Problema (instruccions atòmiques). Suposeu que la vostra màquina té una instrucció *flip* atòmica definida tal com es mostra a continuació. Observeu que la funció *flip* permet generar de forma successiva els números 0, 1, 0, 1, etc.

Un company vostre us proposa una solució per al problema de la secció crítica per a **dos fils**.

Comenteu si la solució proposada funciona o no. Raoneu la vostra resposta.

NOTES:

- Suposeu una màquina amb un sol processador. Per tant, podeu obviar els problemes associats a multiprocessadors (barreres de memòria, etc.).
- Observeu que la funció *flip* realitza aquestes instruccions de forma atòmica: guarda a la variable *old* el valor actual de *lock*, modifica el valor de *lock* i retorna el valor anterior de *lock* guardat a la variable *old*.
- Les línies 13 a 14 corresponen al protocol d'entrada a la secció crítica, mentre que la línia 16 correspon al protocol de sortida de la secció crítica.

```
 1 flip(int *lock):
 2   <int old = *lock;
 3    *lock = (*lock + 1) % 2;
 4    return old;>
 5
 6 variable global:
 7    int lock = 0;
 8
 9 fil:
10    // variable local
11    int value;
12    while (true) {
13      value = flip(&lock);
14      while (value == 1) {};
15      // seccio critica
16      lock = 0;
17      // seccio no critica
18    }
19
```

Problema (instruccions atòmiques). Suposeu que la vostra màquina té dos fils que executen aquestes instruccions

```
variables globals:
  int x = 0;
  int y = 0;
  int z = 0;

fil1:
  x = 1;
  y = 2;

fil2:
  z = x + y;
  printf("El valor de z es %d\n",z);
```

L'objectiu és analitzar quin són els valors que es poden imprimir per pantalla

1. Indiqueu clarament la seqüència d'instruccions i canvis de context que s'ha d'executar perquè el resultat de z pugui ser 0 o 3.
2. Indiqueu clarament la seqüència d'instruccions i canvis de context que s'ha d'executar perquè el resultat de z pugui ser 1.
3. Indiqueu clarament la seqüència d'instruccions i canvis de context que s'ha d'executar perquè el resultat de z pugui ser 2.

Problema (seccions crítiques). El següent codi intenta resoldre el problema de la secció crítica per a **dos** fils

```
 1 variable global:
 2     int torn; // Inicialitzat a 1
 3
 4 fil 1/2:
 5     int i, j;
 6     i = 1 o 2 // respectivament, segons fil
 7     j = 2 o 1 // respectivament, segons fil
 8
 9     while (torn != i) {};
10     // seccio critica
11     torn = j;
12     return;
```

Les condicions són les següents: la variable global "torn" té valor inicial 1; la variable "i" té valor 1 en el fil 1 i valor 2 en el fil 2; la variable "j" té valor 2 en el fil 1 i valor 1 en el fil 2. Per simplificar, suposeu només un sol processador i ignoreu tots els temes relacionats amb la caché. Es demana:

1. Quin tipus d'espera es fa servir, activa o passiva? Quin és el punt d'entrada a la secció crítica? En quin punt se surt de la secció crítica? Comenta la resposta.
2. Comenteu un exemple d'execució del codi. En particular, el fil 1 vol entrar a la secció crítica i en surt; el fil 2 vol entrar a la secció crítica i en surt; el fil 2 vol tornar a entrar a la secció crítica; el fil 1 vol entrar a la secció crítica.
3. Resol aquest codi el problema de la secció crítica? En altres paraules, els dos fils poden ser a dins de la secció crítica? Es pot produir un *deadlock*? Té aquesta solució algun altre problema? Comenteu la resposta suposant que només hi ha dos fils.

Problema (multifil). Suposeu el codi mostrat en aquest full. Aquest codi s'executa diverses vegades tal com es mostra a continuació. Podeu explicar per què no s'imprimeixen els 10 missatges dels fils ?

```
$ ./programa
Soc el fil 0.
Soc el fil 1.
Soc el fil 8.
$ ./programa
Soc el fil 0.
Soc el fil 3.
$ ./programa
Soc el fil 0.
Soc el fil 1.
Soc el fil 2.
```

Codi font de l'aplicació

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <pthread.h>
 4
 5 #define NFILS  10
 6
 7 void *thr_fn(void *arg)
 8 {
 9   long int i;
10
11   i = (long int) arg;
12   printf("Soc el fil %ld.\n", i);
13   return((void *) 0);
14 }
15
16 int main(void)
17 {
18   pthread_t ntid[NFILS];
19   long int i;
20   int err;
21
22   for(i = 0; i < 10; i++) {
23     err = pthread_create(ntid+i, NULL, thr_fn, (void *) i);
24     if (err != 0) {
25       printf("no puc crear el fil %ld.\n", i);
26       exit(1);
27     }
28   }
29
30   exit(0);
31 }
```

Problema (multifil). Suposeu el codi i la sortida que es mostra en aquesta pàgina.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include <pthread.h>
 5
 6 pthread_t ntid[10];
 7
 8 void *thr_fn(void *arg)
 9 {
10   long int i;
11
12   i = (long int) arg;
13   printf("Fil creat amb id %d\n", i);
14
15   return ((void *) 0);
16 }
17
18 int main(void)
19 {
20   char *s;
21   long int i, err;
22
23   for(i = 0; i < 10; i++) {
24     err = pthread_create(&(ntid[i]), NULL, thr_fn, (void *) i);
25     if (err != 0) {
26       printf("no puc crear el fil\n"); exit(1);
27     }
28   }
29
30   for(i = 0; i < 10; i++) {
31     err = pthread_join(ntid[i], NULL);
32     if (err != 0) {
33       printf("no puc fer join al fil.\n"); exit(1);
34     }
35   }
36
37   return 0;
38 }
39
```

NOTA: continua a la següent pàgina...

