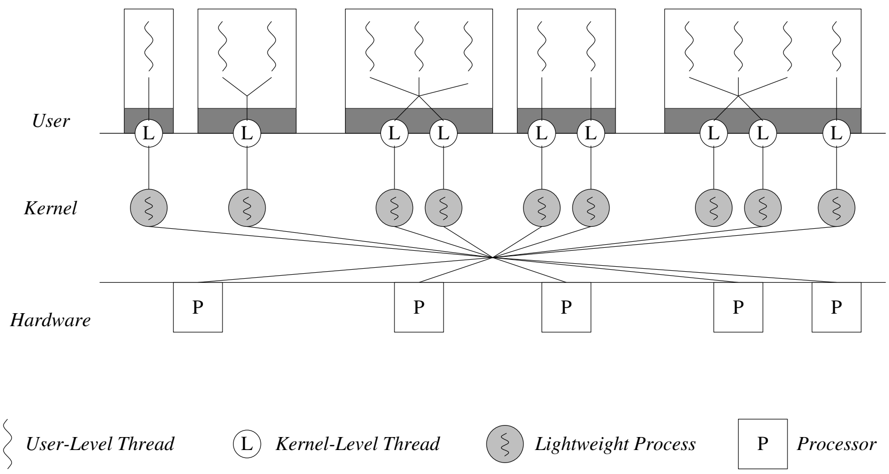
... 2.2.4 - Threads no Solaris

- 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.



... 2.2.4 - Threads no Solaris

- *user-level threads* podem ser criadas, destruídas, bloqueadas, ativadas, etc. sem que o *kernel* seja envolvido;

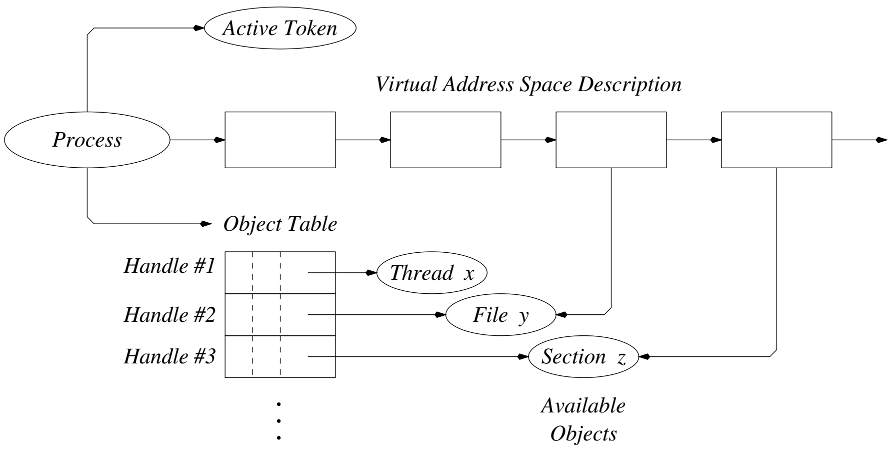
- ... quando uma *thread* do nível de usuário é conhecida pelo *kernel*, o *kernel* terá que alocar estruturas de dados do *kernel* para cada uma e realizar o chaveamento de *threads* — chaveamento no modo *kernel* é mais caro que no modo usuário;

- se uma aplicação involve *threads* que podem bloquear, tais como aquelas que invocam entrada/saída, então com múltiplos *light-weight 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 *light-weight process* é efetivo para algumas aplicações, por exemplo, um cálculo paralelo de *array* poderia dividir as linhas desse *array* entre diferentes *threads* e se essa relação for um para um nenhum chaveamento de *thread* se fará necessário.

... 2.2.4 - Threads no Windows

- processos suportados por diferentes S.O. diferem em muitos aspectos:
  - como os processos são nomeados, representados e protegidos;
  - como oferecer suporte a *threads*;
  - mecanismos para comunicação entre processos;
  - como processos se relacionam com outros processos.



... 2.2.4 - Threads no Solaris

- processo - processo normal no UNIX que inclui espaço de endereçamento, 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 *user-level threads* e as *threads* no *kernel*;
- *kernel threads* - constituem os elementos fundamentais que podem ser escalonados e direcionados para serem executados em algum dos processadores.

