Práctica 6: (Temas 6 y 7) Sincronización por paso de mensajes

Objetivos:

- **practicar la utilización de señales**
- **practicar la utilización de variables condicionales**
- **practicar la sincronización de threads implementada con distintos mecanismos**

Rendezvous con señales

Crear el módulo "mensajes" (ficheros `mensajes.h` y `mensajes.c`)

- **que permite enviar mensajes basados en señales POSIX de tiempo real con información asociada**

Operaciones del módulo "mensajes":

```
void mensajes_send(int canal, int info);
// envía el mensaje con la información asociada 'info' por el
// canal indicado.
// Detalle de implementación: envía la señal de número
// 'canal' con la información asociada 'info'

void mensajes_wait(int canal, int *info);
// recibe un mensaje del canal indicado. La información
// asociada al mensaje se retorna en 'info'.
// Detalle de implementación: espera la señal de número
// 'canal' y retorna en 'info' el campo de información
// asociado con la señal recibida
```


Realización (cont.)

Escribir un programa que cree dos threads: "cliente" y "sumador"

Ambos threads se sincronizarán mediante el mecanismo de "Rendezvous completo" (implementado utilizando el módulo "mensajes")

Descripción del rendezvous:

- **el "cliente" envía un mensaje con un número al "sumador" y se bloquea a la espera del mensaje de respuesta**
- **el "sumador" recibe el mensaje del "cliente", suma 1 al número enviado y envía el resultado de la suma en el mensaje de respuesta**

Sincronización con variable condicional

Crear el módulo "sincroniza" (`sincroniza.h` y `sincroniza.c`)

- **permite enviar mensajes de sincronización (sin información asociada) utilizando una variable condicional y un mutex**

Operaciones del módulo "sincroniza":

```
void sincroniza_send();
// Envía el mensaje de sincronización. Si hay un thread
// esperando el mensaje con sincroniza_wait, dicho thread se
// activa. Si no hay ninguno, el mensaje queda almacenado, de
// forma que el siguiente thread que llame a sincroniza_wait
// no se bloquea y "consume" el mensaje.

void sincroniza_wait();
// Recibe el mensaje de sincronización. Si otro thread ha
// invocado previamente a sincroniza_send el thread no se
// bloquea y "consume" el mensaje. En otro caso el thread
// queda bloqueado a la espera de que otro thread llame a
// sincroniza_send.
```


Realización (cont.)

Escribir un programa que cree dos threads: "waiter" y "sender"

Probar que el mecanismo implementado funciona correctamente realizando diversas secuencias de llamada a `sincroniza_wait()` y `sincroniza_send()`.

¿Qué primitiva de sincronización/exclusión mutua es equivalente a las llamadas `sincroniza_wait()` y `sincroniza_send()`?

Entregar

Enviar por e-mail al profesor (`aldeam@unican.es`):

- **Código desarrollado**