PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Segundo semestre de 2019)

Horario 0781: prof. V. Khlebnikov
Horario 0782: prof. F. Solari A.

Duración: 1 h. 50 min.
Nota: No se puede usar ningún material de consulta.
La presentación, la ortografía y la gramática influirán en la calificación.
La práctica debe ser desarrollada en el cuadernillo usando lapicero.
Lo escrito con lápiz NO será evaluado.
Puntaje total: 20 puntos

Pregunta 1 (5 puntos – 25 min.) While humans can see and hear each other, computers can only read and write. So, one computer can write a note (or send a message) that the other computer will later read, but they cannot see each other. To understand the difficulty with this type of restricted communication, let us examine a simple two-person interactions where communication is restricted to writing and reading of notes. The two people involved, let's call them Alice and Bob, can not see each other and they communicate only by writing and reading of notes. In particular, Alice can not see that Bob is reading a note that she has written to him earlier. Alice and Bob are sharing an apartment. Alice arrives home in the afternoon, looks in the fridge and finds that there is no milk. So, she leaves for the grocery to buy milk. After she leaves, Bob arrives, he also finds that there is no milk and goes to buy milk. At the end they both buy milk and end up with too much milk. So, Alice and Bob are looking for a solution to ensure that: (1) only one person buys milk, when there is no milk; (2) someone always buys milk, when there is no milk.

Notice that a solution in which only Bob is responsible for buying milk would not work. In such a solution, there is a scenario where Alice arrives home and finds out there is no milk, and waits forever for Bob to show up. Alice and Bob have discussed the situation and agreed that in order to synchronize their actions, they will communicate by leaving (signed) notes on the door of the fridge. More specifically, they came up with the following solution:

SOLUTION 1: *If you find that there is no milk and there is no note on the door of the fridge, then leave a note on the fridge's door, go and buy milk, put the milk in the fridge, and remove your note. The code is as follows:*

PROGRAM FOR ALICE:

```
1   if (no note) then
2       if (no milk) then
3           leave note
4           buy milk
5           remove note
6       fi
7   fi
```

PROGRAM FOR BOB:

```
1   if (no note) then
2       if (no milk) then
3           leave note
4           buy milk
5           remove note
6       fi
7   fi
```

a) (1 punto) Comente la Solución 1.

SOLUTION 2: *As soon as you arrive home, you leave a note on the fridge's door. Only then you check, and if you find that there is no milk and there is no note (other than yours), then you go and buy milk, put the milk in the fridge and remove your note. The code is as follows:*

PROGRAM FOR ALICE:

```
1   leave note Alice
2   if (no note Bob) then
3       if (no milk) then
4           buy milk
5       fi
6   fi
7   remove note Alice
```

PROGRAM FOR BOB:

```
1   leave note Bob
2   if (no note Alice) then
3       if (no milk) then
4           buy milk
5       fi
6   fi
7   remove note Bob
```

b) (2 puntos) Comente la Solución 2.

SOLUTION 3:

Alice: *When Alice arrives home, she leaves a note on the fridge's door. Then, if she finds that there is no milk and that there is no note (signed by Bob), she buys milk, puts the milk in the fridge and removes her note. Otherwise, if Bob left a note then she removes her note, and does nothing.*

Bob: *When Bob arrives home, he leaves a note on the fridge's door. Then, if he finds that there is a note signed by Alice, he checks the fridge's door over and over again waiting for Alice to remove her note. Once Bob finds that Alice's note has been removed, he checks if there is milk. If there is no milk, he buys milk, puts the milk in the fridge and removes his note. Otherwise, if there is milk, he removes his note without buying milk.*

PROGRAM FOR ALICE:

```
1       leave note Alice
2       if (no note Bob) then
3           if (no milk) then
4               buy milk
5           fi
6       fi
7       remove note Alice
```

PROGRAM FOR BOB:

```
1       leave note Bob
2       while (note Alice) do skip od
3       if (no milk) then
4           buy milk
5       fi
6
7       remove note Bob
```

c) (2 puntos) Comente la Solución 3.

