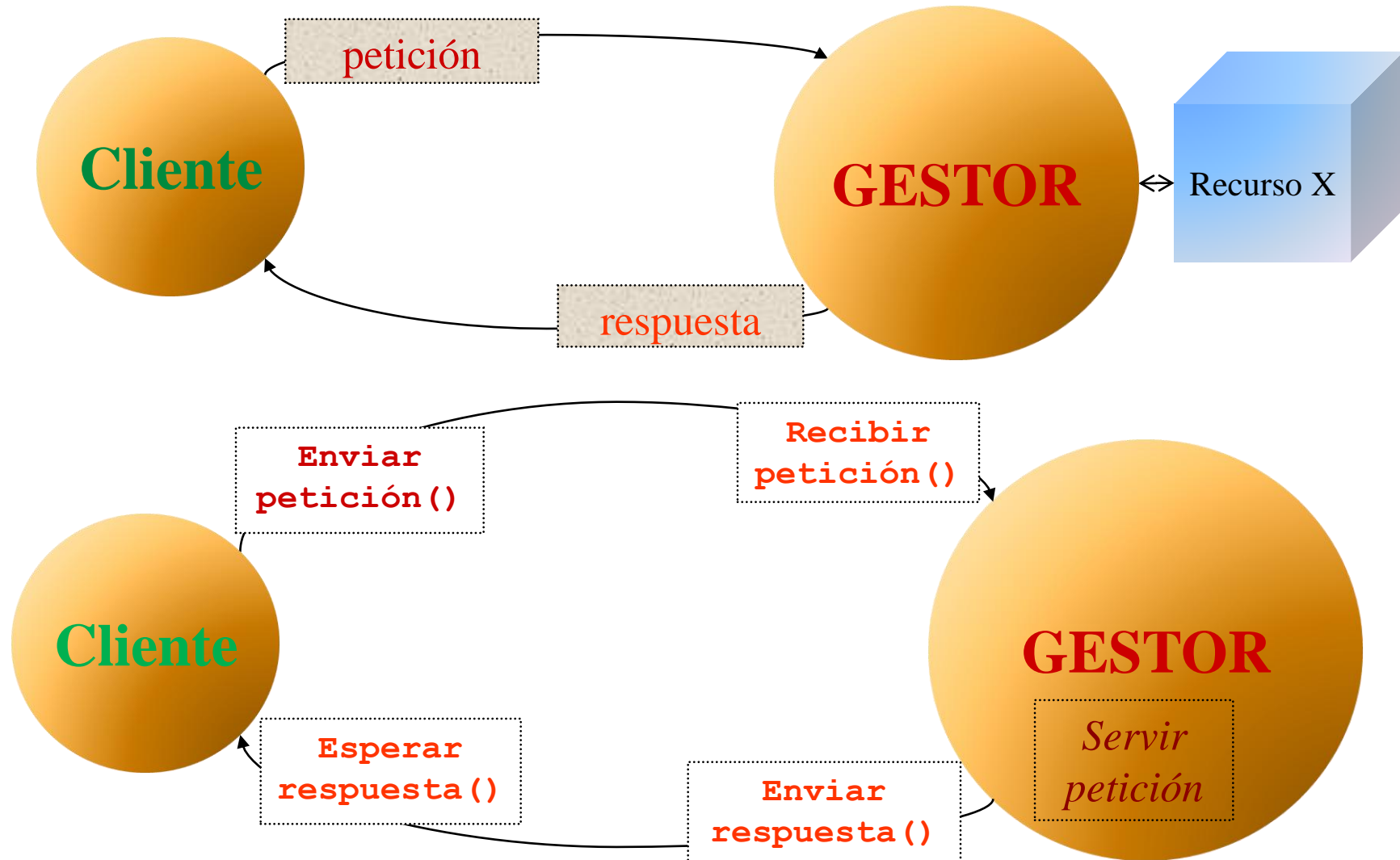


7.6 Modelo de gestión de recursos: esquema cliente-servidor

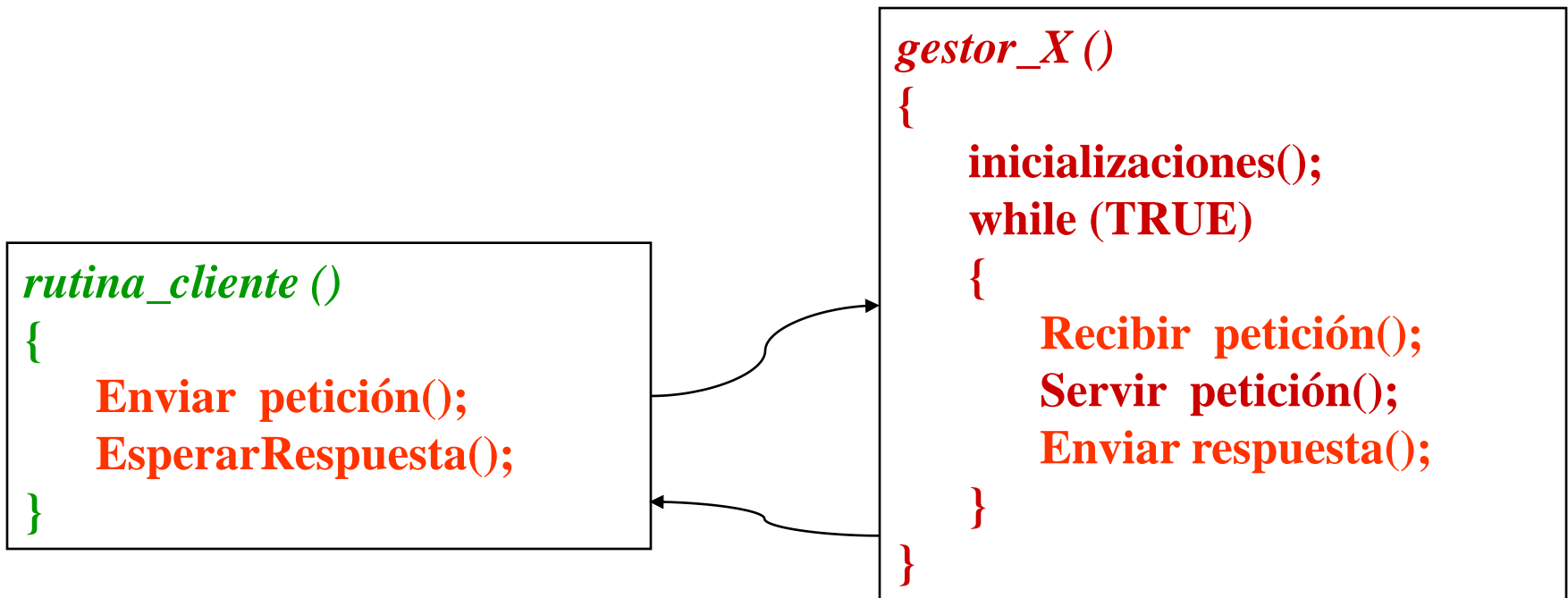
- ❑ Modelo Cliente-Servidor
- ❑ Trasladar código a las capas superiores del SO para reducir el tamaño del núcleo
- ❑ Utilizado para Gestionar ciertas funciones/recursos del SO
- ❑ Muy habitual en sistemas distribuidos y sistemas de red
- ❑ Proceso cliente (proc. usuario que realiza una solicitud)
- ❑ Proceso Servidor (proceso que realiza el trabajo y retorna la respuesta)

Modelo Cliente-Servidor



Modelo Cliente-Servidor

□ Modelo básico



Modelo Cliente-Servidor

□ Cliente (peticiones):

■ **Síncronas**

- Espera respuesta (datos) por parte del servidor
 - a) Cuando la petición se ha recibido
 - b) Cuando la petición comienza a ser servida
 - c) Cuando la petición se ha servido (finalizada)

