Sistemas Operativos

Resumo

Conteúdo

Capítulo 1

Estruturas dos Sistemas Operativos

1.1 Interface com o Utilizador

1.1.1 Linha de Comandos (CLI)

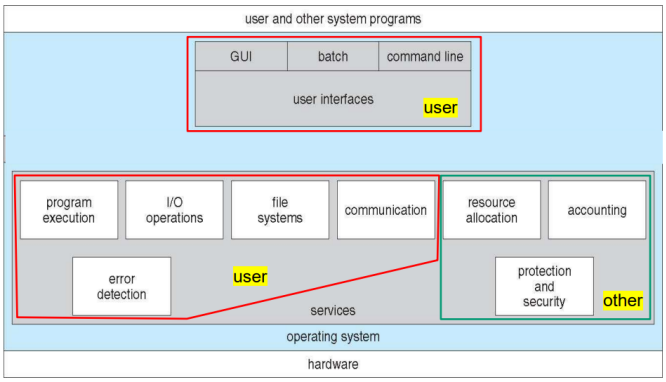
O Command Line Interpreter permite a execução direta de comandos, podendo ser implementado diretamente no kernel ou num programa sistema, como a Shell. Alguns comandos são interpretados pelo CLI e outros por programas específicos

1.1.2 Interface Gráfica (GUI)

A GUI é um tipo de interface mais amigável, desenhada para ser utilizada com rato e teclado, em que os ícones representam ficheiros, pastas e programas.

1.2 Serviços do Sistema Operativo

Como visto anteriormente, o SO fornece diferentes tipos de serviços, que nem sempre estão relacionados com o utilizador. Dada a sua classificação, podem ser agrupados da seguinte forma:



1.2.1 Serviços Relacionados com o Utilizador

- Interface com o Utilizador (UI) – Command Line Interpreter (CLI), Graphics User Interface (GUI), Batch
- Execução de programas – o sistema carrega um programa em memória, executa-o e termina-o, reportando eventuais erros.
- Operações de I/O - um programa em execução requer operações de I/O relativamente a ficheiros ou periféricos
- Manipulação do Sistema de Ficheiros – a grande maioria de operações das aplicações têm a ver com ficheiros (ler, escrever, apagar, mover) e diretórios(listar, procurar, gestão de acesso)
- Comunicações – troca e/ou partilha de dados entre aplicações residentes no mesmo computador ou em computadores ligados por rede
- Deteção de Erros – o SO tem de estar constantemente a par de todos os erros, a tentar reportá-los ou mesmo corrigi-los.

1.2.2 Outros Serviços

- Alocação de recursos – quando múltiplos utilizadores ou processos estão a utilizar o mesmo sistema simultaneamente.
- Accounting/Logging – Guardar historial das acções dos utilizadores e da utilização que fizeram dos recursos do sistema
- Protecção e Segurança – Garantir que a informação que existe ou transita num sistema multiutilizador e ligado em rede não é acedida ou modificada por quem não deve.

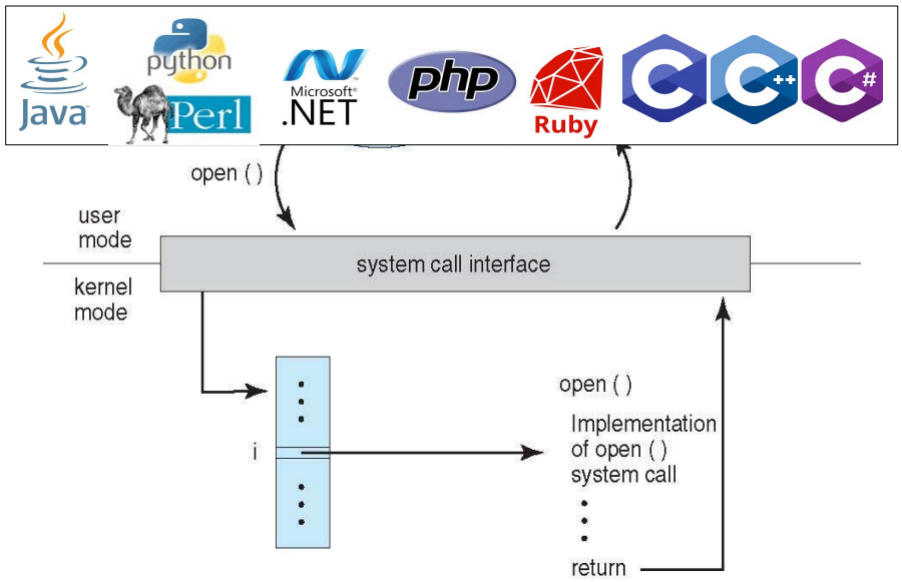
1.3 System Calls

1.3.1 Introdução

Os sistemas operativos utilizam system calls como uma forma de acesso ás suas funções por parte das linguagens de programação. Mais concretamente, trata-se de uma interface de programação fornecida pelo SO.

Geralmente, é escrita em linguagens de alto nível (C, C++ ou Java) e é semelhante a chamadas de funções. São acedidas pelos programas através da Application Program Interface (API), que encapsula o acesso direto às system calls (Win32 em Windows, POSIX em Unix e Java API na JVM), sendo que o acesso está implementado em bibliotecas que são carregadas com as aplicações.

Devem-se usar APIs em vês das system calls diretamente para obter portabilidade (independência da plataforma), esconder complexidade inerente às system call se obter um acréscimo de funcionalidades que otimizam o desempenho.



Cada chamada ao SO é identificada por um número (índice numa tabela).

1.3.2 Exemplo System Call em C

É invocada, em C, uma função (chamada ao SO): $printf(\text{"}Greetings\text{"})$.

A System Call Interface trata dos argumentos (registos, memória e pilha) e executa a instrução TRAP, mudando para o modo kernel/previligiado.

O kernel obtém o identificador da chamada e executa o respetivo código. Este código retorna e muda para modo utilizador. O kernel devolve também o status eo valor de retorno.

O invocador não sabe pormenores de implementação, sabe apenas utilizar a API, sendo que a biblioteca que implementa a API esconde detalhes.

1.4 Desenho e Implementação de Sistemas Operativos

Ao conceptualizar um sistema operativo é necessário definir objetivos para o utilizador (conveniência, facilidade de uso, confiabilidade, segurança, desempenho) e sistema (facilidade de desenho e implementação, flexibilidade, confiabilidade, eficiência). Existe ainda o princípio da separação entre política (decisão sobre o que fazer) e mecanismo (como o fazer).

1.4.1 Estrutura Simples

O MS-DOS é um sistema operativo de estrutura simples, concebido para fornecer o máximo de funcionalidades num mínimo de espaço. Não é um sistema modular e não possuí separação das interfaces e funcionalidades

1.4.2 Estrutura em Camadas

Nesta estruturação, o SO é dividido em vários níveis, cada um construído em cima do anterior. O nível mais baixo é o hardware e o mais alto é a interface com o utilizador. O princípio da modularidade implica que os níveis sejam escolhidos de forma a que cada um só utilize serviços dos níveis inferiores.

1.4.3 Estrutura de Sistema Microkernel

O aumento da dimensão do núcleo dificulta a sua gestão e manutenção. A solução consistem em ter um núcleo pequeno, onde apenas se coloca o essencial: Gestão de memória, gestão de processos e comunicação entre processos (através de mensagens).

Esta estrutura facilita a adição de extensões ao SO, já que novos serviços são adicionados no espaço do utilizador e não requerem alterações no núcleo, o que também trás uma maior fiabilidade pois a maior parte dos serviços correm em modo utilizador. Por outro lado, a comunicação de processos passa pelo núcleo, o que causa transtornos de desempenho.

1.4.4 Estrutura em Módulos

A maioria dos sistemas operativos modernos utilizam o conceito de módulo para implementar o núcleo. Utilizam uma aproximação orientada a objetos, em que cada componente fundamental é isolado. Os módulos comunicam por interfaces bem definidas e podem ser carregados dinamicamente no núcleo. Esta aproximação é semelhante à das camadas mas mais flexível e dinâmica.

1.5 Máquina Virtual

As máquinas virtuais estendem a abordagem por camadas, encapsulando o hardware. Oferecem aos clientes uma interface idêntica à oferecida por determinada arquitetura de hardware e podem ter sistemas operativos a correr sobre o hardware virtualizado. Os recurso físicos do computador são partilhados pelas diferentes instâncias das máquinas virtuais.

1.5.1 Java Virtual Machine (JVM)

As aplicações ignoram o hardware, usando apenas a Java API. O compilador de Java gera java bytecode, cujo verificador permite proteção no acesso á memória sem ajuda do hardware e é útil para hardware restrito (como telemóveis). O runtime executa o bytecode com um interpretador ou um compilador just-in-time (JIT).

1.6 Geração e Inicialização do Sistema Operativo

1.6.1 Gerar

Os sistemas operativos são concretizados de forma a poderem correr em diferentes máquinas e têm de ser configurados e gerados para uma máquina específica. Existe um programa que obtém a informação específica da máquina (system generation - SYSGEN), ficheiros, utilizadores e determina o hardware existente. É usado para:

- Compilar uma versão específica do sistema operativo, ou
- Selecionar os módulos utilizados no processo de ligação (link) dos módulos que constituirão o sistema operativo, ou
- Selecionar o código a executar

1.6.2 Inicializar

Existe um bootstrap program ou bootstrap loader, um pequeno pedaço de código que localiza o núcleo, carrega-o para memória e dá início à sua execução. Este procedimento pode ser feito em duas fases (Um bootstrap loader simples carrega um boot program mais complexo a partir do disco, o qual carrega o núcleo).

Ao receber um evento de reset, o CPU inicializa o registo IP com um endereço de memória pré definido, onde se encontra o bootstrap program. Este programa está em read only memory (ROM).

Capítulo 2

Processos e Escalonamento

2.1 Processos

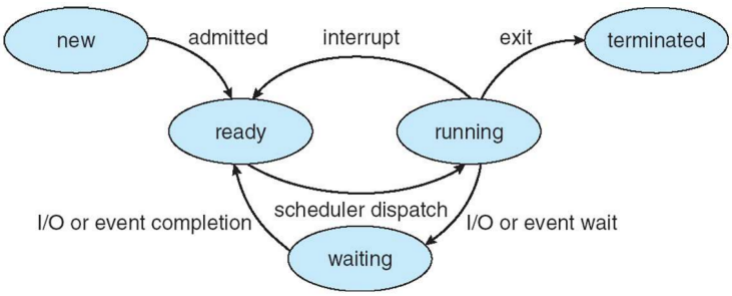
2.1.1 Introdução

Um processo consiste na execução de um programa (conjunto de linhas de código). Existem processos do SO e processos do utilizador, sendo o sistema operativo responsável pela sua criação, extinção, pausa, sincronização, comunicação e evita deadlocks (erros de sincronização). O program counter (IP, instruction pointer) determina a próxima instrução que será executada.

2.1.2 Estados de um Processo

Um processo passa por diversos estados:

- Novo (new): Ao ser criado
- Pronto (ready): À espera de lhe ser atribuído um processador
- Correr (running): A ser executado
- Bloqueado (waiting): À espera de um evento
- Terminado (terminated): Terminou a execução



2.1.3 Bloco de Controle de Processo (PCB)

Toda a informação associada a cada processo está contida no PCB, uma estrutura gerida pelo sistema operativo de forma a que a informação relativa ao processo se mantenha quando este entra e sai de estados de espera. São guardados:

- Estado do Processo
- Número do Processo
- Instruction Pointer
- Cópias dos registos do CPU
- ...

2.2 Operações Sobre Processos

2.2.1 Criação de Processos

Um processo pode criar novos processos durante a sua execução, sendo estes designados por processos filhos do processo que os criou. Estes processos necessitam de recursos, que podem ser obtidos do sistema operativo ou do seu processo pai.

Quando um processo cria um filho, podem executar ambos em paralelo ou o processo pai pode esperar que o filho termine antes de proceder(o processo pai deve terminar sempre em último).
O espaço de endereçamento do filho pode ser um duplicado do processo pai (fork) ou pode ser carregado um programa novo para o filho (exec).

Geralmente, apenas um processo é criado ao iniciar o sistema operativo, sendo todos os outros de alguma forma derivados desse processo inicial (pid = 0).

2.2.2 Terminar Processos

Um processo termina quando acaba de executar a sua última instrução e pede ao sistema operativo que o encerre através da system call exit(). Nessa altura o processo pode retornar um código de resultado para o processo pai e todos os recursos do processo são libertados pelo sistema operativo

2.2.3 Processos em Linux

A criação de um processo no Linux é efetuada através da duplicação do processo atual com a system call fork. Ao ser duplicado, é reservada uma nova zona de memória para o novo processo e é criada a sua PCB, no entanto, os processos são idênticos, à exceção dos seus PIDs, isto é, o código, dados e stack do processo filho são iguais aos do processo pai e ambos continuam a executar o mesmo código, a partir da linha do fork. Um processo filho pode substituir o seu código pelo código de outro programa através da system call exec, mantendo o seu PID e PPID.

