Paradigmas y Lenguajes de Programación
Trabajo Práctico Número 1
Práctica

Ulises C. Ramirez [uli.r19@gmail.com]
Héctor Chripczuk
Verónica Gonzalez

14 de Septiembre, 2018

Código

Todo el código que esté expresado en el documento como respuesta a algún ejercicio esta contenido en la carpeta `PascalFC`, junto con el archivo `*.lst` y el correspondiente `*.obj`.

Versionado

Para el corriente documento se está llevando un versionado a fin de mantener un respaldo del trabajo y además proveer a la cátedra o a cualquier interesado la posibilidad de leer el material en la última versión disponible.

Repositorio: *https://github.com/ulisescolina/UC-PYLP/*

–Ulises

Índice de Contenído

1 Instalación PascalFC

CONSIGNA: **Investigar y describir como instalar el lenguaje PascalFC en su sistema operativo. Lectura recomendada por la cátedra: página del Ing. John Coppens, *http://jcoppens.com/soft/pfc2*.**

Descripción de la instalación: la descripción a brindar se realiza en una máquina con las siguientes características:

Listing 1: Características sistema

```
~$ uname --kernel-name --kernel-release --machine
    --operating-system
Linux 4.15.0-34-generic x86_64 GNU/Linux
~$
```

para iniciar con la instalación se siguió el vínculo a la página del Ing. John Coppens en el apartado de descargas [http://jcoppens.com/soft/pfc2/download.php], luego se procedió a descargar la ultima versión de la compilación del `pfc2`, que para el día 16 de Septiembre del 2018 es `pfc2-0.9.40.x86_64.tar.gz`. Con el archivo comprimido descargado, solamente es necesario descomprimirlo en alguna carpeta que tengamos a mano, y despues de eso utilizar los dos archivos que son el resultado de la compilación del PascalFC, `pfc2` y el `pfc2int`, ahí tendremos el compilador e intérprete.

Finalizados estos pasos ya tendremos instalado el PascalFC, para la compilación y ejecución será necesario realizar en consola los siguientes pasos:

Listing 2: Compilación de y ejecución con pfc2

```
~$ cd /ruta/en/la/que/se/descomprimio/la/descarga/
~$ ./pfc2 archivo_a_ser_compilado.pfc
** Sucede la compilacion **
~$ ./pfc2int archivo_a_ser_compilado.obj
~$
```

2 Codificación 1

CONSIGNA: **Realizar un programa que ejecute paralelamente 2 procesos donde cada uno imprima por pantalla un numero "ID" de proceso**.

Listing 3: TP1, Ejercicio 2

```
program tp1_ej2;

process type print (id : integer);
```

```
4   begin
5     writeln( 'ID de proceso: ', id);
6   end;
7  var
8   p1, p2 : print ;
9  begin
10  cobegin
11    p1 (1) ;
12    p2 (2) ;
13  coend;
14 end.
```

Una cuestión a tener en cuenta en el `Listing 3` es el hecho de que la función `writeln` no es atómica, y es muy probable que se encuentre con intercalamiento aun mas de lo que ocurre con la funcion `write`.

Alivianar un poco esta situación de intercalamiento en el ejercicio, *sin el uso de semáforos* es imprimiendo un parametro al lado del otro utilizando la funcion `write` como se demuestra a continuación:

Listing 4: TP1, Ejercicio 2 (write)

```
1  program tp1_ej2;
2
3  process type print (id : integer);
4    begin
5      write( 'ID de proceso: ', id);
6    end;
7  var
8    p1, p2 : print ;
9  begin
10   cobegin
11     p1 (1) ;
12     p2 (2) ;
13   coend;
14 end.
```

3 Codificación 2

Consigna: **realizar un programa que ejecute paralelamente 3 procesos, 2 de los procesos deben imprimir 5 números pares y el otro 10 números pares.**

Listing 5: TP1, Ejercicio 3

```
1  program tp1_ej3;
2  {
3   Imprime una cantidad determinada de numeros
```

```
4   pares que se encuentren entre 0 y un limite
5   establecido
6   @param id Identificador del proceso
7   @param cantidad cantidad de numeros pares a
8    imprimir
9   @param limSuperior limite establecido para
10   el pool de numeros escogidos
11 }
12 process type pares (
13       id : integer;
14       cantidad : integer;
15       limSuperior : integer
16        );
17   var
18     cant : integer;
19     par : integer;
20   begin
21     Randomize;
22     cant := 0;
23     par := 0;
24     while cant < cantidad do
25     begin
26       par:=Random(limSuperior)+1;
27       if par mod 2 = 0 then
28       begin
29         writeln( 'El proceso ',
30            id, ' imprime ',par);
31         cant := cant + 1;
32           par:=0;
33       end;
34     end;
35   end;
36 var
37   p1, p2, p3 : pares;
38 begin
39   cobegin
40     p1 (1, 5, 100) ;
41     p2 (2, 5, 40) ;
42     p3 (3, 10, 35);
43   coend;
44 end.
```

Cabe mencionar que en el código anterior también se padece de un caso bastante grave de intercalación.