Pregunta 2 (5 puntos – 25 min.) (*POSIX Threads, part2 - Daniel Robbins, IBM developerWorks*) El siguiente programa en lenguaje C, utiliza pthreads para demostrar el trabajo con secciones críticas. Se ha modificado ligeramente para, en lugar de usar mutexes de la librería POSIX pthreads, hacerlo mediante un algoritmo *busy waiting* conocido.

```
alulab@linuxmint18 ~/INF239-Pa2-2/ $ cat thread4.c
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

int myglobal;
int turn;
int interested[2];
#define TRUE 1
#define FALSE 0
void enter_region(int thread) {
    /* main thread 0, thread 1 */
    int other; other = 1 - thread;

    interested[thread] = TRUE;
    turn = thread;
    while(turn == thread && interested[other] == TRUE) ;
}

void leave_region(int thread) {
   interested[thread] = FALSE;
}

void *thread_function(void *arg) {
   int i,j,k,x=0;

   for ( i=0; i<20; i++ ) {
    enter_region(1);
    j=myglobal;
    j=j+1;
    printf(".");
    fflush(stdout);
    /* sleep(1); */
    for(k=0;k<30000;k++) x=x+1;
    myglobal=j;
   leave_region(1);
  }
  return NULL;
}

int main(void) {

pthread_t mythread;
```

```
int i,k,x=0;

if ( pthread_create( &mythread, NULL, thread_function, NULL) ) {
    printf("error creating thread.");
    abort(); }

for ( i=0; i<20; i++) {
    enter_region(0);
    myglobal=myglobal+1;
    printf("o");
    fflush(stdout);
    leave_region(0);
    * sleep(1); */
   for(k=0;k<30000;k++) x=x+1;
}

if ( pthread_join ( mythread, NULL ) ) {
    printf("error joining thread."); abort();
}
printf("\nmyglobal equals %d\n",myglobal);
exit(0);
}
```

a) (3 puntos – 15 min.) Determine cuáles son las líneas que se consideran la Sección Crítica en cada hilo y discuta si son las necesarias para proteger el/los recursos compartidos o puede mejorarse. Explique cómo se protegen las secciones críticas si los hilos intentan a la vez ingresar a su Sección Crítica.

```
alulab@linuxmint18 ~/INF239-Pa2-2/ $ gcc -pthread -lpthread -o thread4 thread4.c
alulab@linuxmint18 ~/INF239-Pa2-2/ $ time ./thread4
o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.
myglobal equals 40
real 0m0.014s
user 0m0.026s
sys  0m0.000s
```

b) (2 puntos – 10 min.) Una crítica que se hace a la solución anterior, es la espera ocupada, o *busy-waiting*. Con la librería POSIX pthreads, se puede ejecutar `pthread_yield()` como cuerpo del lazo de espera. El programa `thread5.c` contiene esta modificación. Su ejecución mediante time y resultado:

```
alulab@linuxmint18 ~/INF239-Pa2-2/ $ time ./thread5
o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.
myglobal equals 40

real 0m0.010s
user 0m0.009s
sys  0m0.009s
```

El tiempo `real` es el tiempo trascurrido durante toda la ejecución, mientras que el tiempo `user` es el utilizado en CPU en modo usuario, y el tiempo `sys` el utilizado en CPU en modo kernel (por el sistema operativo, a cuenta del proceso e hilos) (en estos dos últimos casos, se suman tiempos en cada CPU, si hay más de una).

¿Cómo mejoran los tiempos de ejecución por el uso de `pthread_yield()`? Explique y justifique ayudándose de una gráfica en el tiempo (no escalada, pero algo proporcional), que muestre la ejecución de los hilos para `thread4` y `thread5`.