■ **Asíncronas**

- No espera la respuesta

```
rutina_cliente ()  
{  
    Enviar petición();  
    EsperarRespuesta();  
}
```

```
rutina_cliente ()  
{  
    Enviar petición();  
}
```

□ Servidor

■ **secuencial**

- No se atiende una petición hasta que no finalice la atención de petición precedente. El propio servidor podría atender las peticiones (tareas). Habitual cuando sólo se gestiona **1** recurso

■ **Concurrente**

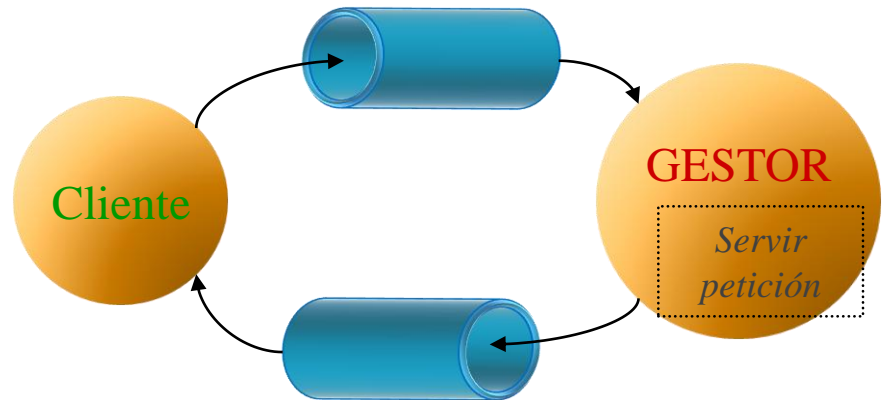
- Se puede atender una petición sin que haya finalizado la atención de una petición previa. El servidor debe encargar a otro proceso (hijo o no) la atención de la petición. Habitual cuando se deben gestionar **N** recursos

Modelo Cliente-Servidor

□ Servidor

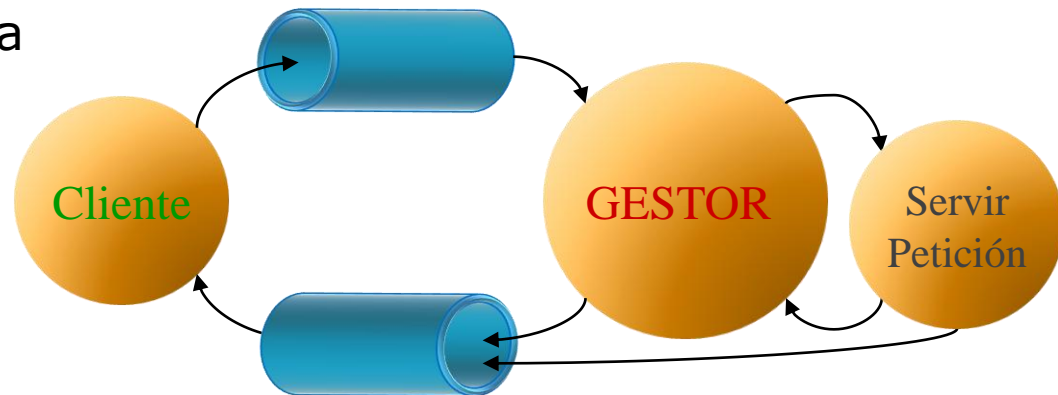
■ **Secuencial**

- No se atiende una petición hasta que la petición previa haya finalizado. El propio servidor puede atender la petición (tarea)



■ **Concurrente**

- Se puede atender a una petición sin que la petición previa haya finalizado. El servidor encarga a otro proceso (hijo o no) la atención de la petición.



