Sistemas Operacionais

André Luiz da Costa Carvalho

andre@icomp.ufam.edu.br

Aula 03 - Mecanismo: Execução Limitada Direta

Aula de Hoje

1. Execução Direta Limitada
   - Modo Kernel e Modo Usuário
   - Chamadas de Sistema e Traps
2. Troca de Contexto
   - Concorrência

Virtualização de CPU

Para virtualizar a CPU, o S.O. deve de alguma forma dividir a CPU entre vários programas (jobs, tasks, processos) ao mesmo tempo.

- Time Share

Desafios:

- Desempenho.
- Controle.

Aula de Hoje

1. Execução Direta Limitada
   - Modo Kernel e Modo Usuário
   - Chamadas de Sistema e Traps
2. Troca de Contexto
   - Concorrência

Execução Direta Limitada

Execução Direta: Programa roda diretamente na CPU.

- S.O.: Cria a struct do Processo, aloca memória, carrega as instruções do executável para a memória, carrega os argc e argv, limpa os registradores, para aí começar a executar a main().
- Programa: A partir daí, é o **código do programa** que está rodando, até que ele termine.
- S.O. Quando o programa termina, a memória que ele usou é liberada e a struct dele é removida da lista de processos.

Simples, não?

Execução Direta Limitada

Problemas:

Como garantir que o processo não fará nada "errado"?

Como parar um processo que está rodando para outro poder usar a CPU (base do **time share**)?

Como fazer isso de forma eficiente?

Operações Restritas

Execução direta é rápida. Programa roda diretamente na CPU, sem intermediários.

Problema: Mas e se o processo tentar executar uma instrução restrita?

- E/S, acesso a mais memória, etc.

Nos S.O.s antigos, a solução para isto era: problema dele.

Proteção

Saída para implementar proteção: Modos de execução:

Modo Usuário:Código que tem restrições nas instruções que pode fazer.

Modo Kernel:O modo em que o S.O. roda, em que todas as instruções são permitidas, incluindo E/S e controle de hardware.

Arquiteturas modernas muitas vezes possibilitam o uso de vários níveis.

Mas como um processo normal faz para fazer uma operação restrita?

Chamada de Sistema

Chamadas de Sistema são a API do S.O. para que processos possam utilizar funções do Hardware, como acessar disco, criar outros processos e alocar memória.

Instruções **trap**: Após estas instruções, o kernel (ou centro) do S.O. volta ao controle e o processador volta a modo Kernel. Isto é necessário para fazer as instruções protegidas.

Ao Final, o S.O. chama uma instrução de Retorno do Trap, que retorna os registradores para o programa atual e atualiza informações pertinentes os trap.

Trap Table

Para identificar o que fazer durante o Trap, o S.O. tem uma tabela de códigos para Trap, que é **inicializada na CPU durante o boot**.

- Por que?

Entre os primeiros comandos feitos pelo S.O., e é necessária antes do boot pela primeira vez.

Momento similar ao preenchimento da **tabela de interrupções**.

Segurança

Definir a tabela de Traps e interrupções são ações **privilegiadas**.

Em modo usuário programas não conseguem acessá-las. O acesso se dá apenas pelo índice na tabela, e não via endereços diretos

A segurança do modo Kernel é **essencial** para todo Sistema Operacional.

Timeline de um processo

OS @ boot (kernel mode)	**Hardware**	
initialize trap table		
	remember address of... syscall handler	

OS @ run (kernel mode)	**Hardware**	**Program (user mode)**
Create entry for process list		
Allocate memory for program		
Load program into memory		
Setup user stack with argv		
Fill kernel stack with reg/PC		
return-from-trap		
	restore regs from kernel stack	
	move to user mode	
	jump to main	
		Run main()
		...
		Call system call **trap** into OS
	save regs to kernel stack	
	move to kernel mode	
	jump to trap handler	
Handle trap		
Do work of syscall		
return-from-trap		
	restore regs from kernel stack	
	move to user mode	
	jump to PC after trap	
		...
		return from main **trap** (via `exit()`)

Aula de Hoje

Troca de Contexto

Troca de contexto é o que ocorre quando o S.O. retira um processo da CPU e coloca outro no lugar.

Parece simples né?

Porém, se um programa está rodando na CPU, por definição o S.O. não está na CPU no mesmo momento.

