Plantillas Maestras Universales de SCD

Monitores Oficiales UGR (`HoareMonitor` / SU) y Semáforos POSIX (Paso de Testigo)

Guía de Código Canónico y Pseudocódigo para el Examen Parcial — Prof. Manuel I. Capel Tuñón

¿Por qué estas plantillas cubren el 100 % de los problemas de examen?

En la Universidad de Granada, el 100 % de los ejercicios prácticos de concurrencia de la última década corresponden exactamente a una de las siguientes **4 familias de patrones**:

1. **Monitores de Dos Bandos / Recursos Incompatibles** (Puente de 1 carril, Aseo mixto con aforo, Centro de Recursos tipo 0/1, Olla de salvajes, Barbería). → **Plantilla 1A** (Página 2).
2. **Monitores de Lectores–Escritores y Turno FIFO Estricto** (Lectores/Escritores con prioridad a escritores, o Cuenta Bancaria para evitar adelantamientos entre solicitudes grandes y pequeñas). → **Plantillas 1B y 1C** (Páginas 4 y 5).
3. **Semáforos con Paso de Testigo (*Passing the Baton*)** (Implementar con semáforos cualquier problema de aforos, bandos o prioridades complejas sin quemar CPU). → **Plantilla 2** (Página 6).
4. **Semáforos para Coordinación de Instrucciones entre P_1, P_2, P_3 (*Handshake / Rendezvous*)** (Garantizar orden: *«P_2 ejecuta 'e' tras el 'b' de P_1 y el 'h' de P_3»*). → **Plantilla 3** (Página 7).

Las 4 Reglas de Oro de Capel para el Examen (Monitores y Semáforos)

1. **Monitores UGR (`scd.h`):** Siempre derivar de `public HoareMonitor`. Toda variable condición `CondVar` **debe inicializarse obligatoriamente en el constructor** mediante `cola = newCondVar();`.
2. **Semántica SU (Hoare) vs SC (Mesa):** En la UGR se usa semántica **Señalar y Espera Urgente (SU)**. El proceso que hace `signal()` cede la posesión inmediata del monitor al proceso despertado. En consecuencia:
   - **Basta con un `if (!cond) cola.wait();`** (no es necesario bucle `while` porque nadie se puede colar).
   - Si el enunciado exige semántica Mesa/SC (Java, POSIX puro), **es obligatorio usar `while (!cond) cola.wait();`**.
3. **Semáforos con Paso de Testigo:** El semáforo `mutex` se inicializa a 1 y los semáforos privados a 0. **Regla sagrada:** Si al salir se despierta a un proceso bloqueado mediante su semáforo privado (`sem_post(&sem_privado)`), **NUNCA se debe hacer `sem_post(&mutex)`**, ya que el testigo de la exclusión mutua se transfiere en mano.
4. **C++ vs Pseudocódigo:** En los exámenes de Capel se acepta tanto la implementación oficial en C++ con `HoareMonitor` como el pseudocódigo formal de Hoare/Pascal. En esta guía se proporcionan ambas sintaxis para máxima seguridad.

Equivalencia Inmediata: Sintaxis C++ UGR ⟷ Pseudocódigo Hoare Canónico

Elemento	C++ Oficial UGR (`scd.h`)	Pseudocódigo Hoare / Capel
Definición monitor	`class M : public HoareMonitor`	`monitor M; ... end;`
Variable de condición	`CondVar cola;`	`var cola: condition;`
Creación / Inicialización	`cola = newCondVar();` (en constr.)	Automática en la declaración del monitor
Bloqueo en condición	`cola.wait();`	`wait(cola);` o `cola.wait;`
Despertar a un proceso	`cola.signal();`	`signal(cola);` o `cola.signal;`
Semáforo inicialización	`sem_init(&s, 0, valor);`	`var s: semaphore := valor;`
Operación de espera sem.	`sem_wait(&s);`	`wait(s);` o `P(s);`
Operación de aviso sem.	`sem_post(&s);`	`signal(s);` o `V(s);`

