INTRODUÇÃO AOS SISTEMAS OPERACIONAIS outubro/2013

SEMANA 10

Operações nos processos.
Processos cooperativos, comunicação entre processos.

1 - Introdução

Um processo, ao longo da sua vida, passa por diferentes estados, que mudam a partir de operações sobre este processo. Além disso, o processo pode tanto cooperar e se comunicar com outros processos quanto ser independente.

2 - Operações nos processos

Operações nos processos ou operações sobre processos são operações realizadas sobre os estados dos processos. A maior parte dessas operações diz respeito às alterações nos possíveis estados do processo, em acordo com as necessidades do sistema operacional.

Uma vez realizada uma operação sobre um processo, este muda de um estado inicial para um estado final.

Considerando um modelo de processo com 5 estados, os possíveis estados são:

i. novo [new]: neste estado o processo está sendo criado no sistema e sendo alocado na memória. Todos os recursos necessários à execução do processo são reservados durante a passagem do processo por este estado, o que acontece uma única vez. Vários processos podem estar simultaneamente neste estado;

ii. pronto [ready]: é o estado onde os processos aguardam a liberação da CPU para que possam iniciar ou continuar seu processamento. É uma fila de processos prontos gerenciada pelo sistema operacional, que a organiza em acordo com as informações contidas no contexto de software [identificação, quotas e privilégios]. Vários processos podem estar simultaneamente neste estado;

iii. rodando [running]: nesse estado o processo efetivamente utiliza a CPU. Ele permanece no processador até que seja interrompido ou termine a sua execução. Para uma máquina com um único processador, apenas um processo pode permanecer neste estado de cada vez;

iv. bloqueado [blocked]: neste estado estão todos os processos que sofreram algum tipo de interrupção de I/O ou que aguardam algum evento, e permanecem neste estado até que a intervenção no sistema operacional seja resolvida ou ocorra o evento que causou o bloqueio. Vários processos podem estar neste estado simultanemante;

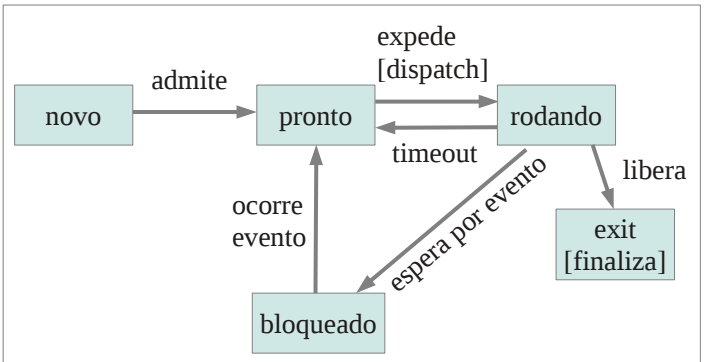

Fig 1 – Modelo de processo com 5 estados

v. exit [finaliza]: é o estado final do processo, quando termina o processamento. Vários processos podem estar neste estado simultaneamente.

Considerando estes 5 estados dos processos, as seguintes operações básicas podem ser realizadas neles:

Operação	Estado inicial do processo	Estado final do processo
criação [create]	-	pronto
destruição [destroy]	rodando/pronto/bloqueado	-
requisição [request]	rodando	bloqueado
liberação [release]	bloqueado	pronto
expiração [time_out]	rodando	pronto
escalonador [scheduler]	pronto	rodando
escalonador [scheduler]	rodando	pronto

As operações básicas podem ser sobre processos ou recursos. Por exemplo, operações básicas sobre processos são criação e destruição, e sobre recursos são requisição [request] e liberação [release].

Num outro exemplo, a operação suspende [supend] sobre um processo rodando leva para o estado parado [stopped]. A operação muda_prioridade [change_priority] altera a prioridade do processo. Num Unix, se uma aplicação [ou comando] roda no terminal, o comando <CTRL>z leva o processo para o estado parado. Ainda no Unix, os comando nice ou renice alteram a prioridade do processo.

3 - Processos cooperativos

Dois ou mais processos podem executar de modo independente ou cooperativo. Para haver cooperação entre processos, deve haver algum meio pela qual eles possam se comunicar.