4 Codificación 3

CONSIGNA: **crear un algoritmo que calcule todos los números primos entre 1 y 100. Distribuir los datos para que cada proceso tome el mismo númeor de elementos. ¿Es una distribución óptima? Justifique.**

Antes de presentar el código del programa se aclara que fue necesario el uso de semáforos aunque el ejercicio no lo pida, para poder así presentar la información solicitada de manera legible, de otra manera no iba a ser discernible si la implementación paralela del algoritmo tuvo éxito.

Listing 6: TP1, Ejercicio 4

```
1  program tp1_ej4;
2  {
3   @return true si el valor de numero es primo
4   @return false si el valor de numero no es primo
5  }
6  function esprimo(numero : integer):boolean
7  var
8    cantDiv : integer;
9    i : integer;
10 begin
11   for i:=2 to numero-1 do
12   begin
13     if (numero mod i = 0) then
14     begin
15       cantDiv:=cantDiv+1;
16     end;
17   end;
18   esprimo := (cantDiv = 0);
19 end;
20
21 {
22  Evalua un rango de numeros y devuelve los numeros
23    primos dentro del rango.
24  @param id Identificador del proceso.
25  @param limInferior limite establecido para el valor
26                     inferior del intervalo.
27  @param limSuperior limite establecido para el valor
28                     superior del intervalo.
29 }
30 process type primos (
31       id : integer;
32       limInferior : integer;
33       limSuperior : integer;
34       var s : semaphore
```

```
35          );
36    var
37      i : integer; {Indice para recorrer}
38    begin
39      for i:=limInferior to limSuperior do
40      begin
41        if (esprimo(i)) then
42        begin
43          wait(s);
44          writeln('El proceso ',id,
45            ' encontro a: ', i);
46          signal(s);
47        end;
48      end;
49    end;
50  var
51    p1, p2, p3, p4 : primos;
52    s : semaphore;
53  begin
54    initial(s,1);
55    cobegin
56      p1 (1, 1, 25, s);
57      p2 (2, 26, 50, s);
58      p3 (3, 51, 75, s);
59      p4 (4, 76, 100, s);
60    coend;
61  end.
```

5 Codificación 4

CONSIGNA: **crear un algoritmo que realice el producto escalar de dos vectores de 10 elementos.**

Listing 7: TP1, Ejercicio 5

```
1  program tp1_ej5;
2  const
3    tamanio = 10;
4    MAX_RANDOM = 51;
5  type
6    vector = array[1..tamanio] of integer;
7
8  {Carga el vector arr con valores aleatorios}
9  procedure cargarVector(var arr : vector);
10 var
```

```
11    i: integer;
12  begin
13    Randomize;
14    for i:=1 to tamanio do
15    begin
16      arr[i] := Random(MAX_RANDOM);
17    end
18  end;
19
20  {Utilidad que permite imprimir el vector arr}
21  procedure imprimir(arr : vector; id : integer);
22  var
23    i: integer;
24  begin
25    write('V', id, ': [');
26    for i:=1 to tamanio-1 do
27    begin
28      write(arr[i],', ');
29    end;
30    write(arr[i],']');
31    writeln;
32  end;
33
34  {Realiza la parte de la multiplicacion}
35  process type multiplicar (
36        id : integer;
37        var vec1 : vector;
38        var vec2 : vector;
39        p_ini : integer;
40        p_fin : integer;
41        var pescalar : integer
42         );
43
44  var
45    suma, producto, i : integer;
46  begin
47    suma := 0;
48    for i:=p_ini to p_fin do
49    begin
50      producto := 0;
51      producto := vec1[i]*vec2[i];
52      suma := suma + producto;
53    end;
54    pescalar := pescalar + suma;
55  end;
56  var
```

```
57    p1, p2, p3, p4, p5 : multiplicar;
58    productoEscalar : integer;
59    vec1, vec2: vector;
60  begin
61    productoEscalar := 0;
62    cargarVector(vec1);
63    cargarVector(vec2);
64    imprimir(vec1, 1);
65    imprimir(vec2, 2);
66    cobegin
67      p1 (1, vec1, vec2, 1, 2, productoEscalar);
68      p2 (2, vec1, vec2, 3, 4, productoEscalar);
69      p3 (3, vec1, vec2, 5, 6, productoEscalar);
70      p4 (4, vec1, vec2, 7, 8, productoEscalar);
71      p5 (5, vec1, vec2, 9, 10, productoEscalar);
72    coend;
73    writeln('El producto escalar de los vectores
74      proporcionados es: ', productoEscalar);
75  end.
```

