Virtualizar a CPU

- Questão de partida
- Execução direta
- Chamadas sistema
- Troca entre processos (switching)
- Concorrência
- Questões

Questão de partida

COMO VIRTUALIZAR A CPU COM CONTROLO DE FORMA EFICIENTE?

Os SOs precisam de virtualizar o CPU de uma forma eficiente enquanto mantêm o controlo do sistema.

Para o fazer é necessário o suporte do *hardware* e do próprio sistema operativo.

Os SOs usam de forma sensata um 'bit' de suporte de *hardware* nesta tarefa.

THE CRUX:
HOW TO EFFICIENTLY VIRTUALIZE THE CPU WITH CONTROL

The OS must virtualize the CPU in an efficient manner while retaining control over the system. To do so, both hardware and operating-system support will be required. The OS will often use a judicious bit of hardware support in order to accomplish its work effectively.

Execução em modo direto

- Correr um programa diretamente no CPU:
  - O SO cria uma entrada para o processo na lista de processos, aloca memória.
  - Carrega o código do programa na memória (a partir do disco).
  - Localiza o ponto de início, *main()*, e inicia a execução do código.

OS	Program
Create entry for process list	
Allocate memory for program	
Load program into memory	
Set up stack with argc/argv	
Clear registers	
Execute **call** main()	
	Run main()
	Execute **return** from main
Free memory of process	
Remove from process list	

Execução em modo direto

- **Questões**
  - Como garantimos que o programa associado ao processo não faz qualquer coisa que não queremos enquanto ele é executado (p.ex. alterar dados na memória)?
  - Quando um processo corre como é que o SO pára a sua execução e troca-o por outro processo, implementando a partilha do tempo - ***time sharing****?*
  - O que acontece se um processo precisa de executar uma operação restrita ao SO tal como requerer um pedido de E/S para um disco ou ganhar acesso a mais recursos tais como CPU e memória?
    - Se o SO deixar os processos executarem o que pretendem executar, o CPU não controla a permissão de acesso aos ficheiros e qualquer processo pode aceder a qualquer ficheiro localizado no disco!

Restringir as operações

- **Modo de utilizador vs. Modo de núcleo**
  - Modo Utilizador (*User mode*)
    - Código que corre no modo de utilizador é restringido nas ações que pode executar.
      - Se lançar um pedido de E/S, o processador dispara uma exceção.
      - O SO pode matar o processo.
    - Modo em que corre qualquer processo de utilizador.
  - Modo de núcleo (*Kernel mode*)
    - Código que corre no modo de núcleo pode executar qualquer ação incluindo ações privilegiadas, tais como lançar pedidos de E/S e executar todo o tipo de instruções restritas.
    - Modo em que corre o SO.

Chamadas Sistema

- Mas como pode um processo executar operações privilegiadas tais como ler informação de um disco?
  - O hardware moderno permite aos programas executar uma **chamada sistema – *system call*.**
  - As chamada sistema permitem que o núcleo (ou *kernel*) exponham certas elementos chave da sua funcionalidade aos programas dos utilizadores, tais como, aceder aos sistemas de ficheiros, a criação ou eliminação de processos, comunicação entre processos e alocação de mais memória.
  - A maior parte dos SOs fornecem centenas de chamadas sistema

Chamadas Sistema

- A execução de uma chamada sistema obedece às seguintes etapas:
  - O programa executa uma instrução especial de exceção, designada por ***trap***.
  - Esta instrução salta para o *kernel* e muda o privilégio para o modo de núcleo.
  - O sistema tem agora agora o privilégio para executar operações desejadas e assim cumprir a tarefa desejada pelo programa.
  - Quando termina a tarefa, o SO executa uma instrução de retorno da exceção designada por ***return-from-trap*** *instruction*.
  - Estas instrução faz retornar o controlo para o código do programa do utilizador, reduzindo os privilégios, passando para o modo de utilizador.

Chamadas Sistema

- Como guarda o *hardware* informação sobre o processo que corre antes de executar a exceção?
  - Tem de garantir que guarda informação dos registos do processo que executa essa chamada.
  - Só dessa forma consegue depois retornar ao estado em que estava quando abandonou o processo, após a instrução *return-from-trap*.
    - Exemplo, nos processadores *x86*
      - Na execução do ***trap*** empilham (*push*) *PC, flags* e alguns registos na pilha do núcleo.
      - Na execução do ***return-from-trap*** retiram da pilha esses valores.

Chamadas Sistema

- Como sabe o processador, na execução da exceção, que código deve correr dentro do SO?
  - O núcleo tem de controlar para onde é executado o código dentro do núcleo.
    - Cria uma tabela de exceções - ***trap table.***
    - Quando a máquina arranca, inicia em modo de núcleo.
    - Configura o hardware.
    - Define qual o código que é executado quando determinadas exceções ocorrem.
      - Por exemplo:
        - Que código correr quando há uma interrupção do disco?
        - Que código correr quando há uma interrupção do teclado?
        - Que código correr quando há uma chamada sistema, ou seja, quando ocorre uma **trap**?

Chamadas Sistema

- O SO informa o *hardware* da localização dod gestores de exceções – **trap handlers**.
- Cada chamada sistema tem associado um número que corresponde a um tipo de chamada sistema:
  - O código do número da chamada sistema é colocada no registo.
  - Quando o SO gere a chamada sistema, examina o número e verifica se ele é válido.
  - Depois salta para o código associado à execução dessa chamada.

