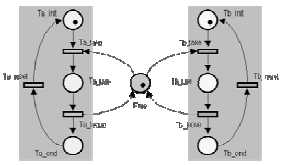
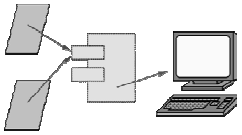
Material Adicional (SOSD – Mod 4)
Prof. Javier Echaiz

Concurrencia
Exclusión mutua y sincronización



Slides del Prof. Samuel Oporto Díaz

CONCURRENCIA



2

Concurrencia

- La concurrencia es la simultaneidad de hechos.
- Un programa concurrente es aquel en el que ciertas unidades de ejecución internamente secuenciales (procesos o threads), se ejecutan paralela o simultáneamente.
- Incluye los siguientes aspectos:
  - comunicación entre procesos.
  - compartición y competencia por los recursos.
  - sincronización de la ejecución de varios procesos.
  - asignación del tiempo de procesador a los procesos.
- Surgen en entornos con un solo procesador, con multiprocesadores y proceso distribuido.

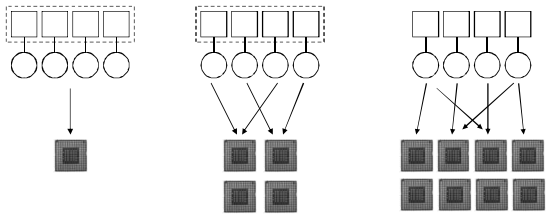
3

Concurrencia

- Un programa concurrente está formado por una colección de procesos secuenciales autónomos que se ejecutan (aparentemente) en paralelo.
- Tres formas de ejecutar procesos concurrentes:
  1. Los procesos multiplexan sus ejecuciones sobre un único procesador (multiprogramación).
  2. Los procesos multiplexan sus ejecuciones sobre un sistema multiprocesador de memoria compartida (multiproceso).
  3. Los procesos multiplexan sus ejecuciones en varios procesadores que no comparten memoria (procesamiento distribuido).
- El término **concurrencia** indica paralelismo potencial.

4

Concurrencia

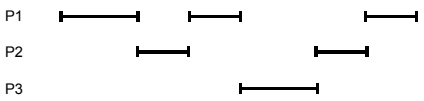


multiprogramación | multiproceso | Procesamiento distribuido

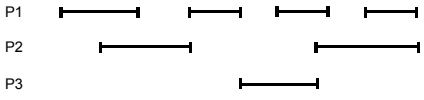
5

Concurrencia

- En un sistema multiprogramado (1 µP), los procesos se intercalan, para dar la apariencia de simultaneidad.



- En un sistema con varios procesadores (n µP), los procesos se superponen.



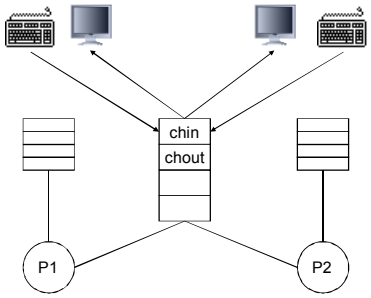
6

Dificultades con la Concurrencia

- Los problemas generados por la multiprogramación surgen por que no es posible predecir la **velocidad relativa de los procesos**:
- La actividad de un proceso depende de la actividad de los otros procesos y de la forma cómo el SO trata las interrupciones y las políticas de planificación.
- Dificultades:
  - Compartir recursos globales (variable global)
  - Manejar la asignación de recursos
  - Detectar un error es difícil, dado que es difícil reproducir la situación..