Sortida:

```
Fil creat amb id 9
Fil creat amb id 8
Fil creat amb id 7
Fil creat amb id 6
Fil creat amb id 5
Fil creat amb id 4
Fil creat amb id 2
Fil creat amb id 1
Fil creat amb id 0
Fil creat amb id 3
```

A continuació modifiquem el codi de la següent forma (les modificacions respecte el codi anterior es mostren en **negreta**).

1. Per què la sortida no imprimeix números diferents per cada fil ? Comenta la teva resposta.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #include <pthread.h>
 5
 6 pthread_t ntid[10];
 7
 8 void *thr_fn(void *arg)
 9 {
10   long int i;
11
12   i = * ((long int *) arg);
13   printf("Fil creat amb id %d\n", i);
14
15   return ((void *) 0);
16 }
17
18 int main(void)
19 {
20   char *s;
21   long int i, err;
22
23   for(i = 0; i < 10; i++) {
24     err = pthread_create(&(ntid[i]), NULL, thr_fn, (void *) &i);
25     if (err != 0) {
26       printf("no puc crear el fil\n"); exit(1);
27     }
28   }
29
30   for(i = 0; i < 10; i++) {
31     err = pthread_join(ntid[i], NULL);
32     if (err != 0) {
33       printf("no puc fer join al fil.\n"); exit(1);
34     }
35   }
36
37   return 0;
38 }
```

Sortida:

Fil creat amb id 1
Fil creat amb id 2
Fil creat amb id 6
Fil creat amb id 9
Fil creat amb id 9
Fil creat amb id 9
Fil creat amb id 9
Fil creat amb id 7
Fil creat amb id 0
Fil creat amb id 5

Problema (multifil). Suposeu que es disposen de dues instruccions, lock(int *var) i unlock(int *var), per sincronitzar els fils. Les funcions utilitzen un paràmetre var (un punter a un sencer), que és la variable sobre la qual sincronitzen. El funcionament de lock i unlock és el següent

- La funció lock(int *var) comprova si *var és zero. En cas afirmatiu, posa *var a 1 i retorna de la funció lock i deixa entrar al fil a la secció crítica. En cas contrari (si *var és 1), el fil farà una espera activa fins que *var sigui 0.
- La funció unlock(int *var) posa *var a 0, de forma que si hi ha un fil esperant a lock podrà entrar a la secció crítica.

Les funcions lock i unlock estan correctament programades i per tant asseguren que només un fil podrà entrar a la secció crítica.

Un company us presenta dues implementacions diferents (functio_fil1 i functio_fil2) per sincronitzar múltiples fils. Li podeu dir si totes dues implementacions són correctes ? Comenteu la vostra resposta.

```
int a = 0;

void funcio_fil2(void *arg)
{
    while (1 == 1) {
      lock(&a);
      // secció crítica
      unlock(&a);

      // secció no crítica
   }
}
```

```
void funcio_fil1(void *arg)
{
    int a = 0;

    while (1 == 1) {
      lock(&a);
      // secció crítica
      unlock(&a);

      // secció no crítica
   }
}
```

Problema (múltiples fils). Un company vostre ha fet algunes proves amb el codi de múltiples fils que incrementa una variable. El codi original, disponible a les transparències, és el següent (el codi compila correctament)

```
#define ITERATIONS 1000

int a;

void *thread_fn(void *arg)
{
  int i;
  for(i = 0; i < ITERATIONS; i++)
    a++;

  return ((void *)0);
}

int main(void)
{
  pthread_t ntid[2];
  int i;

  a = 0;

  for(i = 0; i < 2; i++)
     pthread_create(&(ntid[i]), NULL, thread_fn, NULL);

  for(i = 0; i < 2; i++)
     pthread_join(ntid[i], NULL);

  printf("Valor d'a: %d\n", a);
}
```

El vostre company us pregunta:

1. Quina avantatge té, en un cas general, executar un programa amb múltiples fils encara que només hi hagi un sol processador a l'ordinador ? I si l'ordinador té més d'un processador ?
2. Per què, en executar aquest programa, el resultat de l'execució dóna cada cop un resultat diferent en comptes del resultat esperat, 2000 ?

Un cop heu fet experiments amb aquest programa comenceu a provar amb les funcions de bloqueig. El codi es mostra a la pàgina següent (es mostren en negreta els canvis respecte el codi anterior). El codi compila correctament i dóna sempre el resultat correcte.

