Principios de Sistemas Operativos

Comunicación interprocesos (IPC)

Armando Arce, arce@itcr.ac.cr

Tecnológico de Costa Rica

Comunicación interprocesos (IPC)

Los procesos que se ejecutan concurrentemente pueden ser procesos independientes o procesos cooperativos.

- Un proceso es independiente si no puede afectar o verse afectado por los restantes procesos que se ejecutan en el sistema.
- Cualquier proceso que no comparte datos con ninguno otro proceso es un proceso independiente.
- Un proceso es cooperativo si puede afectar o verse afectado por los demás procesos que se ejecutan en el sistema.

Comunicación interprocesos (IPC)

Hay varias razones para proporcionar un entorno que permita la cooperación entre procesos:

- Compartir información: Dado que varios usuarios pueden estar interesados en la misma información, se debe proporcionar un entorno que permita el acceso concurrente a ella.
- Acelerar cálculos: Si se desea que determinada tarea se ejecute rápidamente, se debe dividir en sus tareas, ejecutando cada una de ellas en paralelo con las demás.
- Modularidad: En ocasiones se desea construir el sistema en forma modular, dividiendo las funciones del sistema en diferentes procesos o hilos.
- Conveniencia: Aveces un único usuario inclusive quiere trabajar en múltiples tareas al mismo tiempo.

Comunicación interprocesos (IPC)

La cooperación entre procesos requieren mecanismos de comunicación interprocesos (IPC, por sus siglas en inglés) que permitan intercambiar datos e información.

- Existen dos modelos fundamentales de comunicación interprocesos: memoria compartida y paso de mensajes.
- En el modelo de memoria compartida, se establece una región de memoria para que sea compartida por los procesos cooperativos.
- De este modo, los procesos puede intercambiar información leyendo y escribiendo datos en la zona compartida.
- En el modelo de paso de mensajes, la comunicación tiene lugar mediante el intercambio de mensajes entre los procesos cooperativos.

Comunicación interprocesos (IPC)

Los dos modelos son bastante comunes en los distintos sistemas operativos y muchos sistemas implementan ambos.

- El paso de mensajes resulta útil para intercambiar pequeñas cantidades de datos, ya que no existe la necesidad de evitar conflictos.
- El paso de mensajes también es más fácil de implementar que el modelo de memoria compartida como mecanismo de comunicación entre computadores.

Comunicación interprocesos (IPC)

La memoria compartida permite una velocidad máxima y una mejor comunicación, ya que puede realizarse a velocidades de memoria cuando se hace en una misma computadora.

- La memoria compartida es más rápida que el paso de mensajes, ya que este último método se implementa normalmente usando llamadas al sistema y, por tanto, requiere que intervenga el kernel, lo que consume más tiempo.
- Por el contrario, en sistemas de memoria compartida, las llamadas al sistema sólo son necesarios para establecer las zonas de memoria compartida.
- Una vez establecida la memoria compartida, todos los accesos se tratan como accesos a memoria rutinarios y no se precisan ayuda del kernel.

Comunicación interprocesos (IPC)