Plantilla 1A: Monitor Universal de Dos Bandos / Recursos Incompatibles (Una Cara)

Ámbito: Puente 1 carril, Aseo mixto, Centro Tipo 0/1, Olla, Barbería. **Regla SU:** Uso de `if` sin `while` (el despertado entra de inmediato).

```
#include "scd.h"
using namespace std; using namespace scd;

class MonitorUniversal : public HoareMonitor {
private:
    int dentro_A, dentro_B;       // Procesos actualmente dentro de cada tipo
    int esperando_A, esperando_B; // Procesos esperando fuera (para evaluar prioridades)
    CondVar cola_A, cola_B;       // Colas de espera donde duermen los procesos

public:
    MonitorUniversal() : dentro_A(0), dentro_B(0), esperando_A(0), esperando_B(0) {
        cola_A = newCondVar(); // ¡OBLIGATORIO en el constructor de la UGR!
        cola_B = newCondVar();
    }

    void entrar_A() {
        // Bloqueo: hay incompatibles dentro O aforo lleno O prioridad rival con esperando_B > 0
        if (dentro_B > 0 /* || dentro_A == AFORO_MAX || esperando_B > 0 */) {
            esperando_A++; cola_A.wait(); esperando_A--;
        }
        dentro_A++; // Entrada autorizada
        // Desbloqueo en cascada opcional: if (dentro_A < AFORO_MAX && esperando_A > 0) cola_A.signal();
    }

    void salir_A() {
        dentro_A--; // Anota salida
        // Si no queda nadie de A y hay rivales esperando, dar paso prioritario a B:
        if (dentro_A == 0 && esperando_B > 0) cola_B.signal();
        else if (dentro_B == 0 && esperando_A > 0) cola_A.signal();
    }

    void entrar_B() {
        if (dentro_A > 0 /* || dentro_B == AFORO_MAX || esperando_A > 0 */) {
            esperando_B++; cola_B.wait(); esperando_B--;
        }
        dentro_B++;
    }

    void salir_B() {
        dentro_B--;
        if (dentro_B == 0 && esperando_A > 0) cola_A.signal();
        else if (dentro_A == 0 && esperando_B > 0) cola_B.signal();
    }
};
```

Equivalente Canónico en Pseudocódigo Teórico de Hoare (Capel)

```
monitor MonitorUniversal;
    var dentro_A, dentro_B, esperando_A, esperando_B: integer;
        cola_A, cola_B: condition;

    procedure entrar_A;
    begin
        if (dentro_B > 0) then begin esperando_A := esperando_A + 1; wait(cola_A); esperando_A :=
             esperando_A - 1; end;
        dentro_A := dentro_A + 1;
    end;

    procedure salir_A;
    begin
        dentro_A := dentro_A - 1;
        if (dentro_A = 0 and esperando_B > 0) then signal(cola_B)
        else if (dentro_B = 0 and esperando_A > 0) then signal(cola_A);
    end;

    procedure entrar_B; begin if (dentro_A > 0) then begin esperando_B := esperando_B + 1; wait(cola_B);
         esperando_B := esperando_B - 1; end; dentro_B := dentro_B + 1; end;
    procedure salir_B;  begin dentro_B := dentro_B - 1; if (dentro_B = 0 and esperando_A > 0) then
         signal(cola_A) else if (dentro_A = 0 and esperando_B > 0) then signal(cola_B); end;
begin dentro_A := 0; dentro_B := 0; esperando_A := 0; esperando_B := 0; end;
```