Processos independentes são aqueles que não afetam ou que não podem ser afetados pela execução de outro processo. Por exemplo, qualquer processo que não compartilhe dados com outro processo é um processo independente, já que qualquer processo que compartilhe dados com outros processos é um processo cooperativo.

Processos cooperativos podem afetar ou ser afetados pela execução de outro processo. As vantagens dos processos cooperativos são:

- compartilhamento de informações: podem haver vários usuários interessados em uma mesma informação. Por exemplo, deve ser permitido acesso concorrente a um arquivo compartilhado;

- aumento na velocidade de processamento: se o computador tem mais de uma unidade de processamento, uma tarefa pode ser dividida em duas ou mais subtarefas e cada uma destas tarefas pode ser executada em paralelo com as demais;

- modularidade: o sistema pode ser construído de maneira modular, que divide as funções em processos separados;

- conveniência: um mesmo usuário pode ter muitas tarefas a serem realizadas simultaneamente. Por exemplo, o usuário pode editar um arquivo, imprimir e compilar um programa, tudo isso simultaneamente.

Uma execução concorrente que requer cooperação entre processos requer também mecanismos que permitam aos processos comunicarem-se uns com os outros para sincronizarem essas ações.

Existem basicamente dois esquemas de comunicação complementares: memória compartilhada e sistema de troca de mensagens. Esses dois esquemas não são mutuamente exclusivos, e podem ser usados simultaneamente num único sistema operacional ou mesmo num único processo.

O método de memória compartilhada requer que processos em comunicação compartilhem o uso de variáveis. Em um sistema de memória compartilhada, a responsabilidade para estabelecer comunicação entre processos é dos programadores da aplicação, pois o sistema operacional fornece o meio que é a memória compartilhada.

Já no método de troca de mensagens, a responsabilidade de estabelecer uma comunicação entre os processos é do próprio sistema operacional.

Processos cooperativos que compartilham diretamente um mesmo espaço de endereços lógicos podem ser vistos como fluxos de execução. Um fluxo de execução ou tarefa, é uma unidade básica de utilização da CPU que compartilha com fluxos irmãos a seção de código, a seção de dados e os recursos do sistema operacional. Uma tarefa não faz nada se não tiver algum fluxo de execução a ela associado, e um fluxo de execução deve estar associado a apenas uma tarefa. O compartilhamento extensivo de recursos entre fluxos de execução irmãos torna mais eficiente a mudança de contexto entre eles, assim como a criação de novos fluxos, em comparação com as trocas de contexto entre processos tradicionais.

Um paradigma para processos cooperativos é a relação produtor/consumidor: o processo produtor produz informação que é consumida por um processo consumidor. Por exemplo, uma aplicação produz caracteres que são consumidos pelo dispositivo de impressão, ou então um compilador produz código assembly que é consumido pelo montador[1].

O produtor e o consumidor devem estar sincronizados para o consumidor não tentar consumir um item ainda não produzido. O consumidor deve esperar pela produção do item.

Numa execução concorrente, enquanto um buffer[2] é preenchido pelo produtor, o consumidor está esvaziando este mesmo buffer.

Quando não há limite prático no tamanho do buffer, chamado de *buffer de capacidade ilimitada* ou não limitado, o produtor sempre pode produzir novos itens.

Já quando é assumido um tamanho fixo para o buffer, chamado de *buffer de capacidade limitada*, o produtor deve esperar caso o buffer esteja cheio.

4 - Comunicação entre processos

Frequentemente os processos do sistema precisam se comunicar com outros processos para a execução de determinadas tarefas. Neste contexto, dois paradigmas são mais utilizados: memória compartilhada [*shared memory*] e troca de mensagem [*message passing*]. No primeiro, os processos tem acesso a uma área comum para leitura e escrita de informações. A segunda abordagem consiste no uso de primitivas para o envio/recebimento [*send/receive*] de mensagens entre processos.

A comunicaçao entre processos [IPC, Inter Process Communication] é uma situação comum que ocorre quando dois ou mais processos precisam compartilhar ou trocar dados entre si.

IPC é um mecanismo de comunicação que usa sistema de mensagens que permite aos processos se comunicarem entre si e sincronizarem as suas ações sem necessidade de compartilhar o mesmo espaço de endereçamento. É particularmente útil na comunicação entre processos em diferentes computadores, como é o caso dos ambientes distribuídos.

