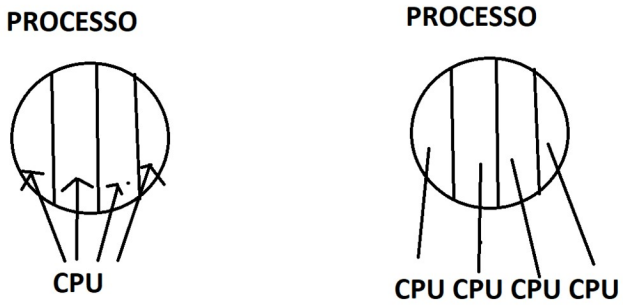
Aula SISTEMAS OPERACIONAIS
PROFESSOR FLÁVIO BRAGANÇA
EXPLICADORES.NET

Utilização de threads



Por que alguém iria querer ter um tipo de processo dentro de um processo? Na realidade, há várias razões para a existência desses **miniprocessos**, chamados threads. Vamos examinar agora algumas delas:

* A principal razão para se ter threads é que em muitas aplicações múltiplas atividades estão ocorrendo simultaneamente e algumas delas podem bloquear de tempos em tempos. Ao decompormos uma aplicação dessas em múltiplos threads sequenciais que são executados em quase paralelo, o modelo de programação **torna-se mais simples.**

- O sistema **aumenta o desempenho** quando utilizamos threads;
- As Threads deixam o sistema **mais simples;**
- Para criar uma Thread é a 10 a 100 vezes mais lento do que criar um processo;
- O modelo de programação se torna mais simples;
- Threds são mais leves que os processos;
- As Threads são vantajosas em sistemas com grande volume de computação e muita E/S;
- MonoThread → Processos não divididos em Threads;
- MultiThread → Sistema que divide os processos em várias Threads;

* Um segundo argumento para a existência dos threads é que como eles são mais leves do que os processos, eles são mais fáceis (isto é, mais rápidos) para criar e destruir do que os processos. Em muitos sistemas, criar um thread é algo de 10 a 100 vezes mais rápido do que criar um processo. Quando o número necessário de threads muda dinâmica e rapidamente, é útil se contar com essa propriedade.

* Uma terceira razão para a existência de threads também é o argumento do desempenho. O uso de threads não resulta em um ganho de desempenho quando todos eles são limitados pela CPU, mas quando há uma computação substancial e também E/S substancial, contar com threads permite que essas atividades se sobreponham, acelerando desse modo a aplicação.

* Por fim, threads são úteis em sistemas com múltiplas CPUs, onde o **paralelismo real** é possível.

Pseudoparalelismo = Quando o processador fica revezando entre as tarefas, quando temos apenas um processador trabalhando com time sharing;

Paralelismo real = Realmente as tarefas são executadas ao mesmo tempo, quando temos um processador com vários núcleos ou vários processadores;

Modelo monothread
- Sistemas onde os processos não são divididos
- Mais lento
- Mais complexo

Modelo multithread
- Sistemas onde os processos são divididos em várias threads
- Mais rápido
- Mais simples

Máquina de estados finitos

Modelo que não é monothread e nem multithread, modelo de computação que tem um estado salvo e algum conjunto de eventos podem modificar esse estado.

Modelo	Características
Threads	Paralelismo, chamadas de sistema **bloqueantes**
Processo monothread	**Não paralelismo**, **chamadas de sistema bloqueantes**
Máquina de estados finitos	Paralelismo, chamadas não bloqueantes, **interrupções**

Modelo semelhante a Thread acrescentando o recurso das interrupções.

Sistema operacional → Chamadas de sistema → Programas

Chamadas de sistema bloqueantes
- O processo e suas threads **não podem sair do estado de execução antes de terminar.**
- **Não preemptivos / não permite preempção**
- Se um processo entra em estado de execução ele só sai quando terminar.
- Desempenho inferior

Chamadas de sistema não bloqueantes
- O processo e suas Threads **podem sair do estado de execução antes de terminar.**
- **Preemptivos / permite preempção**
- Se um processo entra em estado de execução ele pode ser interrompido antes de terminar.
- Tem desempenho superior