```
#define ITERATIONS 1000

int a;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *thread_fn(void *arg)
{
  pthread_mutex_lock(&mutex);

  int i;
  for(i = 0; i < ITERATIONS; i++)
    a++;

  pthread_mutex_unlock(&mutex);
```

```
  return ((void *)0);
}

int main(void)
{
  pthread_t ntid[2];
  int i;

  a = 0;

  for(i = 0; i < 2; i++)
     pthread_create(&(ntid[i]), NULL, thread_fn, NULL);

  for(i = 0; i < 2; i++)
     pthread_join(ntid[i], NULL);

  printf("Valor d'a: %d\n", a);
}
```

El vostre company us pregunta:

3. Per què, tot i que l'ordinador en què s'executa aquest programa té dos processadors, no es reparteix el còmput (de forma paral·lela) entre els dos processadors ? Comenteu la vostra resposta.

Finalment canvieu el codi de la funció thread_fn a

```
void *thread_fn(void *arg)
{
  int i;
  for(i = 0; i < ITERATIONS; i++){
    pthread_mutex_lock(&mutex);
    a++;
    pthread_mutex_unlock(&mutex);
  }

  return ((void *)0);
}
```

El vostre company us pregunta:

4. Per què triga ara el programa tant en executar-se en comparació amb els codis mostrats anteriorment ? Comenteu la vostra resposta.

Problema (mutifil). Responeu a les següents preguntes. Suposeu que es desitja programar una aplicació amb múltiples fils d'execució.

1. Quines avantatges creieu que hi ha si es fan servir processos en comptes de fils ?
2. Quines desavantatges creieu que hi ha si es fan servir processos en comptes de fils ?
3. Té sentit utilitzar tècniques de comunicació inter-procés (pex canonades, fitxers, ...) en un programa multi-fil per comunicar els fils entre sí ? Comenteu la resposta.
4. Poden els processos compartir la memòria RAM entre sí per a la comunicació inter-procés ? Comenteu la vostra resposta.

Problema (multifil). Un company us presenta aquest codi com a exemple d'exclusió mútua. El codi crea múltiples fils i cada fil incrementa una variable global. El programa, però, no funciona correctament ja que a la línia 36 no s'imprimeix el resultat esperat.

1. Podeu indicar per què no funciona correctament el programa? Raoneu la vostra resposta.
2. Com s'hauria de modificar per què doni el resultat correcte? Raoneu la vostra resposta.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <pthread.h>
 4
 5 #define MAXFILS 2
 6 #define NITERS  100000
 7
 8 pthread_t ntid[MAXFILS];
 9 int a = 0;
10
11 void *thr_fn(void *arg)
12 {
13   int i;
14   pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
15
16   pthread_mutex_lock(&mutex);
17   for(i = 0; i < NITERS; i++)
18     a++;
19   pthread_mutex_unlock(&mutex);
20
21   return ((void *) 0);
22 }
23
24 int main(void)
25 {
26   int i;
27
28   for(i = 0; i < MAXFILS; i++) {
29     pthread_create(ntid+i, NULL, thr_fn, NULL);
30   }
31
32   for(i = 0; i < MAXFILS; i++) {
33     pthread_join(ntid[i], NULL);
34   }
35
36   printf("%d\n", a);
37   return 0;
38 }
```

Problema (espera activa). A sota es mostra una implementació d'una barrera. La barrera és útil per a programació paral·lela: els fils processen una sèrie de dades i en acabar criden a la funció barrera per esperar que els altres fils acabin amb la seva part de processament. Així que el darrer fil crida a la funció barrera els tots fils continuen executant amb el següent bloc de dades.

Es demana:

1. Explica quin és el comportament esperat de la funció barrera. En quin punt s'esperen els fils per continuar executant? Quan i com continuen executant els fils?
2. Funciona aquest algorisme en tots el casos? O hi ha algun cas en què l'algorisme no funcioni? Comenta la teva resposta.

```
1 variables globals:
2     int comptador = N;
3
4 barrera:
5     int pos;
6     <pos = comptador; comptador--;>
7     if (pos == 1)
8           <comptador = N>;
9     else {
10          while (comptador != 0) {};
11    }
12    return;
13
14 fil:
15    while (true) {
16          // processa dades
17          barrera();
18    }
```

Problema (semàfors). Suposeu un pont d'un sol carril. Els cotxes que arriben del sud i del nord per la carretera arriben al pont. Els cotxes que van en el mateix sentit poden creuar el pont al mateix temps, però els cotxes no poden creuar el pont en sentit oposat al mateix temps. Es proposa aquí una solució per al problema per a un sol cotxe (és a dir, que al pont només hi pugui haver un cotxe).

1. Comenteu la solució proposada. Soluciona el codi el problema de l'exclusió mútua ?
2. Suposeu que el pont és de 2 carrils. Com modificaríeu el codi perquè hi circulin a tot estirar dos cotxes encara que no vagin en el mateix sentit ?
3. Suposeu que el pont és de 1 carril. Comenteu com modificaríeu el codi perquè puguin passar diversos cotxes a la vegada pel pont, sempre en el mateix sentit. No cal indicar el codi exacte, sinó les idees de base per aconseguir-ho.