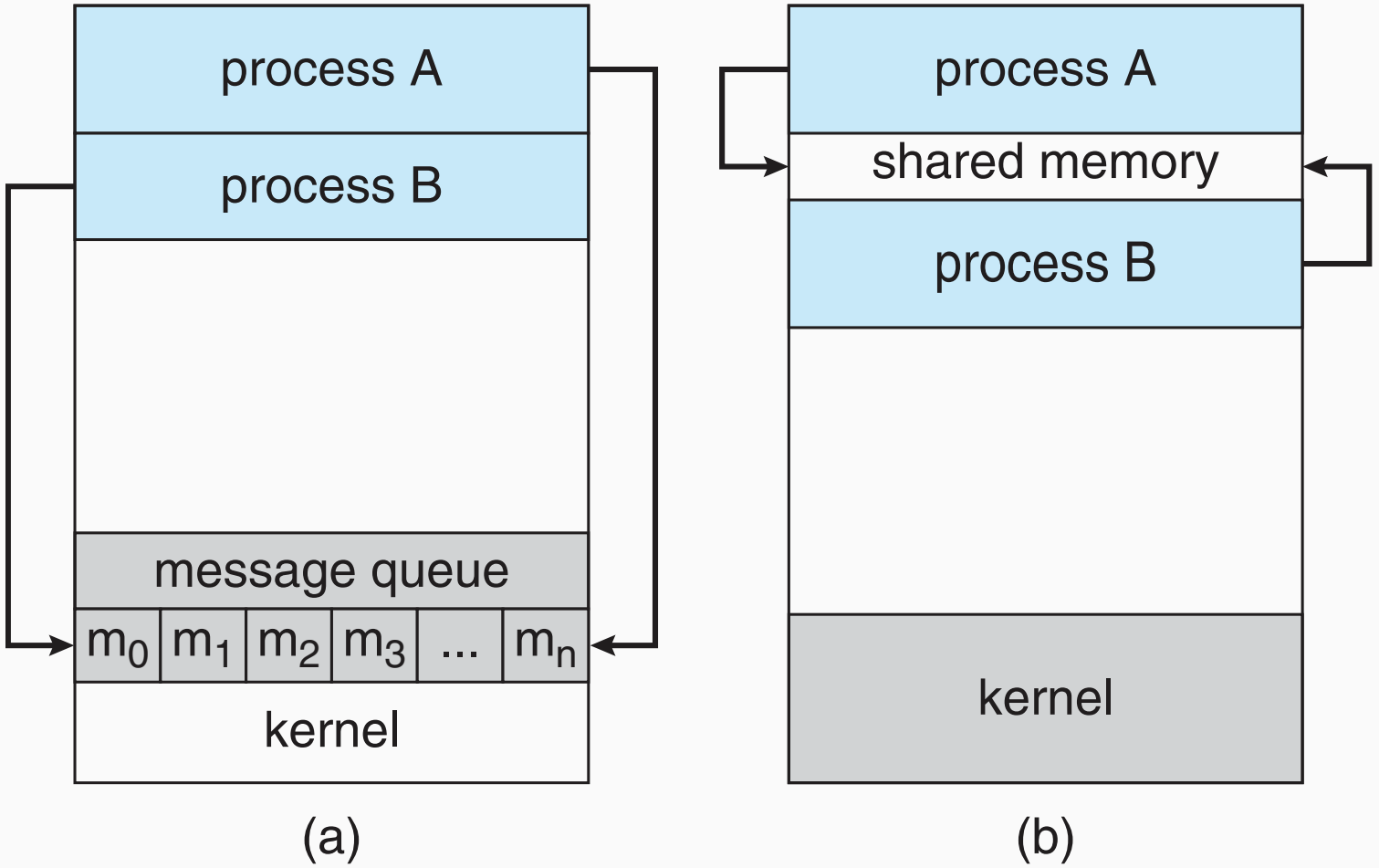

Figure 3.12 Communications models. (a) Message passing. (b) Shared memory.

Memoria compartida

Generalmente, la memoria compartida reside en el espacio de direcciones del proceso que la crea.

- Otros procesos que quieran utilizar la memoria compartida deben conectarse a este espacio de direcciones.
- Es necesario que los procesos acuerden eliminar la protección de memoria sobre este segmento, pues el sistema operativo tratará de evitar que un proceso acceda a la memoria de otro proceso.
- El formato de los datos y su ubicación están determinados por los procesos, y no se encuentran bajo el control del sistema operativo.
- Los procesos también son responsables de verificar que no escriben en la misma posición de memoria simultáneamente.

Memoria compartida

Este mecanismo es altamente eficiente para el intercambio de datos entre procesos de la misma máquina y permite realizar grandes transferencias de datos.

- Sin embargo, un inconveniente que presenta este mecanismo es que no todos los sistemas cuentan con la capacidad de eliminar el esquema de protección de memoria sobre un proceso particular.
- Adicionalmente, la memoria compartida requiere una cuidadosa sincronización de procesos.
- Este esquema requiere que el programador de la aplicación escriba explícitamente el código para acceder y manipular la memoria compartida.