Diccionario de Adaptación Inmediata de la Plantilla 1A

Problema Típico	Variables y Bandos (A y B)	Condición de Bloqueo (`if`) y Aviso (`signal`)
Puente de 1 carril *(Norte vs. Sur)*	A: Coches Norte (`norte`) B: Coches Sur (`sur`)	**Entrada Norte:** `if (sur >0) cola_norte.wait();` **Salida Norte:** `if (norte == 0) cola_sur.signal(); else cola_norte.signal();`
Aseo Mixto *(Hasta 5H ó 1M)*	A: Hombres (H, máx 5) B: Mujeres (M, máx 1)	**Entrada Hombre:** `if (M >0 || H == 5 || (prio_M && esp_M >0)) cola_H.wait();` **Entrada Mujer:** `if (H >0 || M == 1 || (prio_H && esp_H >0)) cola_M.wait();` **Salida Hombre:** `if (H == 0 && esp_M >0) cola_M.signal(); else cola_H.signal();`
Centro de Recursos *(Tipo 0 vs. 1, cap. M)*	A: Procesos 0 (`d_0`) B: Procesos 1 (`d_1`)	**Entrada 0:** `if (d_1 >0 || d_0 == M || esp_1 >esp_0) cola_0.wait();` **Salida 0:** `if (d_0 == 0 && esp_1 >0) cola_1.signal(); else cola_0.signal();`
Olla de Caníbales *(M raciones)*	Salvajes comen (A). Cocinero rellena (B).	**Variable:** `int raciones = M;` **Salvaje come:** `if (raciones == 0) { cola_cocinero.signal(); cola_salvajes.wait(); } raciones-;` **Cocinero rellena:** `if (raciones >0) cola_cocinero.wait(); raciones = M; cola_salvajes.signal();`
Barbería *(Sillas de espera)*	Cliente entra (A). Barbero corta (B).	**Variables:** `bool barbero_libre = true; int esperando = 0;` **Cliente:** `if (barbero_libre) { barbero_libre = false; cola_barbero.signal(); } else cola_sillas.wait();` **Barbero:** `if (esperando == 0) cola_barbero.wait(); else cola_sillas.signal();`

Justificación Teórica Obligatoria para Obtener el 10 en el Examen de Capel

Capel siempre añade la coletilla: *«prestando especial atención al razonamiento sobre la corrección y la justificación de las soluciones propuestas»*. En tu examen, redacta siempre estos 3 puntos tras el código:

1. **Invariante de Exclusión Mutua (Seguridad):**
   - Demostrar que nunca coinciden tipos incompatibles: $I \equiv (\text{dentro_A} = 0 \vee \text{dentro_B} = 0) \wedge (\text{dentro_A} \geq 0 \wedge \text{dentro_B} \geq 0)$.
   - Un proceso A solo incrementa `dentro_A` si dentro_B $= 0$, y mientras dentro_A > 0 ningún proceso B puede superar su guarda de entrada. Por inducción, I se mantiene invariable en todo estado accesible.
2. **Ausencia de Interbloqueo (*Deadlock*):**
   - No puede haber bloqueo mutuo permanente porque las condiciones de bloqueo son disjuntas: si hay procesos esperando en ambas colas, el último proceso en salir del recurso activo transfiere explícitamente el control despertando a la cola contraria (`cola_B.signal()`).
3. **Semántica SU vs Mesa (Por qué no hay inanición ni se usa `while`):**
   - En semántica Señalar y Espera Urgente (Hoare), el proceso señalador pasa el monitor directamente al proceso en la cabeza de la cola, y este pasa a ejecutarse de forma inmediata sin que ningún proceso externo pueda adelantarlo en la cola de entrada. Esto garantiza la validez de la condición de entrada sin reevaluación (`if`).

Plantilla 1B: Monitor de Lectores–Escritores con Prioridad a Escritores (Una Cara)

Objetivo: Permitir múltiples lectores concurrentes, exclusión para escritores y evitar inanición de escritores esperando.

```
#include "scd.h"
using namespace std; using namespace scd;

class LectoresEscritores : public HoareMonitor {
private:
    int      n_lec;            // Numero de lectores actualmente leyendo
    bool     escribiendo;      // true si hay un escritor en la seccion critica
    int      esp_escritores;   // Escritores esperando turno fuera
    CondVar cola_lectores;     // Cola donde esperan los lectores
    CondVar cola_escritores;   // Cola donde esperan los escritores

public:
    LectoresEscritores() : n_lec(0), escribiendo(false), esp_escritores(0) {
        cola_lectores   = newCondVar(); // Inicializacion obligatoria en constructor
        cola_escritores = newCondVar();
    }

    void iniciar_lectura() {
        // PRIORIDAD A ESCRITORES: esperar si hay escritor dentro O esperando fuera
        if (escribiendo || esp_escritores > 0) cola_lectores.wait();
        n_lec++;
        cola_lectores.signal(); // Cascada: despierta al siguiente lector de la cola
    }

    void fin_lectura() {
        n_lec--;
        // El ultimo lector en salir despierta al escritor que espera
        if (n_lec == 0) cola_escritores.signal();
    }

    void iniciar_escritura() {
        esp_escritores++; // Anotamos escritor esperando
        if (escribiendo || n_lec > 0) cola_escritores.wait();
        esp_escritores--;
        escribiendo = true;
    }

    void fin_escritura() {
        escribiendo = false;
        // Prioridad al siguiente escritor; si no hay, liberamos a los lectores
        if (esp_escritores > 0) cola_escritores.signal();
        else cola_lectores.signal();
    }
};
```

Equivalente Canónico en Pseudocódigo Teórico de Hoare (Capel)

```
monitor LectoresEscritores;
    var n_lec, esp_escritores: integer; escribiendo: boolean;
        cola_lectores, cola_escritores: condition;

    procedure iniciar_lectura;
    begin
        if (escribiendo or esp_escritores > 0) then wait(cola_lectores);
        n_lec := n_lec + 1; signal(cola_lectores); // Cascada de lectores
    end;

    procedure fin_lectura;
    begin
        n_lec := n_lec - 1;
        if (n_lec = 0) then signal(cola_escritores);
    end;

    procedure iniciar_escritura;
    begin
        esp_escritores := esp_escritores + 1;
        if (escribiendo or n_lec > 0) then wait(cola_escritores);
        esp_escritores := esp_escritores - 1; escribiendo := true;
    end;

    procedure fin_escritura;
    begin
        escribiendo := false;
        if (esp_escritores > 0) then signal(cola_escritores) else signal(cola_lectores);
    end;
begin n_lec := 0; esp_escritores := 0; escribiendo := false; end;
```

Nota de examen sobre Inanición: La prioridad a escritores puede causar inanición a los lectores ante un flujo continuo de escritores. Si piden equidad absoluta, se añade un contador de turnos consecutivos.

Plantilla 1C: Monitor con Turno FIFO Estricto (Ej. Cuenta Bancaria) (Una Cara)

Problema Clave (Examen 2021): Retiradas de gran importe no pueden ser adelantadas por solicitudes menores. Patrón de **2 Colas**.

```
#include "scd.h"
using namespace std; using namespace scd;

class CuentaBancariaFIFO : public HoareMonitor {
private:
    int      saldo;            // Saldo disponible (Invariante de seguridad: saldo >= 0)
    int      c_primero;        // Cantidad requerida por el proceso en cabeza de turno
    bool     turno_ocupado;    // true si ya hay un cliente con el turno concedido
    CondVar cola_espera;       // Fila general FIFO por estricto orden de llegada
    CondVar cola_turno;        // Cola donde se duerme el cliente en turno si saldo < c_primero

public:
    CuentaBancariaFIFO() : saldo(0), c_primero(0), turno_ocupado(false) {
        cola_espera = newCondVar();
        cola_turno  = newCondVar();
    }

    void depositar(int c) {
        saldo += c;
        // Si el cliente en turno estaba esperando fondos y ahora alcanza el saldo:
        if (turno_ocupado && saldo >= c_primero) {
            cola_turno.signal(); // En semantica SU, despierta y entra de inmediato
        }
    }

    void retirar(int c) {
        // 1. Si ya hay otro cliente con el turno, esperar en la fila general FIFO:
        if (turno_ocupado) {
            cola_espera.wait();
        }

        // 2. Adquirir el turno de retirada en exclusiva:
        turno_ocupado = true;
        c_primero = c;

        // 3. Si no hay saldo suficiente, suspenderse en la cola privada de turno:
        if (saldo < c) {
            cola_turno.wait();
        }

        // 4. Retirada efectiva (en semantica SU, saldo >= c plenamente garantizado):
        saldo -= c;

        // 5. Liberar el turno y despertar al siguiente cliente de la fila de espera:
        turno_ocupado = false;
        c_primero = 0;
        cola_espera.signal();
    }
};
```