Sistema Preemptivo
→ Quando o processo pode ser retirado de execução mesmo antes de ter terminado.
→ Time Sharing;
→ Bom para o sistema;

Sistema Não preemptivo
→ Quando o processo só pode sair de execução quando terminar;
→ Não temos time-sharing;
→ Ruim para o sistema;
→ Predominância de processos CPU-Bound;

Tipos de Threads:

Threads de execução ou simplesmente Threads

- **Thread normal**
- **Divisão dos processos em várias partes;**
- **Thread regular;**

O thread tem um contador de programa que controla qual instrução deve ser executada em seguida. Ele tem registradores, que armazenam suas variáveis de trabalho atuais. Tem uma pilha, que contém o histórico de execução, com uma estrutura para cada rotina chamada, mas ainda não retornada. Embora um thread deva executar em algum processo, o thread e seu processo são conceitos diferentes e podem ser tratados separadamente. Processos são usados para agrupar recursos; threads são as entidades escalonadas para execução na CPU.

Thread despachante

- **Thread que lê requisições da rede.**

Lê as requisições de trabalho que chegam da rede. Depois de examinar a requisição, ele escolhe um thread operário ocioso (bloqueado) e entrega-lhe a requisição, possivelmente colocando um ponteiro para a mensagem em uma palavra especial associada a cada thread. O despachante então acorda o operário que está descansando, tirando-o do estado de bloqueado e colocando-o como pronto.

Obs.:
Multithread também é usado para descrever a situação de permitir múltiplos threads no mesmo processo.

Threads POSIX

- **Threads no padrão de sistemas UNIX**

Para possibilitar que se escrevam programas com threads portáteis, o IEEE definiu um padrão para threads no padrão **IEEE 1003.1c**. O pacote de threads que ele define é chamado Pthreads. A maioria dos sistemas UNIX dá suporte a ele. O padrão define mais de 60 chamadas de função.

Threads Pop-Up

- **Thread gerada a partir de mensagens da rede**

Normalmente empregrada em sistemas distribuídos, a chegada de uma mensagem faz o sistema criar um novo thread para lidar com a mensagem. Esse thread é chamado de thread pop-up.

Thread – Vield

- **Thread não bloqueante/preeemptiva = Sai da CPU antes de terminar para dar lugar a outra Thread.**

Thread desiste voluntariamente da CPU para deixar outro thread executar.
Vantagens → possibilitam que múltiplas atividades ocorram no mesmo tempo, é mais fácies de criar e destruir que o processo, tem ganhado em desempenho na execução baseada Entrada/Saída.

Thread de execução → Normal / Regular
Thread despachante → Rede
Tread Pop-up → Mensagens de Rede
Thread Vield → Dá lugar a outra Thread
Thread POSIX → Baseado em Unix

Jacket ou wrapper
Jacket → Jaqueta
Wrapper → Embrulho

Código que envolve a chamada de sistema para **evitar eventuais bloqueios** futuros, bloqueios que podem parecer para o sistema corretos, **mas podem prejudicar o desempenho de execução das Threads**.

Comunicação entre processos (IPC – Inter Processor Communications)

Processos quase sempre precisam comunicar-se com outros processos. Por exemplo, em um pipeline do interpretador de comandos, a saída do primeiro processo tem de ser passada para o segundo, e assim por diante até o fim da linha. Então, há uma necessidade por comunicação entre os processos, de preferência de uma maneira bem estruturada sem usar interrupções.

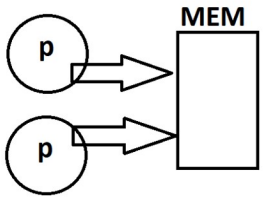
- Processos se comunicam o tempo inteiro entre si;
- Um processo pode necessitar de informações de outros processos para prosseguir.

Condições de disputa/Corrida

Em alguns sistemas operacionais, processos que estão trabalhando juntos podem compartilhar de alguma memória comum que cada um pode ler e escrever. A memória compartilhada pode encontrar-se na memória principal (possivelmente em uma estrutura de dados de núcleo) ou ser um arquivo compartilhado; o local da memória compartilhada não muda a natureza da comunicação ou os problemas que surgem.

Dois processos, um processo lê e outro escreve em um arquivo compartilhado.

