```
/* SOLUCIÓN ORIGINAL */

#define N 5              /* número de filósofos */

void filosofo(int i) {
    while(true) {
     pensar();         /* está pensando */
     coger_tenedor(i);        /* coge el tenedor izquierdo */
     coger_tenedor((i + 1) % N)       /* coge el tenedor derecho */
     comer();         /* come */
     dejar_tenedor(i);        /* deja el tenedro izquierdo */
     dejar_tenedor((i + 1 ) % N)      /* deja el tenedor derecho */
    }
}


/* SOLUCIÓN CON VECTORES Y SEMÁFOROS. SECCIÓN CRÍTICA */

#define N 5              /* número de filósofos */
#define IZQ (i - 1) % N            /* num del adyacente izq. de i */
#define DER (i + 1) % N             /* num del adyacente der. de i */
#define PENSANDO 0             /* está pensando */
#define HAMBRIENTO 1             /* está hambriento */
#define COMIENDO 2             /* está comiendo */

int estado[N];              /* vector estado de filósofos */
semaforo mutex = 1;             /* mutex para las secciones críticas */
semaforo s[N];              /* un semáforo por cada filósofo */

void filosofo(int i) {            /* i es el num de filosofo */
    while(true) {
     pensar();         /* está pensando */
     coger_tenedores();        /* obtiene 2 tenedores, se bloquea si
                no puede */
     comer();         /* está comiendo*/
     dejar_tenedores();        /* deja los 2 tenedores en la mesa */
    }
}

void coger_tenedores(int i) {
    wait(mutex);         /* entra en la seccion crítica */
    estado[i] = HAMBRIENTO;        /* el filósofo i tiene hambre */
    prueba(i);          /* intenta coger los dos tenedores */
    signal(mutex);          /* sale de la sección crítica */
    wait(s[i]);          /* se bloquea si no consiguió los
                dos tenedores */
}

void dejar_tenedores(int i) {
    wait(mutex);         /* entra en la sección crítica */
    estado[i] = PENSANDO;        /* el filósofo i ha dejado de comer */
    prueba(IZQ);          /* comprueba si el adyacente por la izq
                puede comer ahora */
```

```
    prueba(DER);          /* comprueba si el adyacente por la der
                puede comer ahora */
    signal(mutex);          /* sale de la sección crítica */
}

void prueba(int i) {
    if(estado[i] == HABRIENTO && estado[IZQ] != COMIENDO
      && estado[DER] != COMIENDO) {
        estado[i] = COMIENDO;     /* el filósofo i está comiendo */
        signal(s[i]);
    }
}
```