Laboratório de Pesquisa em Redes e Multimídia

Escalonamento no Unix

Universidade Federal do Espírito Santo
Departamento de Informática

Sistemas Operacionais

Introdução

- Unix é um S.O. multiprogramado de tempo compartilhado:
  - Permite que vários processos sejam executados concorrentemente.
  - Processos ativos competem pelos recursos do sistema (memória, periféricos e, em particular, a CPU).
- A concorrência é emulada intercalando processos com base na atribuição de uma fatia de tempo (*time slice* ou *quantum*).
  - Processo executa durante a fatia de tempo a ele atribuída.
- O escalonador (*scheduler*) é o componente do S.O. que determina qual processo receberá a posse da CPU em um dado instante.

Projeto do Escalonador

- O projeto de um escalonador deve focar em dois aspectos:
  - *Política de escalonamento* – estabelece as regras usadas para decidir para qual processo ceder a CPU e quando chaveá-la para um outro processo.
  - *Implementação* – definição dos algoritmos e estruturas de dados que irão executar essas políticas.
- A política de escalonamento deve buscar:
  - Tempos de resposta rápidos para aplicações interativas.
  - Alto *throughput* (vazão) para aplicações em *background.*
  - Evitar *starvation* (um processo não deve ficar eternamente esperando pela posse da CPU).
- Os objetivos acima podem ser conflitantes!
  - O escalonador deve prover uma maneira eficiente de balancear esses objetivos, minimizando overhead.

Objetivos do Escalonador

- Sendo um sistema **de tempo compartilhado**, o Unix deve alocar a CPU de maneira **justa** para todos os processos.
- Naturalmente, quanto maior for a **carga no sistema, menor** será a **porção de CPU** dada a cada processo e, portanto, mais lentamente eles executarão.
- É função do escalonador **garantir uma boa performance** para todos os processos, considerando todas as expectativas de carga do sistema.
- Aplicações que concorrem pela CPU podem apresentar **características diversas** entre si, implicando em **diferentes requisitos** de escalonamento.
  - Ex: aplicações interativas, aplicações *batch* e aplicações de tempo real

Objetivos do Escalonador (cont.)

- Processos interativos (*shells*, editores, interfaces gráficas, etc.)
  - Passam grande parte do tempo esperando por interações
  - Interações devem ser processadas rapidamente; do contrário, os usuário observarão um alto tempo de resposta.
  - Requisito: reduzir os tempos médios (e a variância) entre uma ação de usuário e a resposta da aplicação de forma que o usuário não detecte/perceba este atraso (50-150 ms)
- Processos *batch* (que rodam em *background*, sem interação com o usuário)
  - Medida da eficiência do escalonamento: o tempo para completar uma tarefa na presença de outras atividades, comparado ao tempo para completá-la em um sistema "inativo", sem outras atividades paralelas.
- Processos de tempo real
  - Requisito: como as aplicações são "*time-critical*", o escalonador deve ter um comportamento previsível, com limites garantidos nos tempos de resposta.
  - Ex: aplicações de vídeo.

Objetivos do Escalonador (cont.)

- As funções do kernel – tais como gerência de memória, tratamento de interrupções e gerência de processos – devem ser executadas prontamente, sempre que requisitadas.
- Num S.O. bem comportado:
  - Todas as aplicações devem sempre progredir
  - Nenhuma aplicação deve impedir que as outras progridam, exceto os casos em que o usuário explicitamente permita isso
  - O sistema deve ser sempre capaz de receber e processar entradas interativas de usuário para que o mesmo possa controlar o sistema
    - Ex: "Ctrl+Alt+Del"

Troca de Contexto

- Ocorre sempre que o escalonador decide entregar a CPU a um outro processo.
- É uma operação custosa:
  - O contexto de hardware é salvo no respectivo PCB (*u area*).
  - Os valores dos registradores do próximo processo a ser executado são carregados na CPU
  - Adicionalmente, tarefas específicas a cada arquitetura de hardware podem ser realizadas, por exemplo: atualizar cache de dados, instruções e tabela de tradução de endereços; se houver pipeline de instruções, ele deve ser "esvaziado", etc.
- Esses fatores confirmam o alto custo da operação. Isso pode ter influência na implementação ou até mesmo na própria escolha da política de escalonamento a ser adotada.

Modos de Operação da CPU

- Com finalidade principal de proteção, a maioria dos computadores atuais fornece pelo menos dois modos de operação.
- O Unix requer do hardware a implementação de apenas 2 modos de operação:
  - *user mode* (menos privilegiado)
  - *kernel mode* (com mais privilégios).
- Processos de usuário rodam em *user mode*; logo, não podem – acidental ou maliciosamente –, corromper outro processo ou mesmo o kernel.

