Threads

Um conceito muito interessante no contexto de processos é a thread. Até agora, vimos um modelo em que o processo é um programa que executa um único encadeamento de execução (de forma sequencial). A maioria dos sistemas operacionais modernos estendeu o conceito de processo para permitir que ele tenha vários encadeamentos de execução e, assim, possa executar mais de uma tarefa por vez.

A ideia é a seguinte: o sistema operacional mantém vários processos na memória simultaneamente. O sistema operacional escolhe e começa a executar um desses processos. Eventualmente, o processo pode ter que esperar que alguma tarefa, como uma operação de E/S, seja concluída. Em um sistema não multiprogramado, a CPU ficava inativa. Em um sistema multiprogramado, o sistema operacional simplesmente alterna e executa outro processo. Quando esse processo precisa esperar, a CPU alterna para outro processo e assim por diante. Eventualmente, o primeiro processo termina de esperar e recupera a CPU. Desde que pelo menos um processo precise ser executado, a CPU nunca ficará inativa.

Uma thread de execução é a menor sequência de instruções programadas que pode ser gerenciada independentemente por um planejador, que normalmente faz parte do sistema operacional. A implementação de threads e processos difere entre sistemas operacionais. mas na maioria dos casos, um thread é um componente de um processo. Vários threads podem existir em um processo, executando simultaneamente e compartilhando recursos, como a memória, por exemplo, enquanto diferentes processos não compartilham esses recursos. Em particular, os threads de um processo compartilham seu código executável e os valores de suas variáveis a qualquer momento (SILBERSCHATZ; GAGNE; GALVIN, 2012).

Os threads diferem dos processos do sistema operacional multitarefa tradicional nas seguintes características: Processos são tipicamente independentes, enquanto os threads existem como subconjuntos de um processo. Os processos carregam consideravelmente mais informações de estado do que os threads, enquanto vários threads dentro de um processo compartilham o estado do processo, bem como a memória e outros recursos. Os processos têm espaços de endereço separados, enquanto os segmentos compartilham seu espaço de endereço. Os processos interagem somente por meio de mecanismos de comunicação entre processos fornecidos pelo sistema. A alternância de contexto entre threads no mesmo processo é tipicamente mais rápida que a alternância de contexto entre processos.

O escalonamento em tais sistemas depende se as threads são suportadas em nível de usuário ou em nível de núcleo (ou ambos). Vamos considerar primeiro as threads em nível de usuário. Como o núcleo não tem conhecimento da existência de threads, ele funciona normalmente, selecionando um processo, digamos, A, e dando a esse processo o controle de seu quantum. Dentro do processo A existe um escalonador de threads que seleciona qual thread vai executar, digamos, A1. Como não existem interrupções de relógio para multiprogramar as threads, essa thread pode continuar executando o quanto quiser. Se ela consumir o quantum inteiro do processo, o núcleo selecionará outro processo para executar. Quando o processo A finalmente for executado outra vez, a thread A1 retomará sua execução. Ela continuará a consumir todo o tempo de A, até que termine. Entretanto, seu comportamento antissocial não afetará outros processos. Eles receberão o que o escalonador considerar como sua fatia apropriada,

independentemente do que estiver acontecendo dentro do processo A. Agora, considere o caso em que as threads de A têm relativamente pouco trabalho a fazer por rajada de CPU, por exemplo, 5 ms de trabalho, dentro de um quantum de 50 ms. Consequentemente, cada uma deles é executada por um pouco de tempo e, então, devolve a CPU para o escalonador de threads. Isso pode levar à sequência A1, A2, A3, A1, A2, A3, A1, A2, A3, A1, antes do núcleo comutar para o processo B.