Sistemes Operatius II
Avaluació contínua – Parcial part 2 – 17 de desembre del 2015

Nom i Cognoms: _____

En total hi ha 4 problemes. Es poden utilitzar apunts però no es pot programar el codi proposat. Només cal respondre breument a les preguntes (no escriviu un llibre!).

Problema 1 (2.5 punts, intruccions atòmiques). Suposeu que la vostra màquina **un únic processador**. Heu dissenyat un programa amb 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'han d'executar perquè el resultat de z pugui ser 0 o 3.

Suposem que les variables x, y i z estan inicialitzades a 0. Perquè el resultat de z pugui ser 0 cal que s'executi la suma z = x + y del fil 2 abans que s'executin les inicialitzacions del fil 1.

Perquè el resultat de z pugui ser 3 cal que s'executin les inicialitzacions del fil 1 abans que s'executi la suma del fil 2.

2. Indiqueu clarament la seqüència d'instruccions i canvis de contest que s'han d'executar perquè el resultat de z pugui ser 1.

De forma similar a abans, suposem que les variables x, y i z estan inicialitzades a 0. A continuació es mostra l'ordre en què s'han executar les instruccions perquè el resultat pugui ser 1.

Fil 1	Fil 2
Assignació x = 1	
Canvi de context	
	Operació z = x + y
	Impressió del resultat per pantalla
	Canvi de context
Assignació y = 2	

3. (Difícil, deixeu-la pel final si no surt) Indiqueu clarament la seqüència d'instruccions i canvis de context que s'han d'executar perquè el resultat de z pugui ser 2.

De forma similar a abans, suposem que les variables x, y i z estan inicialitzades a 0. A continuació es mostra l'ordre en què s'han executar les instruccions perquè el resultat pugui ser 2. La idea per aconseguir-ho es basa en el fet que l'operació z = x + y no és atòmica i, per tant, necessita múltiples instruccions per ser executada.

Fil 1	Fil 2
	Carregar en un registre el valor de x (val 0).
	Canvi de context
Assignació x = 1 (es desa a memòria)	
Assignació y = 2 (es desa a memòria)	
Canvi de context	
	Carregar en un registre el valor de y (val 2)
	Operació z = x + y = 0 + 2
	Impressió del resultat per pantalla

Hi ha una altra solució proposada per companys vostres i que ha sigut considerada vàlida en aquest apartat. La forma d'aconseguir-ho és que el compilador re-ordeni les instruccions del fil 1, de forma que primer s'executi l'assignació y = 2 i després l'assignació x = 1. Aconseguir que el resultat sigui igual a 2 es fa de forma similar a l'apartat 2 d'aquest problema.

Problema 2 (2.5 punts, 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 no 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).

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

```
1  variables globals:
2     sem_t pont_buit (= 1);
3
4  cotxe que vé del sud:
5     sem_wait(&pont_buit);
6     // creuar pont
7     sem_post(&pont_buit);
8
9
10 cotxe que vé del nord:
11    sem_wait(&pont_buit);
12    // creuar pont
13    sem_post(&pont_buit);
```

1. Comenteu la solució proposada. Soluciona el codi el problema de l'exclusió mútua?

El codi soluciona el problema de l'exclusió mútua. Per això s'utilitza el semàfor pont_buit que s'inicialitza a 1. Si les zones crítiques es protegeixen bé, només es permetrà l'accés d'un cotxe al pont. Suposem doncs que un cotxe ve del sud. Farà un sem_wait i entrarà al pont. Mentrestant, si un altre cotxe que arriba del nord vol creuar-lo haurà de fer també un sem_wait. En aquest cas aquest cotxe es quedarà adormir al sem_wait. Quan el cotxe que és al pont faci sem_post, el cotxe que està adormit al sem_wait es despertarà i podrà creuar el pont. Cap altre cotxe podrà creuar-lo fins que aquest no faci sem_post.

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?

Si permetem que hi circulin dos cotxes, l'únic que cal fer es inicialitzar el semàfor pont_buit a 2. D'aquesta forma en tot moment podrà haver-hi un màxim de dos cotxes a la secció crítica. Hi podrà haver dos cotxes creuant en el mateix sentit així com dos cotxes creuant en sentit oposat.

Alguns companys han proposat fer servir dos semàfors. Per exemple, alguns han proposat fer servir un semàfor per a cada sentit. Aquesta solució no és vàlida doncs restringeix que només hi pugui haver dos cotxes en cas que aquests circulin en sentit oposat. Però no permet que hi hagi dos cotxes circulant en el mateix sentit. Altres companys han proposat fer servir solucions basades en consultar (fent servir un "if") quin dels dos semàfors està lliure. Aquesta solució tampoc és vàlida ja que no hi ha cap funció que permeti saber el valor d'un semàfor. Les dues úniques funcions que hi ha disponibles per manipular un semàfor són sem_wait i sem_post (i la funció d'inicialització).