NOTA: Per solucionar aquest problema **no** es poden fer servir monitors.

```
1  variables globals:
2     sem pont_buit = 1;
3
4  cotxes que venen del sud:
5     sem_down(pont_buit);
6     // creuar pont
7     sem_up(pont_buit);
8
9
10 cotxes que venen del nord:
11    sem_down(pont_buit);
12    // creuar pont
13    sem_up(pont_buit);
```

Problema (semàfors). Considerem una versió modificada de la instrucció *tee* del sistema operatiu Linux. Aquesta comanda s'executa invocant

```
$ tee nomfitxer
```

Aquesta comanda llegeix les dades de l'entrada estàndard i escriu aquestes dades al fitxer nomfitxer. És a dir, aquesta comanda realitza una còpia de l'entrada estàndard al fitxer nomfitxer.

Se us planteja escriure un programa paral·lel en pseudocodi que utilitzi dos fils d'execució: un fil per llegir les dades de l'entrada estàndard i un per a escriure les dades al fitxer nomfitxer.

Qüestions:

1. Quin paradigma dels vistos a classe és el que s'ha d'utilitzar per resoldre el problema ?
2. Indiqueu el pseudocodi que implementa la funcionalitat demanada utilitzant semàfors. Tingueu en compte que caldrà sincronitzar correctament els fils entre sí.
3. Suposeu que la comanda permet especificar múltiples fitxers a la línia de comandes tal com s'indica a continuació:

   ```
   $ tee nomfitxer1 nomfitxer2 normfitxer3 ... normfitxerN
   ```

   En aquest cas l'entrada estàndard s'escriu a cadascun dels fitxers indicats a la línia de comandes. Suposeu que es crea un fil per a cada fitxer i un pel fil que llegeix les dades. Indiqueu el pseudocodi que implementa la funcionalitat demanada utilitzant semàfors.

NOTA: Per resoldre aquest problema podeu utilitzar un pseudocodi similar a l'utilitzat a les transparències de teoria. Podeu suposar que els fils han estat creats correctament.

Problema (semàfors). El departament on treballeu té un lavabo unisex. Pot ser utilitzat tant per homes i dones però no al mateix temps. Es proposa a continuació una solució al problema fent servir semàfors.

1. Comenteu la solució proposada. Soluciona el codi el problema de l'exclusió mútua ? Suposeu que no hi ha límit en el nombre de persones que hi pugui haver al lavabo.
2. Per a què serveixen els semàfors mutexD, mutexH i lababo? Comenteu la vostra resposta.
3. Té algun defecte aquesta solució ? Comenteu i raoneu la vostra resposta.
4. (Difícil) Comenteu com modificaríeu el codi perquè a tot estirar hi pugui haver quatre persones al lavabo en un mateix instant de temps. Assegureu-vos que no provoca un deadlock.

```
variables globals:
 1   int dones = 0, homes = 0;
 2   sem_t mutexD (= 1), mutexH (= 1), lavabo (= 1);
 3
 4 dones:
 5   sem_wait(&mutexD);
 6   dones = dones + 1;
 7   if (dones == 1) sem_wait(&lavabo);
 8   sem_post(&mutexD);
 9   // anar al lavabo
10   sem_wait(&mutexD);
11   dones = dones - 1;
12   if (dones == 0) sem_post(&lavabo);
13   sem_post(&mutexD);
14
15 homes:
16   sem_wait(&mutexH);
17   homes = homes + 1;
18   if (homes == 1) sem_wait(&lavabo);
19   sem_post(&mutexH);
20   // anar al lavabo
21   sem_wait(&mutexH);
22   homes = homes - 1;
23   if (homes == 0) sem_post(&lavabo);
24   sem_post(&mutexH);
```

Problema (semàfors). Suposeu un sistema en què només es disposa dels semàfors binaris. L'objectiu és implementar un semàfor general fent servir semàfors binaris.

1. Proposeu una solució al problema. Implementeu l'algorisme d'un semàfor general fent servir semàfors binaris. Us podeu inspirar en l'algorisme dels lectors-escriptors per fer una proposta. Utilitzeu la variable sencera "s" com a variable associada al semàfor general. El valor de la variable "s" està inicialitzat a M, igual que es fa a les transparències.
2. Comenteu la solució proposada. En particular, poden accedir múltiples fils al mateix temps a la variable "s"? Comenteu i raoneu la resposta.
3. Si la variable "s" està inicialitzada a M, què passa si per exemple M+3 fils intenten entrar a la secció crítica? Podran entrar tots els fils a la secció crítica? Indica on es quedaran esperant/dormint els fils que no poden entrar. Comenta la resposta.
4. A partir de la resposta a la pregunta 3, indica com aniran entrant els fils que s'esperen per entrar a la secció crítica a mesura que els fils que són a dins en surten.

NOTA: Es donen alguns detalls dels semàfors binaris

- El semàfor binari només pot prendre els valors 0 i 1.
- La funció sem_wait només pot entrar a la secció crítica si el seu valor és 1.
- La funció sem_post posa el valor del semàfor a 1. Crides successives al sem_post no tenen cap efecte. El valor del semàfor no canvia.

