Sistemas Operativos

Colas de mensajes

Curso 2024

Facultad de Ingeniería, UDELAR

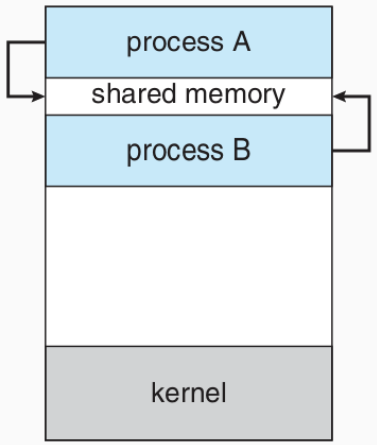
Agenda

1. Modelos para la comunicación entre procesos
2. Comunicación basada en pasaje de mensajes
3. Equivalencia con semáforos
4. Equivalencia con monitores
5. Problemas clásicos de concurrencia

Modelos para la comunicación entre procesos

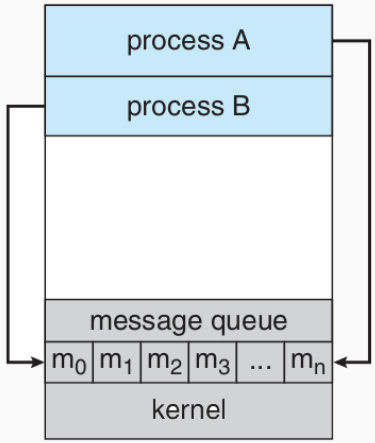
Modelo de memoria compartida

- Un mismo espacio de direcciones es compartido entre dos o más procesos.
  - Acceso a los datos compartidos es por *referencia*.
- SO: provee syscalls para definir el espacio compartido.
- Los procesos acceden directamente a la memoria compartida.
- Sincronización: mecanismos como semáforos y/o monitores.



Modelo de pasaje de mensajes

- Se intercambian mensajes. Sin compartir un mismo espacio de direcciones.
  - El acceso a los datos compartidos es por *valor*.
- SO: provee syscalls para el envío y recepción de mensajes.
  - Garantiza orden en el intercambio de mensajes.
- Sincronización: los mecanismos están incluidos en las primitivas de intercambio de mensajes.



Comunicación basada en pasaje de mensajes

Primitivas para el pasaje de mensajes

Primitivas fundamentales

```
enviar(mensaje)
recibir(mensaje)
```

Implementación de la comunicación

- Directa o indirecta.
- Sincrónica o asincrónica.
- Capacidad finita o infinita.

Comunicación directa e indirecta

Comunicación directa

- El receptor se nombra de forma explícita.
- `enviar(P, mensaje)` — envía un mensaje al proceso P.
- `recibir(Q, mensaje)` — recibe del proceso Q.
- Cardinalidad Nx1: múltiples emisores y un único receptor.

Comunicación indirecta

- Utiliza estructuras independientes llamadas mailboxes (buzones de correo o cola de mensajes).
- `enviar(M, mensaje)` — envía mensaje al mailbox M.
- `recibir(M, mensaje)` — recibe mensaje del mailbox M.
- Cardinalidad NxM: múltiples emisores y múltiples receptores.

Tipo de sincronización

- Primitivas bloqueantes o no bloqueantes.
- Enviar bloqueante: El emisor se bloquea hasta que el mensaje es recibido por el mailbox.
- Enviar no bloqueante: El emisor envía el mensaje al mailbox y continúa su ejecución (puede fallar si está lleno).
- Recibir bloqueante: El receptor se bloquea hasta recibir un mensaje del mailbox.
- Recibir no bloqueante: El receptor recibe un mensaje si hay uno disponible y si no continúa su ejecución.

Capacidad del mailbox

- Capacidad finita (N): Pueden existir a lo más N mensajes en el mailbox. Si el mailbox tiene su capacidad colmada, el emisor (bloqueante) se bloqueará hasta que se libere un lugar.
  - La capacidad puede ser 0.
- Capacidad infinita: La capacidad del mailbox es "infinita". El emisor nunca se bloquea.

Equivalencia con semáforos

Semáforos de conteo con mailbox

Implementación

Enviar no bloqueante y recibir bloqueante.