... 2.2.3 - 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;

... 2.2.3 - 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*).

2.2.3 - Microkernel

- **idéia:** 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).

- **algumas questões:**
  - 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 ?

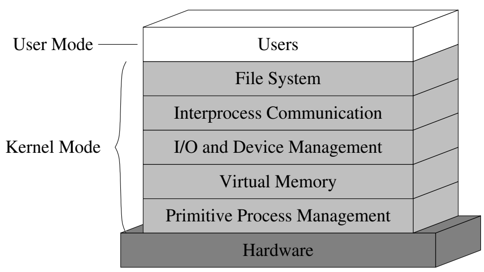
- **histórico:** o seu uso se popularizou no Sist. Oper. Mach.

- ... em teoria esta abordagem provê alto grau de flexibilidade e modularidade — já no Windows NT a modularidade e portabilidade são benefícios chaves.

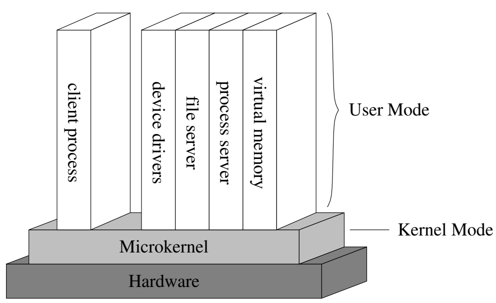
... 2.2.3 - Microkernel

- *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.



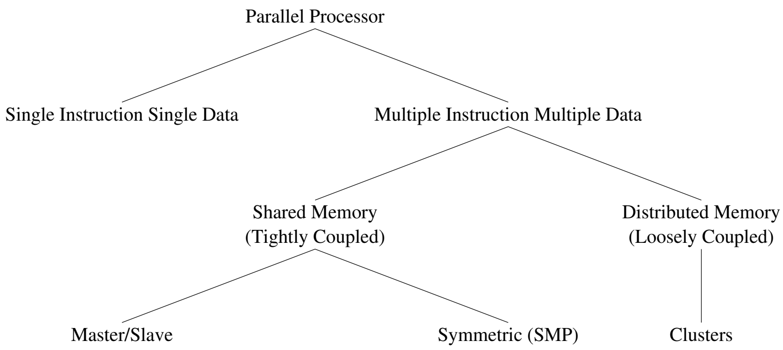
Layered Kernel



Microkernel

2.2.2 - 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;



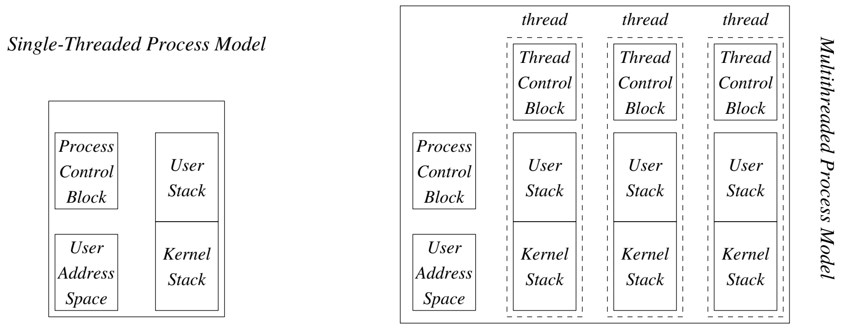
... 2.2.2 - Symmetric Multiprocessing - SMPs

- *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.

... 2.2.1 - Threads ou 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.



... 2.2.1 - Threads ou 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.

2.2.1 - Threads ou 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 caracterísiticas são distintas e portanto podem ser tratadas de forma independente;
- *lightweight process* pertencem a um processo e são caracterizadas pelo *program counter*, registradores e pilha;
- *lightweight process* 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.

... 2.2.1 - Threads ou Lightweight Process

- *multhreading* refere-se a habilidade do sist. oper. suportar múltiplas *threads* dentro de um único processo — algumas abordagens são apresentadas a seguir: