Cap. 2 – GESTÃO DO PROCESSADOR

1. A modelação do ambiente de programação através da activação e desactivação de um conjunto de processadores virtuais, cada um deles associado a um processo particular, supõe que 2 factos essenciais relativos ao comportamento dos processos sejam garantidos. Quais são eles?

Para que tal modelo seja viável é preciso garantir que: a execução dos processos não é afectada pelo instante ou local no código onde ocorre a comutação, e que não são impostas quaisquer restrições relativamente aos tempos de execução, totais ou parciais, dos processos.

2. Qual é a importância da PCT na operacionalização de um ambiente de multiprogramação? Que tipo de campos devem existir em cada entrada da tabela?

A implementação de um ambiente multiprogramado implica a existência de uma gama variada de informação acerca de cada processo. Esta informação é mantida numa tabela, que é usada intensivamente pelo "scheduler" para fazer a gestão do processador e de outros recursos do S.C..

Os campos que existem em cada entrada da tabela são os seguintes:

- Identificadores: do processo, do processo-pai e do utilizador ao qual o processo pertence;
- Caracterização do espaço de endereçamento: sua localização (em memória principal ou na área de "swapping"), de acordo com o tipo de organização de memória estabelecido;
- Contexto do processador: valores de todos os registos internos do processador no momento em que se deu a comutação do processo;
- Contexto de I/O: informação sobre os canais de comunicação e "buffers" associados;
- Informação de estado e de "scheduling": estado de execução do processo (de acordo com o diagrama de estados) e outra informação associada.

3. O que é o "scheduling" do processador? Que critérios devem ser satisfeitos pelos algoritmos que o põem em prática? Quais são os mais importantes num sistema multiutilizador de uso geral, num sistema de tipo "batch" e num sistema de tempo real?

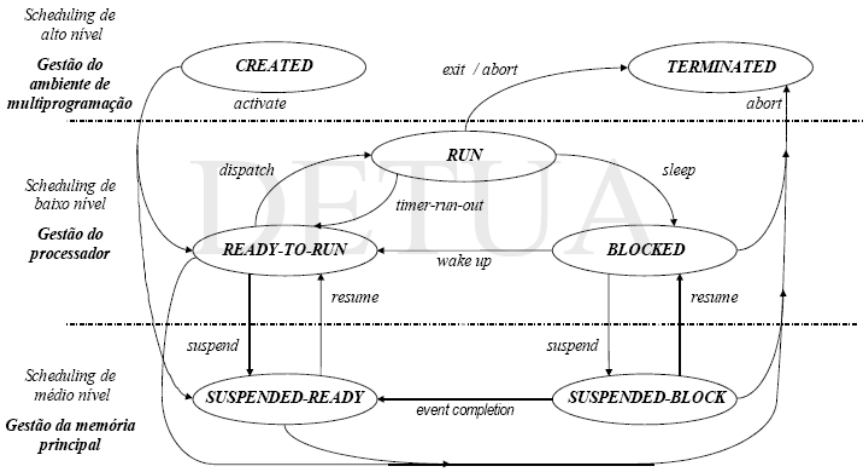
O "scheduling" do processador tem como finalidade determinar quando é que um processo deverá ser atribuído. Note-se que as funções básicas de qualquer "scheduler" do processador são a escolha do próximo processo a executar, atribuir-lhe um "time-slot" de tempo do processador e dar-lhe o controlo sobre o processador. Quando o processo é interrompido, deve determinar se está bloqueado ou se terminou a execução e colocá-lo no estado correcto, escolhendo de seguida o próximo processo a ser executado. Em sistemas com definição de prioridades, o "scheduler" pode ainda ter que reajustar a prioridade do processo em função da ocupação do último "time-slot" que lhe foi atribuído, bem como a duração do próximo "time-slot".