Modelo Cliente-Servidor

□ N° recursos disponibles (en el servidor)

■ **1 Recurso**

- Las peticiones se atienden de una en una

■ **N Recursos**

- El servidor debe servir peticiones mientras disponga de recursos libres.
- Es importante conocer si hay que identificar de forma única a cada recurso: Problema de identificar los recursos libres/ocupados/liberados

□ N° recursos por petición

■ **1 Recurso**

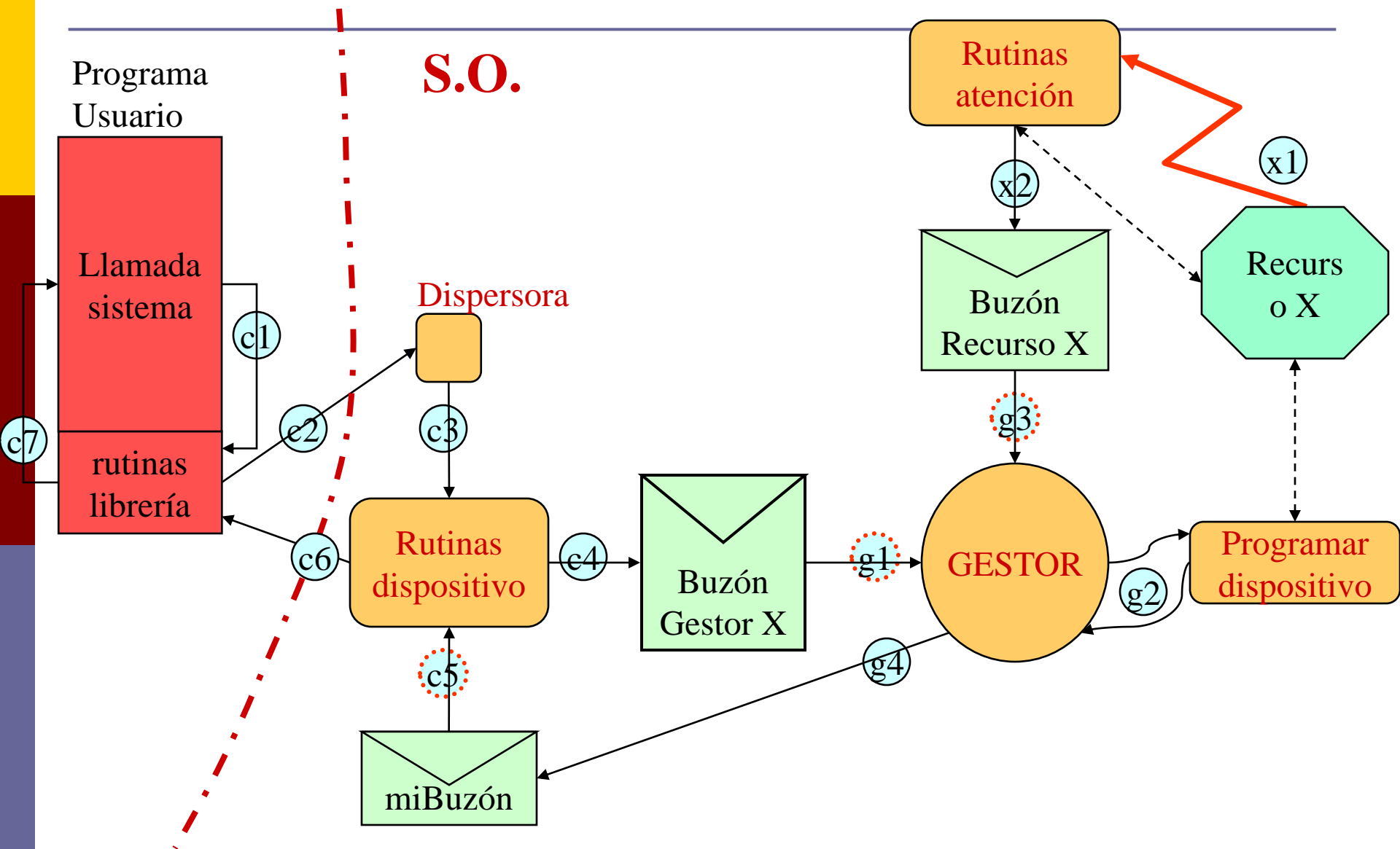
■ **k Recursos**

- Servidor: Si no hay suficientes recursos no se atiende (y no se debería atender a ninguna otra por el problema de la **inanición**)
- Si los recursos tienen identificador único aparece el problema de identificar los recursos libres/ocupados/liberados

7.7 Ejemplos de gestores de recursos

- Gestor de entrada/salida
- Gestor de impresora
- ...