7

Un ejemplo simple



8

un ejemplo simple

```
void echo()
{
  char_in = getchar();
  char_out = char_in;
  putchar(char_out);
}
```

Se asume que esta función es usada por muchos procesos en el SO y dado que existe una sola pantalla y un solo teclado, todos los programas interactúan con este programa. Por lo que lo que es recomendable tenerlo siempre en memoria

9

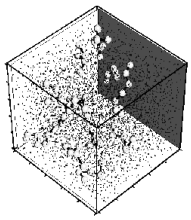
un ejemplo simple

```
Process P1              Process P2
.                       .
chin  = getchar();            .          chin = a
.                       chin  = getchar(); chin = b
chout = chin;           chout = chin;
putchar(chout);         .
.                       putchar(chout);
.                       .
```

Conclusiones:
Proteger las variables compartidas.
¿cómo?
Evitando que otro proceso ejecute a `echo` hasta que el proceso actual termine.
Evitando que el proceso actual sea interrumpido hasta que termine

10

INTERACCION ENTRE PROCESOS



11

Tipos de proceso

Independientes	Cooperantes
Su estado no es compartido	Su estado es compartido
Son deterministas	Su funcionamiento no es determinista
Son reproducibles	Su funcionamiento puede ser irreproducible
Ejemplo: Un programa que calcula 1000 cifras decimales de π	Ejemplo: un proceso que escribe en el terminal la cadena “abc” y en otro la cadena “cba”

12

Interacción entre procesos

- La concurrencia se presenta cuando existen:
  - multiprogramación de aplicaciones independientes.
  - aplicaciones con varios procesos.
  - el uso de las estructuras de datos del SO.
  - el computador se dispone de varios procesadores.

(multiprogramación).
(multiproceso).
(procesamiento distribuido).

13

Interacción entre procesos

- Existen tres maneras en que los procesos interactúan:

① **competencia por recursos escasos**.
  - Los procesos no conocen a los demás
  - •Interbloqueo •Inanición •Exclusión mutua

② **cooperación por compartición**
  - Los procesos conocen indirectamente a los demás
  - •Interbloqueo •Inanición •Exclusión mutua •Coherencia de datos

③ **cooperación por comunicación**
  - Los proceso conocen directamente a los otros.
  - •Interbloqueo •Inanición

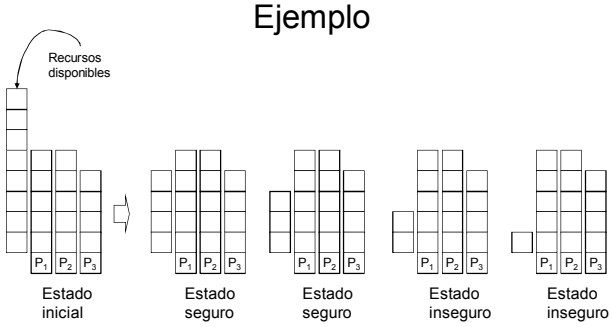
14

① Competencia por recursos escasos

- Los procesos compiten por recursos escasos.
- Si dos procesos compiten por un recurso X
- el SO asigna el recurso a uno y el otro espera
- Problemas.
  - Interbloqueo (estancamiento)
  - Inanición
  - Exclusión mutua

15

Ejemplo



16

② La cooperación por compartición

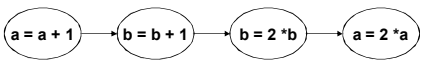
- Los procesos no se conocen.
- Los procesos cooperan para evitar inconsistencias
- Recursos compartidos:
  - variables compartidas
  - ficheros
  - bases de datos
- Problema
  - Confianza
- Solución
  - Exclusión mutua
  - Sección crítica
  - La escritura debe ser mutuamente excluyente.
  - La lectura puede ser concurrente

17

Ejemplo

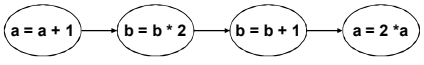
Ejecución consistente

P1: a = a + 1
    b = b + 1
P2: b = 2 * b
    a = 2 *a



Ejecución concurrente

P1: a = a + 1
P2: b = b * 2
P1: b = b + 1
P2: a = 2 *a



18

③La cooperación para la comunicación

- Los procesos se conocen explícitamente.
- La comunicación se da con mensajes: **`send`**, **`receive`**.
- No se comparten recursos no exclusión mutua.
- Problemas.
  - Interbloqueo. Cada proceso espera un mensaje del otro proceso
  - Inanición. Dos procesan envían sus mensaje mutuamente mientras que esperan un mensaje del otro proceso