Paso de mensajes

El paso de mensajes proporciona un mecanismo que permite a los procesos comunicarse y sincronizar sus acciones sin compartir el mismo espacio de direcciones.

Paso de mensajes

El paso de mensajes proporciona al menos dos operaciones: envío de mensajes (*send*) y recepción de mensajes (*receive*).

- Los mensajes enviados pueden tener un tamaño fijo o variable. Si sólo se pueden enviar mensajes de tamaño fijo, la implementación en el nivel de sistema es directa.
- Sin embargo, esta restricción hace que la tarea de programación sea más complicada.
- Por el contrario, los mensajes de tamaño variable requieren una implementación más compleja en el nivel del sistema, pero la tarea de programación es más sencilla.

Paso de mensajes

Para que dos procesos ubicados en máquinas independientes se puedan comunicar, es necesario que exista un enlace de comunicación entre ellas. Existen varios métodos para implementar lógicamente un enlace las operaciones de envío y recepción:

- comunicación directa o indirecta
- comunicación síncrona o asíncrona
- almacenamiento en búfer explícito o automático

Nombrado

Los procesos que se van a comunicar deben disponer de un modo de referenciarse entre sí. Pueden usar comunicación directa o indirecta. En el caso de comunicación directa, cada proceso que desea establecer una comunicación debe nombrar de forma explícita al receptor o transmisor de la comunicación. En este esquema, las primitivas *send()* y *receive()* se definen del siguiente modo:

- send(P,mensaje): Envía un mensaje al proceso P
- receive(Q,mensaje): Recibe un mensaje del proceso Q