Problema (semàfors). Suposeu el següent codi utilitzat per a realitzar un càlcul en paral·lel.

```
1 variables globals:
2     int a = 0;
3     int N = número de fils;
4     sem_t semafor (= 1);
5
6 fil:
7     int i = 0;
8     sem_wait(&semafor);
9     for(i = 0; i < 1000; i += N)
10          a++;
11    sem_post(&semafor);
```

Es demana

1. Aquest codi s'executa en un ordinador amb múltiples processadors. Per què no es reparteix el còmput de forma paral·lela entre els múltiples processadors? Comenta com s'està executant el codi.

Es modifica el codi tal com es mostra a continuació

```
1 variables globals:
2     int a = 0;
3     int N = número de fils;
4     sem_t semafor (= 1);
5
6 fil:
7     int i = 0;
8     for(i = 0; i < 1000; i += N) {
9           sem_wait(&semafor);
10          a++;
11          sem_post(&semafor);
12    }
```

2. En aquest exemple el codi triga molt en executar-se, inclús respecte el cas N = 1. Per què?
3. Com modificaríeu el codi per tal de fer un codi eficient? Comenteu la vostra resposta per a un cas general, és a dir, que la solució no estigui adaptada específicament per a l'exemple proposat.

Problema (semàfors). Suposeu un sistema en què només es disposa dels semàfors binaris. L'objectiu és implementar un semàfor general fent servir semàfors binaris.

1. Proposeu una solució al problema. Us podeu basar en l'algorisme dels lectors-escriptors per fer una proposta. Utilitzeu la variable sencera "s" com a variable associada al semàfor general. El valor de la variable "s" està inicialitzat a M, igual que es fa a les transparències.
2. Comenteu la solució proposada. En particular, poden accedir múltiples fils al mateix temps a la variable "s"? Comenteu i raoneu la resposta.
3. Si la variable "s" està inicialitzada a M, què passa si per exemple M+3 fils intenten entrar a la secció crítica? Podran entrar tots els fils a la secció crítica? En cas afirmatiu, indica per què. En cas negatiu, indica on es quedaran esperant/dormint els fils que no poden entrar. Comenta la resposta.

Problema (monitors). En una granja un pagès té animals separats per mascles i femelles. Tots dos han d'accedir al mateix estable per poder menjar però no hi poden coincidir al mateix temps. És a dir, els mascles no poden accedir a l'estable per menjar si hi ha femelles menjant, i a la inversa, les femelles no poden accedir a l'estable si hi ha mascles menjant. El pagès té un sistema automàtic que controla l'accés a l'estable. Es proposa a sota l'algorisme de controla l'accés a l'estable.

1. Per què serveixen les variables n1 i n2 ? Què indiquen ?
2. Soluciona la proposta el problema proposat ? Comenteu la vostra resposta.
3. En cas que només hi hagi un únic mascle i una única femella, es pot substituir el “while” de la línia 8 i de la línia 23 per un “if” ? Comenteu la vostra resposta.

```
 1 variables globals:
 2    monitor_mutex mutex;
 3    cond cua;
 4    int n1 = 0, n2 = 0;
 5
 6 femelles:
 7    monitor_lock(mutex);
 8    while (n2 != 0) {
 9       wait(cua, mutex);
10    }
11    n1++;
12    monitor_unlock(mutex);
13    // menjar
14    monitor_lock(mutex);
15    n1--;
16    if (n1 == 0) {
17       signalAll(cua);
18    }
19    monitor_unlock(mutex);
20
21 mascles:
22    monitor_lock(mutex);
23    while (n1 != 0) {
24       wait(cua, mutex);
25    }
26    n2++;
27    monitor_unlock(mutex);
28    // menjar
29    monitor_lock(mutex);
30    n2--;
31    if (n2 == 0) {
32       signalAll(cua);
33    }
34    monitor_unlock(mutex);
35
```

Problema (monitors). Suposeu el mateix enunciat del problema 2. Considerem una versió modificada de la instrucció *tee* del sistema operatiu Linux. Aquesta comanda s'executa invocant

```
$ tee nomfitxer1 nomfitxer2 normfitxer3 ... normfitxerN
```

Aquesta comanda llegeix les dades de l'entrada estàndard i l'escriu a cadascun dels fitxers indicats a la línia de comandes. Es demana escriure un programa paral·lel en pseudocodi i que utilitzi un fil d'execució per a cada fitxer a escriure i un fil per llegir les dades de l'entrada estàndard.

Indiqueu clarament el pseudocodi que implementa la funcionalitat demanada utilitzant monitors. Tingueu en compte que caldrà sincronitzar correctament els fils entre sí.

NOTA: Per resoldre aquest problema podeu utilitzar un pseudocodi similar a l'utilitzat a les transparències de teoria. Podeu suposar que els fils han estat creats correctament.

Problema (monitors). Es presenta a continuació un codi perquè una clau sigui *reentrant*, és a dir, que un fil pugui cridar múltiples vegades a una funció de lock sense que es produeixi un *deadlock*. Per entrar a la secció crítica els fils han de cridar a lock_reentrant. Per sortir-ne, han de cridar a unlock_reentrant. El codi és correcte i ha estat provat satisfactòriament en llenguatge C.

