Sistemas Operacionais de Tempo Real

Sérgio Campos

Sistemas Operacionais de Tempo Real

- Escalonador, Threads, Tasks
- Tarefas periódicas
- Tarefas aperiódicas
- Microkernel
- Interrupções
- Relógios e temporização

Sistemas Operacionais

Tarefas e jobs são implementados como threads.

- cada thread executa um job
- tarefas consistem de múltiplos jobs executados em tempos diferentes.

Escalonador escolhe qual thread executa a cada instante

Threads se comunicam através de memória compartilhada ou mensagens.

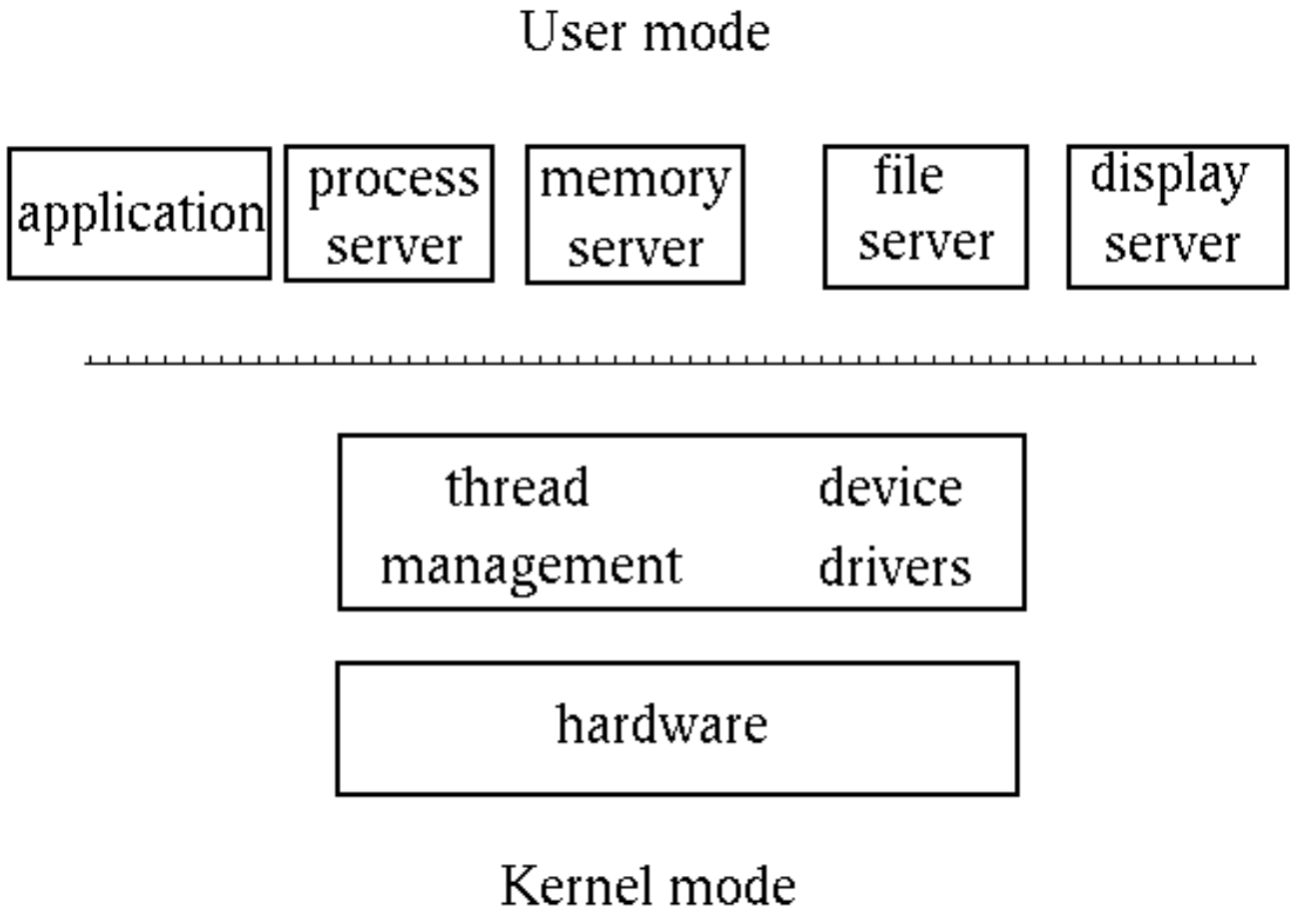
Tarefas Periódicas

- Uma tarefa periódica é implementada como um thread que executa periodicamente.
- Após sua execução o thread é posto para dormir e reiniciado quando do seu próximo período.
- A maioria dos SOs comerciais não implementa thread periódicos
  - Implementação em espaço de usuário com timers, tabelas, ...

Tarefas Aperiódicas

- São implementados como threads que executam em resposta a eventos
  - Tipicamente interrupções
- Após sua execução são colocados para dormir até o próximo evento
- Também não são implementados pos SOs comerciais em geral...