19

PROBLEMAS POTENCIALES A RESOLVER

20

1. Exclusión mutua

- Es un mecanismo empleado en el diseño de los sistemas operativos para evitar los problemas de competencia por recursos, se basa en definir una zona o región crítica la cual está marcada por las instrucciones que hacen uso del recurso en competencia (recurso crítico).

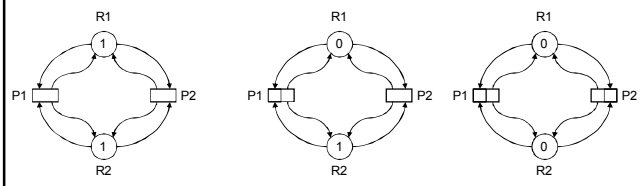
21

2. Interbloqueo (deadlock)

condiciones : P1 requiere R1 y R2
P2 requiere R1 y R2

acciones:

1. P1 obtiene R1
2. P2 obtiene R2
3. P1 Y P2 están bloqueados esperando cada uno al otro



22

3. Inanición

- Los procesos siempre están bloqueados y nunca acceden a los recursos que necesitan

Sean 3 procesos.

- P1 solicita recurso.
- P2 y P3 solicitan recursos
- P1 suelta el recurso
- Se asigna el recurso a P2
- P1 solicita el recurso
- P2 suelta el recurso
- Se asigna el recurso a P1

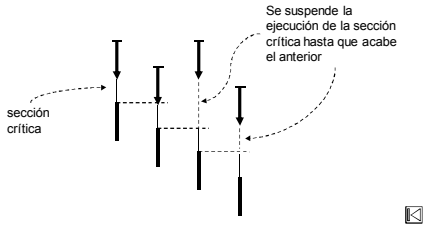
23

SECCION CRÍTICA

24

Sección Crítica

- Es la parte del programa que accede a un recurso compartido
- Solo un programa puede acceder a su sección crítica en un momento dado
- NO se puede confiar simplemente en el SO para hacer cumplir esta restricción



Ejemplo: solo un proceso en cada momento estará enviando comandos a la impresora

25

Requerimientos para exclusión mutua

1. Exclusión mutua. Sólo un procese accede a la vez a su SC
2. Un proceso suspendido en su SC no debe estorbar a otros
3. No inanición, no interbloqueo. Si un proceso solicita acceso a su SC no debe demorar su atención.
4. Si ningún proceso está en su SC, no se puede evitar que otro proceso entre a su SC.
5. No suponer la velocidad relativa de los procesos.
6. Se permanece en la SC por un tiempo finito.

26

SOLUCIONES DE SOFTWARE

27

Intento inicial

PROCESO_UNO	PROCESO_DOS	PROCESO_PRINCIPAL

procedure proceso_uno
begin
while TRUE do
begin
obtener_entrada
entrar_exclusion_mutua
count = count + 1
salir_exclusion_mutua
procesar_entrada
end
end;
	procedure proceso_dos
	begin
	while TRUE do
	begin
	obtener_entrada
	entrar_exclusion_mutua
	count = count + 1
	salir_exclusion_mutua
	procesar_entrada
	end
	end;
		begin
		count = 0;
		parbegin
		proceso_uno;
		proceso_dos;
		parend;
		end.

- parbegin, parend hacen que proceso_uno y proceso_dos operen concurrentemente.
- estos procesos se encuentran en un ciclo infinito de entrada/salida a sus secciones críticas.