Nombrado

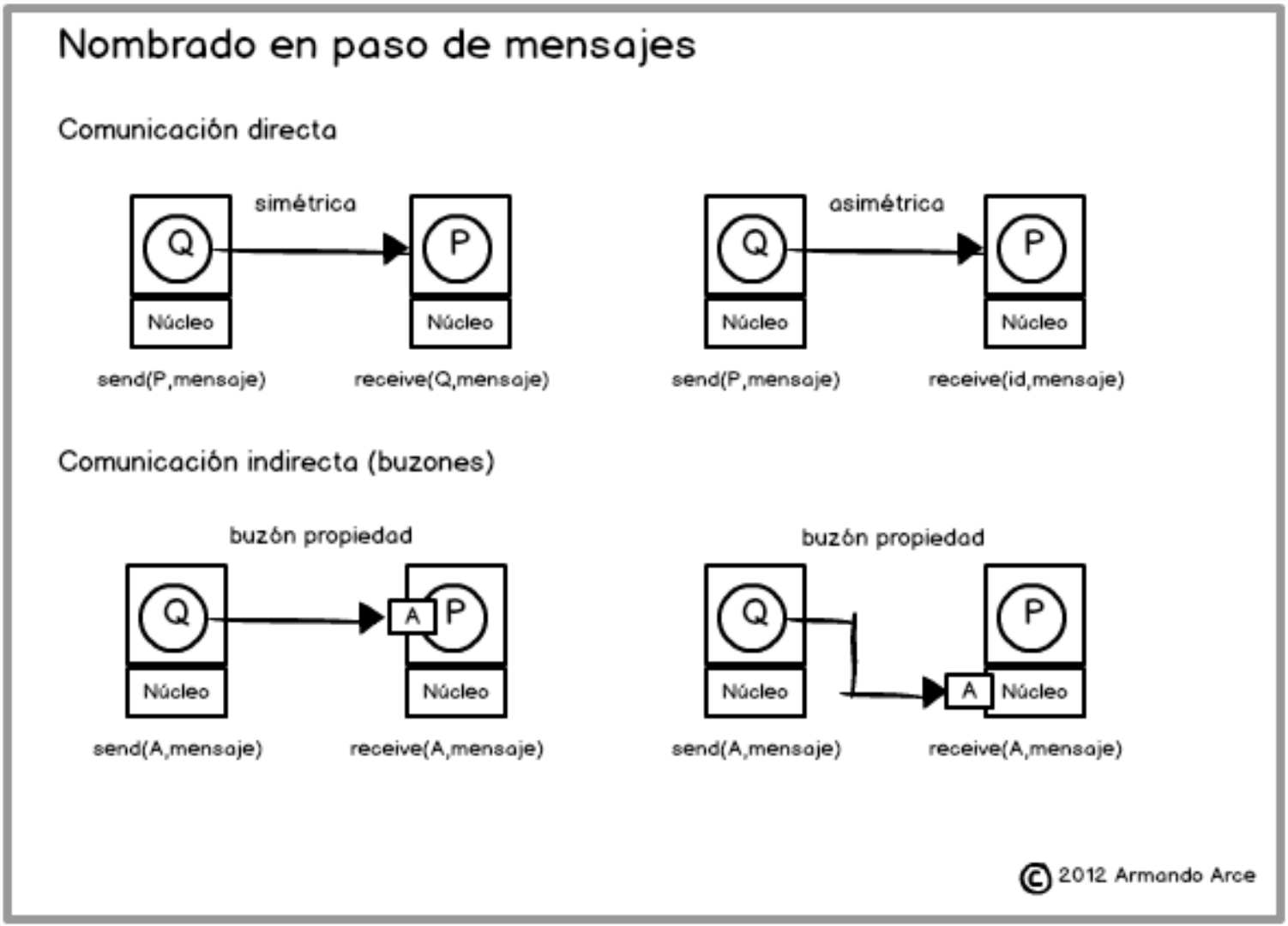

Figure 2:

Nombrado

Un enlace de comunicaciones, según este esquema, tiene las siguientes propiedades:

- Los enlaces se establecen de forma automática entre cada par de procesos que quieran comunicarse. Los procesos sólo tiene que conocer la identidad del otro para comunicarse.
- Cada enlace se asocia con exactamente dos procesos.
- Entre cada par de procesos existe exactamente un enlace.

Nombrado

Este esquema presenta simetría en lo que se refiere al direccionamiento, es decir, tanto el proceso transmisor como el proceso receptor debe nombrar al otro para comunicarse.

- Existe una variante de este esquema que emplea asimetría en el direccionamiento.
- En este caso, solo el transmisor nombra al receptor; el receptor no tiene que nombrar al transmisor. En este esquema, las primitivas *send()* y *receive()* se definen del siguiente modo:
- send(P,mensaje): Envía un mensaje al proceso P
- receive(id,mensaje): Recibe un mensaje de cualquier proceso; a la variable *id* se le asigna el nombre del proceso con el que se ha llevado acabo la comunicación.

Nombrado

La desventaja de estos dos esquemas (simétrico y asimétrico) en la limitada modularidad de las definiciones de procesos resultantes.

- Cambiar el identificador de un proceso puede requerir que se modifiquen todas las restantes definiciones de procesos.
- Deben localizarse todas las referencias al identificador antiguo, para poder sustituirlas por un nuevo identificador.
- En general, cualquier técnica de precodificación, en la que los identificadores deban establecerse explícitamente, es menos deseable que las técnicas basadas en la indirección.

Nombrado

Con el modelo de comunicación indirecta, los mensajes se envían y reciben en buzones de correo o puertos.

- Un buzón de correo puede verse de forma abstracta como un objeto en el que los procesos pueden colocar mensajes y del que pueden eliminar mensajes.
- Cada buzón de correo tiene asociada una identificación unívoca. En este esquema, un proceso puede comunicarse con otros procesos a través de una serie de buzones de correo diferentes.
- Sin embargo, dos procesos sólo se puede comunicar si tienen un buzón de correo compartido.

Nombrado

Las primitivas *send()* y *receive()* se definen del siguiente modo:

- send(A,mensaje): Envía un mensaje al buzón de correo A
- receive(A,mensaje): Recibe un mensaje del buzón de correo A

En este esquema, un enlace de comunicaciones tiene las siguientes propiedades:

- Puede establecerse un enlace entre un par de procesos solo si ambos tienen un buzón de correo compartido.
- Un enlace puede asociarse con más de dos procesos.
- Entre cada par de procesos, puede haber una serie de enlaces diferentes, correspondiendo cada enlace a un buzón de correo.