Máquina de Estados do Unix

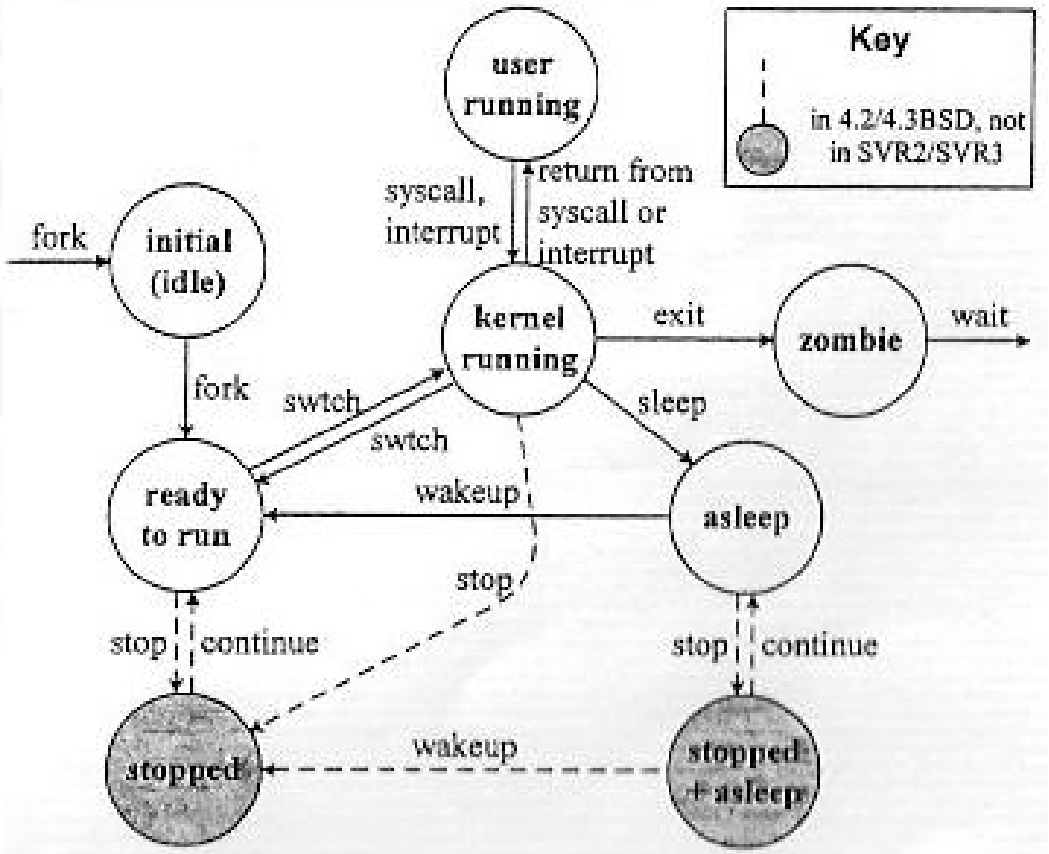

Figure 2-3. Process states and state transitions.

Escalonamento Tradicional

- Usado nos antigos sistemas Unix SVR3 e 4.3BSD. Projetado para lidar com os seguintes requisitos:
  - Tempo compartilhado
  - Ambientes interativos
  - Processos *background* (batch) e *foreground* rodando simultaneamente
- A política de escalonamento adotada objetiva:
  - Melhorar os tempos de resposta para os usuários interativos, e
  - Garantir, ao mesmo tempo, que processos *background* não sofram *starvation*
- O escalonamento tradicional do Unix é baseado em prioridades dinâmicas
  - A cada processo é atribuída uma prioridade de escalonamento, que é alterada com o passar do tempo.
  - O escalonador sempre seleciona o processo com a prioridade mais alta dentre aqueles no estado pronto-para-execução (*ready/runnable process*).

Escalonamento Tradicional (cont.)

- Se o processo não está no estado *running*, o kernel periodicamente aumenta a sua prioridade.
- Quanto mais o processo recebe a posse da CPU mais o kernel reduz a sua prioridade.
- Esse esquema previne a ocorrência de *starvation*.

- O valor da prioridade de um processo é calculado com base em dois fatores:
  - *Usage factor* – é uma medida do padrão de uso recente da CPU pelo processo.
  - *Nice value* – valor numérico relacionado ao uso da SVC *nice*.
- É a alteração dinâmica do *usage factor* que permite ao kernel variar dinamicamente a prioridade do processo

Escalonamento Tradicional (cont.)

- O escalonador tradicional usa o esquema de "*preemptive round robin*", isto é, escalonamento circular com preempção, para aqueles processos com a mesma prioridade.
- O *quantum* possui um valor fixo, tipicamente de 100 ms.
- A chegada de um processo de mais alta prioridade na fila de prontos força a preempção (troca de contexto) do processo em execução se esse está executando em *user mode*, mesmo que ele não tenha terminado o seu quantum.

Escalonamento Tradicional (cont.)

- O kernel do Unix tradicional é não-preemptivo. Isso significa que um processo executando em *kernel mode* nunca é preemptado (nunca perde a posse da CPU para um outro processo).
- Um processo pode voluntariamente perder a posse da CPU ao bloquear-se esperando por algum recurso.
- Pode haver preempção do processo em execução quando esse retorna de *kernel mode* para *user mode.* Nesse instante, se existir um processo mais prioritário na fila de prontos, ocorre a preempção (troca de contexto).