28

Primer intento

PROCESO_UNO	PROCESO_DOS	PROCESO_PRINCIPAL

procedure proceso_uno
begin
while TRUE do
begin
while turno=2 do espera;
seccion_critica_uno;
turno = 2;
otras_tareas_dos;
end
end;
	procedure proceso_dos
	begin
	while TRUE do
	begin
	while turno=1 do espera;
	seccion_critica_dos;
	turno = 1;
	proceso_dos_proceso;
	end
	end;
		begin
		turno=1;
		parbegin
		proceso_uno;
		proceso_dos;
		parend;
		end.

- Los procesos entran a sus SC alternativamente, y el proceso 1 entra primero.
- Si uno de los procesos falla entonces el otro proceso quedará bloqueado indefinidamente.
- *problemas:*
  - supuesto de velocidad, prioridad.
  - sincronización en tándem. (por dos lados).

29

Segundo intento

PROCESO_UNO	PROCESO_DOS	PROCESO_PRINCIPAL

procedure proceso_uno
begin
while TRUE do
begin
while p2entro do espera;
plentro = TRUE;
seccion_critica_uno;
plentro = FALSE
otras_tareas_dos;
end
end;
	procedure proceso_dos
	begin
	while TRUE do
	begin
	while plentro do espera;
	p2entro = TRUE
	seccion_critica_dos;
	p2entro = FALSE
	proceso_dos_proceso;
	end
	end;
		begin
		plentro = FALSE;
		p2entro = FALSE;
		parbegin
		proceso_uno;
		proceso_dos;
		parend;
		end.

- Cada proceso puede ver el estado del otro pero no alterarlo.
- Si un proceso quiere entrar a su SC verifica el estado del otro (**false**) y luego pone su estatus en **true**. Al salir pone **false** en su estado.

problema:
que P1 y P2 intenten entrar al mismo tiempo a su SC.

El momento en que un proceso determina si puede entrar a su sección critica y el momento en que informa que esta en exclusión mutua. puede ser usado por otro proceso para verificar si puede entrar.

```
pp:   plentro = FALSE;
      p2en tro = FALSE;
p1:   while p2entro do;
p2:   while plentro do;
p1:   plentro = TRUE;
p2:   p2entro = TRUE;
```

30

Tercer intento

PROCESO_UNO	PROCESO_DOS	PROCESO_PRINCIPAL

procedure proceso_uno
begin
while TRUE do
begin
p1deseaentrar = TRUE;
while p2deseaentrar do espera;
seccion_critica_uno;
p1deseaentrar = FALSE
otras_tareas_dos;
end
end;
	procedure proceso_dos
	begin
	while TRUE do
	begin
	p2deseaentrar = TRUE
	while p1deseaentrar do espera;
	seccion_critica_dos;
	p2deseaentrar = FALSE
	proceso_dos_proceso;
	end
	end;
		begin
		p1deseaentrar=FALSE;
		p2deseaentrar=FALSE;
		parbegin
		proceso_uno;
		proceso_dos;
		parend;
		end.

Este algoritmo trata de asegurar que mientras un proceso pasa su verificación ningún otro proceso puede pasar su verificación.
Se hace uso de las variables p1deseaentrar y p2deseaentrar.

problema:
cada proceso puede asignar TRUE a su indicador antes de realizar la verificación y por lo tanto ambos procesos se bloquearían uno al otro (interbloque).

Cuarto intento

PROCESO_UNO	PROCESO_DOS	PROCESO_PRINCIPAL

procedure proceso_uno
begin
while TRUE do
begin
p1deseaentrar = TRUE;
while p2deseaentrar do
begin
p1deseaentrar = FALSE
espera de algunos ciclo
p1deseaentrar = TRUE
end
seccion_critica_uno;
p1deseaentrar = FALSE
otras_tareas_uno;
end
end;
	procedure proceso_uno
	begin
	while TRUE do
	begin
	p2deseaentrar = TRUE;
	while p1deseaentrar do
	begin
	p2deseaentrar = FALSE
	espera de algunos ciclo
	p2deseaentrar = TRUE
	end
	seccion_critica_uno;
	p2deseaentrar = FALSE
	otras_tareas_dos;
	end
	end;
		begin
		p1deseaentrar=FALSE;
		p2deseaentrar=FALSE;
		parbegin
		proceso_uno;
		proceso_dos;
		parend;
		end.