Ejemplo de Modelo Cliente-Servidor: Esquema Gestor de E/S.



Ejemplo de Modelo Cliente-Servidor:

Llamada al sistema de E/S

```
rutina_dispositivo_X (char * vector, int num, asin_sin,  
                    buzon_expl)  
{  
    if (asin_sin == SINCRO) {  
        construir_petición(vector, num, mi_buzon, &pet);  
        enviar_mensaje(buzon_Gestor_X, &pet); c4  
        recibir_mensaje(mi_buzon, &fin_pet); c5  
        return(fin_pet.codigo_resultado); c6  
    } else {  
        construir_petición (vector, num, buzon_expl, &pet);  
        enviar_mensaje(buzon_Gestor_X, &pet); c4  
    }  
}
```

Ejem. de Modelo Cliente-Servidor: Gestor de E/S

```
gestor_X ()
```

```
{  
    struct peticion pet, fin_pet;  
    int num, buzon_Gestor, su_buzon, buzon_recurso_X;  
    char * vector;  
    inicializar_estructuras(Gestor_X);  
    inicializar_buzones(Gestor_X);  
    while (TRUE) {  
        recibir_mensaje(buzon_Gestor, &pet); g1  
        sacar_peticon(&pet, &vector, &num, &su_buzon);  
        programar_dispositivo_X(&vector, &num); g2  
        recibir_mensaje(buzon_recurso_X, &fin_pet); g3  
        enviar_mensaje(su_buzon, &fin_pet); g4  
    }  
}
```

