Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ciencias de la Computación e Informática

CI-0123

Grupo 01

Proyecto
Fase 1
Comunicación punto a punto

Profesores:

Gabriela Barrantes

Francisco Arroyo

Elaborado por:

Luis Carvajal Rojas B31494

Marcial Carrillo Vega A81379

Diego Contreras Estrada B62074

Andrés Navarrete Boza B55017

Índice

1. Problema

El problema se expresa en el enunciado de la Fase 1 del proyecto, el cual consta de 2 ejercicios:

- **2.1** Levantar (de forma independiente) dos procesos: uno que lea una oración por teclado (llamémoslo "lector") y se la envíe al otro, que cuente la cantidad de palabras (llamémoslo "contador") El contador imprime tanto la oración como el número de palabras.
- **2.2** Ahora el "lector" deberá almacenar la oración y esperar una respuesta del "contador" con el número de palabras. - Una vez que la obtiene (la respuesta), debe almacenarla junto con el número de palabras. Esto no puede bloquear el ingreso. Una vez leída una oración, se debe leer la siguiente, independientemente de si el "contador" haya respondido o no. - Si la oración consiste únicamente del número "1", debe imprimir todos los pares (oración, palabras) que se hayan almacenado al momento. El "contador", por su parte, debe ser capaz de manejar las solicitudes continuas y responderle al "lector"

2. Parte 2.1

2.1. Especificaciones y decisiones

1. Se trabajó con C++ utilizando MPI para trabajar con procesos de la máquina.
2. Se definió como palabra cualquier conjunto de caracteres distintos del espacio.
3. No se controló la entrada al usuario.
4. Se usaron dos procesos, un lector y un contador.
5. Se usaron las funciones **MPI_Send** y **MPI_Recive**.

2.2. Pseudocódigo

```
1 Entran 2 procesos
2   if proceso_id == 0{ // El proceso 0 lee
3     Pedir al usuario una oracin
4     MPI_Send(Oracin del usuario.
```

```
  } else { // El proceso 1 cuenta las oraciones
    MPI_Recive(Oracin del usuario)
    Contar
    Imprimir resultado
  }
```

2.2.1. Código fuente

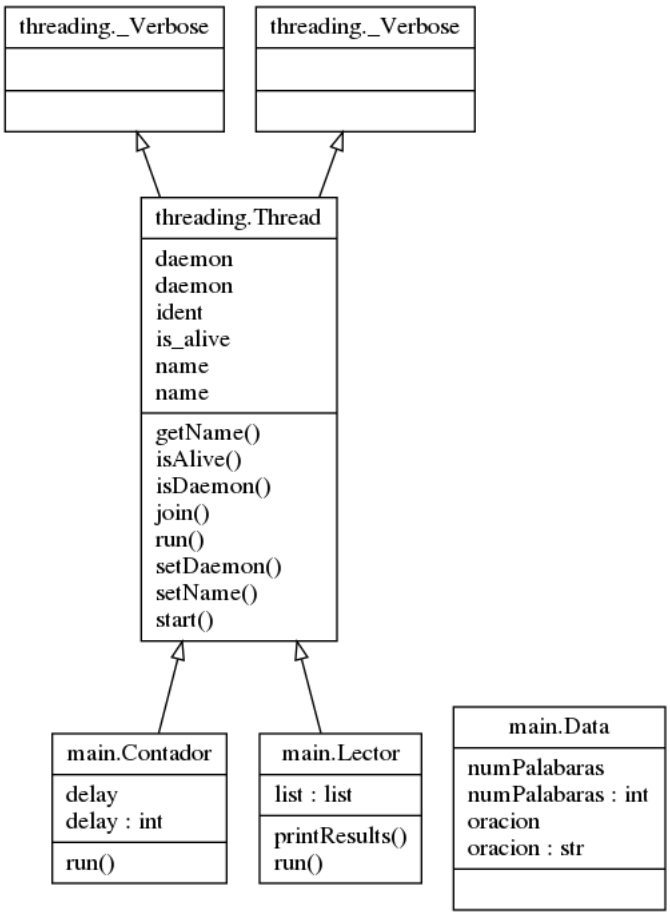
```
#include <mpi.h>
#include <iostream>
#include <cstdio>
#include <stdio.h>
#include <string.h>

using namespace std;

int main(int argc, char **argv)
{
    MPI_Init(NULL, NULL);
    char sentence[1000];

    int totalWords = 0;
    int process_rank, total_processes;
    MPI_Comm_rank(MPI_COMM_WORLD, &process_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &total_processes);

    if(process_rank != 0) {
        MPI_Recv(sentence, 1000, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
    MPI_STATUS_IGNORE);


  for(int i = 0; i < strlen(sentence); ++i)
            if(sentence[i] == ' ' && i > 0 && sentence[i - 1] != ' ')
                totalWords++;
        printf("La oracion es: %stotalWords: %d\n",sentence,totalWords+1);
    }
    else { // Este es el proceso 0
        printf("Ingrese una oracion:\n");
        fgets(sentence, 1000, stdin);
        MPI_Send(sentence, 1000, MPI_CHAR, 1, 0, MPI_COMM_WORLD);

    }
    MPI_Finalize();
    return 0;
}
```