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.

Si el pont és d'un sol carril, es pot aconseguir que diversos cotxes creuin el pont a la vegada fent servir l'algorisme dels lectors tant pels cotxes del sud com pels cotxes del nord. Pels cotxes del sud l'algorisme és aquest

```
variables globals:
      sem_t pont_buit (= 1), cotxes_nord  (= 1), cotxes_sud (= 1);
      int sud = 0, nord = 0;

cotxe que ve del sud:
      sem_wait(&cotxes_sud);
      sud = sud + 1;
      if (sud == 1) sem_wait(&pont_buit);
      sem_post(&cotxes_sud);
      // creuar pont
      sem_wait(&cotxes_sud);
      sud = sud - 1;
      if (sud == 0) sem_post(&pont_buit);
      sem_post(&cotxes_sud);
```

L'algorisme pels cotxes del nord és equivalent al dels cotxes del sud canviant les variables corresponents. Aquest algorisme té el defecte que si hi ha cotxes que venen del sud, els cotxes del nord s'hauran d'esperar fins que no hi hagi cap cotxe del sud creuant el pont.

Hi ha una altra solució proposada per un company vostre i que és d'interès mencionar aquí. La solució "simula" una variable condicional d'un monitor fent servir semàfors. La proposta es mostra a continuació.

```
variables globals:
      sem_t sem_var (= 1);
      int counter;
      int direccio;

cotxes que volen creuar el pont (siguin del nord o del sud):
      sem_wait(&sem_var);
      while (direccio != my_dir and counter != 0) {
            sem_post(&sem_var);
            sem_wait(&sem_var);
      }
      if (direccio != my_dir) direccio = my_dir;
      counter++;
      sem_post(&sem_var);

cotxes que volen sortir del pont (siguin del nord o del sud):
      sem_wait(&sem_var);
      counter--;
      sem_post(&sem_var);
```

Se suposa que cada fil té una variable local "my_dir" que indica la seva procedència (del sud o del nord).

Problema 3 (2.5 punts, 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 continuació soluciona el problema plantejat i funciona correctament. La idea està basada en assignar a cada fil un número (o tiquet). Un fil podrà entrar a la secció crítica així que el torn sigui igual al seu tiquet.

Per entrar a la secció crítica el fil ha de cridar a la funció lock_tiquet; per sortir-ne ha de cridar a unlock_tiquet. Observar que al codi suposa se suposa que els fils no necessàriament estan adormits al wait (línia 14) en l'ordre en què hi arriben. És per això que es fa un broadcast a la línia 22 i no pas un signal.

```
 1 variables globals:
 2     pthread_cond_t cond;
 3     pthread_mutex_t mutex;
 4     unsigned long torn_actual = 0, tiquet = 0;
 5
 6 lock_tiquet:
 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_tiquet:
20      pthread_mutex_lock(&mutex);
21      torn_actual++;
22      pthread_cond_broadcast(&cond);
23      pthread_mutex_unlock(&mutex);
24      return;
```

Es demana:

1. Quina és la utilitat de les variables tiquet i torn_actual? Comenteu la resposta.

La variable "tiquet" indica en tot moment el següent número de torn que se li assignarà a un fil (assignació que es fa a la línia 10 i que va seguit d'un increment de "tiquet" per donar-li el següent número a un altre fil).

La variable "torn_actual" indica el fil que està executant en cada moment. La variable augmenta quan un fil acaba de fer unlock. Quan es fa el broadcast, els fils que estan adormits a la línia 14 es desperten i comproven si és el seu torn. Només continuarà executant aquell fil a qui li toqui executar en aquell moment. La resta es tornaran a dormir.

2. Quina és la utilitat de la variable el_meu_tiquet i per què s'ha declarat de forma local? Comenteu la vostra resposta.

La variable "el_meu_tiquet" és una variable pròpia de cada fil que s'encarrega d'emmagatzemnar el número de torn que té el fil en qüestió. S'ha d'emmagatzemar localment ja que cada fil ha de tenir el seu propi torn. Si fos global tots els fils tindrien el valor de "tiquet" més recent i l'algorisme no funcionaria correctament.

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.

Es descriu breument el funcionament de l'algorisme

- El fil 1 fa la crida a lock_tiquet i se li assigna el tiquet 0. Ja que no es compleix la condició de la línia 12, el fil 1 retorna de la funció lock_tiquet i comença a executar el codi protegit.
- El fil 3 fa la crida a lock_tiquet mentre el fil 1 està executant el codi protegit. Se li assigna el tiquet 1 i el fil 3 s'adorm a la línia 14.
- El fil 2 fa la crida a lock_tiquet. Se li assigna el tiquet 2 i s'adorm a la línia 14.
- El fil 1 fa ara la crida unlock_tiquet. S'incrementa el valor de torn i es desperten els fils adormits a la línia 14. Només el fil 3 entrarà a la zona protegida. El fil 2 es tornarà a adormir.
- El fil 3 crida a unlock_tiquet. S'incrementa el valor de torn i es desperta el fil 2 que executa la zona protegida.

