PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS

2da práctica (tipo a)
(Primer semestre de 2021)

Horario 0781: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 1 (6 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P2_P1** de la **Práctica 2** en PAIDEIA **antes de las 09:40**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_21.txt`. Por ejemplo, `20171903_21.txt`.

a) (3 puntos – 15 min.) Considere su propio código de estudiante de la PUCP como la secuencia de 8 dígitos: $d_7\, d_6\, d_5\, d_4\, d_3\, d_2\, d_1\, d_0$. Calcule $n_i = d_i + d_{i+4}$, donde $0 \le i \le 3$. Hay 4 hilos th_0, th_1, th_2, th_3, y cada hilo th_i ejecuta un bucle de n_i iteraciones del incremento de la variable compartida x, cuyo valor inicial es 0, en 2:

var $x = 0$

thread $th[i]$:
    $j = 0$
    while $j < n_i$:
        x += 2

Presente los valores finales posibles de la variable x. Explique el escenario (*RAE: "Escenario: 4. m. Conjunto de circunstancias que rodean a una persona o un suceso."*) de ejecución cuando el resultado final de la variable x será el valor siguiente al mínimo.

b) (3 puntos – 15 min.) For positive values of `K`, what are the possible final values of `n` in the following algorithm? Explíquelo.

Algorithm 2.10: Incrementing and decrementing	
`integer n ← 0`	
p	**q**

`integer temp`
`p1: do K times`
`p2: temp ← n`
`p3: n ← temp + 1`
	`integer temp`
	`q1: do K times`
	`q2: temp ← n`
	`q3: n ← temp - 1`

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.1 "Ulyssa"

Profesor del curso: V. Khlebnikov

Lima, 14 de mayo de 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
2da práctica (tipo a)
(Primer semestre de 2021)

Horario 0781: prof. V. Khlebnikov

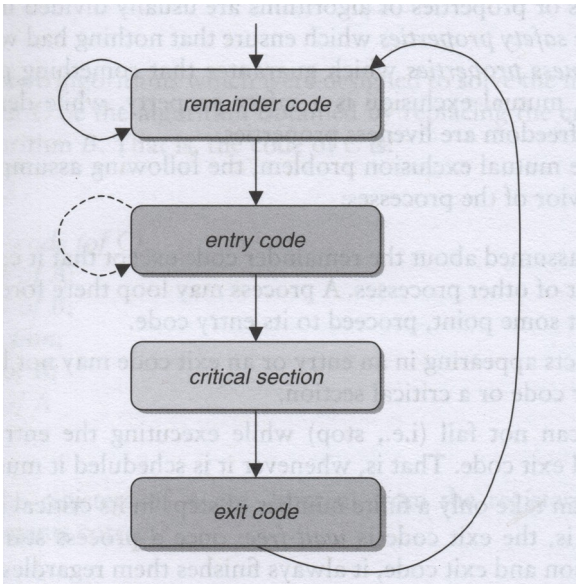
Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 2 (6 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P2_P2** de la **Práctica 2** en PAIDEIA **antes de las 10:20**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_22.txt`. Por ejemplo, `20171903_22.txt`.

"The ***mutual exclusion*** **problem** is the guarantee of mutually exclusive access to a single shared resource when there are several competing processes. The problem is defined as follows: it is assumed that each process is executing a sequence of instructions in an infinite loop. The instructions are divided into four continuous sections of code: the ***remainder***, ***entry***, ***critical section***, and ***exit***. Thus the structure of a mutual exclusion solution looks as follows:

loop forever
remainder code;
entry code;
critical section;
exit code
end loop

A process starts by executing the remainder code. At some point the process might need to execute some code in its critical section. In order to access its critical section a process has to go through an **entry code** which **guarantees** that while it is executing its critical section, no other process is allowed to execute its critical section. In addition, once a process finishes its critical section, the process executes its **exit code** in which it **notifies** other processes that it is no longer in its critical section. After executing the exit code the process returns to the remainder.



The mutual exclusion problem is to write the code for the ***entry code*** and the ***exit code*** in such a way that the following two basic requirements are satisfied.

Mutual exclusion: *No two processes are in their critical section at the same time*.

Deadlock-freedom: *If a process is trying to enter its critical section, then some process, not necessarily the same one, eventually enters its critical section.*

However, deadlock-freedom may still allow "starvation" of individual processes. That is, a process that is trying to enter its critical section, may never get to enter its critical section, and wait forever in its entry code.