La funció *fil_id*() (en negreta) retorna l'identificador del fil, un valor sencer positiu. Cada fil té assignat un identificador diferent. La variable *id* s'utilitza per saber quin fil ha agafat la clau. Observar que *id* s'inicialitza a -1 (línia 5 i 25) per indicar que cap fil té la clau.

L'algorisme es basa en la següent idea: en cridar un fil a la funció *lock_reentrant* es comprova primer si la clau ja està adquirida abans pel mateix fil (línia 9). En cas que un altre fil tingui adquirida la clau el fil s'adorm a la línia 11.

Contesteu a les següents preguntes:

1. Quina és la utilitat de la variable *comptador* ? Comenteu la vostra resposta.
2. Comenta esquemàticament el funcionament de les funcions *lock_reentrant* i *unlock_reentrant* amb un exemple. Per això suposeu una aplicació amb 2 fils amb id = -1 i que succeeix el següent temporalment: el fil 0 crida a lock_reentrant; el fil 1 crida a lock_reentrant; el fil 0 crida a lock_reentrant; el fil 0 crida a unlock_reentrant; el fil 0 crida a unlock_reentrant.
3. Suposeu que a la línia 26 del codi es posa un “signal” en comptes d'un “broadcast”. Es pot canviar aleshores el “while” de la línia 10 per un “if” ? Comenteu la vostra resposta suposant que hi pot haver qualsevol nombre de fils.

```
 1 variables globals:
 2   pthread_mutex_t mutex;
 3   pthread_cond_t cond;
 4   int comptador = 0;
 5   int id = -1;
 6
 7 lock_reentrant:
 8   pthread_mutex_lock(&mutex);
 9   if (id != fil_id()) {
10      while (id != -1) {
11          pthread_cond_wait(&cond, &mutex);
12      }
13      id = fil_id();
14      comptador = 1;
15   } else {
16      comptador++;
17   }
18   pthread_mutex_unlock(&mutex);
19   return;
20
21 unlock_reentrant:
22   pthread_mutex_lock(&mutex);
23   comptador--;
24   if (comptador == 0) {
25      id = -1;
26      pthread_cond_broadcast(&cond);
27   }
28   pthread_mutex_unlock(&mutex);
29   return;
```

Problema (monitors). El problema de "la barberia" és un dels problemes clàssics de la sincronització de fils. Es presenta a continuació una versió simplificada del problema original. La barberia és una botiga en què hi ha una sala d'espera, amb infinites cadires, i la sala del barber on el client és atès. Les condicions de funcionament de la barberia són les següents: a) Si no hi ha cap client, el barber se'n va a dormir (línies 12-13 del codi), b) Si el barber està ocupat, el client espera (línies 23-24). Se suposa que el barber arriba abans a la botiga que els clients.

```
 1 global:
 2     int clients = 0;          // Nombre de clients
 3     boolean ocupat = false;  // Permet saber si el barber esta ocupat
 4
 5     mutex clau;
 6     cond cond_barber, cond_clients;
 7
 8 barber: // Hi ha nomes un barber. El barber obre la botiga.
 9     while (true) {
10         lock(clau)
11         ocupat = false
12         if (clients == 0) // Si no hi ha clients, el barber s'adorm
13             wait(cond_barber, clau)
14         signal(cond_clients)
15         clients--
16         unlock(clau)
17         tallar_cabell()
18     }
19
20 client: // El client crida aquesta funcio quan es vol tallar el cabell
21     lock(clau)
22     clients++
23     while (ocupat) // Si el barber esta ocupat, el client s'adorm
24         wait(cond_clients, clau)
25     signal(cond_barber)
26     ocupat = true
27     unlock(clau)
28     em_tallen_el_cabell()
29     return
```

Es demana el següent:

1. Quin és l'objectiu de protegir amb la variable "clau" determinades parts del codi al client i barber? Comenteu i raoneu la resposta per aquest codi en particular.
2. Observar que es fa servir la mateixa variable "clau" per protegir el codi del barber i del client. Creieu que es poden protegir amb claus diferents el codi del barber i del client? Comenteu i raoneu la resposta per aquest codi en particular.
3. Comenteu el funcionament el codi amb un exemple: el barber obre la botiga i ho hi ha cap client; un client A entra a la botiga; mentre el client A és atès els clients B i C entren a la botiga; el barber acaba d'atendre el client A i vol passar a atendre els clients que queden a la botiga. Suposeu que mentrestant no arriben nous clients.
4. Observar que a la línia 12 es protegeix el "wait" amb un "if" en comptes d'un "while". Raoneu per què no fa falta un "while".
5. De forma similar, observar que a la línia 23 es protegeix el "wait" amb un "while" en comptes d'un "if". Raonar per què no es pot posar un "if". Es pot produir algun problema? Comenteu i raoneu la resposta per aquest codi en particular.
6. Suposeu que a la botiga hi ha múltiples barbers en comptes d'un únic barber. Com modificaríeu el codi per tal d'adaptar-ho a aquest cas? A la pàgina següent se us torna a

mostrar el codi que teniu en aquest full. Utilitzeu-lo per indicar les modificacions que hi faríeu. Comenteu la vostra proposta.

