PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Primer semestre de 2023)

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 (2 puntos – 10 min.)

(a) (1 punto) ¿Qué son *a physical program counter* y *a logical program counter*? ¿Cómo se modifica cada uno de ellos?
(b) (0,5 puntos) ¿Por qué los procesos no deben ser programados en base de las suposiciones sobre el tiempo?
(c) (0,5 puntos) Además de un programa, una entrada, una salida, ¿qué tiene más un proceso?

Pregunta 2 (4 puntos – 20 min.)

(a) (1 punto) La terminación de procesos se clasifica en dos categorías. ¿Cuáles son estas? Cada categoría abarca dos condiciones. Indique estas.
(b) (2 puntos) Considerando tres estados de procesos, indique todas las transiciones posibles entre ellos y las causas que los provocan.
(c) (1 punto) ¿Cuáles son las condiciones para cumplir que proporcionan una buena solución que permite evitar *race conditions*?

Pregunta 3 (2 puntos – 10 min.) En la figura 2.15 del libro MOS4E se presenta un ejemplo de programa que usa hilos:

```
$ cat mos4e_fig2.15_v0.c
#include <sys/types.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NUMBER_OF_THREADS   1

void *
print_hello_world(void *tid)
{
    /* This function prints the process's and the thread's identifier and
       then exits. */
    printf("Hello World. Greetings from thread %d\n",
           (int)(intptr_t)tid);
    pthread_exit(0);
}

int
main(int argc, char *argv[])
{
    /* The main program creates NUMBER_OF_THREADS threads and then exits. */
    pthread_t threads[NUMBER_OF_THREADS];
    int status, i;

    for (i=0; i<NUMBER_OF_THREADS; i++) {
        printf("Main here. Creating thread %d\n", i);
        status = pthread_create(&threads[i], NULL, print_hello_world,
                                (void *)(intptr_t)i);

        if (status != 0) {
            printf("Oops. pthread_create returned error code %d\n", status);
            exit(-1);
        }
    }
    exit(0);
```

```
}
$ gcc mos4e_fig2.15_v0.c -lpthread
$ ./a.out
Main here. Creating thread 0
$
```

¿Por qué el programa se ejecuta no como se espera?

Pregunta 4 (3 puntos – 15 min.) Dado el siguiente programa:

```
$ cat -n 2023-1_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 2023-1_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 13-18) 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 13-18) se ejecuta exactamente dos veces?
c) (1 punto – 5 min.) ¿Con qué guion de ejecución ambos lazos (las líneas 13-18 y 27-32) se ejecutan de forma infinita?

Pregunta 5 (4 puntos – 20 min.) Dado el siguiente programa y su ejecución:

```go
package main

import (
    "fmt"
    "sync"
)

var (
    buf  [3]int = [3]int{0,0,0}
    pos     int
    wg      sync.WaitGroup
    mutex   sync.Mutex
    full    sync.Mutex
    empty   sync.Mutex
)

func producer() {
    for n := 1; n < 11; n++ {
        item := n * n
        pos = n % 3
        if buf[pos] != 0 {
            full.Lock()
        }
        mutex.Lock()
        buf[pos] = item
        mutex.Unlock()
        fmt.Printf("producer: pos=%d, item=%d, buf=%v\n", pos, item, buf)
    }
    wg.Done()
}

func consumer() {
    var item int
    for n := 1; n < 11; n++ {
        pos = n % 3
        mutex.Lock()
        if buf[pos] == 0 {
            empty.Lock()
        }
        item = buf[pos]
        buf[pos] = 0
        mutex.Unlock()
        fmt.Printf("consumer: pos=%d, item=%d, buf=%v\n", pos, item, buf)
    }
    wg.Done()
}

func main() {
    wg.Add(2)
    go consumer()
    go producer()
    wg.Wait()
}
```

```
$ go build ej9_v0.go
$ ./ej9_v0
producer: pos=1, item=1, buf=[0 1 0]
producer: pos=2, item=4, buf=[0 0 4]
producer: pos=0, item=9, buf=[9 0 4]
producer: pos=1, item=16, buf=[9 16 4]
producer: pos=2, item=25, buf=[9 16 25]
consumer: pos=1, item=1, buf=[9 16 25]
consumer: pos=2, item=25, buf=[9 16 0]
consumer: pos=0, item=9, buf=[0 16 0]
consumer: pos=1, item=16, buf=[0 0 0]
consumer: pos=2, item=0, buf=[0 0 0]
fatal error: all goroutines are asleep - deadlock!

goroutine 1 [semacquire]:
sync.runtime_Semacquire(0x49de18?)
    /usr/lib/go-1.18/src/runtime/sema.go:56 +0x25
sync.(*WaitGroup).Wait(0x4862e0?)
    /usr/lib/go-1.18/src/sync/waitgroup.go:136 +0x52
```

```
main.main()
     /home/vk/Descargas/ej9_v0.go:52 +0x49

goroutine 6 [semacquire]:
sync.runtime_SemacquireMutex(0x25?, 0xa0?, 0x3?)
     /usr/lib/go-1.18/src/runtime/sema.go:71 +0x25
sync.(*Mutex).lockSlow(0x551f20)
     /usr/lib/go-1.18/src/sync/mutex.go:162 +0x165
sync.(*Mutex).Lock(...)
     /usr/lib/go-1.18/src/sync/mutex.go:81
main.consumer()
     /home/vk/Descargas/ej9_v0.go:38 +0x189
created by main.main
     /home/vk/Descargas/ej9_v0.go:50 +0x31

goroutine 7 [semacquire]:
sync.runtime_SemacquireMutex(0x28?, 0xa0?, 0x3?)
     /usr/lib/go-1.18/src/runtime/sema.go:71 +0x25
sync.(*Mutex).lockSlow(0x551f28)
     /usr/lib/go-1.18/src/sync/mutex.go:162 +0x165
sync.(*Mutex).Lock(...)
     /usr/lib/go-1.18/src/sync/mutex.go:81
main.producer()
     /home/vk/Descargas/ej9_v0.go:22 +0x155
created by main.main
     /home/vk/Descargas/ej9_v0.go:51 +0x3d
$
```

Sin explicar el programa, interprete el mensaje de error e indique en qué consiste exactamente el *deadlock* ("abrazo mortal") anunciado aquí.

Pregunta 6 (3 puntos – 15 min.) (*Ben-Ari, PCDP/2E*) Consider the following algorithm (cada sentencia `Pi` y `Qj` es atómica):

```
integer n ← 0

Hilo P                      Hilo Q

P1:  while n < 2            Q1:  n ← n + 1
P2:      write(n)           Q2:  n ← n + 1
```

Construct scenarios that give the output sequences: 012, 002, 02. (**1,5 puntos** por tres secuencias)
Must the value 2 appear in the output? Explíquelo. (**0,5 puntos**)
How many times can the value 2 appear in the output? (**0,5 puntos**)
How many times can the value 1 appear in the output? (**0,5 puntos**)

Pregunta 7 (2 puntos – 10 min.) Analice el siguiente pseudocódigo y determine si hay *race condition*:

```
volatile int lock;

/* i = 0,1 */
Hilo(i)

    lock=1-i
    while(lock);
    <sección crítica>
    lock=0
    <sección no crítica>
```

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 21.1 "Vera"

Profesor del curso: V. Khlebnikov

Pando, 4 de mayo de 2023