Equivalente Canónico en Pseudocódigo Teórico de Hoare (Capel)

```
monitor CuentaBancariaFIFO;
    var saldo, c_primero: integer; turno_ocupado: boolean;
        cola_espera, cola_turno: condition;

    procedure depositar(c: integer);
    begin
        saldo := saldo + c;
        if (turno_ocupado and saldo >= c_primero) then signal(cola_turno);
    end;

    procedure retirar(c: integer);
    begin
        if (turno_ocupado) then wait(cola_espera);
        turno_ocupado := true; c_primero := c;
        if (saldo < c) then wait(cola_turno);
        saldo := saldo - c;
        turno_ocupado := false; c_primero := 0;
        signal(cola_espera);
    end;
begin saldo := 0; c_primero := 0; turno_ocupado := false; end;
```

Garantía de No Inanición: La cola `cola_espera` de `HoareMonitor` asegura orden FIFO estricto. Ninguna petición posterior puede saltar a `cola_turno` ni retirar saldo mientras el cliente actual mantenga `turno_ocupado == true`, erradicando la inanición.

Plantilla 2: Semáforos con Paso de Testigo (*Passing the Baton*) (Una Cara)

Fundamento: Semáforos POSIX para aforos y prioridades. **Regla de Oro:** En la salida, al hacer `sem_post(&privado)`, **NO** hacer `sem_post(&mutex)`.

```c
#include <semaphore.h>

sem_t mutex;                // Mutex de exclusion mutua sobre las variables (inic. a 1)
sem_t sem_A, sem_B;         // Semaforos privados de bloqueo (inicializados a 0)
int dentro_A = 0, esp_A = 0; // Contadores de procesos dentro y esperando
int dentro_B = 0, esp_B = 0;

void inicializar() {
    sem_init(&mutex, 0, 1);
    sem_init(&sem_A, 0, 0);  // Inic 0: quien haga wait aqui se bloquea obligatoriamente
    sem_init(&sem_B, 0, 0);
}

void entrar_A() {
    sem_wait(&mutex); // Pedir el testigo para consultar el estado
    if (dentro_B > 0 || dentro_A >= 5 /* || esp_B > 0 si B tiene prioridad */) {
        esp_A++;          // Anotarse en lista de espera
        sem_post(&mutex); // Liberar mutex para que otros puedan avanzar
        sem_wait(&sem_A); // BLOQUEO EFECTIVO en el semaforo privado
    } else {
        dentro_A++;
        sem_post(&mutex); // Entrada autorizada de forma directa
    }
}

void salir_A() {
    sem_wait(&mutex); // Adquirir el testigo para actualizar el estado
    dentro_A--;

    // REGLA SAGRADA DEL PASO DE TESTIGO (evaluar transferencias):
    // Caso 1: Pasar el testigo a un proceso B si ya no quedan A dentro y hay B esperando:
    if (dentro_A == 0 && esp_B > 0) {
        esp_B--; dentro_B++;
        sem_post(&sem_B); // ¡SE TRANSFIERE EL TESTIGO EN MANO (NO hacer post a mutex)!
    }
    // Caso 2: Si no, transferir el testigo a otro A que estuviera en espera:
    else if (dentro_B == 0 && esp_A > 0 && dentro_A < 5) {
        esp_A--; dentro_A++;
        sem_post(&sem_A); // ¡SE TRANSFIERE EL TESTIGO A OTRO A (NO hacer post a mutex)!
    }
    // Caso 3: Nadie puede entrar; se devuelve el testigo general al sistema:
    else {
        sem_post(&mutex); // Unico caso donde se libera el mutex
    }
}
```

