Práctica 6: Generador aleatorio de nombres.

Karina Flores García y Gabriel Alejandro Herrera Gandarela

***Abstract*— In this practice, the star architectural pattern is implemented in order to distribute the tasks among the different processes. The main task is to create names randomly.**

I. OBJETIVOS.

- Aplicar los métodos de comunicación colectiva.
- Implementar el patrón arquitectónico de Estrella.
- Realizar un procesamiento masivo de datos con técnicas de HPC.

II. INTRODUCCIÓN.

El tópico más urgente y exitoso en arquitectura de software en los últimos cuatro o cinco años es, sin duda, el de los patrones (patterns), tanto en lo que concierne a los patrones de diseño como a los de arquitectura. Inmediatamente después, en una relación a veces decomplementariedad, otras de oposición, se encuentra la sistematización de los llamados estilos arquitectónicos.

Patrón arquitectónico de estrella
Es de tipo Emborro ssingly parallel (Paralelismo engorroso) ya que los procesadores no necesitan comunicación entre ellos. Cuenta con un maestro y un grupo de esclavos.
MAESTRO: asigna tareas a esclavos.
Este patrón es óptimo para algoritmos de datos.

III. MODELO.

IV. ANÁLISIS DEL PROBLEMA.

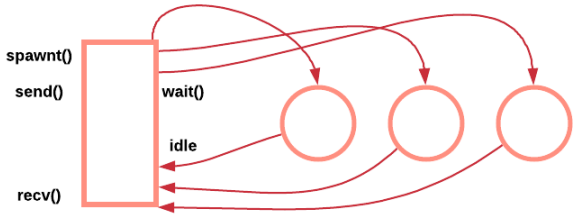
La problemática de esta práctica consta en generar 20000 nombres aleatorios a través de las combinaciones de nombres y apellidos de una base de datos y generar el registro de cada uno en un archivo de extensión csv.
Para incluir una base de datos, hacemos uso de dos arreglos de 100 elementos cada uno, con distintos nombres y apellidos. La resolución del problema de forma paralela la realizaremos con ayuda del patrón arquitectónico de estrella.
Para empezar a realizar el código, debemos de entender cómo es el patrón arquitectónico *Estrella*. Basicamente, el patrón consta de un maestro y al menos un esclavo; un maestro se encargará de enviar información a cada uno de los esclavos, estos se encargarán de recibirla y hacer uso de ella, cuando terminen su tarea, enviarán información al maestro.
El envío y recepción de información se realizará con la ayuda de las funciones *send()* y *recv()* de MPI.

V. IMPLEMENTACIÓN

A. Desarrollo

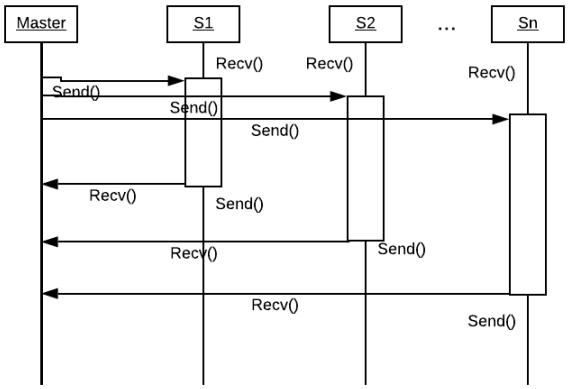
Entendiendo la lógica del algoritmo, el maestro manda el número de nombres que debe realizar cada esclavo, y cada esclavo, tendrá una copia de cada uno de los arreglos para ir generando los nombres y escribir en el archivo con extensión *csv*. Al principio, se pensaba en que el maestro mandara los dos arreglos a cada esclavo y cada uno lo reciba para ir generando los nombres, sin embargo, los arreglos de cadenas de caracteres necesitan de un trato especial, por lo que, puede traer complicaciones para la ejecución en paralelo, es por eso que cada esclavo tenga una copia de los arreglos y sobre esos, tomé la información.
Cuando termine la tarea cada esclavo, mandará un dato al maestro y éste sabrá que ya acabó. Algo que no se mencionó anteriormente es que el maestro desde que inicia, empieza a contar el tiempo, y para saber hasta dónde contar, si sabemos el número de esclavos y cada uno regresa un dato, entonces, no parará de contar el tiempo hasta que el número de datos recibidos sea igual al número de esclavos.