A comunicação entre processos pode ocorrer em várias situações diferentes, tais como:

- redirecionamento da saída de um comando para outro [exemplo: uso de pipes[3] "|" na linha de comando];
- envio de arquivos para impressão;

1 Montador: o montador traduz as instruções em assembly para um arquivo com instruções de máquina binária e dados binários.

2 Buffer: retentor, é uma área para armazenamento temporário de dados que geralmente usa a memória primária (RAM).

3 Pipe: é um canal de comunicação de mão única onde um processo escreve e outro processo lê. O sistema operacional que implementa pipes [por exemplo, Unix] usa sinais para indicar aos processos envolvidos que um pode escrever no bloco pipe enquanto o outro pode ler.

- transmissão de dados através da rede;
- transferência de dados entre periféricos, etc.

A comunicação entre processos ocorre, geralmente, através da utilização de recursos comuns aos processos envolvidos nessa comunicação, e não envolve mecanismos de interrupção devido a sua complexidade e limitações de performance. Devido a isso, as interrupções são reservadas para a administração do sistema em si e não para a comunicação entre processos.

Para a comunicação entre processos é necessário algum mecanismo bem estruturado, como:

- buffers: buffer [retentor] é uma região temporária de memória utilizada para escrita e leitura de dados. Esses dados podem ser originados de dispositivos ou processos. Os buffers podem ser implementados em software [é o mais usado] ou hardware. Buffers normalmente são usados quando existe uma diferença entre a taxa com que os dados são recebidos e a taxa com que eles podem ser processados, ou mesmo no caso em que essas taxas são variáveis;

- semáforos: semáforo é uma variável de sincronização que pode ter valores inteiros não-negativos. Foi inventado por Edsger Dijkstra em 1965;

- memória compartilhada: na memória compartilhada, o código para implementar o buffer é explicitamente escrito pelo programador da aplicação.

Dois mecanismos são possíveis para comunicação entre processos:

- comunicação por meio de memória compartilhada: os processos usam uma área de armazenamento comum, e o código para implementar as operações de armazenagem e leitura de valores nesta área são escritos explicitamente pelo programador da aplicação;

- sistema de troca de mensagens: o sistema operacional permite que os processos se comuniquem uns com os outros e realizem operações sincronizadamente, através da troca de mensagens.

A estrutura básica da comunicação entre processos com troca de mensagens têm por função permitir que os processos se comuniquem sem a necessidade de usar variáveis compartilhadas. Um mecanismo de suporte a esse sistema deve fornecer pelo menos duas operações, uma para enviar e outra para receber a mensagem. O mecanismo IPC fornece duas primitivas:

i. send [message]: envia mensagem de tamanho fixo ou variável;

ii. receive [message]: recebe mensagem.

Desse modo, se os processos P e Q desejam se comunicar, eles precisam estabelecer um link [canal] de comunicação e trocar mensagens com send/receive, por exemplo:

- send (P, message) para enviar uma mensagem para P;
- receive (Q, message) para receber uma mensagem de Q.

As propriedades do canal [link] de comunicação são:

- o canal é estabelecido automaticamente;
- o canal é associado exatamente a um par de processos;
- entre cada par existe exatamente um canal;
- o canal pode ser unidirecional, mas normalmente é bidirecional.

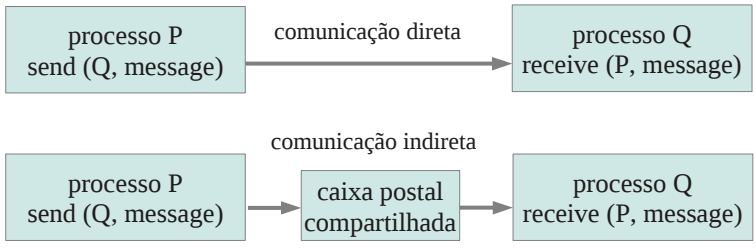

Fig 2 – Sistema de troca de mensagens entre processos P e Q

Do ponto de vista da sincronização de primitivas, as troca de mensagens podem ser bloqueante [síncrona] ou não bloqueante [assíncrona]. As formas mais comuns para as primitivas send/receive são:

- send [envio] bloqueante: o processo de envio é bloqueado até que a mensagem seja recebida pelo processo receptor ou pela caixa postal. Ou seja, o processo de envio aguarda até que a mensagem seja recebida;

- send [envio] não bloqueante: o processo de envio envia a mensagem e retorna à operação;

- receive [recepção] bloqueante: o processo receptor é bloqueado até que a mensagem esteja disponível. Ou seja, o receptor aguarda até que uma mensagem esteja disponível;

- receive [recepção] não bloqueante: o processo receptor recebe uma mensagem válida ou então nada.