<u>Pregunta 3</u> (5 puntos – 25 min.) El siguiente código corresponde a una función que será invocada desde un programa principal para ser ejecutada por dos hilos concurrentes.

```
void thread_thing(int *pnum_times)
{
  int i, j, x;

  for (i = 0;  i < 4; i++) {
            printf("doing one thing\n");
            for (j = 0; j < 10000; j++) x = x + i;
            (*pnum_times)++;
          }
}
```

a) (1 punto – 5 min.) Para el caso en que a cada hilo se le indica como argumento la dirección de una variable global diferente, digamos "`&r1`" y "`&r2`", siendo `r1=r2=0` inicialmente. ¿Cuál sería el resultado de `r1+r2`? Justifique su respuesta.

b) (2 puntos – 10 min.) Analice, ¿que ocurriría si en el programa principal se indica a ambos hilos la misma dirección, digamos "`&r`" de una variable global? ¿Cuál debería ser el resultado correcto de "`r`" ? Justifique su respuesta.

Se le pide a algún estudiante del curso que agregue lo necesario para que el resultado al utilizarse la misma variable global en ambos hilos, sea el resultado correcto, presentando la siguiente solución:

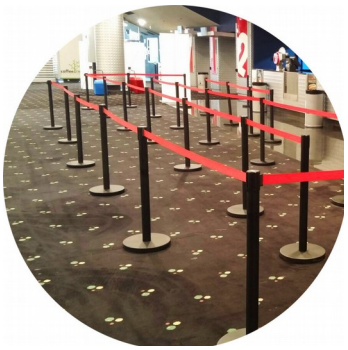
```
void thread_thing(int *pnum_times)
{
  int i, j, x;

  pthread_mutex_lock(s);
  for (i = 0;  i < 4; i++) {
            printf("doing one thing\n");
            for (j = 0; j < 10000; j++) x = x + i;
            (*pnum_times)++;
          }
  pthread_mutex_unlock(s);
}
```

c) (1 punto – 5 min.) ¿El resultado será el esperado? ¿Cómo será la ejecución concurrente de los hilos en este caso?

d) (1 punto – 5 min.) ¿Qué cambios en el código deben hacerse para que cumpla los criterios de una buena solución?

Pregunta 4 (5 puntos – 25 min.) En un museo, de postes separadores con cinta extensible retráctil, se forma un pasillo, abierto en su entrada pero cerrado en su salida que está justo en la puerta de entrada al museo. Es que las visitas al museo son guiadas por un miembro del personal del museo que acompaña a los grupos de 8 visitantes. Por eso, el pasillo está configurado exactamente para 9 personas. Cuando en el pasillo aparecen los primeros visitantes, a ellos se acerca el guía y todos esperan hasta que se complete el grupo. El último del grupo cierra la entrada al pasillo y se abre su salida permitiendo a todas 9 personas entrar al museo pero uno por uno.



Se pide desarrollar el código del **monitor** *GroupOf9* con el procedimiento *waiting_lane* que deben ejecutar todos los 9 procesos. Los procesos simulan el comportamiento de las personas en un modelo de este pasillo donde ellos deben esperar la formación del grupo completo.

Como un ejemplo de la estructura de un monitor, se le proporciona el código presentado en el libro de texto (Figure 2-34):

```
monitor ProducerConsumer
    condition full, empty;
    integer count;

    procedure insert(item: integer);
    begin
        if count = N then wait(full);  /* The buffer has N slots. */
        insert_item(item);
        count := count + 1;
        if count = 1 then signal(empty)
    end;

    function remove: integer;
    begin
        if count = 0 then wait(empty);
        remove = remove_item;
        count := count - 1;
        if count = N - 1 then signal(full)
    end;

    count := 0;
end monitor
```

La práctica ha sido preparada por FS(2,3) y VK(1,4)
con LibreOffice Writer en Linux Mint 19.2 “Tina”.

Profesor del curso: (0781) V. Khlebnikov
(0782) F. Solari A.

Pando, 27 de septiembre de 2019