Quando um filho termina, é enviado um sinal SIGCHLD ao processo pai, sendo que a forma que o processo pai tem para aceitar o status do processo filho é através do wait. O wait retorna o identificador (PID) do processo filho que terminou e retorna também o código de termino do processo filho como parâmetro. Existem suas macros para verificar e obter o valor passado do processo filho para o pai:

- WIFEXITED - verifica se foi recebido algum valor pela função exit
- WEXITSTATUS - acede ao valor recebido

Só é possível passar valores inteiros e pequenos até 8 bits. Para passar outros valores e tipos de dados de maior dimensão recorre-se a zonas de memória partilhada.

2.3 Comunicação Entre Processos (IPC)

Os processos concorrentes que executam num sistema operativo podem ser independentes ou cooperativos. Um processo diz-se independente se não pode afetar ou ser afetado por outros processos em execução, caso contrário é cooperativo.

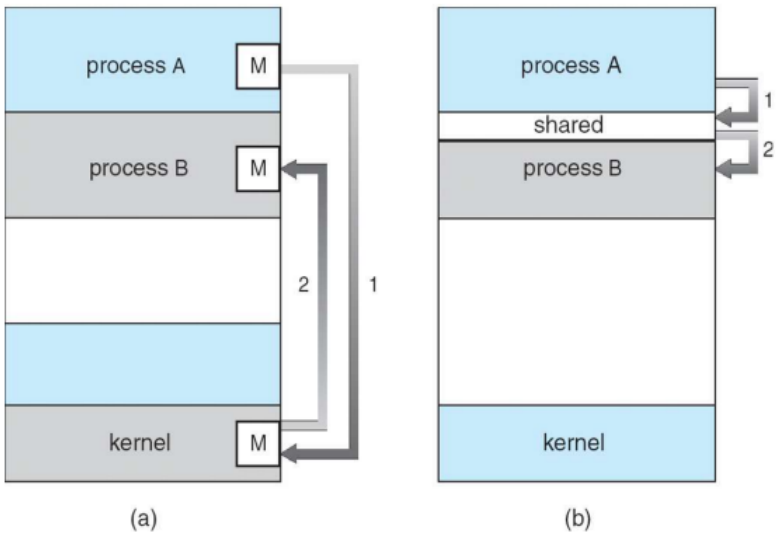
Existem várias razões para a existência de processos cooperativos, entre elas a partilha de informação, melhorias de desempenho, modularidade e conveniência.

2.3.1 Comunicação Baseada em Mensagens

Segundo este tipo de comunicação, a troca de dados entre processos é baseada em cópia (mensagens). Os processos comunicam sem recurso a espaços comuns aos dois processos, fornecendo duas operações: send e recieve.

2.3.2 Comunicação Baseada em Memória Partilhada

Neste tipo de comunicação, os processos pedem ao sistema para criar uma zona de memória comum e trocam informação escrevendo e lendo nesta zona de memória, havendo a necessidade de sincronização. A memória partilhada pode ser alocada com um tamanho fixo (estaticamente) ou variável (dinamicamente).



(a) Mensagens (b) Memória partilhada

2.3.3 IPC em Linux

Memória Partilhada

A Memória partilhada permite definir uma zona da memória que pode aparecer no espaço de endereçamento privado dos processos. Se pensarmos que existe uma tabela de tradução de endereços de memória virtuais em endereços de memória físicos (RAM) então partilhar memória consiste em ter tabelas de tradução de processos diferentes a apontar para o mesmo endereço físico.

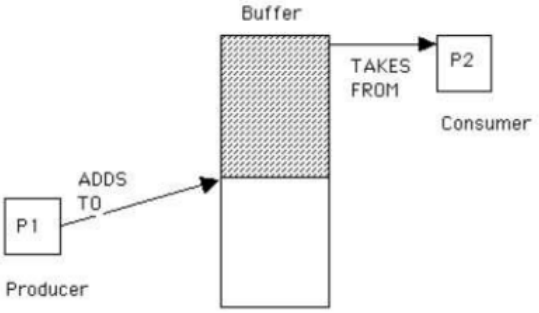
Pipes Sem Nome

Trata-se de um mecanismo muito simples, o pipe é um canal de comunicação em que os dados fluem de uma extremidade (de escrita) para a outra (de leitura).

Um pipe pode ser criado através de: int fd[2]; pipe(fd). Esta instrução cria um unnamed pipe e retorna dois descritores de stream: um para leitura (read fd[0]) e outro para escrita (write fd[1]).

Produtor - Consumidor

Um produtor produz informação que é consumida por um processo consumidor através de um buffer.