Utilitzeu aquest codi per respondre al punt 6 del problema.

```
1  global:
2      int clients = 0;          // Nombre de clients
3      boolean ocupat = true;    // Permet saber si el barber esta ocupat
4
5      mutex clau;
6      cond cond_barber, cond_clients;
7
8  barber: // Hi ha nomes un barber. El barber obre la botiga.
9      while (true) {
10         lock(clau)
11         ocupat = false
12         if (clients == 0) // Si no hi ha clients, el barber s'adorm
13             wait(cond_barber, clau)
14         signal(cond_clients)
15         ocupat = true
16         unlock(clau)
17         tallar_cabell()
18     }
19
20 client: // El client crida aquesta funcio quan es vol tallar el cabell
21     lock(clau)
22     clients++
23     while (ocupat) // Si el barber esta ocupat, el client s'adorm
24         wait(cond_clients, clau)
25     signal(cond_barber, clau)
26     clients--
27     unlock(clau)
28     em_tallen_el_cabell()
29     return
```

Problema (monitors). Les funcions lock i unlock dels monitors serveixen per bloquejar l'accés a una secció crítica. Aquestes funcions tenen però, un (petit) defecte: en cas que hi hagi múltiples fils esperant per entrar a la secció crítica no sabem quin fil hi podrà accedir primer així que el fil que és a dins faci la crida a unlock.

L'algorisme que es presenta a sota soluciona el problema plantejat i funciona correctament. La idea és similar al d'una peixateria en un supermercat: en arribar el client a la peixateria ha d'agafar un tiquet; aquest tiquet li indica quan és el seu torn; així que a la pantalla aparegui el seu número serà atès (és a dir, accedirà a la secció crítica).

Es demana:

1. Quina és la utilitat de les variables tiquet i torn_actual? Comenteu la resposta i feu parel·lisme amb l'exemple de la peixateria si us serveix.
2. Quina és la utilitat de la variable el_meu_tiquet i per què s'ha declarat de forma local? Comenteu la vostra resposta.
3. Comenta molt esquemàticament el funcionament de l'algorisme amb un exemple. En particular, suposeu que l'aplicació té tres fils (1, 2, i 3) i que els fils criden a lock (línia 6) en aquest ordre: primer el fil 1, després el fil 3, i després el fil 2. Comenteu què succeeix fins que el fil 2 entra a la secció crítica.

```
 1 variables globals:
 2     pthread_cond_t cond;
 3     pthread_mutex_t mutex;
 4     unsigned long torn_actual = 0, tiquet = 0;
 5
 6 lock:
 7     unsigned long el_meu_tiquet;
 8
 9     pthread_mutex_lock(&mutex);
10     el_meu_tiquet = tiquet;
11     tiquet++;
12     while (el_meu_tiquet != torn_actual)
13     {
14         pthread_cond_wait(&cond, &mutex);
15     }
16     pthread_mutex_unlock(&mutex);
17     return;
18
19 unlock:
20     pthread_mutex_lock(&mutex);
21     torn_actual++;
22     pthread_cond_broadcast(&cond);
23     pthread_mutex_unlock(&mutex);
24     return;
```

Problema (multifil). (Aquest exercici forma part d'un codi de les pràctiques de l'assignatura d'aquell any) Suposeu l'enunciat de la pràctica 4 d'aquesta assignatura, en què s'han de controlar una sèrie de fils “secundaris” des del fil principal. Els fil principal desperta a una sèrie de fils que estan dormint. Els fils executen i realitzen un determinat càlcul. En finalitzar, s'adormen i desperten al fil principal.

Es mostra aquí el codi dels fils “secundaris”

```
 1 #define N_THREADS 10
 2
 3 int comptador = N_THREADS;
 4 pthread_mutex_t mutex  = PTHREAD_MUTEX_INITIALIZER;
 5 pthread_cond_t  cond1  = PTHREAD_COND_INITIALIZER;
 6 pthread_cond_t  cond2  = PTHREAD_COND_INITIALIZER;
 7
 8 void *thr_fn(void *arg)
 9 {
10   while (1)
11   {
12     pthread_mutex_lock(&mutex);
13     comptador--;
14     if (comptador == 0)
15     {
16       comptador = N_THREADS;
17       pthread_cond_signal(&cond1);
18     }
19     pthread_cond_wait(&cond2, &mutex);
20     pthread_mutex_unlock(&mutex);
21
22     /* PROCESSO LES DADES*/
23   }
24
25   return ((void *)0);
26 }
```

Suposeu que els fils “secundaris” han estat creats correctament i estan tots dormint a la línia 19. A continuació es proposen 3 solucions per controlar, des del fil principal, els fils “secundaris”. El fil principal ha de despertar als fils “secundaris” i posar-se a dormir.

Solució 1.

```
pthread_cond_broadcast(&cond2);
pthread_cond_wait(&cond1, &mutex);
```

Solucio 2.

```
pthread_cond_broadcast(&cond2);
pthread_mutex_lock(&mutex);
pthread_cond_wait(&cond1, &mutex);
pthread_mutex_unlock(&mutex);
```

Solució 3.

```
pthread_mutex_lock(&mutex);
pthread_cond_broadcast(&cond2);
pthread_cond_wait(&cond1, &mutex);
pthread_mutex_unlock(&mutex);
```

Comenteu cadascuna de les tres solucions.