El problema de los algoritmos anteriores es que los procesos podrían bloquearse durante la verificación.
Este algoritmo asigna FALSE a su propio indicador durante cierto tiempo, si es que se encuentra durante la verificación.
Esto permite que el otro proceso salga de la espera con TRUE en su indicador.
La exclusión mutua esta garantizada y no puede haber interbloqueo.

Cuarto intento

problema:

- Pero se puede dar el **aplazamiento indefinido,** dado que no se puede hacer ningún supuesto respecto a la velocidad de los procesos.
- Así, si los procesos se ejecutan en tándem (alternativamente),
- se puede dar la siguiente secuencia de instrucciones, en la cual ninguno de los procesos entra a su sección critica:

Ciclo reloj	PROCESO_UNO	PROCESO_DOS
1	p1deseaentrar = TRUE	
2		p2deseaentrar = TRUE
3	while p2deseaentrar do	
4	p1deseaentrar = FALSE	
5	:	
6	p1deseaentrar = TRUE	
7		while p1deseaentrar do
8		p2deseaentrar = FALSE
9		:
10		p2deseaentrar = TRUE
11	while p2deseaentrar do	
12	p1deseaentrar = FALSE	
13	:	
14	p1deseaentrar = TRUE	

Solución correcta

PROCESO_UNO	PROCESO_DOS	PROCESO_PRINCIPAL

procedure proceso_uno
begin
while TRUE do
begin
p1deseaentrar = TRUE;
while p2deseaentrar do
begin
if proceso_activo = DOS then
begin
p1deseaentrar = FALSE;
while proceso_activo=DOS do;
p1deseaentrar = TRUE;
end
end;
seccion_critica_uno;
proceso_activo = DOS
p1deseaentrar = FALSE
otras_tareas_uno;
end
end;
	procedure proceso_uno
	begin
	while TRUE do
	begin
	p2deseaentrar = TRUE;
	while p1deseaentrar do
	begin
	if proceso_activo = UNO then
	begin
	p2deseaentrar = FALSE;
	while proceso_activo=UNO do;
	p2deseaentrar = TRUE;
	end
	end;
	seccion_critica_dos;
	proceso_activo = UNO
	p2deseaentrar = FALSE
	otras_tareas_dos;
	end
	end;
		begin
		p1deseaentrar = FALSE;
		p2deseaentrar = FALSE;
		proceso_activo = UNO;
		parbegin
		proceso_uno;
		proceso_dos;
		parend;
		end.

- Cada proceso consigue una vuelta a la sección crítica
- Si un proceso quiere la sección crítica, pone su bandera y puede tener que esperar a su vuelta. No hay aplazamiento indefinido.
- Pero se puede presentar la posibilidad de que uno de los procesos sea interrumpido luego de salir de su sección critica.

Algoritmo de Peterson

PROCESO_UNO	PROCESO_DOS	PROCESO_PRINCIPAL

procedure proceso_uno
begin
while TRUE do
begin
p1deseaentrar = TRUE;
proceso_activo = DOS
while p2deseaentrar and
proceso_activo = DOS
do espera;
seccion_critica_uno;
p1deseaentrar = FALSE
otras_tareas_uno;
end
end;
	procedure proceso_dos
	begin
	while TRUE do
	begin
	p2deseaentrar = TRUE;
	proceso_activo = UNO
	while p2deseaentrar and
	proceso_activo = UNO
	do espera;
	seccion_critica_uno;
	p2deseaentrar = FALSE
	otras_tareas_dos;
	end
	end;
		begin
		p1deseaentrar = FALSE;
		p2deseaentrar = FALSE;
		proceso_activo = UNO;
		parbegin
		proceso_uno;
		proceso_dos;
		parend;
		end.