Nombrado

Un buzón de correo puede ser propiedad de un proceso o del sistema operativo. Si es propiedad de un proceso, es decir, si el buzón de correo forma parte del espacio direcciones del proceso, entonces se puede diferenciar entre el propietario (aquel que sólo recibe mensajes a través este buzón) y el usuario (aquel que sólo puede enviar mensajes a dicho buzón de correo).

- Puesto que cada buzón de correo tiene un único propietario, no puede haber confusión acerca de quién recibirá un mensaje enviado a ese buzón de correo. Cuando un proceso que posee un buzón de correo termina, dicho buzón desaparece. A cualquier proceso que con posterioridad envíe un mensaje a ese buzón debe notificársele que dicho buzón ya no existe.

Nombrado

Por el contrario, un buzón de correo que sea propiedad del sistema operativo tiene existencia propia: es independiente y no está asociado a ningún proceso concreto. El sistema operativo debe proporcionar un mecanismo que permita a un proceso hacer lo siguiente:

- crear un buzón de correo nuevo.
- enviar y recibir mensajes a través del buzón de correo.
- eliminar un buzón de correo.

Nombrado

Por omisión, el proceso que crea un buzón de correo nuevo es el propietario del mismo.

- Inicialmente, el propietario es el único proceso que puede recibir mensajes a través de este buzón.
- Sin embargo, la propiedad y el privilegio de recepción se pueden pasar a otros procesos mediante las apropiadas llamadas al sistema.
- Por supuesto, esta medida puede dar como resultado que existen múltiples receptores para cada buzón de correo.

Sincronización

La comunicación entre procesos tiene lugar a través de llamadas a las primitivas *send()* y *receive()*. Existen diferentes opciones de diseño para implementar cada primitiva. El paso de mensajes puede ser con bloqueo o sin bloqueo, mecanismos también conocidos como síncrono y asíncrono.

- Envío con bloqueo: el proceso que envía se bloquea hasta que proceso receptor o el buzón de de correo reciben el mensaje.
- Envío sin bloqueo (IPC_NOWAIT): el proceso transmisor envía el mensaje y continúa operando.