1. Quina o quines són solucions correctes per implementar l'algorisme de sincronització ? Comenteu la vostra resposta.
2. En cas que la solució no funcioni, indiqueu un exemple que resulti en una execució incorrecta.

Problema (semàfors i monitors). Una biblioteca té 10 espais d'estudi idèntics preparats per ser utilitzats per un sol estudiant a la vegada. Per evitar disputes, els estudiants han d'anar al mostrador per demanar l'ús d'un d'aquests espais. Quan un estudiant finalitza d'utilitzar un d'aquests espais ha de tornar al mostrador i indicar que un dels espais ha quedat lliure. Si no hi ha espais lliures, els estudiants han d'esperar al mostrador fins que algú deixa lliure un espai.

El bibliotecari del mostrador no sap quin dels 10 espais està ocupat o qui l'està utilitzant. Només sap el nombre d'espais que hi ha lliures. Quan un estudiant demana un espai, el bibliotecari redueix en 1 el nombre d'espais lliures. Quan un estudiant allibera un espai, l'incrementa.

Respon a les següents preguntes:

1. En aquest problema, què representa cada fil ? Els estudiants, els espais d'estudi, o el bibliotecari ? Comenta la teva resposta.
2. Indica la solució al problema fent servir pseudo-codi i/o codi C i fent servir **semàfors**. Només cal que mostreu el codi dels fils per accedir als espais, no cal que mostreu l'algorisme per seleccionar l'espai d'estudi (indiqueu simplement "Agafar aula d'estudi lliure" per agafar un espai o "Deixar aula d'estudi" per alliberar l'espai), Tampoc cal que mostreu el codi per crear els fils. Per respondre aquesta pregunta podeu utilitzar un pseudo-codi similar a l'utilitzat a les transparències de teoria.
3. Comenta, amb l'ajut del codi, com es controla l'entrada i sortida d'estudiants a les aules d'estudi. En particular, comenta què passa si un estudiant vol agafar un espai i tots els espais estan ocupats. Comenta també com s'assabenten els estudiants que esperen al mostrador si un estudiant allibera un espai.
4. Suposeu que hi ha un vector L (L de Lliure) de sencers de mida 10, L[0] a L[9]. Una aula x està ocupada si L[x] és 0 i està lliure si L[x] és 1. Suposeu que L s'inicialitza amb tots els seus valors a 1. Indiqueu l'algorisme requerit per seleccionar i alliberar un espai d'estudi, tenint en compte que múltiples fils poden intentar executar aquest algorisme a la vegada. Utilitzeu **semàfors** per sincronitzar els fils correctament.
5. Indiqueu la solució al punt 2 fent servir **monitors**. Comenta, amb l'ajut del codi, com es controla l'entrada i sortida d'estudiants a les aules d'estudi. En particular, comenta què passa si un estudiant vol agafar un espai i tots els espais estan ocupats. Comenta també com s'assabenten els estudiants que esperen al mostrador si un estudiant allibera un espai.
6. Indiqueu la solució al punt 4 fent servir **monitors**.

Problema (semàfors i monitors). Una empresa desitja implementar un servidor web amb múltiples fils. L'esquema bàsic de funcionament d'aquest servidor web és el següent: al servidor s'executen 1 fil principal i N fils secundaris. El fil principal del servidor escolta les peticions de connexió de clients (és a dir,usuaris que volen veure una pàgina web del servidor). A la petició de connexió s'inclou informació sobre la pàgina web que l'usuari desitja veure. En produir-se una petició de connexió el fil principal "redirigeix" aquesta petició a un dels fils secundaris i torna de seguida a escoltar peticions de connexió. El fil secundari processa la petició responent a l'usuari amb la pàgina web demanada.
Es demana escriure en pseudo-codi l'algorisme de sincronització dels fils. Podeu utilitzar les següents funcions

- h = escoltar_peticio() per escoltar una petició de connexió d'un client. La funció retorna tota la informació necessària a la variable h. Podeu suposar que la variable h és de tipus Peticio.
- respondre_peticio(h) per respondre a una petició d'un usuari.

Responeu a les següents preguntes

1. Indiqueu la solució al problema fent servir **semàfors**. Només cal que mostreu el codi per sincronitzar els fils entre sí, no cal que mostreu el codi per crear els fils.
2. Comenteu, fent servir la solució anterior, com se sincronitzen els fils entre sí. En particular, comenteu com "redirigeix" el fil principal una petició a un fil secundari ? Què passa si tots els fils secundaris estan ocupats responent a usuaris i arriba una nova petició de connexió al fil principal ?
3. Indiqueu la solució al problema fent servir **monitors**. Només cal que mostreu el codi per sincronitzar els fils entre sí, no cal que mostreu el codi per crear els fils.
4. Suposeu que els fils secundaris han d'accedir a una base de dades compartida en el moment de respondre la petició. Comenteu com gestionaríeu l'accés a la base de dades per evitar deixar-la inconsistent.