Práctica 3: Exclusión mutua y mecanismos de sincronización.

Objetivos: El objetivo de la práctica es introducir los conceptos básicos de la gestión de la exclusión mutua y de la sincronización de procesos mediante el uso de semáforos binarios y contadores.

Como primera aproximación al uso de semáforos se incluyen dos documentos como parte de la información de la práctica:

- El documento "Teoría práctica 3: Hilos, comunicación y sincronización" incluye una descripción pormenorizada del uso de hilos POSIX así como los mecanismos de sincronización disponibles.
- El documento "Semáforos" incluye un breve resumen de los diferentes tipos de semáforos, así como una breve descripción de las variables de condición, que aunque no son objeto de la práctica son un mecanismo potente que merece la pena conocer.

Bajo el epígrafe "Ejemplos de uso de semáforos" se incluye ejemplos del uso de semáforos en las siguientes situaciones:

- **hilos.c**: Ejemplo de existencia de sección crítica sin gestionar adecuadamente que produce resultados erróneos.
- **critical.c**: Ejemplo de uso de semáforos binarios para gestionas la exclusión mutua de una sección crítica.
- **named_sem.c**: Ejemplo de uso de semáforos con nombre para la gestión de la exclusión mutua de procesos que comparten una zona de memoria.
- **unnamed_sem_t.c**: Ejemplo de **MAL** uso de semáforos sin nombre para la gestión de la exclusión mutua de procesos que comparten una zona de memoria.
- **unnamed_sem_t_ok.c**: Corrección del ejemplo anterior para el uso correcto de semáforos sin nombre para la gestión de la exclusión mutua de procesos que comparten una zona de memoria.

Tareas: La práctica consistirá en la implementación de las siguientes tareas. El alumno podrá elegir en cada ejercicio realizarlo mediante hilos y memoria global o mediante procesos y memoria compartida.

Ejercicio 1: Compilar y ejecutar los programas ejemplo citados anteriormente hasta comprender correctamente su funcionamiento, las diferencia entre procesos e hilos y los diferentes tipos de semáforos. Comprobar el funcionamiento incorrecto de los programas cuando la sección crítica no se trata adecuadamente.

Ejercicio 2: Tomando como base el ejemplo hilos.c implementar la exclusión mutua usando el algoritmo de la panadería de Lamport.

Seudocódigo del algoritmo de Lamport:

```
// Variables globales compartidas
BOOLEAN Eligiendo[N];
int Turno[N];

// Inicialización de variables
Eligiendo[] = {false, false,..., false};
Turno[] = {0, 0,..., 0};

// Proceso o hilo I-ésimo
void Proceso(int I)
{
   extern BOOLEAN Eligiendo[N];
   extern int Turno[N];
   int j;
```

```
  while (true) {
    Eligiendo[I] = true;
    Turno[I] = max(Turno[0], Turno[1],..., Turno[N - 1]) + 1;
    Eligiendo[I] = false;
    for (j = 0; j < N; j++) {
      while (Eligiendo[j]);
      while ((Turno[j] != 0) && (Turno[j], j) < (Turno[I], I));
    }
    //-------------------
    // SECCIÓN CRÍTICA
    //-------------------
    Turno[I] = 0;
    // Sección residual
  }
}
```

Ejercicio 3: Implemente esta misma solución usando semáforos binarios, semáforos contadores y semáforos contadores con nombre.

Ejercicio 4: Implemente una solución al problema de los productores-consumidores para P productores y C consumidores con las siguientes características:

- Los productores y consumidores comparten un búfer limitado de tamaño N = 100.
- Cada uno de los P productores produce 1000 números aleatorios en el intervalo [0, 10] y los deposita en el búfer. Cuando finaliza cada productor devuelve la suma de todos los números que ha generado al programa principal.
- Cada uno de los C consumidores consume los números depositados por los productores del búfer hasta que todos los productores han finalizado. Cuando un consumidor finaliza devuelve la suma total de los valores consumidos.
- Cuando han finalizado todos los productores y consumidores el programa principal compara las sumas de los elementos producidos y consumidos para comprobar si cada elemento producido se ha consumido una vez y solo una vez.

Ejercicio 5 (OPCIONAL): Modifique el ejercicio anterior para que cada elemento producido sea consumido por todos los consumidores.

Seudocódigo del problema del productor-consumidor:

```
// Variables globales compartidas
item bufer[N];
semaforo empty, full, mutex;

// Inicialización de variables
init(mutex, 1);
init(full, 0);
init(empty, N);


// Proceso productor
```

```
void Productor()
{
   extern semaforo mutex, full, empty;
   T dato;

   while (true) {
      ProducirDato(dato);
      wait(empty);
      wait(mutex);
      EntrarDato(dato);
      signal(mutex);
      signal(full);
   }
}

// Proceso consumidor
void Consumidor()
{
   extern semaforo mutex, full, empty;
   T dato;

   while (true) {
      wait(full);
      wait(mutex);
      SacarDato(dato);
      signal(mutex);
      signal(empty);
      ConsumirDato(dato);
   }
}
```