Es pot veure doncs que els fils s'executen en l'ordre en què arriben a lock_tiquet.

Problema 4 (2.5 punts, canonades). El vostre amic informàtic maldestre està fent experiments amb la funció pipe de C. El seu objectiu és aconseguir una comunicació bidireccional entre el pare i fill i, per això, us presenta el codi que hi ha a continuació. El codi compila correctament.

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3
 4 int main()
 5 {
 6   int fd[2];
 7   int res;
 8
 9   pipe(fd);
10
11   if (fork() == 0) {
12     res = 1;
13     write(fd[1], &res, sizeof(int));
14     read(fd[0], &res, sizeof(int));
15     printf("Valor rebut fill: %d\n", res);
16   } else {
17     read(fd[0], &res, sizeof(int));
18     printf("Valor rebut pare: %d\n", res);
19     res = 2;
20     write(fd[1], &res, sizeof(int));
21   }
22
23   return 0;
24 }
```

El vostre amic us comenta que creu que a) el fill escriu a la línia 13 el valor 1 que serà llegit pel pare a la línia 17, b) a continuació el pare escriure a la línia 20 el valor 2 que serà llegit pel fill a la línia 14. Tu i el vostre amic feu alguns experiments:

```
$ ./pipe
Valor rebut pare: 1
Valor rebut fill: 2
$ ./pipe
Valor rebut pare: 1
Valor rebut fill: 2
$ ./pipe
Valor rebut fill: 1
<espereu 10 segons sense que passi res i atureu el procés amb Ctrl+C>
```

Observar que els dos primers experiments surten tal com s'espera (el pare imprimeix 1, el fill imprimeix 2) mentre que el tercer no surt tal com s'espera (el fill imprimeix 1 i l'aplicació es queda "penjada" indefinidament). Es demana comentar breument les respostes a les següents preguntes:

1. Com és que, al tercer experiment, el imprimeix un 1? Indiqueu quines instruccions s'han d'executar perquè pugui passar.

A la línia 15 s'imprimeix un 1 en cas que no hi hagi cap canvi de context entre la línia 13 i 14. És a dir, que el procés fill escriu un 1 a la canonada i el mateix fill torna a llegir aquest valor de la canonada. S'imprimeix doncs un 1 per pantalla. En fer un canvi de context al pare, aquest es queda "penjat" a la línia 17 esperant que un altre procés hi escrigui alguna cosa.

2. Al tercer experiment, a quina línia es queda "penjada" l'aplicació? Per què?

Tal com s'ha comentat fa un moment, l'aplicació es queda bloquejada a la línia 17. La raó és que la funció de lectura "read" és bloquejant. Per tant, el pare es quedarà esperant que un altre procés escrigui alguna cosa a la canonada. Com que el fill no hi escriu res, el pare es quedarà bloquejat de forma indefinida.

3. Es pot "arreglar" el problema amb un sleep? On el col·locaríeu?

Tot i que no és pas una solució elegant, es pot posar un "sleep" entre la línia 13 i 14 per intentar forçar un canvi de context entre el write i el read del fill. D'aquesta forma podem aconseguir que el pare faci el read abans que el fill ho faci. Si després el fill intenta fer el seu read i el pare no ha fet el write, no passa res ja que el fill simplement es quedarà bloquejat a la línia 14 a l'espera que el pare hi escrigui alguna cosa a la canonada.

4. I fent servir dues canonades, es pot arreglar el problema? Indica com modificaríeu el codi si això serveix.

Fent servir dues canonades sí que es pot solucionar el problema. El que es vol aconseguir és que el pare llegeixi el valor 1 escrit pel fill i que el fill llegeixi el valor 2 enviat pel pare. Per tant, el codi queda així:

```
 1 #include <stdio.h>
 2 #include <unistd.h>
 3
 4 int main()
 5 {
 6   int fd1[2], fd2[2];
 7   int res;
 8
 9   pipe(fd1); pipe(fd2);
10
11   if (fork() == 0) {
12     res = 1;
13     write(fd1[1], &res, sizeof(int));
14     read(fd2[0], &res, sizeof(int));
15     printf("Valor rebut fill: %d\n", res);
16   } else {
17     read(fd1[0], &res, sizeof(int));
18     printf("Valor rebut pare: %d\n", res);
19     res = 2;
20     write(fd2[1], &res, sizeof(int));
21   }
22
23   return 0;
24 }
```

Fins que el fill no faci write a fd1[1] el pare estarà bloquejat a read fd1[0]. De forma similar, fins que el pare no faci el write a fd2[1] el fill estarà bloquejat a read fd2[0]. I sempre es llegiran els valors correctes.