Equivalente Canónico en Pseudocódigo Clásico (Dijkstra / Andrews)

```
var mutex: semaphore := 1; sem_A, sem_B: semaphore := 0;
    dentro_A, esp_A, dentro_B, esp_B: integer := 0;

procedure entrar_A;
begin
    wait(mutex);
    if (dentro_B > 0 or dentro_A >= 5) then begin esp_A := esp_A + 1; signal(mutex); wait(sem_A); end
    else begin dentro_A := dentro_A + 1; signal(mutex); end;
end;

procedure salir_A;
begin
    wait(mutex); dentro_A := dentro_A - 1;
    if (dentro_A = 0 and esp_B > 0) then begin esp_B := esp_B - 1; dentro_B := dentro_B + 1; signal(sem_B);
        end
    else if (dentro_B = 0 and esp_A > 0 and dentro_A < 5) then begin esp_A := esp_A - 1; dentro_A :=
        dentro_A + 1; signal(sem_A); end
    else signal(mutex);
end;
```

Plantilla 3: Coordinación Asimétrica entre Procesos (P_1, P_2, P_3 / *Handshake*) (Una Cara)

Problema Estrella (Examen 2025): P_2 ejecuta 'e' tras el 'b' de P_1 al menos una vez, y tras el 'h' de P_3 exactamente 1 vez por cada 'e'.

```
#include <semaphore.h>

// 1. Coordinacion 1 a 1 estricta (P3 produce 'h' y P2 consume 'e'):
sem_t sem_h_hecho;   // Inic 0: P3 senala a P2 que 'h' ha concluido
sem_t sem_h_ack;     // Inic 1: P2 autoriza a P3 a ejecutar su siguiente 'h'

// 2. Condicion "Al menos una vez" (desbloqueo permanente tras la primera ocurrencia):
sem_t sem_b_primera; // Inic 0: barrera que se desbloquea tras el primer 'b'
bool b_ya_ocurrio = false;
sem_t mutex_b;       // Inic 1: protege el acceso concurrente a b_ya_ocurrio

void inicializar() {
    sem_init(&sem_h_hecho, 0, 0);
    sem_init(&sem_h_ack, 0, 1);       // Permite a P3 realizar su primer 'h'
    sem_init(&sem_b_primera, 0, 0);
    sem_init(&mutex_b, 0, 1);
}

void P1() {
    while (true) {
        a(); b();
        // Notificar que 'b' se ejecuto al menos una vez en la historia:
        sem_wait(&mutex_b);
        if (!b_ya_ocurrio) {
            b_ya_ocurrio = true;
            sem_post(&sem_b_primera); // Abre la barrera para P2
        }
        sem_post(&mutex_b);
        c();
    }
}

void P2() {
    while (true) {
        d();
        // Paso 1: Comprobar condicion permanente de P1 ("al menos una vez"):
        sem_wait(&mutex_b);
        if (!b_ya_ocurrio) {
            sem_post(&mutex_b);
            sem_wait(&sem_b_primera); // Espera al primer 'b'
            sem_post(&sem_b_primera); // ¡TRUCO CANONICO!: se deja abierto para siempre
        } else {
            sem_post(&mutex_b);
        }

        // Paso 2: Handshake 1 a 1 con P3:
        sem_wait(&sem_h_hecho); // Espera que P3 complete su 'h'
        e();                    // EJECUCION DE LA SECCION PROTEGIDA
        sem_post(&sem_h_ack);   // Concede permiso a P3 para su siguiente 'h'
        f();
    }
}

void P3() {
    while (true) {
        g();
        sem_wait(&sem_h_ack);   // Espera autorizacion de P2
        h();
        sem_post(&sem_h_hecho); // Senala a P2 que 'h' termino
        i();
    }
}
```