Mailboxes infinitos.

```
espera: mailbox of NIL;

procedure Init() (n: integer)
   for i in 1..n do
      enviar(espera, NIL);
   end for
end procedure
```

```
procedure P()
   recibir(espera, m);
end procedure

procedure V()
   enviar(espera, NIL);
end procedure
```

Mailbox infinito con semáforos

```
cantidad: semaforo;
mutex: semaforo;
mensajes: cola;

procedure recibir(var m)
   P(cantidad);
   P(mutex);
   m := remove(mensajes);
   V(mutex);
end procedure
```

Es el productor-consumidor.

```
procedure enviar(m)
   P(mutex);
   add(mensajes, m);
   V(mutex);
   V(cantidad);
end procedure

Begin
   INIT(cantidad, 0);
   INIT(mutex, 1);
End
```

Equivalencia con monitores

Monitores con mailbox

```
mutex, cond: mailbox of NIL;
cantCond: mailbox of integer;

procedure function_N()
   recibir(mutex, m);
   ...
   enviar(mutex, NIL);
end procedure

procedure wait()
   recibir(cantCond, cant);
   enviar(cantCond, cant + 1);
   enviar(mutex, NIL)
   recibir(cond, m);
   recibir(mutex, m);
end procedure
```

```
procedure signal()
   recibir(cantCond, cant);
   if cant > 0 then
      enviar(cantCond, cant - 1);
      enviar(cond, NIL)
   else
      enviar(cantCond, cant);
   end if
end procedure

Begin
   enviar(mutex, NIL);
   enviar(cantCond, 0);
End
```

Mailbox infinito con monitores

```
monitor mailbox
mensajes: cola;
espera: condition;

procedure enviar(m)
   add(mensajes, m);
   espera.signal();
end procedure

procedure recibir(var m)
   if size(mensajes) = 0 then
      espera.wait();
   end if
   m := mensajes.remove();
end procedure
```

Problemas clásicos de concurrencia

Configuración de mailboxes

Vamos a suponer que los mailboxes que usamos son:

- Infinitos (enviar no bloqueante)
- Recibir bloqueante

Problema de Alicia y Bernardo

```
patio: mailbox of NIL;

procedure alicia()
   recibir(patio, m);
   Pasear perro;
   enviar(patio, NIL);
   Otras tareas;
end procedure

procedure bernardo()
   recibir(patio, m);
   Pasear perro;
   enviar(patio, NIL);
   Otras tareas;
end procedure
```

```
Begin
   enviar(patio, NIL);
   Cobegin
      alicia();
      bernardo();
   Coend
End
```

Problema de productor-consumidor con buffer finito

```
capacidad: mailbox of NIL;
buffer: mailbox of producto;      ▷ es infinito, trivial si es finito

procedure productor()
   while true do
      p := producir();
      recibir(capacidad, m);
      enviar(buffer, p);
   end while
end procedure

procedure consumidor()
   while true do
      recibir(buffer, p);
      enviar(capacidad, NIL);
      consumir(p);
   end while
end procedure
```

```
Begin
   for i in 1..N do
      enviar(capacidad, NIL);
   end for
   Cobegin
      productor();
      ...     ▷ No se puede usar for
      productor();
      consumidor();
      ...
      consumidor();
   Coend
End
```

Problema de lectores-escritores (sin prioridad)

```
mtxdoc: mailbox of NIL;
cantlect: mailbox of integer;


procedure lector()
   while true do
      recibir(cantlect, cant);
      if cant = 0 then
         recibir(mtxdoc, m);
      end if
      enviar(cantlect, cant + 1);
      leer();
      recibir(cantlect, cant);
      if cant = 1 then
         enviar(mtxdoc, NIL);
      end if
      enviar(cantlect, cant - 1);
   end while
end procedure
```

```
procedure escritor()
   recibir(mtxdoc, m);
   escribir;
   enviar(mtxdoc, NIL);
end procedure

Begin
   enviar(mtxdoc, NIL);
   enviar(cantlect, 0);
   Cobegin
      lector();
      ...
      lector();
      escritor();
      ...
      escritor();
   Coend
End
```

Problema de N filósofos comensales

```
mesa: mailbox of NIL;
tenedor: array[1..N] of mailbox of NIL;
```

```
procedure filosofo(i: integer)
   pensar;
   recibir(mesa, m);
   recibir(tenedor[i], m);
   recibir(tenedor[i+1 mod N], m);
   comer;
   enviar(tenedor[i], NIL);
   enviar(tenedor[i+1 mod N], NIL);
   enviar(mesa, NIL);
end procedure
```

```
Begin
   for i in 1..N do
      enviar(tenedor[i], NIL);
   end for
   for i in 1..N-1 do
      enviar(mesa, NIL);
   end for
   Cobegin
      filosofo(1);
      ...
      filosofo(N);
   Coend
End
```