- Quando um ou mais processos compartilham uma área de memória comum



Regiões críticas

Parte do programa que tem acesso à memória compartilhada.

- Área do **código** do programa que tem **acesso a memória compartilhada**, chamada assim por gerar a condição de **disputa** ou de **corrida**.

Exclusão Mútua

Como evitar as condições de corrida? A chave para evitar problemas aqui e em muitas outras situações envolvendo memória compartilhada, arquivos compartilhados e tudo o mais compartilhado é encontrar alguma maneira de proibir mais de um processo de ler e escrever os dados compartilhados ao mesmo tempo. Colocando a questão em outras palavras, o que precisamos é de exclusão mútua, isto é, alguma maneira de se certificar de que se um processo está usando um arquivo ou variável compartilhados, os outros serão impedidos de realizar a mesma coisa.

Técnica que não permite que dois ou mais processos executem tarefas ao mesmo tempo em uma determinada área de memória.

Condição de disputa → Vários processos acessando uma mesma área;
Região crítica → Código do programa que possui a condição de disputa;
Exclusão mútua → Processo que resolve a condição de disputa;

Algum modo de impedir acesso a um recurso compartilhado que esteja em uso.

Boa solução:

- Nunca dois processos podem estar simultaneamente em suas regiões críticas.
- Nada pode ser afirmado sobre velocidade ou números de CPUs.
- Nenhum processo executando fora da região crítica pode bloquear outro.
- Nenhum processo deve esperar **eternamente** para entrar em sua região crítica.

Deadlocks

Podem ocorrer tantos em recursos de **hardware** quanto de **software**. Geralmente ocorre com recursos **não preemptivos**.

Sinuca de bico;

Quando um processo fica aguardando eternamente um recurso que está bloqueado por outro processo, que está aguardando o primeiro processo terminar.

Posse e espera;

Um processo tem um recurso e outro está esperando

Recursos preemptível → É aquele que pode ser retirado de execução sem nenhum prejuízo.

Recursos não-preemptível → Se por acaso for retirado de execução pode causar danos.
Ex.: gravador de CD.

Quatro condições para que <u>ocorra</u> o deadlock:

- **Condição de exclusão mútua** (recurso está associado a 1 único processo ou disponível),
- **Condição de posse e espera** (1 processo pode requisitar mais de um recurso),
- **Condição de não preempção** (recurso deve ser explicitamente liberados pelo processo que o retém),
- **Condição de espera circular** (mais de um processo esperando o recurso utilizado).

Quatro estratégias para tratar deadlocks:

- Ignorar o problema (Algoritmo do avestruz);
- Detecção e recuperação;
- Anulação dinâmica (por meio de uma alocação cuidadosa de recursos);
- Prevenção (negando uma das quatro condições).

Prevenção de deadlocks

- Estabelecer critérios de que todos os recursos sejam previamente alocados, antes do processo ganhar acesso à UCP.
  Que todos os processos recebam seus recursos antes de entrar em execução.
- Admitir a prática da preempção, isto é, o sistema ter a possibilidade de retirar um recurso alocado para um processo e dar a outro processo.
  Quando um processo está travando um recurso podemos retirar de execução.
- Forçar que um processo não aloque mais do que um recurso de cada vez.
  Não deixar um processo utilizar mais de um recurso por vez;

Qualquer que seja a estratégia de prevenção adotada, ela é sempre muito onerosa, uma vez que precisa ser executada a todo instante. A estratégia mais comum e menos onerosa é detectar a ocorrência de um deadlock e, uma vez adotada, executar rotinas de resolução do problema.

Ignorar o problema – Algoritmo do avestruz.

