Programación Sobre Redes

T6: Comunicación entre Procesos I

Nicolás Mastropasqua

May 14, 2020

Instituto Industrial Luis A. Huergo

Contenidos

1. Repaso

2. Comunicación entre procesos

Repaso

Previamente...

- Procesos
  - Estados de un proceso
  - Representación de procesos: PCBs
  - Context Switch
- Manejo de Procesos en C

Menú del día

- ¿Pueden dos procesos hablar entre ellos?
- ¿Cuáles son los **modelos** de comunicación?

Comunicación entre procesos

Las dos alternativas

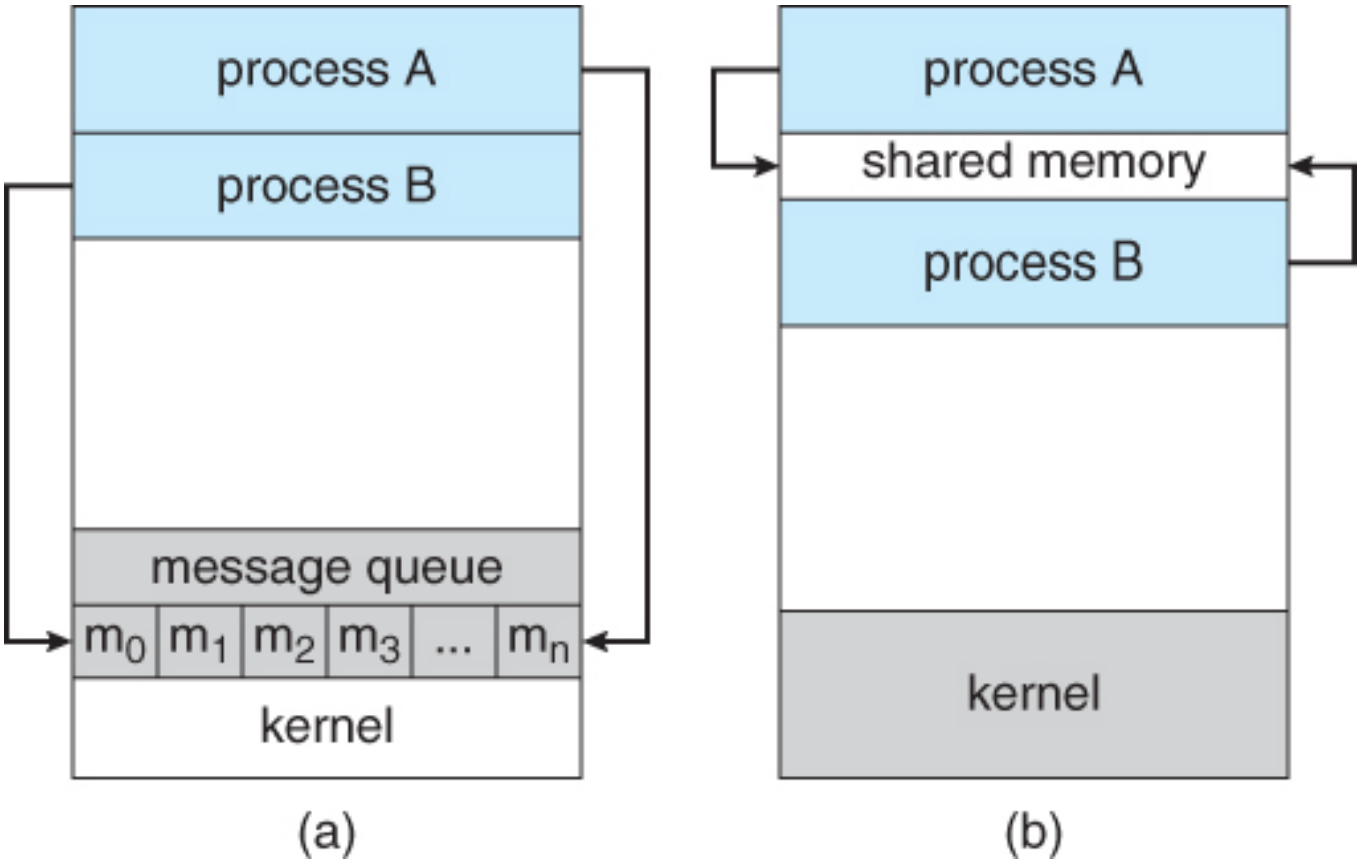


Figure 1: Modelo de pasaje de mensajes(izq) y memoria compartida(der)