3. Parte 2.2

3.1. Especificaciones y decisiones

1. Se usó Python con la librería de multithreading.
2. Se definió como palabra cualquier conjunto de caracteres distintos del espacio.
3. Se usaron listas como estructuras de datos de almacenamiento.
4. El *delay* se ingresa por consola.
5. Si el lector se encuentra leyendo y resulta ser el 1 y a su vez también tiene información de contador que no ha sido almacenada esta información se almacena y se considera la última y acaba el programa.
6. Se usaron dos hilos, uno llamado lectura y cuenta.

3.2. Máquina de estado



3.3. Pseudocódigo

```
Pseudocdigo lector:
    Mientras no reciva un 1 y enter.
        Se lee una oracion.
            Almacena la oracin en una estructura de datos privada.
            Si el contador puede recibir una oracin .
                Enva , por medio de una estructura de datos compartida, la oracin .
            Si el contador ha dado una repuesta.
                Se almacena la misma.

Pseudocdigo Contador:
    Espera si hay una oracin en la estructura de datos.
        Cuenta las palabras en la oracion.
        Se crea un retraso configurable
        Almacena la respuesta en la estructura compartida.
```

3.4. Requerimientos adicionales

1. Compilador de Pyhton 3
2. Función *Delay*

3.5. Manual de Usuario

1. Ir a la carpeta donde se encuentra el código fuente.
2. Abrir una terminal.
3. Escribir: *python3 main2.py (cantidad del delay)*

3.6. Código fuente

```
import time
import threading
import datetime
import sys

class Data:
    oracion = ""
    numPalabaras = 0
    def __init__(self,oracion,numPalabaras):
        self.oracion = oracion
        self.numPalabaras = numPalabaras

class Lector(threading.Thread):
    list = []
    def __init__(self):
        threading.Thread.__init__(self)
        print("Im lector")

    def printResults(self):
        for data in self.list:
            if data.numPalabaras != -1:
                print(" %s \t Palabras: %d" % (data.oracion,data.numPalabaras))
            else:
                print(" %s \t Palabras: no procesada" % (data.oracion))

    def run(self):
        index = 0
```

```
28          lastIdexSend = 0
29          while True:
30
31           oracion= input("inserte la oracion: ")
32
33           if working[0] == 2:
34                   temp = self.list[lastIdexSend]
35                   temp.numPalabaras = resultadoContador[0]
36                   ##sem.release()
37                   lastIdexSend += 1
38                   working[0] = 0
39
40           if len(oracion) == 1:
41            if oracion[0] == '1':
42                 workDone[0]=True
43                 sem.release()
44                 self.printResults()
45                 break
46
47           else:
48                 self.list.append(Data(oracion, -1))
49
50                 if working[0] == 0: ##Si esta libre
51                      data = self.list[index]
52                      buffer[0] = data.oracion
53                      sem.release()
54                      index += 1
55
56  class Contador (threading.Thread):
57      delay = 0
58      def __init__(self, final_delay):
59           print("Im counter")
60           self.delay = final_delay
61           threading.Thread.__init__(self)
62
63      def run(self):
64           while True:
65                ##print("waiting on semaphore!")
66                sem.acquire()
67
68                if workDone[0]:
69                     ##print("workDone set to TRUE")
70                     break
71
72                working[0] = 1
73
74                resultado = len(buffer[0].split(" "))##Cuenta las palabras
75
76                time.sleep(int(delay))
```

```
77                  resultadoContador[0] = resultado
78                  working[0] = 2
79
80  if __name__ == '__main__':
81
82      sem = threading.Semaphore(0)
83      buffer = []
84      resultadoContador = []
85      working = []
86      workDone = []
87
88      delay = sys.argv[1]
89      working.append(0) ## 0 = not working ## 1 = working ## 2 finished counting
         the word
90      buffer.append("")
91      resultadoContador.append(0)
92      workDone.append(False)
93      # Create new threads
94      thread1 = Lector()
95      thread2 = Contador(delay)
96
97      # Start new Threads
98      thread1.start()
99      thread2.start()
100
101     thread1.join()
102     ##print("thread1 is done")
103     thread2.join()
104     ##print("thread2 is done")
105     print ("Exiting the Program!!!")
```