6 Investigar

Consignas:

- **¿Qué diferencia existe entre la multiprogramación y el multiproceso?**
- **Describa que es una *instrucción atómica* y que es *intercalacion***
- **¿Qué son los semáforos en la programación paralela? ¿Para qué sirven?**
- **¿Cuáles son sus instrucciones y para qué se utiliza cada una?**

6.1 Multiprogramación y Multiproceso

Ambos describen una forma de compartir el tiempo de computo de un sistema por programas, que en ultima instancia ayuda a dar una explicación conceptual de lo que significa concurrencia. La diferencia la expuesta en [Gortázar Bellas, et al, 2012] es la siguiente: La ***Multiprogramacion***, se da cuando los programas se ejecutan en un único procesador disponible y sus procesos internos comparten el tiempo de cómputo del procesador mencionado. Se habla de ***Multiproceso***, cuando el sistema en el cual se ejecuta el programa posee multiples procesadores, entonces es posible asignar diferentes procesos a diferentes procesadores.

6.2 ¿Qué es una *instrucción atómica* y que es la *intercalación*?

Se considera una *instrucción atómica* [Gortázar Bellas, et al, 2012] a aquella que se ejecuta completamente antes de que se ejecute ninguna otra instrucción de cualquier otro proceso del programa, una *intercalación* en un programa es una secuencia de ejecución de las instrucciones atómicas con las que cuente dicho programa, en el caso de que hayan 2 instrucciones atómicas ejecutadas por 2 procesos concurrentes, existiran 4 posibles intercalaciones.

6.3 ¿Qué son los semáforos en la programacion paralela?

Los semáforos son una herramienta que se destina para la sincronización de procesos, en PascalFC estos son un tipo abstracto de datos, y como tal tiene operaciones y estructuras de datos internos. Todo semáforo tiene un contador que toma valores positivos, y una lista de procesos asociados.

6.4 ¿Para qué sirven?

Estos sirven para garantizar que recursos en el sistema que deben ser utilizados por un proceso a la vez, efectivamente sean utilizados por un proceso a la vez, este conjunto de instrucciones que acceden a las areas mencionadas son denominados *sección crítica*.

6.5 ¿Cuales son sus instrucciones y para que se utiliza cada una?

Las operaciones que se pueden invocar sobre una variable de tipo semáforo son:

- `initial(s,v)` inicializa el contador del semáforo `s` al valor `v`. Este procedimiento solo se puede invocar una vez y debe ser llamado desde el programa principal. Debe ser inicializado antes de utilizarse.
- `wait(s)` Este procedimiento solo se puede invocar desde un proceso. Funcionamiento:
  - Si el contador `s` tiene un valor mayor que cero, el proceso continúa su ejecución y el valor del contador se decrementa en uno.
  - Si el contador del semáforo tiene el contador igual a cero, el proceso se queda bloqueado y se añade a la lista de procesos bloqueados del semáforo.
- `signal(s)` Este procedimiento solo se puede invocar desde un proceso. Funcionamiento:
  - Si no hay procesos bloqueados en el semáforo `s`, se incrementa el valor de contador en una unidad.
  - Si hay procesos bloqueados en el semáforo, se elige aleatoriamente a uno de ellos y se desbloquea para que continue su ejecución.