Starvation-freedom: *If a process is trying to enter its critical section, then this process must eventually enter its critical section.*

In solving the mutual exclusion problem, the following assumptions are made about the behavior of the processes:

1. Nothing is assumed about the **remainder code** except that it can not influence the behavior of other processes. A process may loop there forever, it may halt or it may, at some point, proceed to its entry code.
2. Shared objects appearing in an **entry** or an **exit code** may not be referred to in a remainder code or a critical section.
3. A process can not fail (i.e. stop) while executing the **entry code**, **critical section**, and **exit code**. That is, whenever it is scheduled it must take a step.
4. A process can take only a finite number of steps in its **critical section** and **exit code**. That is, the exit code is *wait-free*: once a process starts executing its critical section and exit code, it always finishes them regardless of the activity of the other processes.
5. While the collection of processes is concurrent, individual processes are sequential.

Let ***A*** be an arbitrary **deadlock-free mutual exclusion** algorithm that does no satisfy requirement 4, that is, the number of steps in the exit code depends on the activity of the other processes. In the following, ***A*** is modified, so that the new algorithm satisfies requirement 4. One additional shared bit, called *flag*, is used.

Initially: *flag* = *false*.

```
1   Former entry code of A;
2   while flag do skip od;
3   flag := true;
4   Former exit code of A;
5   critical section;
6   flag := false;
```

Does the modified algorithm satisfy deadlock-freedom and/or mutual exclusion (for any `A`)? Assuming that `A` satisfies also starvation-freedom, does the modified algorithm satisfy starvation-freedom?"

Si el algoritmo `A`, dado inicialmente, según el enunciado, garantiza la exclusión mutua y está libre de *deadlock*, entonces el primer proceso que ejecute la línea 1 obtendrá el acceso exclusivo al recurso, teniendo la condición de **while** en *false*, pasará a la línea 3, cambiando el valor de la variable *flag* a *true.* Si ahora aparecen 2 procesos más que quieren usar el mismo recurso compartido, entonces …

Complete la traza de ejecución y responda a las preguntas sobre este algoritmo modificado.

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.1 "Ulyssa"

Profesor del curso: V. Khlebnikov

Lima, 14 de mayo de 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

SISTEMAS OPERATIVOS
2da práctica (tipo a)
(Primer semestre de 2021)

Horario 0781: prof. V. Khlebnikov

Duración: 1 h. 50 min.
Nota: **La presentación, la ortografía y la gramática influirán en la calificación.**
Puntaje total: 20 puntos

Pregunta 3 (8 puntos – 30 min.) Su respuesta debe estar en la carpeta **INF239_0781_P2_P3** de la **Práctica 2** en PAIDEIA **antes de las 11:00**. Por cada 3 minutos de retardo son -2 puntos.
El nombre de su archivo debe ser <*su_código_de_8_dígitos*>`_23.txt`. Por ejemplo, `20171903_23.txt`.

a) (4 puntos) Hay tres procesos: A, B y C con tres partes secuenciales de su código: A1, A2, A3 en A; B1, B2, B3 en B; y C1, C2, C3 en C. Se necesita sincronizarlos de tal manera que

A2 se ejecute después de B1, A3 se ejecute después de C2;
B2 se ejecute después de A1, B3 se ejecute después de C1; y
C2 se ejecute después de B2, C3 se ejecute después de A2.

Para evitar la espera ocupada de los cambios en las variables-indicadoras, la solución debe ser con semáforos, cuyas operaciones son `up` y `down`. Presente la solución.

b) (4 puntos) Se desea simular semáforos usando monitores con sus variables condicionales y las operaciones `waitc`, `signalc` y `empty` sobre ellas. En el siguiente código complete las líneas que faltan de modo que `monP` y `monV` se comporten como semáforos.

```
/* Monitor implementation of semaphore. */
monitor monSemaphore {
    int semvalue;
    condition notbusy;

    void monP() {
    . . .
    }

    void monV() {
    . . .
    }

    init{ semvalue = 1; }
} // end of monSemaphore monitor

int n;

void inc(int i) {
    monP();
        n = n + 1;
    monV();
}

void main() {
    cobegin {
        inc(1); inc(2);
    }
}
```

La práctica ha sido preparada por VK
con LibreOffice Writer en Linux Mint 20.1 “Ulyssa”

Profesor del curso: V. Khlebnikov

Lima, 14 de mayo de 2021