STARVATION

Inanição

Quando o processo não é atendido por motivos de determinações de prioridades;

Exemplo:

Servidor de impressão

Primeiro os documentos com menor tamanho;

Extensão	Significado
.bak	Cópia de segurança
.c	Código-fonte de programa em C
.gif	Imagem no formato Graphical Interchange Format
.hlp	Arquivo de ajuda
.html	Documento em HTML
.jpg	Imagem codificada segundo padrões JPEG
.mp3	Música codificada no formato MPEG (camada 3)
.mpg	Filme codificado no padrão MPEG
.o	Arquivo objeto (gerado por compilador, ainda não ligado)
.pdf	Arquivo no formato PDF (Portable Document File)
.ps	Arquivo PostScript
.tex	Entrada para o programa de formatação TEX
.txt	Arquivo de texto
.zip	Arquivo compactado

Gerenciamento de entrada e saída

- Gerenciamento dos periféricos de entrada e saída
- Entrada : Teclado
- Saída : Monitor
- Entrada e saída : Porta USB;

Princípios do hardware de E/S Diferentes pessoas veem o hardware de E/S de maneiras diferentes. Engenheiros elétricos o veem em termos de chips, cabos, motores, suprimento de energia e todos os outros componentes físicos que compreendem o hardware. Programadores olham para a interface apresentada ao software — os comandos que o hardware aceita, as funções que ele realiza e os erros que podem ser reportados de volta. Neste livro, estamos interessados na programação de dispositivos de E/S, e não em seu projeto, construção ou manutenção; portanto, nosso interesse é saber como o hardware é programado, não como ele funciona por dentro. Não obstante isso, a programação de muitos dispositivos de E/S está muitas vezes intimamente ligada à sua operação interna. Nas próximas três seções, apresentaremos uma pequena visão geral sobre hardwares de E/S à medida que estes se relacionam com a programação.

Dispositivos de E/S

Dispositivos de E/S podem ser divididos de modo geral em duas categorias:

Dispositivos de blocos

Armazena informações em blocos de tamanho fixo, cada um com seu próprio endereço. Tamanhos de blocos comuns variam de 512 a 65.536 bytes. Todas as transferências são em unidades de um ou mais blocos inteiros (consecutivos). A propriedade essencial de um dispositivo de bloco é que cada bloco pode ser lido ou escrito independentemente de todos os outros. Discos rígidos, discos Blu-ray e pendrives são dispositivos de bloco comuns;

HD / CD / DVD / BLURAY / SSD (SETORES)

Dispositivos de caractere.

Um dispositivo de caractere envia ou aceita um **fluxo de caracteres**, desconsiderando qualquer estrutura de bloco. Ele não é endereçável e não tem qualquer operação de busca. Impressoras, interfaces de rede, mouses (para apontar), ratos (para experimentos de psicologia em laboratórios) e a maioria dos outros dispositivos que não são parecidos com discos podem ser vistos como dispositivos de caracteres.

Teclado, mouse, Scanner

Controladores de dispositivos

Unidades de E/S consistem, em geral, de um componente mecânico e um componente eletrônico. É possível separar as duas porções para permitir um projeto mais modular e geral. O componente eletrônico é chamado de controlador do dispositivo ou adaptador. Em computadores pessoais, ele muitas vezes assume a forma de um chip na placa-mãe ou um cartão de circuito impresso que pode ser inserido em um slot de expansão (PCIe). O componente mecânico é o dispositivo em si.

Dispositivo	Taxa de dados
Teclado	10 bytes/s
Mouse	100 bytes/s
Modem 56 K	7 KB/s
Scanner em 300 dpi	1 MB/s
Filmadora *camcorder* digital	3,5 MB/s
Disco Blu-ray 4x	18 MB/s
Wireless 802.11n	37,5 MB/s
USB 2.0	60 MB/s
FireWire 800	100 MB/s
Gigabit Ethernet	125 MB/s
Drive de disco SATA 3	600 MB/s
USB 3.0	625 MB/s
Barramento SCSI Ultra 5	640 MB/s
Barramento de faixa única PCIe 3.0	985 MB/s
Barramento Thunderbolt2	2,5 GB/s
Rede SONET OC-768	5 GB/s

Controlador (placa)	**Dispositivo**	
Controladora IDE	Dispositivos IDE	(hd/cd/dvd ANTIGOS)
Controladora SATA	Dispositivos SATA	(hd/cd/dvd ATUAIS)
Controladora Floppy	Drive de disquete	

Controlador → Dispositivos

SISTEMA OPERCIONAL → DRIVER → CONTROLADORA → DISPOSITIVO

DRIVER = **SOFTWARE** QUE INSTALAMOS PARA CONTROLAR O DISPOSITIVO. O SISTEMA OPERACIONAL PODE NÃO TER O DRIVER DO DISPOSITIVO, LOGO PRECISAMOS DO DRIVER DO DISPOSITIVO;