2.4 Escalonamento

2.4.1 Multiprogramação

Um único utilizador não consegue manter o CPU e os dispositivos de E/S sempre ocupados. O objetivo da multiprogramação é organizar os jobs de forma a que o CPU tenha sempre um para executar. Alguns jobs são mantidos em memória e quando um job em execução tem de esperar, outro toma o seu lugar.

Dado que há mais processos que recursos, o sistema operativo é responsável pelo seu escalonamento. Para este efeito são mantidas três filas de espera:

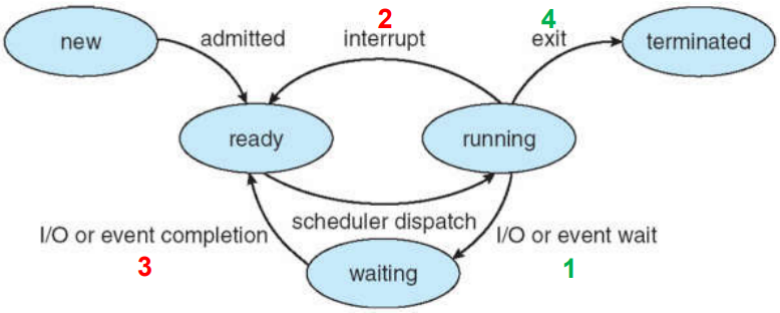
- Job queue - todos os processos
- Ready queue - todos os processos em memória prontos e à espera de executar
- Device queue - contem os processos à espera de I/O num dispositivo (uma para cada dispositivo)

Os processos migram frequentemente entre as diversas filas de espera e sempre que o CPU se encontra parado (idle) o sistema operativo tem que escolher um dos processos que se encontra na fila de processos prontos (ready queue) para ser atribuído ao CPU (running).

2.4.2 Comutação de Processos (Scheduler)

A comutação de processos ocorre quando o scheduler (escalonador) retira um processo do estado running e o substitui por outro, escolhido através de um algoritmo de escalonamento. Existem 4 situações em que o CPU pode efetuar escalonamento:

- Processo passa de estado em execução para estado em espera
- Processo passa de estado em execução para estado ready
- Processo passa de estado em espera para estado ready
- Processo termina



2.4.3 Dispacher

O dispacher é o componente que atribui o controlo do CPU ao processo seleccionado pelo escalonador do CPU.

Quando há uma comutação entre dois processos existe uma mudança de contexto, pois o SO deve salvaguardar o estado do antigo processo na PCB deste e carregar o estado do novo processo da sua PCB. Esta operação é indispensável para que os processos voltem ao estado running sem perder o seu contexto anterior. Para além disso existe uma comutação para modo utilizador e o CPU deve posicionar-se no programa para continuar a sua execução.

Existe ainda o conceito de dispach latency, o tempo que o dispatcher demora a parar um processo e a colocar outro em execução (tempo que demora a mudança de contexto)

2.4.4 Tipos de Escalonamento

Escalonamento Não Preemptivo

Neste tipo de escalonamento, o processo liberta o CPU por sua iniciativa: termina ou operação de I/O. Ocorre nas situações 1 e 4.

Escalonamento Preemptivo

Neste tipo de escalonamento um processo pode ser retirado do estado de execução sem ser por sua vontade (desafetação forçada), por exemplo quando o sistema operativo estipula a duração máxima da sequência de CPU. Ocorre nas situações 2 e 3. Este tipo de escalonamento gera problemas de partilha de informação (sincronização) ou no timer.

2.4.5 Critérios de Escalonamento

- Utilização do CPU - Manter o CPU ocupado tanto quanto possível. Critério a Maximizar
- Throughput - Resultados do trabalho efetuado pelo CPU: número de processos executados por unidade de tempo. Critério a Maximizar.

- Tempo total (turn arount time) - Tempo total que um processo leva a ser executado e tempo de espera para ser carregado em memória, tempo de espera na fila ready, tempo de I/O, tempo em execução. Critério a Minimizar.
- Tempo de espera - Tempo que um processo esteve na ready queue antes de ser ativado. Critério a Minimizar.
- Tempo de resposta (interatividade) - Tempo desde a submissão de um pedido até à obtenção da primeira resposta - Sistemas interativos. Critério a Minimizar.

2.4.6 Algoritmos de Escalonamento

Existem diversos algoritmos de escalonamento, cujo propósito é determinar qual o processo que irá ser executado num dado instante, em função de vários parâmetros.