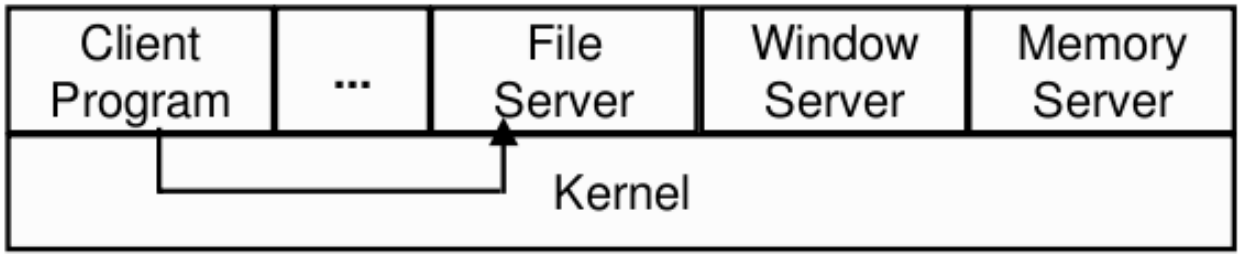
Mach Micro Kernel



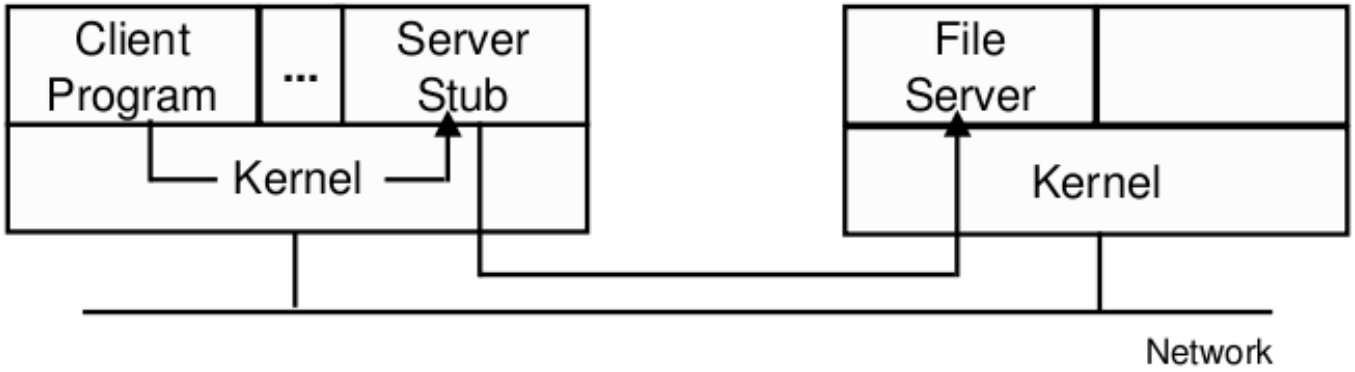
Micro Kernel

Comunicação com microkernels:

- Centralizado:



- Distribuído:



Interrupções

São usadas para:

- Eventos
- Tempo

Interrupções são tratadas pelo microkernel

- São detectadas, marcadas e ack'ed pelo microkernel
- Mas o tratamento é feito pelo usuário
  - Tratamento no microkernel custa caro
- O tratador da interrupção é chamado pelo escalonador

Relógios e Temporização

Todos os sistemas provêem ao menos um contador de tempo via hardware

- É iniciado com um valor específico e conta `c--`.
- Quando chega a zero, gera uma interrupção
- Podem ser programados periodicamente ou individualmente

A resolução do relógio é a granularidade do contador

- No hardware pode ser em nanosegundos
- A aplicação somente enxerga em geral milisegundos
  - Por causa do tratamento que precisa ser feito para executar tudo
  - IRQ $\rightarrow$ ACK $\rightarrow$ Escalonador $\rightarrow$ Escolhe $\rightarrow$ Atualiza
- Existe então interrupção de tempo, e a cada x destas uma interrupção de relógio

Relógios — Timers

A maioria dos SOs permitem que cada thread tenha seus próprios relógios

- Threads podem criar, setar, cancelar, destruir relógios
- Kernel mantém para cada relógio:
  - Thread que o criou
  - Valor (hora que vai expirar)
  - Tipo: periódico ou não
  - Tratador de evento 0

Relógios podem ser síncronos ou não

- Assíncronos podem ser usados para monitorar deadlines perdidas
- Quando expira verifica-se se o job terminou.

Sistemas Operacionais Comerciais de Tempo Real

- LynxOS, QNX and VxWorks
- Real-Time POSIX API
  - Preemptive, fixed priority scheduling
  - Primitivas de sincronização padrão (mutex e mensagens).
  - Threads XOR processes.
- Modulares e escaláveis
  - kernel pequeno para sistemas embutídos
  - I/O, arquivos, network são módulos opcionais

Sistemas Operacionais Comerciais de Tempo Real

Rápidos e eficientes:

- A maioria é microkernel
- Troca de contexto rápida, assim como interrupções, lock/unlock. Normalmente microsegundos
- Partes do kernel não interrompíveis são otimizadas e pequenas
- Escalonamento flexível
  - Ao menos 32 níveis de prioridades. A maioria 128 ou 256.
  - FIFO ou RR para threads de prioridade igual
  - sem EDF
- Resolução de relógios boa: nominal $\rightarrow$ nanoseg; real $\rightarrow$ microseg
- Sem paginação ou swapping