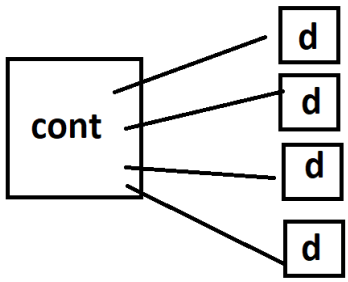
DRIVE = **DISPOSITIVO** (DRIVE DE DISQUETE, DRIVE DE CD)

A comunicação entre o controlador e os dispositivos pode acontecer de diversas maneiras:

E/S programada

- Busy Waiting;
- Espera ocupada;
- O controlador fica perguntando o tempo inteiro para os dispositivos se os mesmos querem transmitir até que a resposta seja positiva;
- Lento;
- Obsoleto;

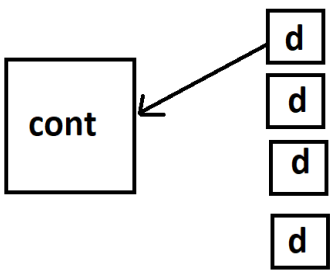
O aspecto essencial da E/S programada, claramente ilustrado nessa figura, é que, após a saída de um caractere, a CPU continuamente verifica o dispositivo para ver se ele está pronto para aceitar outro. Esse comportamento é muitas vezes chamado de espera ocupada (busy waiting) ou polling. A E/S programada é simples, mas tem a desvantagem de segurar a CPU o tempo todo até que toda a E/S tenha sido feita. Se o tempo para "imprimir" um caractere for muito curto (pois tudo o que a impressora está fazendo é copiar o novo caractere para um buffer interno), então a espera ocupada estará bem. No entanto, em sistemas mais complexos, em que a CPU tem outros trabalhos a fazer, a espera ocupada será ineficiente, e será necessário um método de E/S melhor.



E/S orientada a interrupções

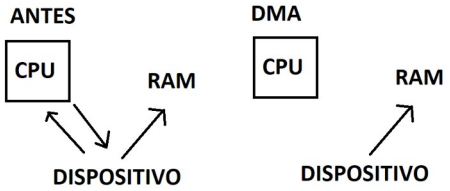
- IRQ = Interruption request;
- Agora o dispositivo interrompe o controlador quando necessita;
- Sistema mais rápido;
- Utilizado atualmente;

E/S orientada a interrupções Agora vamos considerar o caso da impressão em uma impressora que não armazena caracteres, mas imprime cada um à medida que ele chega. Se a impressora puder imprimir, digamos, 100 caracteres/segundo, cada caractere levará 10 ms para imprimir. Isso significa que após cada caractere ter sido escrito no registrador de dados da impressora, a CPU vai permanecer em um laço ocioso por 10 ms esperando a permissão para a saída do próximo caractere. Isso é mais tempo do que o necessário para realizar um chaveamento de contexto e executar algum outro processo durante os 10 ms que de outra maneira seriam desperdiçados. A maneira de permitir que a CPU faça outra coisa enquanto espera que a impressora fique pronta é usar interrupções. Quando a chamada de sistema para imprimir a cadeia é feita, o buffer é copiado para o espaço do núcleo, como já mostramos, e o primeiro caractere é copiado para a impressora tão logo ela esteja disposta a aceitar um caractere. Nesse ponto, a CPU chama o escalonador e algum outro processo é executado. O processo que solicitou que a cadeia seja impressa é bloqueado até que a cadeia inteira seja impressa.