Os critérios que devem ser satisfeitos pelos algoritmos são:

- Justiça → Todo o processo deve ter direito à sua fracção de tempo do processador;
- Previsibilidade → O tempo de execução de um processo deve ser razoavelmente constante e independente da sobrecarga pontual a que o S.C. possa estar sujeito;
- "Throughput" → Deve-se procurar maximizar o número de processos terminados por unidade de tempo;
- Tempo de resposta → Deve-se procurar minimizar o tempo de resposta às solicitações feitas pelos processos interactivos;
- Tempo de "turnaround" → Deve-se procurar minimizar o tempo de espera pelo completamento de um "job" no caso de utilizadores num sistema "batch";
- "Deadlines" → Deve-se procurar garantir o máximo de cumprimento possível das metas temporais impostas pelos processos em execução;
- Eficiência → Deve-se procurar manter o processador o mais possível ocupado com a execução dos processos dos utilizadores (minimizar os tempos mortos);

Os critérios mais importantes num sistema multiutilizador de uso geral (interactivo) são os critérios de previsibilidade e de tempo de resposta; num sistema de tipo "batch" são os critérios de tempo de "turnaround", "throughput" e de eficiência; num sistema de tempo real são os critérios de "deadlines" e eficiência.

4. Descreva o diagrama de estados do "scheduling" do processador em 3 níveis. Qual é o papel desempenhado por cada nível? Num sistema de tipo "batch" multiprogramado fará sentido a existência de 3 níveis de "scheduling"?



O "scheduling" de baixo nível diz respeito à gestão do processador; o de médio nível diz respeito à gestão da memória principal e o "scheduling" de alto nível diz respeito à gestão do ambiente de multiprogramação.

Num ambiente multiprogramado, o número de processos que coexistem é muito elevado, pelo que a memória principal se torna limitativa. Para contrariar a situação, cria-se em memória de massa a área de "swapping", libertando-se assim espaço em memória principal e potencia-se um aumento da taxa de utilização do processador, surgindo 2 novos estados inerentes à transferência de endereçamento para a área de "swapping" – SUSPENDED-READY e SUSPENDED-BLOCK. No entanto, um outro aspecto a ter em conta é o controlo do grau de multiprogramação que deve ser o maior possível, mas garantindo sempre um equilíbrio adequado entre o serviço aos processos que coexistem e a ocupação do processador, surgindo 2 novos estados – CREATED e TERMINATED. Logo, faz sentido usar 3 níveis de "scheduling".

5. Os estados READY-TO-RUN e BLOCKED, entre outros, têm associadas filas de espera de processos que se encontram nestes estados. Conceptualmente, porém, existe apenas 1 fila de espera associada ao estado READY-TO-RUN, mas filas de espera múltiplas associadas ao estado BLOCKED. Em princípio, 1 por cada dispositivo ou recurso. Porque é que é assim?

Em ready-to-run , aguardam a atribuição do processador pra começar ou continuar a sua execução, mas tem apenas uma entidade responsável pela sua atribuição. Em blocked estão impedidos de continuar ate que um acontecimento ocorra, mas esses acontecimentos podem ser provocados por vários dispositivos ou recursos, logo a existência de filas de espera para cada um deles.

6. Indique quais são as funções principais desempenhadas pelo "kernel" de um S.O.. Neste sentido, explique porque é que a sua operação pode ser considerada como um serviço de excepções.

O "kernel" é responsável pelo tratamento das interrupções e por agendar a atribuição do processador e de muitos outros recursos do S.C.. Os processadores actuais têm 2 níveis de funcionamento: nível supervisor e nível utilizador. Note-se que só ocorre mudança de contexto imediatamente após o serviço de excepção, o qual pode ser causado por uma interrupção proveniente de um dispositivo de I/O ou quando ocorre uma interrupção do RTC, isto é, quando se esgota o "time-slot", ou quando um processo executa uma chamada ao sistema, ou seja, faz um pedido de atendimento a um dispositivo de I/O, sendo que o responsável pelo tratamento das interrupções é o "kernel".

7. O que é uma comutação de contexto? Descreva detalhadamente as operações mais importantes que são realizadas quando há uma comutação de contexto.