LynxOS

LynxOS 3.0 é um microkernel de 28KB que oferece:

- Escalonador
- Tratamento de interrupções
- Sincronização

Kernel Plug-Ins (KPIs) são módulos que podem crescer até LynxOS virar um Unix

- Self-hosted system $\rightarrow$ desenvolvimento na mesma plataforma de uso
- Pode ter até proteção de memória e paginação

QNX / Neutrino

Capaz de escalar de um microkernel de 12KB até um sistema multi processado Unix-like distribuído

Microkernel provê somente:

- threads
- troca de contexto
- mensagens

VxWorks

Usado no Mars Pathfinder...

Monolítico ao invés de microkernel

- Mas permite desligar proteção de memória, ... Até inversão de prioridades!

Uniprocessador, mas permite multiprocessamento com memória compartilhada


Linux vs. Windows

Windows NT RT

O Bom:

- Implementa threads, interrupçoes com prioridade, eventos
- Resolução de relógio boa

O Mau:

- Usa muita memória. Muita!
- Suporte fraco para escalonamento e controle de recursos de RT
- Comunicação e tratamento de interrupções imprevisível

O Feio:

- Microsoft!

Windows NT RT

- Somente 32 níveis de prioridade — de 16 a 31 para RT
- Muitos threads do kernel rodam em pri 16, menor que outros RT
- Threads de pri igual escalonam RR — FIFO limitado
- Eventos são usados para sincronizar threads
  - `signal` $\rightarrow$ `wait`
  - Mas... eventos são FIFO!!!

Linux RT

O Bom:

- 100 níveis de prioridade
- Políticas de escalonamento flexíveis
- May the source be with you!

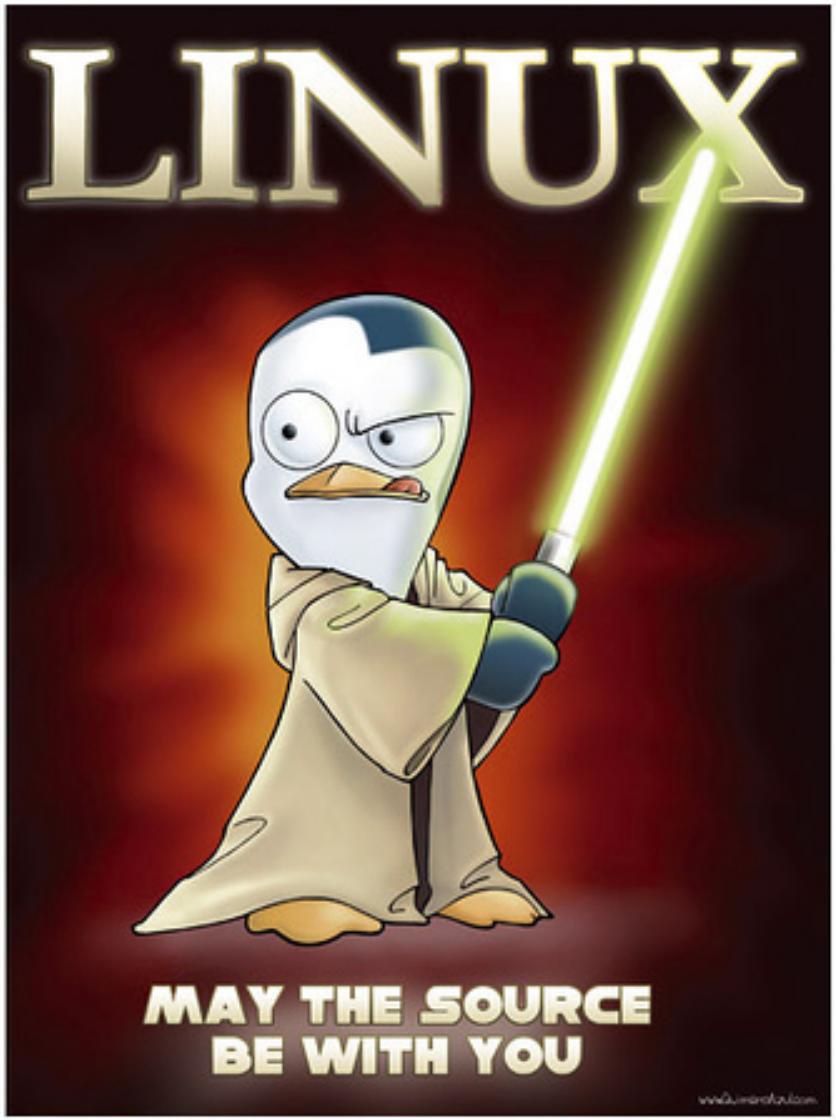
O Mau:

- A maioria dos subsistemas do kernel desabilita interrupções nas seções críticas (que feio!)
- Erros nos relógios pode ser grandes e imprevisíveis

O Feio:

- Não é microkernel.

Linux RT



E para quem não conhece...