E para que dois processos se comuniquem, eles devem ter uma maneira de se referir um ao outro. Para isso, eles podem usar tanto comunicação direta quanto comunicação indireta:

i. comunicação direta: cada processo que queira se comunicar com outro deve usar explicitamente o nome do processo receptor ou remetente da mensagem. O estabelecimento dessa comunicação é automático, cada canal de comunicação usualmente é bidirecional e liga um par de processos;

ii. comunicação indireta: as mensagens são enviadas e recebidas de caixas postais. Uma caixa postal pode ser vista abstratamente como um local com identificação única no qual os processos podem colocar mensagens e do qual mensagens podem ser retiradas.

No esquema de comunicação indireta, um processo pode se comunicar com outro por intermédio de diversas caixas postais diferentes, e a comunicação ocorre pelo compartilhamento no uso de alguma caixa postal. Um canal [link] tem sua capacidade determinada pelo número máximo de mensagens que pode estar na fila de mensagens associada ao canal.

Nos modelos de comunicação entre processos, o buffering da fila de mensagens pode ser implementado de três maneiras:

i. capacidade zero: o canal não guarda mensagens, então o processo de envio deve esperar pelo receptor [rendevous]. É o caso *send bloqueante*, onde o processo de envio aguarda até

que a mensagem seja recebida;

ii. capacidade limitada: existe um tamanho limitado para a fila de mensagens, ou seja, existe um número máximo de mensages que pode residir na fila. Enquanto essa fila não atingir o seu limite, o *send* [envio] é não bloqueante e o processo de envio envia a mensagem e retorna à operação. Mas se o canal estiver cheio, o *send* [envio] é bloqueante e o processo de envio é bloqueado até que haja espaço disponível nessa fila;

iii. capacidade ilimitada ou não limitada: a fila não tem limites e admite qualquer número de mensagens. Aqui, não existe *send* [envio] bloqueante.

A comunicação entre processos também pode ocorrer através de sockets, RPC ou RMI. Essa forma de comunicação é bastante comum nos sistemas tipo cliente-servidor.

- sockets: soquetes são definidos como pontos finais de comunicação entre processos remotos. O soquete abre uma conexão de rede para as aplicações se comunicarem, e permite o envio e recebimento de dados pela rede. Soquete é software, implementado em sistemas da família Unix e família Windows, entre outros. Por exemplo, uma aplicação pode abrir um soquete para enviar e receber mensagens TCP/IP para um sistema remoto, que simplifica a tarefa de comunicação em rede pois para enviar dados basta escrever neste soquete, e para receber dados basta ler deste socket. Exemplo de soquete: 192.168.1.10:53, onde 192.168.1.10 é o endereço IP remoto e 53 a porta remota. No protocolo TCP/IP, a comunicação em rede ocorre através de um par de soquetes;

- RPC [Remote Procedure Calls]: chamadas de procedimentos remotos é uma forma de comunicação entre processos que permite a uma aplicação ou subrotina executar em outro espaço de endereçamento, normalmente em outro computador da rede. E isso sem que o programador precise explicitamente escrever o código dessa interação remota. O RPC foi criado pela Sun Microsystems nos anos 1980, e é muito usado no Unix como interface para o serviço NFS [Network File System, sistema de arquivo em rede]. Outros serviços que também usam RPC são NIS [Network Information System, sistema de informação de rede] e rsh [remote shell, que é um comando Unix para atuação remota];

- RMI [Remote Method Invocation, Java]: é uma API [Application Program Interface, interface de aplicação de programa] do Java que provê funcionalidade similar ao método padrão RPC do Unix.