A comutação de contexto está inteiramente relacionada com a multiprogramação, onde é criada a imagem de aparente simultaneidade na execução de diferentes programas pelo mesmo processador. A comutação de contexto ocorre quando um processo bloqueia ou quando ocorre uma interrupção gerada por um dispositivo externo. Nestas situações, é salvaguardado o estado do processo na PCT (onde cada entrada possui identificadores, espaço de endereçamento, contexto do processador e de I/O e informação de estado e de "scheduling"). O processo passa nestas situações para o estado BLOCKED ou READY-TO-RUN. O "scheduler" selecciona um processo para execução e é restaurado o estado deste a partir da entrada respectiva da PCT.

8. Classifique os critérios que devem ser satisfeitos pelos algoritmos de "scheduling" segundo as perspectivas sistémica e comportamental, e respectivas subclasses. Justifique devidamente as suas opções.

- Perspectiva sistémica:
  - Critérios orientados para o utilizador: estão relacionados com o comportamento do S.O. na perspectiva dos processos ou dos utilizadores;
  - Critérios orientados para o sistema: estão relacionados com o uso eficiente dos recursos do S.O.;
- Perspectiva comportamental:
  - Critérios orientados para o desempenho: são quantitativos e passíveis de serem medidos;
  - Outro tipo de critérios: são qualitativos e difíceis de serem medidos de forma directa.

9. Distinga disciplinas de prioridade estática das de prioridade dinâmica. Dê exemplos de cada uma delas.

Os processos podem ser agrupados em níveis de prioridade distinta no que respeita ao acesso ao processador. As prioridades podem ser: estáticas, quando o método de definição é determinístico, ou dinâmicas, quando o método de definição depende da história passada da execução do processo.

Como exemplos temos:

- Prioridade estática: sistemas multiutilizador (por exemplo, o Unix SVR4 usa esta disciplina para os processos utilizador);
- Prioridade dinâmica: sistemas do tipo "batch".

10. Num S.O. multiutilizador de uso geral, há razões diversas que conduzem ao estabelecimento de diferentes classes de processos com direitos de acesso ao processador diferenciados. Explique porquê.

Em muitas circunstâncias concretas, considerar todos os processos em pé de igualdade não é o procedimento mais adequado. A minimização do tempo de resposta exige, por exemplo, que seja dada preferência a processos I/O-intensivos sobre processos CPU-intensivos na calendarização para execução. Em sistemas de tempo real, por outro lado, há processos associados com o tratamento de condições de alarme ou de acções meramente operacionais, cuja execução efectiva tem que ser mantida dentro de intervalos de tempo bem definidos.

11. Entre as políticas de "scheduling" "preemptive" e "non-preemptive", ou uma combinação das duas, qual delas escolheria para um sistema de tempo real? Justifique claramente as razões da sua opção.

- "Non-preemptive": quando, após a atribuição do processador a um dado processo, este o mantém na sua posse até bloquear ou terminar; a transição *TIMER-RUN-OUT* não existe neste caso. É característica dos sistemas de tipo "batch";
- "Preemptive": quando o processador pode ser retirado ao processo que o detém, por esgotamento do "time-slot" que lhe foi atribuído ou por necessidade de execução de um processo de prioridade mais elevada. É característica dos sistemas interactivos.

Em sistemas de tempo real, e quando uma aplicação é específica, escolheria a política de "scheduling" "non-preemptive". No caso de poder haver processos "bug", a política "preemptive" seria a mais indicada.

12. Foi referido nas aulas que os S.O.'s de tipo "batch" usam principalmente uma política de "scheduling" "non-preemptive". Será, porém, uma política pura, ou admite excepções?

Tendo em conta que os sistemas de tipo "batch" são caracterizados pelo fornecimento em bloco de "jobs" a serem processados sequencialmente, então a política "non-preemptive" é a mais indicada, pois garante a atribuição do processador ao processo até este terminar ou bloquear.

Pode admitir excepções no caso de o processo em execução apresentar "bugs". Neste caso, uma política "preemptive" seria a mais indicada pois retiraria de imediato a posse do processador ao processo em questão e atribui a outros que possam estar na fila de espera do estado READY-TO-RUN, o que contribui para um melhor aproveitamento e desempenho do processador.

13. Justifique claramente se a disciplina de "scheduling" usada em Linux para a classe SCHED_OTHER é uma política de prioridade estática ou dinâmica.

