PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Segundo semestre de 2022)

Horarios 0781, 0782: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **Sin apuntes de clase y sin calculadora o computadora.**
La presentación, la ortografía y la gramática influirán en la calificación.
Puntaje total: 20 puntos

Pregunta 1 (5 puntos – 25 min.) Se pretende la organización del trabajo intercalado de 2 hilos creados con los identificadores `th_flip` y `th_flop`, ejecutando las funciones respectivas `thread_flip()` y `thread_flop()`. Para sincronización se usan las funciones `thr_continue()` y `thr_suspend()`:

`thr_suspend()` immediately suspends the execution of the thread specified by target_thread.
On successful return from `thr_suspend()`, the suspended thread is no longer executing.
Once a thread is suspended, subsequent calls to `thr_suspend()` have no effect.
Signals cannot awaken the suspended thread; they remain pending until the thread resumes execution.

`thr_continue()` resumes the execution of a suspended thread.
Once a suspended thread is continued, subsequent calls to `thr_continue()` have no effect.

Al terminar su parte de trabajo, cada hilo despierta al otro y suspende a sí mismo:

```
...
thread_t th_flip, th_flop;
...

void * thread_flip(void *arg) {
    ...
    while (1) {
        ...
        thr_continue(th_flop);
        thr_suspend(th_flip);
    }
}
```

```
void * thread_flop(void *arg) {
    ...
    while (1) {
        thr_suspend(th_flop);
        ...
        thr_continue(th_flip);
    }
}
```

Encuentre *race condition* en este código.

Pregunta 2 (3 puntos – 15 min.) Dado el siguiente programa:

```
$ cat -n 2022-2_p2.c | expand
     1  #include <pthread.h>
     2  #include <stdlib.h>
     3  #include <stdio.h>
     4
     5  int n=1;
     6  pthread_t p, q;
     7  static pthread_mutex_t LOCK = PTHREAD_MUTEX_INITIALIZER;
     8
     9  void *
    10  thread_p(void *arg)
    11  {
    12      int i=0;
    13      while (n<1) {
    14          pthread_mutex_lock(&LOCK);
    15          n++;
    16          pthread_mutex_unlock(&LOCK);
    17          i++;
    18      }
    19      fprintf(stderr,"p: %d iterations\n",i);
    20      pthread_exit(0);
    21  }
    22
    23  void *
```

```
24  thread_q(void *arg)
25  {
26      int i=0;
27      while (n>=0) {
28          pthread_mutex_lock(&LOCK);
29          n--;
30          pthread_mutex_unlock(&LOCK);
31          i++;
32      }
33      fprintf(stderr,"q: %d iterations\n",i);
34      pthread_exit(0);
35  }
36
37  int
38  main(void)
39  {
40      int i;
41
42      if (pthread_create(&p,NULL,thread_p,NULL)) {
43          printf("error creating thread p"); abort(); }
44      if (pthread_create(&q,NULL,thread_q,NULL)) {
45          printf("error creating thread q"); abort(); }
46      if (pthread_join(p,NULL)) {
47          printf("error joining thread p"); abort(); }
48      if (pthread_join(q,NULL)) {
49          printf("error joining thread q"); abort(); }
50      exit(0);
51  }
```

```
$ gcc 2022-2_p2.c -pthread
$ ./a.out
p: 0 iterations
q: 2 iterations
$ ./a.out
p: 0 iterations
q: 2 iterations
$ ./a.out
q: 2 iterations
p: 2 iterations
...
```

a) (1 punto – 5 min.) ¿Con qué guion de ejecución el lazo en *p* (las líneas 13-17) se ejecuta exactamente una sola vez? Indique la secuencia de los números de las líneas.

b) (1 punto – 5 min.) ¿Con qué guion de ejecución el lazo en *p* (las líneas 13-17) se ejecuta exactamente dos veces?

c) (1 punto – 5 min.) ¿Con qué guion de ejecución ambos lazos (las líneas 13-17 y 27-31) se ejecutan de forma infinita?

<u>Pregunta 3</u> (4 puntos – 20 min.) A continuación se muestra el algoritmo de Peterson:

```
var  flag : array [0..1] of boolean;
     turn : 0..1;

begin
     flag[0] := false;
     flag[1] := false;

     Parbegin
          {P0}                                  {P1}
          repeat                                repeat
               flag[0] := true;                      flag[1] := true;
               turn := 1;                            turn := 0;
               while flag[1] and turn = 1            while flag[0] and turn = 0
                    do {nothing};                         do {nothing};
               { Critical Section }                  { Critical Section }
               flag[0] := false;                     flag[1] := false;
               { Remainder of the cycle }            { Remainder of the cycle }
          forever                               forever
     Parend
end.
```

a) (2 puntos) Se hacen los siguientes cambios en el algoritmo de Peterson: en P0 la instrucción `flag[0] := true` se cambia por `flag[0] := false` y la instrucción `flag[0] := false` se cambia por `flag[0] := true`, y de forma análoga se hace el cambio en P1. ¿Qué propiedades de las secciones críticas no se cumplen?

b) (2 puntos) La instrucción `while flag[1] and turn = 1` en P0 del algoritmo de Peterson presentado originalmente, sin cambios, se cambia por `while flag[1] or turn = 1`, y de forma análoga para el P1. ¿Qué propiedades de las secciones críticas no se cumplen?

<u>Pregunta 4</u> (6 puntos – 30 min.)

a) (2 puntos) Según MOS4E, ¿cómo funcionará en un proceso la secuencia de operaciones `signal(foo); wait(foo);`, donde `foo` es una variable de condición de un monitor que se usa por primera vez?

b) (4 puntos) ¿Cómo funcionará el siguiente código para 10 procesos que, al mismo tiempo, hacen `call Sincroniza.seguir`? Describa todos los casos posibles. Use los números de las líneas del código. ¿Y si los procesos sean solamente 7, cómo funcionaría?

```
1    monitor Sincroniza
2        condition barrera
3        integer cuenta, dormidos
4
5        procedure seguir;
6        begin
7            if cuenta = 7 then
8            begin
9                cuenta := 1
10               dormidos := dormidos - 1
11               signal(barrera)
12           end
13           else
14           begin
15               cuenta := cuenta + 1
16               dormidos := dormidos + 1
17               wait(barrera)
18               if dormidos = 1 then
19               begin
20                   dormidos := dormidos - 1
21                   signal(barrera)
22               end
23           end
24       end
25
26       cuenta := 1
27       dormidos := 0
28   endmonitor
```

<u>Pregunta 5</u> (2 puntos – 10 min.) Five jobs are waiting to be run. Their expected run times are 9, 6, 3, 5, and *X*. In what order should they be run to minimize average response time?

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 21 “Vanessa”

Profesor del curso: V. Khlebnikov

Pando, 30 de septiembre de 2022