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);
```

```
        begin
                writeln( 'ID de proceso: ', id);
        end;
var
        p1, p2 : print ;
begin
        cobegin
                p1 (1) ;
                p2 (2) ;
        coend;
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)

```
program tp1_ej2;

process type print (id : integer);
        begin
                write( 'ID de proceso: ', id);
        end;
var
        p1, p2 : print ;
begin
        cobegin
                p1 (1) ;
                p2 (2) ;
        coend;
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

```
program tp1_ej3;
{
 Imprime una cantidad determinada de numeros
```

```
 pares que se encuentren entre 0 y un limite
 establecido
 @param id Identificador del proceso
 @param cantidad cantidad de numeros pares a
  imprimir
 @param limSuperior limite establecido para
  el pool de numeros escogidos
}
process type pares (
                        id : integer;
                        cantidad : integer;
                        limSuperior : integer
                    );
        var
                cant : integer;
                par : integer;
        begin
                Randomize;
                cant := 0;
                par := 0;
                while cant < cantidad do
                begin
                        par:=Random(limSuperior)+1;
                        if par mod 2 = 0 then
                        begin
                          writeln( 'El proceso ',
                             id, ' imprime ',par);
                          cant := cant + 1;
                          par:=0;
                        end;
                end;
        end;
var
        p1, p2, p3 : pares;
begin
        cobegin
                p1 (1, 5, 100) ;
                p2 (2, 5, 40) ;
                p3 (3, 10, 35);
        coend;
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

```
program tp1_ej4;
{
 @return true si el valor de numero es primo
 @return false si el valor de numero no es primo
}
function esprimo(numero : integer):boolean
var
        cantDiv : integer;
        i : integer;
begin
        for i:=2 to numero-1 do
        begin
                if (numero mod i = 0) then
                begin
                        cantDiv:=cantDiv+1;
                end;
        end;
        esprimo := (cantDiv = 0);
end;

{
 Evalua un rango de numeros y devuelve los numeros
   primos dentro del rango.
 @param id Identificador del proceso.
 @param limInferior limite establecido para el valor
                    inferior del intervalo.
 @param limSuperior limite establecido para el valor
                    superior del intervalo.
}
process type primos (
                        id : integer;
                        limInferior : integer;
                        limSuperior : integer;
                        var s : semaphore
```

```
                    );
        var
                i : integer; {Indice para recorrer}
        begin
                for i:=limInferior to limSuperior do
                begin
                        if (esprimo(i)) then
                        begin
                                wait(s);
                                writeln('El proceso ',id,
                                        ' encontro a: ', i);
                                signal(s);
                        end;
                end;
        end;
var
        p1, p2, p3, p4 : primos;
        s : semaphore;
begin
        initial(s,1);
        cobegin
                p1 (1, 1, 25, s);
                p2 (2, 26, 50, s);
                p3 (3, 51, 75, s);
                p4 (4, 76, 100, s);
        coend;
end.
```

5 Codificación 4

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

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.