Troca entre processos (switching)

- Como é que é realizada a troca de processos?
  - O SO tem de decidir parar um processo e iniciar outro.
  - Mas se um processo está a ser executado, então o SO não está a ser executado. Como pode ele fazer alguma coisa?
  - O SO tem de reganhar controlo do CPU.

Troca entre procesos (switching)

COMO REGANHAR O CONTROLO DO CPU?

Como é que os sistemas operativos reganham o controlo de um CPU de forma a que possam trocar processos?

THE CRUX: HOW TO REGAIN CONTROL OF THE CPU

How can the operating system **regain control** of the CPU so that it can switch between processes?

Troca entre Processos (switching)

- **Estratégia cooperativa**
  - O SO confia que os processos se portam bem (antigo Macintosh e Xerox).
    - Os processos que demoram mais tempo é assumido que cedem periodicamente o comando ao SO de forma a que este possa decidir executar outras tarefas.
  - Como é que um *processo simpático* passa o CPU para o SO?
    - A maior parte dos processos acabam por passar o comando do CPU para o SO através das **chamadas sistema.**
      - **ex.** Quando precisam de abrir e ler ficheiros, ler uma mensagem de outra máquina, ou criar um novo processo, iniciam o processo de chamada sistema que os leva a ceder o controlo ao SO.
  - Neste tipo de sistemas há uma chamada de sistema fictícia, designada por chamada sistema de produção (**yield system call)**
    - Permite dar acesso a outros processos para garantir a partilha de CPU entre processos.

Troca entre Processos (switching)

- **Estratégia não-cooperativa**
  - O que acontece quando há erros ou um processo se recusa a fazer chamadas sistema?
    - **ex.** Quando um processo entrava num ciclo infinito, apenas se podia ganhar controlo através de um *reboot*.
  - **Interrupção com temporizador (timer interrupt)**
    - Um temporizador pode ser programado para lançar um sinal de interrupção ao fim de alguns milissegundos.
    - Quando o sinal é lançado, os processos que estão em execução páram e o gestor de interrupções (I*nterrupt Handler*) é executado.
    - O SO ganha controlo do CPU e, desta forma, pode decidir parar o processo atual e iniciar outro.
  - No arranque (***boot time***) o SO informa o ***hardware*** de qual é o código que se deve correr quando o ***timer interrupt*** ocorre.
    - SO inicia o temporizador durante a sequência de arranque do computador.
    - O temporizador pode, eventualmente ser desligado.

Troca entre processos (switching)

- **Guardar e restaurar contexto**
  - O SO tem de decidir se continua a executar o processo atual ou se escolhe outro.
  - A decisão é feita pelo **Escalonador – *dispatcher***
    - As políticas de escalonamento de processos serão discutidas nos próximos slides.
    - Se a decisão é para realizar a troca, o SO executa código de baixo-nível (linguagem máquina) que é designado por *troca de contexto* – **context switch**.
    - Uma troca de contexto executa as seguintes tarefas:
      - Guarda valores dos registos do processo em execução (na pilha do núcleo do SO) **[executado pelo hardware]**– e mais tarde na área do processo se este não for mais executado **[executado pelo software]**.
      - Restaura alguns valores do processo que irá ser executado (da pilha do núcleo do SO) **[software e depois hardware]**.
      -

Troca entre processos (switching)

OS @ boot (kernel mode)	Hardware
initialize trap table	
	remember addresses of...
	syscall handler
	timer handler
start interrupt timer	
	start timer
	interrupt CPU in X ms

OS @ run (kernel mode)	Hardware	Program (user mode)
		Process A
		...
	timer interrupt	
	save regs(A) to k-stack(A)	
	move to kernel mode	
	jump to trap handler	
Handle the trap		
Call `switch()` routine		
save regs(A) to proc-struct(A)		
restore regs(B) from proc-struct(B)		
switch to k-stack(B)		
return-from-trap (into B)		
	restore regs(B) from k-stack(B)	
	move to user mode	
	jump to B's PC	
		Process B
		...

Concorrência?

- O que é que acontece quando, durante uma chamada ao sistema ocorre uma interrupção provocada pelo relógio (**timer interrupt**) ?
- O que é que acontece quando se está a tratar de uma interrupção e ocorre outra interrupção?
  - Estas questões serão tratadas mais à frente, na concorrência.
  - Os *SO* podem desligar as interrupções durante o processamento de uma interrupção.
    - Desta forma garantem que mais nenhuma interrupção surge quando estão a tratar de uma outra.
    - Têm de ter cuidado para esta situação não durar muito tempo (porquê?).
  - SO têm incorporados *mecanismos de fechadura* (***locking schemes***) para proteger o acesso concorrente a determinadas estruturas de dados.

Questões

- Qual é a diferença entre o modo de utilizador e modo de núcleo na execução de um sistema operativo?
- Defina sucintamente o que é uma chamada sistema.
- Explique as principais etapas de uma chamada sistema:
  - Executar uma instrução especial de exceção – trap;
  - Mudar para o modo kernel;
  - Executar a instrução de retorno – return-from-trap;
  - Guardar informação sobre os registos do processo;
- Explique qual é o papel do temporizador e por que razão ele permite a execução de um escalonador?