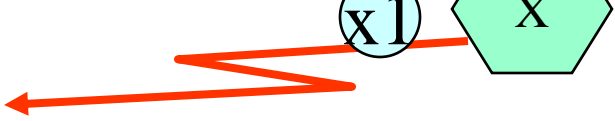
Servir
petición

}


}

Ejem. de Modelo Cliente-Servidor: rutinas de atención y sincronización

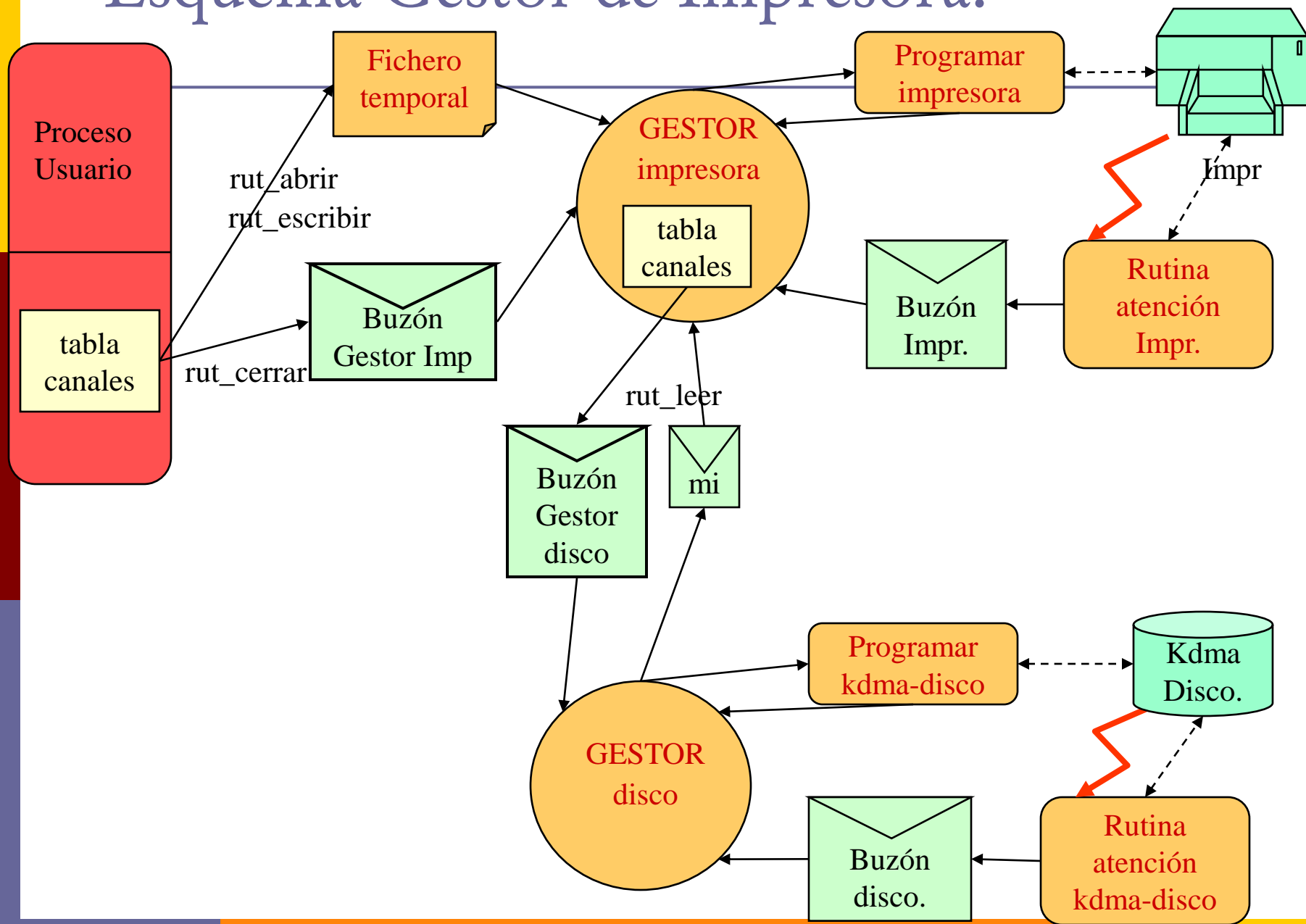
```
rutina_atención_dispositivo () {  
    struct peticion fin_pet;  
    obtiene_envia_dato();  
    if (FINAL) {  
        fin_pet.codigo_resultado = cod_result;  
        enviar_mensaje(buzon_recurso_X, &fin_pet) x2  
    }  
}
```



```
rutina_sincronizacion (int buzon_expl) {  
    struct peticion fin_pet;  
    recibir_mensaje(buzon_expl, &fin_pet); c5  
    return(fin_pet.codigo_resultado); c6  
}
```



Esquema Gestor de Impresora.



Gestor de la impresora.

Rut_Abrir y Rut_Cerrar (es cliente)

```
int rut_abrir (char * nombre)
{
    if (es_impresora(nombre)) {
        temporal(nom);
        canal = rut_abrir_fichero_impresora(nom);
        return (canal);
    } else {
        /* rut_abrir_X(nombre); según el dispositivo X*/
        return (canal);
    }
}
```

```
int rut_cerrar (int canal)
{
    if (impresora(canal)) {
        obtener_nom(canal, nom);
        construir_petición(nom, &pet);
        enviar_mensaje(buzon_Gestor_IMP, &pet);
    }
    liberar_canal(canal);
}
```

Gestor de la impresora. (Gestor)

```
gestor_impresora ()
```

```
{
```

```
...
```

```
inicializar_buzones(buzon_Gestor_impresora, buzon_impresora);
```

```
while (TRUE) {
```

```
    recibir_mensaje(buzon_Gestor_impresora, &pet);
```

```
    canal=rut_abrir_fichero(pet.nombre);
```

```
    while ((num = rut_leer(canal, vector,TAM)) != 0){
```

```
        programar_dispositivo Impr (vector, &num);
```

```
        recibir_mensaje(buzon_Impresora, &fin_pet);
```

```
    }
```

```
    rut_cerrar_fichero(canal);
```

```
    rut_borrar(pet.nombre);
```

```
}
```

```
}
```

Servir
petición