Sincronización

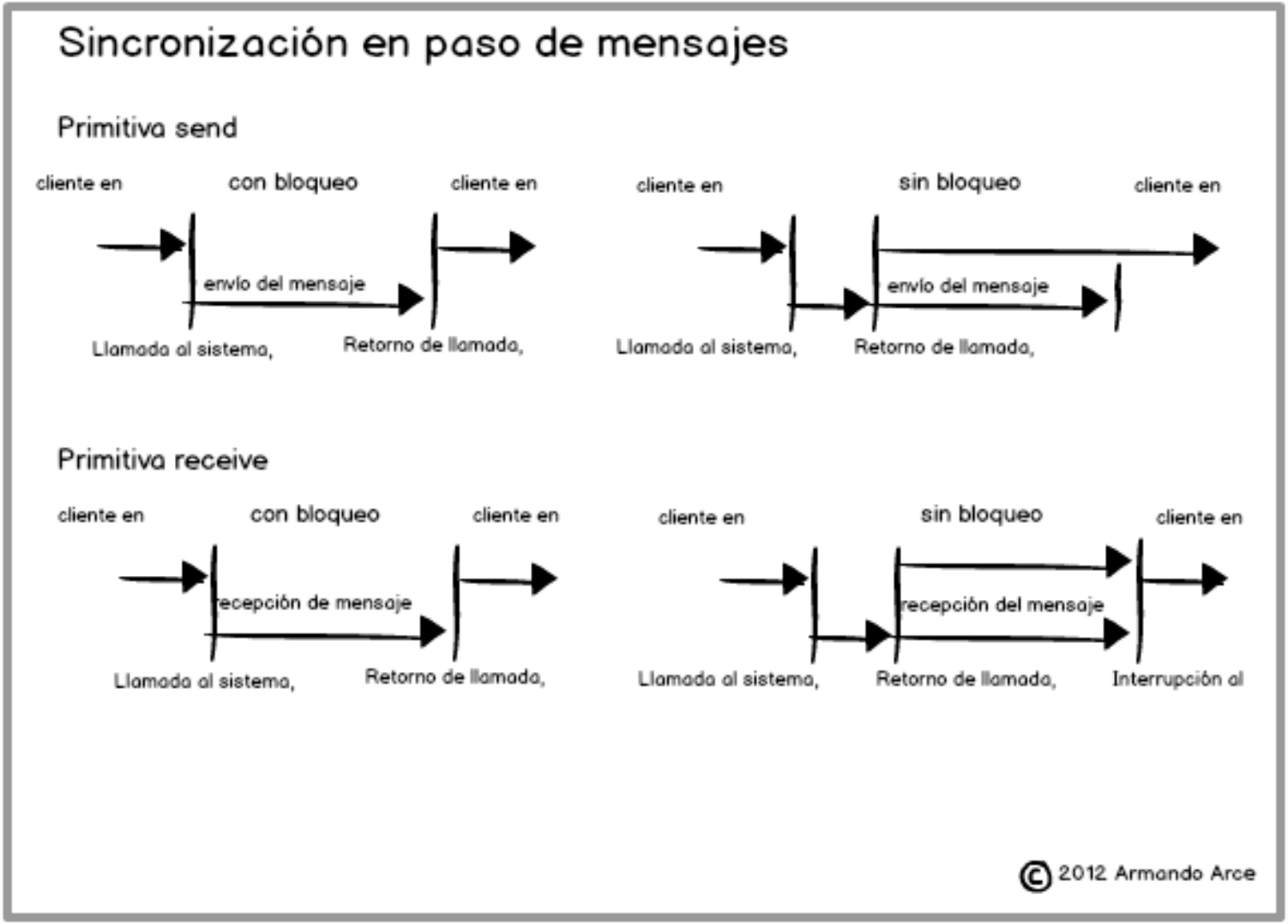

Figure 3:

Sincronización

- Recepción con bloqueo: el receptor se bloquea hasta que hay un mensaje disponible.
- Recepción sin bloqueo: el receptor extrae un mensaje válido un mensaje nulo.

Son posibles combinaciones diferentes de las operaciones *send* y *receive*. Cuando ambas operaciones se realizan con bloqueo, se tiene lo que se denomina *rendezvous* entre el transmisor y receptor.

Almacenamiento en buffer

Sea la comunicación directa o indirecta, los mensajes intercambiados por los procesos que se están comunicando residen en una cola temporal. Básicamente, tales colas se puede implementar de tres maneras:

Almacenamiento en buffer

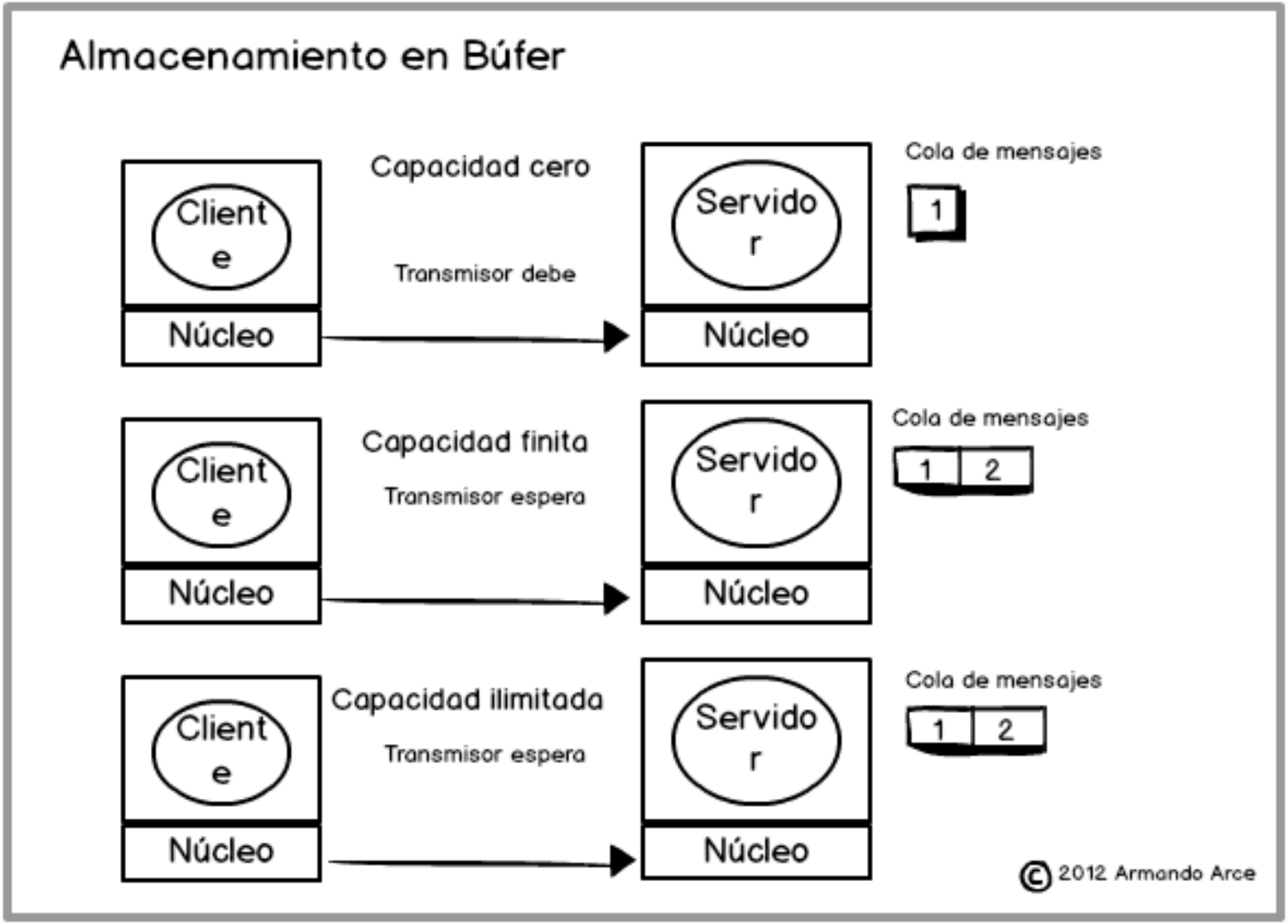

Figure 4:

Almacenamiento en buffer

Sea la comunicación directa o indirecta, los mensajes intercambiados por los procesos que se están comunicando residen en una cola temporal. Básicamente, tales colas se puede implementar de tres maneras:

- Capacidad cero: la cola tiene una longitud máxima de cero; por tanto, no puede haber ningún mensaje esperando en el enlace. En este caso, el transmisor debe bloquearse hasta que el receptor reciba el mensaje. En ocasiones, se dice que el caso de capacidad cero es un sistema de mensajes sin almacenamiento en buffer; los otros casos se conocen como sistemas con almacenamiento en buffer automático.

Almacenamiento en buffer

- Capacidad limitada: la cola tiene una longitud finita N; por tanto, puede haber en ella N mensajes como máximo. Si la cola no está llena cuando se envió el mensaje, el mensaje se introduce en la cola (se copia el mensaje o se almacena un puntero al mismo), y el transmisor puede continuar la ejecución sin esperar. Sin embargo, la capacidad del enlace es finita. Si el enlace está lleno, el transmisor debe bloquearse hasta que haya espacio disponible en la cola.
- Capacidad ilimitada: la longitud de la cola es potencialmente infinita; por tanto, puede haber cualquier cantidad de mensajes esperando en ella. El transmisor nunca se bloquea.

Referencia bibliográfica

A.SILBERSCHATZ, P. GALVIN, y G. GAGNE, Operating Systems Concepts, Cap. 3, 9a Edición, John Wiley, 2013.