O processador só é atribuído a processos da classe SCHED_OTHER se não houver outro tipo de processos prontos a serem executados. Esta classe está associada aos processos utilizador. Para esta classe, o Linux usa um algoritmo baseado em créditos no estabelecimento da sua prioridade:

- No instante de recreditação *i*, a prioridade do processo *j* (equivalente ao número de créditos de execução que lhe são atribuídos) é calculada pela fórmula:

$$CPU_j(i) = \frac{CPU_j(i-1)}{2} + PBase_j + nice_j$$

- em que $CPU_j(i)$ representa a prioridade do processo *j* (o número de créditos que lhe são atribuídos) no instante de recreditação *i*, $CPU_j(i-1)$ o número de créditos não usados pelo processo *j* no intervalo de recreditação *i-1*, $Pbase_j$ a prioridade base do processo *j* e $nice_j$ o valor de alteração de prioridade dependente do utilizador (valor no intervalo -20 a 19);
- O "scheduler" calendariza para execução o processo com mais créditos (maior prioridade);
- Sempre que ocorre uma interrupção do RTC, o processo perde 1 crédito;
- Quando o número de créditos atinge o valor 0, o processo perde a posse do processador por esgotamento do "time-slot" e outro processo é calendarizado;
- Quando já não há processos na fila de espera dos processos prontos a serem executados com créditos não-nulos, procede-se a nova operação de recreditação que envolve todos os processos da classe, mesmo os bloqueados;

Este algoritmo combina assim 2 factores: a história passada de execução do processo e a sua prioridade, e maximiza o tempo de resposta dos processos I/O-intensivos sem produzir adiamento indefinido para os processos CPU-intensivos. Logo, é uma política de prioridade dinâmica.

14. O que é o "aging" dos processos? Dê exemplos de 2 disciplinas de "scheduling" com esta característica, mostrando como ela é implementada em cada caso.

O "aging" dos processos é descrito da seguinte forma: se ao fim de um determinado tempo o processo não for calendarizado para execução, aumenta-se a prioridade, prevenindo o seu adiamento indefinido.

Um meio de alterar a situação é incorporar no cálculo da prioridade o tempo que o processo aguarda a atribuição do processador no estado READY-TO-RUN, sendo que:

$$P = \frac{1 + bR}{fe_N}$$

onde *P* é a prioridade do processo, *R* é o tempo normalizado em termos da duração do intervalo de execução, e fe_N é a estimativa da fracção de ocupação da N-ésima janela de execução do processo.

Outro meio é usar múltiplas filas de espera dos processos READY-TO-RUN, e outro é o algoritmo usado pelo Linux para a classe SCHED_OTHER, que é baseado em créditos no estabelecimento da sua prioridade.

15. Distinga "threads" de processos. Assumindo que pretende desenvolver uma aplicação concorrente usando um dos paradigmas, descreva o modo como cada um afecta o desenho da arquitectura dos programas associados.

O conceito de *processo* corporiza as propriedades seguintes: *pertença de recursos* – um espaço de endereçamento próprio e um conjunto de canais de comunicação com os dispositivos de entrada / saída;
fio de execução (*thread*) – um *program counter* que sinaliza a localização da instrução que deve ser executada a seguir, um conjunto de *registos internos do processador* que contêm os valores actuais das variáveis em processamento e um *stack* que armazena a história de execução (um *frame* por cada rotina invocada e que ainda não retornou).
Estas propriedades, embora surjam reunidas num *processo*, podem ser tratadas separadamente pelo sistema de operação. Quando tal acontece, os *processos* dedicam-se a agrupar um conjunto de recursos e os *threads*, também conhecidos por *light weight processes*, constituem entidades executáveis independentes dentro do contexto de um mesmo processo. **Exemplo de um doc** em que existem 3 thread um que interage com o utilizador, o outro reformata o doc atravez das instrucoes dadas e o ultimo escreve o conteodo da memoria para o disco periodicamente. Se for ao nível do kernel vai ter um par de satck (sistema /utilizador) comum a todos od processos