SOLUCIONES DE HARDWARE

Inhabilitación de interrupciones

- Un proceso corre hasta que invoca un servicio del SO o hasta que es interrumpido.
- Deshabilitar interrupciones garantiza la exclusión mutua.
- Pero el µP es limitado en su habilidad para intercalar programas

- Multiproceso
  - inhabilitar interrupciones puede no garantizar la exclusión mutua

37

Instrucciones especiales de máquina

- Ejecutó en un ciclo de instrucción sencillo
- No sujeto a la interferencia de otras instrucciones
- Leyendo y escribiendo
- Leyendo y probando

- *Instrucción comparar y fijar (*test y set*)*
- *Instrucción intercambiar*

38

Instrucciones test y set

```
boolean testset (int i) {
          if (i == 0) {
                i = 1;
                return true;
          }
          else {
                return false;
          }
    }
```

39

Instrucción intercambiar

```
void exchange(int register,
     int memory) {
     int temp;
     temp = memory;
     memory = register;
     register = temp;
  }
```

40

Ventajas de soluciones con hardware

- Pertinente a muchos procesos en un procesador sencillo o procesadores múltiples dividiendo la memorial principal
- Es simple y por lo tanto fácil de verificar
- Puede estar acostumbrado a soportar secciones críticas múltiples

41

Ventajas de soluciones con hardware

- Que espera ocupado consuma tiempo de unidad central de proceso
- El hambre es posible cuando un proceso deja una sección crítica y más de un proceso están esperando.
- Estancamiento
  - Si un bajo proceso de prioridad tiene la región crítica y unas necesidades de proceso de prioridad más altas, el proceso de prioridad más alto obtendrá el procesador para esperar a la región crítica

42

SEMAFOROS

43

Semáforos

- La variable especial llamado un semáforo es usado para comunicar
- Si un proceso está esperando para una señal, es suspendido hasta esa señal se envía
- Espera y operaciones de señales no pueden ser interrumpidas
- Forme fila están acostumbrado a tener los procesos esperando en el semáforo

44

Semáforos

- El semáforo es una variable que tiene un valor de entero, número entero
  - El mayo se inicializa a un número no negativo
  - La operación de espera decrementa el valor de semáforo
  - Los incrementos de operación de señales comunican por señales valor

45

Problema del Productor - Consumidor

- Unos o más productores están generando datos y poniendo estos en un parachoques
- un consumidor sencillo está tomando artículos fuera del parachoques un a ratos
- Sólo un productor o consumidor puede acceder el parachoques a cualquier un tiempo

46

Productor

```
producer:
while (true) {
  /* produce item v */
  b[in] = v;
  in++;
}
```

47

Consumidor

```
consumer:
while (true) {
  while (in <= out)
     /*do  nothing */;
  w = b[out];
  out++;
  /* consume item w */
}
```

48

El productor con Buffer Circular

```
producer:
while (true) {
  /* produce item v */
  while ((in + 1) % n == out) /* do nothing
  */;
  b[in] = v;
  in = (in + 1) % n
}
```

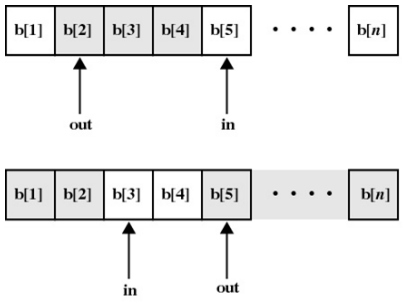
49

El consumidor con Circular Buffer

```
consumer:
while (true) {
  while (in == out)
     /* do nothing */;
  w = b[out];
  out = (out + 1) % n;
  /* consume item w */
}
```