Prioridades dos Processos

- Prioridades recebem valores entre 0 e 127 (quanto menor o valor numérico maior a prioridade)
  - 0 – 49 : processos do kernel (*kernel priorities*)
  - 50 – 127 : processos de usuário (*user priorities*)
- Por ordem decrescente de prioridade...
  - Swapper
  - Controle de dispositivos de E/S orientados a
  - Manipulação de arquivos
  - Controle de dispositivos de E/S orientados a
  - Processos de usuário

Prioridades dos Processos (cont.)

- Campos de prioridade na *proc struct*
  - p_pri prioridade de escalonamento atual
  - p_usrpri prioridade em *user mode*
  - p_cpu medida de uso recende da CPU
  - p_nice fator *nice* controlável pelo usuário (SVC *nice*)
- O escalonador usa o campo *p_pri* para decidir qual processo escalonar. Se o processo roda em modo usuário então *p_pri = p_usrpri.*

Prioridades dos Processos (cont.)

- A prioridade do processo em modo usuário (*p_usrpri*) depende de dois fatores:
  - Uso recente da CPU (*p_cpu*)
  - Fator *p_nice*=[0-39], que é controlado pelo usuário (via SVC *nice()*)
    - Default: *p_nice* = 20
    - Quanto MAIOR o *p_nice*, MAIOR o *p_usrpri* (=> menor prioridade).
    - O comando *nice(x)* : -20 ≤ x ≤ +39 **=>** p_nice = p_nice + x
      - Usuários comuns só podem chamar o comando nice() com valores positivos (o que diminui a prioridade final do processo).
      - Comando nice com valores negativos (que aumentam o valor de *p_usrpri*, diminuindo a prioridade final do processo), somente pelo *superuser*.
        - *normal user*: Δ↑ *p_nice* ➔ Δ↓ prioridade
        - *superuser*: Δ↓ *p_nice* ➔ Δ↑ prioridade
- Processos *background* recebem automaticamente grandes valores de *p_nice* , por isso são menos prioritários.

Prioridades dos Processos (cont.)

- Quanto mais um processo ocupa a CPU, mais seu *p_cpu* aumenta, maior será seu *p_usrpri* e, consequentemente, menor será a sua prioridade.
- Quanto mais um processo espera para ser escalonado, menor será o seu *p_cpu* e, consequentemente, menor será o de *p_usrpri.* Com isso, a sua prioridade diminui menos do que os processos que usam muito a CPU.

Algoritmo de Escalonamento

- Na criação do processo: *p_cpu* = 0
- A cada *CPU tick (1tick = 10ms)* a rotina de interrupção do relógio incrementa *p_cpu* do processo em execução (até o máximo de 127), de modo a refletir o seu uso relativo de CPU.
- A cada 100 *CPU ticks* (100 x 10ms = 1000ms), ou seja, a cada segundo, via *callout*, o kernel invoca a rotina *schedcpu()*, que reduz o *p_cpu* de todos os processos de um "fator de decaimento" (*decay factor*) e recomputa a prioridade *p_usrpri* de todos os processos segundo a fórmula:

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2 \times p\_nice$$

- PUSER = 50, que é a prioridade base dos processos de usuário.
- No Unix SVR3 o *decay* possui valor fixo igual a ½.
- No Unix 4.3BSD a fórmula usada é: $$decay = \frac{2 \times load\_average}{(2 \times load\_average + 1)}$$

  onde *load_average* é o número médio de processos aptos a executar (*ready*) no último segundo.

Algoritmo de Escalonamento (cont.)

- Em função do recálculo de prioridades feito pela rotina *schedcpu()* pode haver troca de contexto
- Isto acontece quando o processo em execução fica com prioridade mais baixa do que qualquer outro processo pronto para executar (*ready*), considerando-se os novos valores de *p_usrpri* de todos os processos.
- Esssa estratégica previne *starvation* e favorece processos *I/O bound*.

Sleep Priority

- Após um processo ter sido bloqueado, por exemplo, dentro de uma SVC, ao ser acordado o kernel seta o valor de *p_pri* como sendo igual ao mesmo valor da prioridade do recurso pelo qual o processo esteve esperando.
  - Ex: 28 para terminais, 20 para disco
- Com a prioridade (maior) de kernel recém adquirida o processo será escalonado na frente de outros processos de usuário e continuará a sua execução em modo kernel a partir do ponto de bloqueio.
- Quando a SVC é finalmente completada, imediatamente antes de retornar ao modo usuário, o kernel faz *p_pri* = *p_usrpri*, restaurando o valor original da prioridade do processo em modo usuário (antes dele ter feito a SVC).

Referências

- VAHALIA, U. Unix Internals: the new frontiers. Prentice-Hall, 1996.
  - Capítulo 5 (até seção 5.5)