**Comunicação Entre Processos**

Nos sistemas operacionais tais como Windows e Linux muitos dos seus recursos operam sobre a forma de processos, e processos são instâncias de programas em execução, ou seja, um programa gera um processo, e também existem processos que rodam em segundo plano disponibilizando algum tipo de serviço, a seguir alguns exemplos:

* alsa – usado para inicializar o Alsa (Advanced Linux Sound Architecture).
* crond – usado para executar programas de usuários em tempo periódicos pré-determinados.
* dhcpd – usado para inicializar o servidor de Host Dinâmico dchp.
* httpd – usado para inicializar o servidor Web Apache.
* mysql – usado para inicializar o servidor de Banco de Dados MySQL.
* sshd – é um daemon que serve conexões ssh.
* syslog – usado para fornecer um arquivo unificado de log do sistema.

Quando o sistema operacional lança um processo, esse processo tem uma memória reservada para poder realizar a sua execução, ou seja, sua memória fica protegida somente para o seu uso, e para se comunicar com outros processos é necessário algum mecanismo que permita essa comunicação ou alguma outra forma que os dados sejam compartilhados entre os processos, é nesse momento que entra em cena o IPC.

O que é IPC?

O IPC é um mecanismo que permite que dois ou mais processos realizem a troca de dados entre si. Isso é extremamente útil devido ao fato que um processo possui sua própria região de memória onde outros processos não tem a permissão de acessar aquele espaço de memória.

Existem dois tipos de IPC o pull-based e o push-based:

Pull-based

Requer um meio, como um armazenamento para compartilhar os dados, isso porque os processos que querem ler esses dados precisam dar um pull desses dados. Neste caso os dados serão lidos através de um elemento intermediário, como um repositório de dados, onde um processo pode escrever dados nesse repositório enquanto outro processo realiza a leitura desses dados. Para esse grupo de IPC é fortemente recomendado realizar a sincronia para que não haja concorrência, os IPC referente a esse grupo são:

* FIFO
* Shared File
* Shared Memory
* Message Queue
* MMAP
* Semaphore

[Diagrama

Descrição gerada automaticamente](https://github.com/NakedSolidSnake/Raspberry_IPC_Intro/blob/main/img/pull-based.png)

Fonte: <https://embarcados.com.br/comunicacao-entre-processos/>

Push-based

Neste caso os dados serão entregues diretamente ao processo que está realizando a leitura, os IPC referente a esse grupo são:

* Signal
* Socket
* PIPE
* DBUS

[Diagrama, Esquemático

Descrição gerada automaticamente](https://github.com/NakedSolidSnake/Raspberry_IPC_Intro/blob/main/img/push-based.png)

Fonte: <https://embarcados.com.br/comunicacao-entre-processos/>

**Semáforos**

Semáforos em sistemas operacionais são estruturas de dados usadas para controlar o acesso a recursos compartilhados entre processos ou threads, evitando condições de corrida e sincronizando sua execução. Existem dois tipos principais de semáforos:

Semáforo Binário (Mutex):

* + Funciona como um cadeado que pode estar aberto (1) ou fechado (0).
  + Operações principais:
    - ‘wait’ (ou ‘P’): Decrementa o semáforo. Se estiver fechado (0), bloqueia o processo ou thread até que seja aberto.
    - ‘signal’ (ou ‘V’): Incrementa o semáforo, liberando o recurso e permitindo que outros processos ou threads possam acessá-lo.

Semáforo de Contagem:

* + Pode ter qualquer valor inteiro não negativo.
  + Operações principais:
    - ‘wait’ (ou ‘P’): Decrementa o semáforo. Se o valor for zero, bloqueia o processo ou thread até que seja incrementado.
    - ‘signal’ (ou ‘V’): Incrementa o semáforo.

Uso de Semáforos:

* Exclusão Mútua: Proteção de seções críticas do código para garantir que apenas um processo ou thread acesse um recurso por vez.
* Sincronização: Coordenação da ordem de execução entre processos ou threads.
* Prevenção de Deadlock: Evitam a ocorrência de situações onde processos ou threads ficam bloqueados indefinidamente aguardando uns aos outros.

Vantagens:

* Flexibilidade: Podem ser usados para exclusão mútua e sincronização.
* Eficiência: Operações de semáforos são geralmente rápidas e eficientes.

Desvantagens:

* Complexidade: Pode ser complexo garantir o uso correto dos semáforos para evitar deadlocks e condições de corrida.
* Espera Ocupada: Em algumas implementações, os processos ou threads podem gastar tempo de CPU aguardando, o que pode não ser eficiente.

Semáforos são uma ferramenta poderosa para a programação concorrente em sistemas operacionais, permitindo o controle seguro e eficiente de recursos compartilhados entre processos ou threads.

Referências

<https://embarcados.com.br/comunicacao-entre-processos/>

<https://profandreluisbelini.wordpress.com/wp-content/uploads/2012/03/aula-08-cap-07-sincronizac3a7c3a3o-e-comunicac3a7c3a3o-entre-processos.pdf>