7 Aplicar semáforos a los ejercicios en la Sección 2 y la Sección 3

7.1 Ejercicio Sección 2

Listing 8: TP1, Ejercicio 2 (Semáforo)

```
1  program tp1_ej2_semaforo;
2
3  process type imprimir (
4        id : integer;
5        var s : semaphore
6            );
7  begin
8    wait(s);
9    writeln( 'ID de proceso: ', id);
10   signal(s);
11 end;
12 var
13   p1, p2 : imprimir;
14   s : semaphore;
15 begin
16   initial(s, 1);
17   cobegin
18     p1 (1, s) ;
19     p2 (2, s) ;
20   coend;
21 end.
```

7.2 Ejercicio Sección 3

Listing 9: TP1, Ejercicio 3 (Semáforo)

```
1  program tp1_ej3_semaforo;
2  {
3   Imprime una cantidad determinada de numeros
4   pares que se encuentren entre 0 y un limite
5   establecido
6   @param id Identificador del proceso
7   @param cantidad cantidad de numeros pares a
8     imprimir
9   @param limSuperior limite establecido para
10    el pool de numeros escogidos
11 }
12 process type pares (
```

```
13        id : integer;
14        cantidad : integer;
15        limSuperior : integer;
16        var s : semaphore
17         );
18  var
19    cant : integer;
20    par : integer;
21  begin
22    Randomize;
23    cant := 0;
24    par := 0;
25    while cant < cantidad do
26    begin
27      par:=Random(limSuperior)+1;
28      if par mod 2 = 0 then
29      begin
30        wait(s);
31        writeln( 'El proceso ',
32            id, ' imprime ',par);
33        signal(s);
34        cant := cant + 1;
35          par:=0;
36      end;
37    end;
38  end;
39  var
40    p1, p2, p3 : pares;
41    s : semaphore;
42  begin
43    initial(s, 1);
44    cobegin
45      p1 (1, 5, 100, s) ;
46      p2 (2, 5, 40, s) ;
47      p3 (3, 10, 35, s);
48    coend;
49  end.
```

8 Inconvenientes de código en el `Anexo 9.1`

Consigna:

- **Explique brevemente las inconsistencias que posee el programa en el `Anexo 9.1`**
- **Solucione el inconveniente utilizando semáforos**

8.1 Inconsistencias en `Anexo 9.1`

El segundo punto deja al descubierto el inconveniente mayor que posee el código, el hecho que el ejercicio solicite que se solucione el problema con semaforos implica un caso de condicion de carrera en el acceso a la variable compartida `count`.

Ambos procesos estan compartiendo el espacio de memoria con la variable global mencionada y ninguno de los procesos asegura la manipulacion de la misma mediante la solicitud de un cerrojo y consecuentemente tampoco se solicita que se ceda dicho cerrojo. Esto causaría problemas de inconsistencia con lo que ven los dos procesos, porque al momento de acceder cargar al registro la variable para modificación en un proceso puede que tambien se este modificando en el otro, esto llevaria a que no se tome uno de los cambios, y por tanto se pierda información.

8.2 Solucion mediante semáforos

Listing 10: TP1, Código de anexo (Semaáforo)

```
1  program gardens1;
2  var
3    count: integer;
4    s : semaphore;
5  process turnstile1(var s:semaphore);
6  var
7    loop: integer;
8  begin
9    for loop := 1 to 5 do
10   begin
11     wait(s);
12     count := count + 1;
13     signal(s);
14   end;
15 end; (* turnstile1 *)
16
17 process turnstile2(var s:semaphore);
18 var
19   loop: integer;
20 begin
21   for loop := 1 to 5 do
22   begin
23     wait(s);
24     count := count + 1;
25     signal(s);
26   end;
27 end; (* turnstile2 *)
```

```
28
29 begin
30   initial(s,1);
31   count := 0;
32   cobegin
33     turnstile1(s);
34     turnstile2(s);
35   coend;
36   writeln('Total admitted: ',count)
37 end.
```

9 Anexos

9.1 Programa 1

Listing 11: TP1, Código de anexo

```
1  program gardens1;
2  var
3    count: integer;
4  process turnstile1;
5  var
6    loop: integer;
7  begin
8    for loop := 1 to 5 do
9    begin
10     count := count + 1;
11   end;
12 end; (* turnstile1 *)
13   process turnstile2;
14 var
15   loop: integer;
16 begin
17   for loop := 1 to 5 do
18   begin
19     count := count + 1;
20   end;
21 end; (* turnstile2 *)
22 begin
23   count := 0;
24   cobegin
25     turnstile1;
26     turnstile2;
27   coend;
28   writeln('Total admitted: ',count)
29 end.
```

Referencias

[Gortázar Bellas, et al, 2012] Gortázar Bellas, Francisco; Martínez Unanue, Raquel; Fresno Fernández, Victor. *Lenguajes de Programación y Procesadores - Capítulo 3.5*. Editorial Universitaria Ramon Areces, Madrid, 2012. ISBN: 9788499610702.