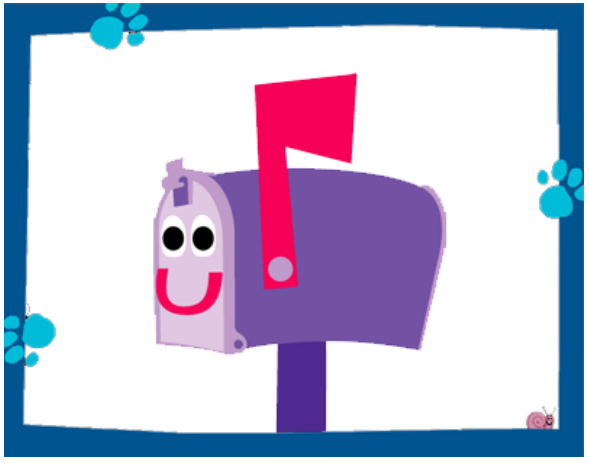
Pasaje de mensajes

- Los procesos se comunican intercambiando mensajes entre ellos
- La comunicación puede ser directa o indirecta
  - Si es directa, cada proceso debe tener una forma de referirse al otro
  - Si es indirecta, los procesos hablan entre ellos a través de un buzón (o puertos)

Ejemplos

- Para comunicación directa, tendría sentido tener una función SEND(P, MESSAGE), RECEIVE(P, MESSAGE) siendo P un proceso
- Para comunicación indirecta, SEND(A, MESSAGE), RECEIVE(A, MESSAGE) siendo A la dirección del buzón

Buffering



- Si el buzón tiene capacidad 0, no se pueden encolar mensajes. Por lo tanto, el emisor se bloquea hasta que el receptor recibe el mensaje
- Si el buzón tiene capacidad total $n \geq 1$ y si hay lugar libre, el emisor puede colocar su mensaje allí. Si no hay lugar, entonces se bloquea hasta que lo consiga

Sincronización

Lo anterior, da lugar a distintos tipos de sincronización entre los procesos según la característica de **send**() y **receive**():

- send bloqueante: El emisor se bloquea hasta que el receptor recibe el mensaje
- send no-bloqueante: El emisor envía el mensaje y continúa su ejecución
- receive bloqueante: El receptor se bloquea hasta recibir un mensaje
- receive no-bloqueante: El receptor obtiene un mensaje valido o un error si no había

Memoria compartida

Otra alternativa es un modelo de memoria compartida

Memoria compartida

- Normalmente, el sistema operativo intenta prevenir que un proceso acceda al espacio de memoria de otro
- Sin embargo, si ambos están de acuerdo, pueden hacerlo. De esta manera escriben o leen datos del mismo bloque de memoria
- Cada proceso es responsable de mantener la coherencia del modelo

memoria vs mensajes

- El modelo de mensajes es interesante a la hora de intercambiar poca información, ya que no incurre en conflictos de coherencia y puede ser más fácil de implementar
- El modelo de memoria permite mayor velocidad, ya que no tiene el *overhead* provocado por el manejo de mensajes y se puede trabajar, en general, a la velocidad que lo permita la memoria(si son procesos locales).
- Sin embargo, para el anterior, es necesario una complejidad adicional para sincronizar los procesos y proteger los datos

Memoria compartida en C

- ¿Podemos elegir que una variable sea compartida entre procesos? Sí
- En C, podemos utilizar la función **mmap** para hacer esto
- Vamos a omitir los detalles de su uso para no entrar en la parte oscura de C ahora...

Ejercicio

Queremos construir un programa de manera que:

- un proceso lea la variable ping y escriba su contenido incrementado en uno en la variable pong
- otro proceso lea la variable pong y escriba su contenido incrementado en uno en la variable ping

Para esto, tenemos a disposición la función share_mem(int var) que permite compartir la referencia, de tipo entero, entre todos los hijos de un proceso y el siguiente código con algunas funciones útiles.

```
int ping;
int pong;
int temp;

void pong() {
  while (true) {
     temp = ping;
     temp++;
     pong = temp;
  }
}

void ping() {
  while (true) {
     temp = pong;
     temp++;
     ping = temp;
  }
}
```

a) ¿Qué variables deben residir en el área de memoria compartida?
b) ¿Existe alguna variable que no deba residir en el espacio de memoria compartida?
c) Escribir un procedimiento main() para el problema usando el código presentado y las funciones para comunicación entre procesos provistas. Si creés que es necesario podés asumir que el scheduler siempre da la ejecución que fijes.