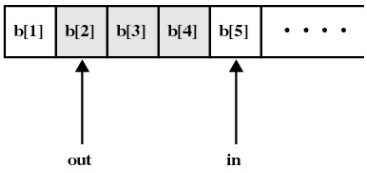
50

Buffer Circular Finito



51

Parachoques infinito

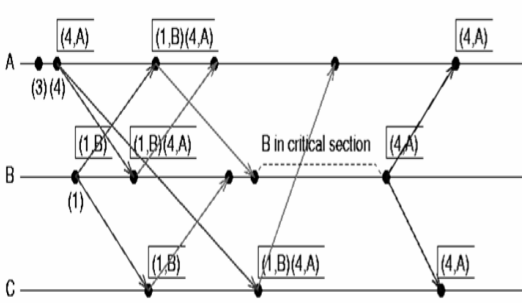


Note: shaded area indicates portion of buffer that is occupied

Figure 5.11 Infinite Buffer for the Producer/Consumer Problem

52

Problema de barbería



53

MONITORES

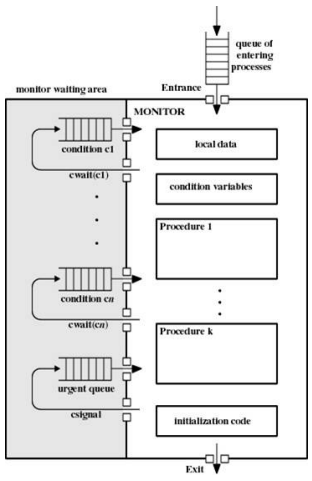
54

Monitores

- El monitor es un módulo de software
- Características principales
  - Las variables de datos locales son accesibles sólo por el monitor
  - Procese entre monitor invocando unos de sus procedimientos
  - Sólo un procese pueda estar ejecutando en el monitor a la vez

55

Estructura de un monitor



PASO DE MENSAJES

57

Paso de mensajes

- Imponga exclusión mutua
- Información de cambio

```
envie ( destino, envie mensajes)
reciba ( fuente, envie mensajes)
```

58

Sincronización

- Remitente y receptor no pueden o no pueden estar bloqueando (esperando para mensaje)
- El entramado envia, bloqueando reciba
  - Ambos remitente y receptor es bloqueado hasta el mensaje se da
  - Llame una cita

59

Sincronización

- El Nonblocking envia, bloqueando reciba
  - El remitente continúa el proceso tales como mensajes de transmisión tan pronto como sea posible
  - El receptor es bloqueado hasta el mensaje pedido llegue
- El Nonblocking envia, nonblocking reciba
  - Ni la fiesta se requiere para esperar

60

Direccionamiento

- Direccionamiento directo
  - envie primitivo incluya un identificador específico del proceso de destino
  - reciba el primitivo pudo saber adelantado que procese un mensaje es esperado
  - reciba el primitivo pudo usar parámetro de fuente para retornar un valor cuando la operación de receive ha sido ejecutado

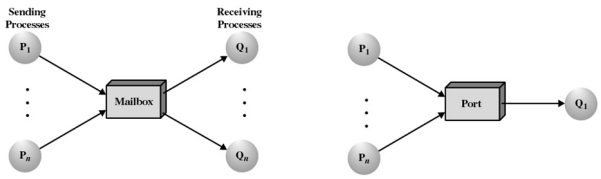
61

Direccionamiento

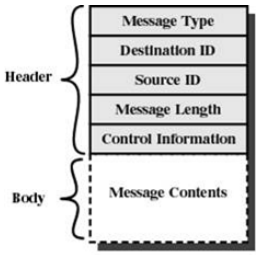
- Direccionamiento indirecto
  - los mensajes son enviados a una estructura de datos dividida consistiendo de las colas
  - las colas son llamados los buzones
  - un proceso envia a un mensaje para el buzón y el otro proceso romper y remover el mensaje del buzón

62

Comunicación de procesos indirectos



Formato de mensaje



64

Problema de los lectores y escritores

- Muchos lectores pueden leer simultáneamente el archivo
- Sólo un escritor a la vez puede escribir al archivo
- Si un escritor está escribiendo al archivo, ningún lector puede leerlo

65