16. Indique justificadamente em que situações um ambiente "multithreaded" pode ser vantajoso.

O "multithreading" representa a situação em que é possível criar-se "threads" múltiplos de execução no contexto de um processo. As vantagens de um ambiente "multithreaded" são as seguintes, com as respectivas descrições das situações em que o ambiente "multithreaded" pode ser vantajoso:

- Maior simplicidade na decomposição da solução e maior modularidade na implementação: programas que envolvem múltiplas actividades e atendimento a múltiplas solicitações são mais fáceis de conceber e mais fáceis de implementar numa perspectiva concorrencial do que numa perspectiva puramente sequencial;
- Melhor gestão de recursos do S.C.: havendo uma partilha do espaço de endereçamento e do contexto de I/O entre os "threads" em que uma aplicação é dividida, torna-se mais simples gerir a ocupação da memória principal e o acesso aos dispositivos de I/O de uma maneira eficaz;
- Eficiência e velocidade de execução: uma decomposição da solução em "threads" por oposição a processos, ao envolver menos recursos por parte do S.O., possibilita que operações como a sua criação e destruição e a mudança de contexto se tornem menos pesadas e, portanto, mais eficientes.

17. Que tipo de alternativas pode o S.O. fornecer à implementação de um ambiente "multithreaded"? Em que condições é que num multiprocessador simétrico os diferentes "threads" de uma mesma aplicação podem ser executados em paralelo?

Como alternativa à implementação de um ambiente "multithreaded", temos que esse ambiente pode ser fornecido por bibliotecas, as quais fornecem primitivas para a sua criação, gestão e "scheduling", que correm em modo utilizador.

Tendo em conta que num multiprocessador simétrico a memória é a mesma para todos os processadores, sendo que o tempo de acesso é o mesmo, então num multiprocessador deste tipo os diferentes "threads" de uma mesma aplicação podem ser executados em paralelo quando as tarefas de cada "thread" são independentes entre si.

18. Explique como é que os "threads" são implementados em Linux.

À parte da chamada ao sistema *fork( )* que cria um novo processo a partir de um já existente por cópia integral do seu contexto alargado (espaço de endereçamento, contexto de I/O e contexto do processador), existe uma outra, *clone( )*, que cria um novo processo a partir de um já existente por cópia apenas do seu contexto restrito (contexto do processador), partilhando o espaço de endereçamento e o contexto de I/O e iniciando a sua execução pela invocação de uma função que é passada como parâmetro. Logo, o Linux não distingue efectivamente processos e "threads", os quais designa por "tasks" que são tratadas de igual forma pelo "kernel".

19. O principal problema da implementação de "threads", a partir de uma biblioteca que fornece primitivas para a sua criação, gestão e "scheduling" no nível utilizador, é que quando um "thread" particular executa uma chamada ao sistema bloqueante, todo o processo é bloqueado, mesmo que existam "threads" que estão prontos a serem executados. Será que este problema não pode ser minimizado?

Sim atravez do *kernel threads* – em que os *threads* são implementados directamente ao nível do *kernel* que providencia as operações de criação, gestão e *scheduling* de *threads*; a sua implementação é menos eficiente do que no caso anterior, mas o bloqueio de um *thread* particular não afecta a calendarização para execução dos restantes.

20. Num ambiente "multithreaded" em que os "threads" são implementados no nível utilizador, vai haver um par de "stacks" (sistema / utilizador) por "thread", ou um único par comum a todo o processo? E se os "threads" forem implementados ao nível do "kernel"? Justifique.

Quando os "threads" são implementados ao nível do utilizador, existe um único par de "stacks" (sistema / utilizador) comum a todo o processo, na medida em que o utilizador não tem noção do que se passa a este nível. O utilizador vê este ambiente como um processo único em execução com um "program counter" e respectiva "stack" sem se preocupar com as diferentes tarefas que o constituem.

Ao nível do "kernel", que é o responsável pelo tratamento das interrupções e pela calendarização da atribuição do processador e de muitos outros recursos do S.C., já tem uma perspectiva diferenciada sobre cada "thread", havendo portanto um par de "stacks" por "thread", isto é, cada "thread" deve ter um "program counter" e um "stack" (chamadas a funções).