Mas se o S.O. não está rodando, como ele pode fazer qualquer coisa?
Como ele pode ganhar de volta o controle da CPU?

S.O.s cooperativos

Sistemas mais antigos tinham uma abordagem **cooperativa**.

O S.O. confia que seus processos se comportarão e entregarão a CPU de volta ao S.O. de tempos em tempos.

- Quando fizerem chamadas de sistema.
- Quando fizerem algo ilegal (divizão por zero, p. ex.)
- Quando o processo dá a preferência (chamada de sistema **yield**)

Mas e aí, e se der algo de errado com um processo?

S.O. não cooperativo

Sem hardware especial, não há nada que o S.O. possa fazer para retomar o controle da CPU de um processo à revelia deste.

Temporizador: um timer que gera interrupções após a passagem de um tempo definido (curto).

A cada intervalo, o processo rodando é interrompido, e um handler de interrupção definido é chamado.
Ele será responsável por definir se o processo atual será parado ou continua.

S.O. não cooperativo

O S.O. no boot deve:

- Informar o código que roda sempre que o timer gerar interrupção
- Iniciar o timer (modo Kernel)
  - Isso garante que mesmo q o S.O. entregue o controle da CPU para outro processo, a CPU volte para ele.

Quando ocorre uma interrupção, o Hardware salva automaticamente as informações dos registradores para poder voltar para o mesmo ponto assim que a mesma for tratada.

Timeline de um processo

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	

Troca de Contexto

Após o controle da CPU voltar para o S.O. (via chamada de sistema ou interrupção do temporizador), ele deve decidir se continua rodando o mesmo programa ou **troca** por outro.

Esta troca é decidida pelo Escalonador (scheduler).

Mas qual o Mecanismo desta troca?
Troca de contexto.

Troca de Contexto

Nela, o S.O. tem que salvar as informações do processo atual e em seguida carregar as informações do novo processo por cima deles.

Assim, quando ocorrer o retorno da interrupção do timer, o novo processo que será executado.
Mas como salvar um contexto?

Troca de Contexto

Código simples em assembler que copia os registradores gerais, Program Counter, ponteiros para pilha e depois troca pelos do processo novo.

```
# void swtch(struct context **old, struct context *new);
#
# Save current register context in old
# and then load register context from new.
.globl swtch
swtch:
  # Save old registers
  movl 4(%esp), %eax  # put old ptr into eax
  popl 0(%eax)        # save the old IP
  movl %esp, 4(%eax)  # and stack
  movl %ebx, 8(%eax)  # and other registers
  movl %ecx, 12(%eax)
  movl %edx, 16(%eax)
  movl %esi, 20(%eax)
  movl %edi, 24(%eax)
  movl %ebp, 28(%eax)
```

```
# Load new registers
movl 4(%esp), %eax  # put new ptr into eax
movl 28(%eax), %ebp # restore other registers
movl 24(%eax), %edi
movl 20(%eax), %esi
movl 16(%eax), %edx
movl 12(%eax), %ecx
movl 8(%eax), %ebx
movl 4(%eax), %esp  # stack is switched here
pushl 0(%eax)       # return addr put in place
ret                 # finally return into new ctxt
```

Concorrência

E se durante uma chamada de sistema ocorrer uma interrupção de hardware?

E se tem uma interrupção de timer durante o tempo do S.O.?

Duas saídas:

- Desativar interrupções
- Locks/travas/semáforos

Reboots

Nos sistemas cooperativos, uma solução simples para processos em loops infinitos ou que não devolvem o processador era **reiniciar** a máquina.
Parece uma gambiarra safada, mas na verdade até hoje é uma técnica muito útil para aumentar a robustez dos sistemas.
Isso porque reiniciar volta o sistema a um estado conhecido(e mais testado).
Reboots também recuperam recursos parados ou vazados (como memória). Além de serem simples de automatizar. Não é incomum sistemas complexos terem reboots periódicos totais ou parciais.

Finalizando

Execução Direta Limitada é: Os programas podem usar a CPU, mas apenas nos limites que o S.O. deixa.

O S.O. age em modo Kernel, onde todas as operações são permitidas, e pode recuperar o controle da CPU via chamadas de sistemas de processos e interrupções (como o temporizador).

Troca de Contexto é o processo de salvar os dados de um processo para poder trocar o mesmo por outro e depois, quando for novamente a vez dele, continuar de onde salvou.

Proxima Aula: Escalonamento: Parte 1.