Equivalente Canónico en Pseudocódigo Formal (Estilo Capel)

```
var sem_h_hecho: semaphore := 0; sem_h_ack: semaphore := 1;
    sem_b_primera: semaphore := 0; mutex_b: semaphore := 1; b_ya_ocurrio: boolean := false;

process P1; begin loop a(); b(); wait(mutex_b); if not b_ya_ocurrio then begin b_ya_ocurrio := true;
     signal(sem_b_primera); end; signal(mutex_b); c(); end loop; end;
process P2; begin loop d(); wait(mutex_b); if not b_ya_ocurrio then begin signal(mutex_b);
     wait(sem_b_primera); signal(sem_b_primera); end else signal(mutex_b); wait(sem_h_hecho); e();
     signal(sem_h_ack); f(); end loop; end;
process P3; begin loop g(); wait(sem_h_ack); h(); signal(sem_h_hecho); i(); end loop; end;
```

Tabla de Decisión Rápida: Identificar el Problema en 10 Segundos

Frase clave en el enunciado	Herramienta a usar	Acción inmediata y plantilla
«Diseñar un monitor para un puente / aseo / olla / centro...»	**HoareMonitor** (Semántica SU)	**Plantilla 1A** (Pág. 2). Crear colas con `newCondVar()`. Usar `if (!cond) cola.wait()` y avisar en la salida con `cola.signal()`.
«Atender en estricto orden de llegada sin que las pequeñas adelanten a las grandes»	**HoareMonitor** con cola privada	**Plantilla 1C** (Pág. 5). Usar dos colas: `cola_espera` (fila de llegada) y `cola_turno` (fila del que tiene el turno si saldo $< c$).
«Lectores y escritores con preferencia a escritores / lectores»	**HoareMonitor** de dos colas	**Plantilla 1B** (Pág. 4). Bloquear lectores si `esp_escritores >0`. Desbloqueo en cascada para lectores con `signal()`.
«Resolver el aseo / puente con semáforos POSIX...»	**Paso de Testigo** (*Baton*)	**Plantilla 2** (Pág. 6). `sem_t mutex = 1`, semáforos privados a 0. En salida, si alguien puede pasar, hacerle `post(sem_X)` sin soltar mutex.
«P2 no puede ejecutar X hasta que P1 ejecute Y...»	Semáforos **Handshake**	**Plantilla 3** (Pág. 7). Semáforo de fin `hecho = 0` y autorización previa `ack = 1`.
«Búfer de N elementos con semáforos...»	Semáforos contadores clásicos	`sem_t libres = N, ocupados = 0, mutex = 1;` Productor: `wait(libres); wait(mutex); ... post(mutex); post(ocupados);` Consumidor: `wait(ocupados); wait(mutex); ... post(mutex); post(libres);`

Checklist de Seguridad antes de Entregar el Examen de Capel

- ☐ **¿Es un monitor?**: ¿He puesto `#include "scd.h"` y heredado de `public HoareMonitor`?
- ☐ **¿He inicializado todas las `CondVar` en el constructor?**: Recordar `cola = newCondVar();`. ¡Olvidarlo aborta el programa!
- ☐ **¿Es semántica SU (UGR)?**: BASTA con usar `if (!cond) cola.wait();`. Si me piden semántica Mesa, entonces cambio a `while (!cond) cola.wait();`.
- ☐ **¿Son semáforos con Paso de Testigo?**: Comprobar que en la salida **NO** se hace `sem_post(&mutex)` si transfieres el testigo haciendo `sem_post(&sem_privado)`.
- ☐ **¿Inanición discutida?**: Si una clase puede acaparar el recurso indefinidamente (ej. flujo continuo de coches norte o lectores), mencionar la adición de un cupo de paso consecutivo para alternar.
- ☐ **¿Justificación escrita?**: Escribir en 3 líneas el Invariante de exclusión mutua, la ausencia de interbloqueo y por qué no hay inanición.