Algoritmo First-Come, First-Served (FCFS)

Este algoritmo é implementado através de uma fila FIFO, sendo que os tempos médios de espera são geralmente longos pois pode acontecer que processos curtos fiquem atrás de processos longos na fila.

Algoritmo Shortest-Job-First (SJF)

Este algoritmo associa a cada processo a duração expectável do seu próximo ciclo de utilização do CPU (service time) e seleciona o processo com menor CPU burst (em caso de empate utiliza FCFS).

Existem duas versões deste algoritmo:

- Não preemptivo - o CPU só será libertado quando o processo completar o seu CPU burst
- Preemptivo - se um novo processo for colocado na ready queue com um CPU burst menor do que o restante tempo do processo em execução, este é substituído (Shortest-Remaining-Time-First, SRTF)

Algoritmo com Prioridades

A prioridade é representada por um valor inteiro associado ao processo e o CPU é atribuído ao processo da ready queue com maior prioridade. Novamente existem duas versões:

- Não preemptivo - o processo em curso é executado até finalizar o seu ciclo CPU
- Preemptivo - um processo com maior prioridade toma o lugar de um com menor prioridade

Um problema especifico deste algoritmo é a starvation: processos com baixa prioridade podem não ser executados. A solução consiste num processo de aging: aumentar a prioridade dos processos em espera à medida que o tempo passa.

Algoritmo Round-Robin (Estafeta)

Neste algoritmo, usado sobretudo em sistemas time sharing, cada processo usa o CPU um determinado tempo, o time quantum (10 a 100 ms). Ao terminar este tempo o processo é preemptivo e é colocado no final da ready queue.

Se existirem n processos com q time quantum então cada processo utiliza $1/n$ tempo de CPU em slots de, no máximo, q tempo de cada vez. Os processos podem esperar até $(n-1) \times q$. Em termos de desempenho, se q é longo então é igual ao FCFS. Se q é pequeno perde-se tempo demais na comutação de processos

Algoritmo com Filas Multi-Nível

Neste algoritmo a ready queue é dividida em várias filas, de acordo com a classificação de processos (foreground, background).

Por sua vez, cada fila tem o seu próprio algoritmo de escalonamento (foreground - RR, background - FCFS).

Existe ainda escalonamento entre filas:

- Cada fila tem prioridade diferente
- Cada fila tem uma percentagem de tempo

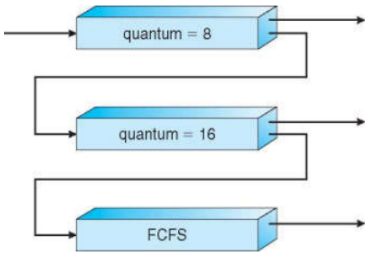
Algoritmo com Filas Multi-Nível com Retorno

Este algoritmo é uma combinação entre o algoritmo com filas multi-nível e o algoritmo com prioridades, no sentido em que existem várias filas e um processo de aging, que atua no sentido de mudar os processos de fila com o passar do tempo. Existem ainda os conceitos de promoção e demoção de processos entre as diferentes filas.

Exemplo:

Nestas três filas tem-se:

- Q0 – RR com time quantum de 8 ms
- Q1 – RR com time quantum de 16 ms
- Q2 - FCFS



Um novo processo entra em Q0, corre 8 ms e se não terminar nesse período vai para Q1, onde recebe mais 16 ms de CPU. Caso ainda não tenha terminado vai para Q2 e termina quando chegar a sua vez, nos 20% de tempo CPU que são atribuídos à fila FCFS

2.5 Escalonamento em Multi-Processadores

O scheduling torna-se mais complexo quando se dispõe de vários CPUs. Existem vários casos possíveis:

- Processadores diferenciados ou assimétricos
  - Escalonamento efectuado apenas por um processador – master server
  - Outros processadores apenas executam código do nível utilizador
  - Apenas um processador acede a dados do sistema
- Processadores indiferenciados ou simétricos (SMP)
  - Uma ready queue única: cada processador retira processos da fila sempre que fica livre (problemas de afinidade)
  - Uma ready queue por processador: os processos tendem a ficar sempre no mesmo processador (só migram quando houver grandes assimetrias)

2.5.1 Load Balancing

- Permite distribuir a carga de forma homogénea por todos os processadores
- Técnica de push: o scheduler global examina a carga de cada processador periodicamente e distribui os processos
- Tecnica de pull: o scheduler de cada processador vai buscar processos sempre que a sua ready queue fica vazia