E/S usando DMA

- Direct memory access;
- Sistema de acesso direto à memória;
- Quando o dispositivo necessita acessar a memória esse acesso é direto, sem o auxílio do processador;
- Muito mais rápido;



Uma desvantagem óbvia do mecanismo de E/S orientado a interrupções é que uma interrupção ocorre em cada caractere. Interrupções levam tempo; portanto, esse esquema desperdiça certa quantidade de tempo da CPU. Uma solução é usar o acesso direto à memória (DMA). Aqui a ideia é deixar que o controlador de DMA alimente os caracteres para a impressora um de cada vez, sem que a CPU seja incomodada. Na essência, o DMA executa E/S programada, apenas com o controlador do DMA realizando todo o trabalho, em vez da CPU principal. Essa estratégia exige um hardware especial (o controlador de DMA), mas libera a CPU durante a E/S para fazer outros trabalhos. Uma linha geral do código é dada na Figura 5.10. A grande vantagem do DMA é reduzir o número de interrupções de uma por caractere para uma por buffer impresso. Se houver muitos caracteres e as interrupções forem lentas, esse sistema poderá representar uma melhoria importante. Por outro lado, o controlador de DMA normalmente é muito mais lento do que a CPU principal. Se o controlador de DMA não é capaz de dirigir o dispositivo em velocidade máxima, ou a CPU normalmente não tem nada para fazer de qualquer forma enquanto esperando pela interrupção do DMA, então a E/S orientada à interrupção ou mesmo a E/S programada podem ser melhores. Na maioria das vezes, no entanto, o DMA vale a pena.

Espera ociosa → O controlador pergunta em looping se o dispositivo quer transmistir;
IRQ → O dispositivo interrompe o processador quando necessário;
DMA → O dispositivo acessa a RAM sem precisar do processador;

Drivers dos dispositivos

- SOFTWARE
- DRIVER(SOFTWARE) <> DRIVE (Dispositivo)
- Cada dispositivo possui um driver
- Ensina ao sistema operacional a trabalhar com o dispositivos

Sistema operacional → Driver → Dispositivo

Em consequência, cada dispositivo de E/S ligado a um computador precisa de algum código específico do dispositivo para controlá-lo. Esse código, chamado driver do dispositivo, geralmente é escrito pelo fabricante do dispositivo e fornecido junto com ele. Tendo em vista que cada sistema operacional precisa dos seus próprios drivers, os fabricantes de dispositivos comumente fornecem drivers para vários sistemas operacionais populares.

Sistemas com múltiplos processadores

Multicomputadores
Vários computadores trabalhando como um único computador;
Exemplo : Rede Bitcoin;

Multiprocessadores
Quando um computador tem vários processadores;
Exemplo : Vários processadores na mesma placa mãe, processador com vários núcleos;

Multiprocessadores

- Vários processadores na mesma placa mãe;
- Vários núcleos em um mesmo processador;
- O acesso a RAM tem sempre a mesma velocidade;

Um multiprocessador de memória compartilhada ou simplesmente multiprocessador é um sistema de computador onde duas ou mais CPUS compartilham acesso total a uma memória RAM comum. Um programa executado em qualquer uma das CPUS enxerga o espaço de endereçamento virtual normal (geralmente paginado). Uma propriedade deste sistema é que uma CPU pode escrever uma informação em uma palavra de memória e depois ler a palavra de volta e obter um valor diferente, pois outra CPU pode ter alterado este valor.

FRACAMENTE ACOPLADOS
A LIGAÇÃO ENTRE OS PROCESSADORES E AS MEMÓRIAS É POR MEIO DE **REDE**;
NUMA = ACESSO NÃO UNIFORME A MEMÓRIA;

FORTEMENTE ACOPLADOS
A LIGAÇÃO ENTRE OS PROCESSADORES E AS MEMÓRIAS É POR MEIO DE **BARRAMENTO**;
UMA = ACESSO UNIFORME A MEMÓRIA;

Hardware dos multiprocessadores

Tipos de multiprocessadores

UMA (Uniform Memory Access – Acesso uniforme da memória) – Onde todas as palavras de memória são lidas com a mesma velocidade. (smp)

- Acesso a memória uniforme (sempre na mesma velocidade);
- Processadores estão sempre na mesma placa que a memória.
- Meio de comunicação é o barramento
- Multiprocessadores
- Sistemas fortemente acoplados

NUMA (Non uniform memory access – Acesso não uniforme à memória) - Onde existe há diferença de velocidade de leitura de palavra de memória lida em cada processador.

- Acesso a memória não uniforme (em velocidades diferentes)
- Processadores estão em uma mesma rede.
- Meio de comunicação é a rede
- Multicomputadores
- Sistemas fracamente acoplados