B. Diagrama de patrón arquitectónico

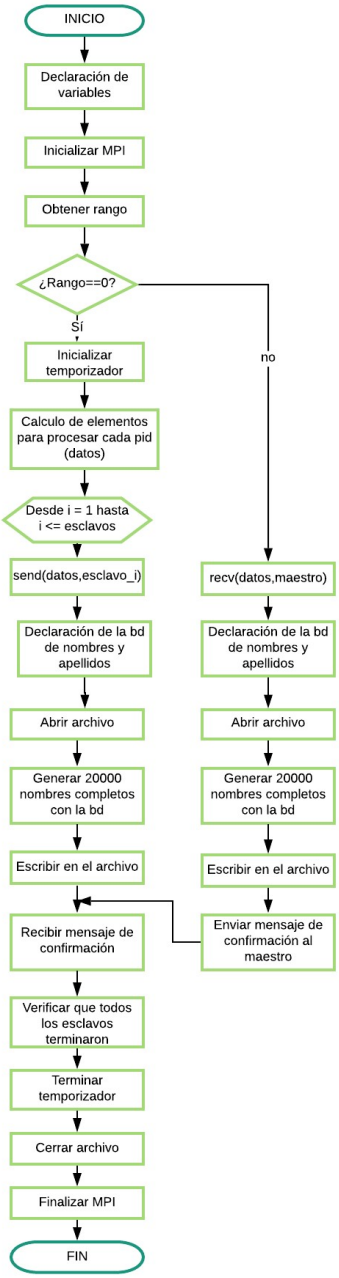


- Si es dinámico, crea 3 nodos con spawnt().
- Si es estático no necesita spawnt().
- **idle:** No hace nada en un proceso de tiempo.

C. Diagrama UML de secuencia



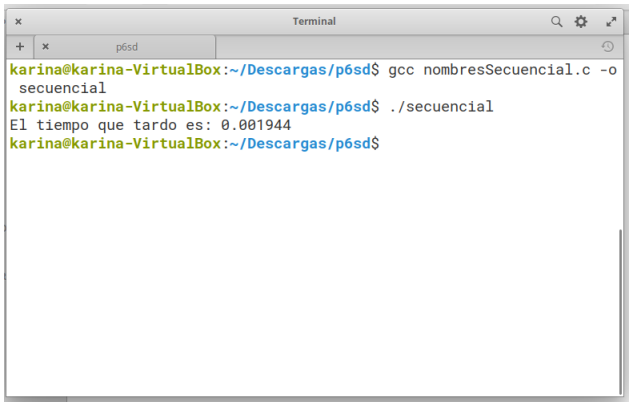
D. Flujo del programa



VI. CÓDIGO FUENTE

A. Código secuencial

```
1   FILE * archivo = fopen("nombres.csv","w");
2   if(archivo == NULL){
3     perror("Error en la apertura del archivo");
4     return 1;
5   }else{
6     tiempo_inicio = clock();
7     //Crea 20000 nombres aleatorios y los guarda en
       un archivo
8     for(int i=0;i<total_nomb;i++){
9       char nombreCompl[40];
10      numero = rand() % 50;
11      strcpy(nombreCompl, nombres[numero+2]);
12      strcat(nombreCompl,apellidos[numero+10]);
13      strcat(nombreCompl,apellidos[numero-4]);
14      strcat(nombreCompl,"\n");
15      fputs(nombreCompl,archivo);
16    }
17    fclose(archivo);
18    printf("El tiempo que tardo es: %f\n", (clock()
      -tiempo_inicio)/(double)CLOCKS_PER_SEC);
19  }
20 }
```



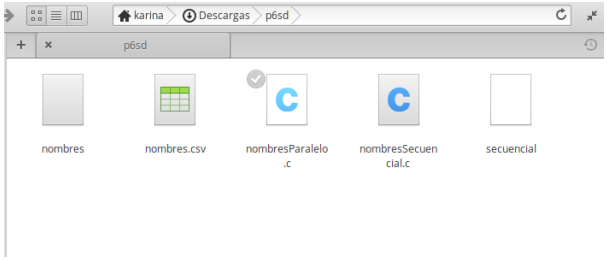
B. Código paralelo.

```
1     if(pid==0){
2       tiempo_inicio = MPI_Wtime();
3         int datos = 20000/(size-1);
4         tag = 0;
5         for(int i =1; i<=size-1;i++){
6           MPI_Send(&datos, 1,MPI_INT, i, tag,
      MPI_COMM_WORLD);
7         }
8     }else{
9       tag = 0;
10      MPI_Recv(&datos, 1,MPI_INT, 0, tag,
      MPI_COMM_WORLD, &info);
11      //Se omite todo el contenido de los arreglos
      para evitar c digo de declaraciones
12      //Arreglo de 100 nombres para elegir
13      char *nombres[]={"Karina ","Alejandro ","
      Gabriel "...};
14      //Arreglo de 100 apellidos para elegir
15      char *apellidos[]={"Flores ","Garc a ","
      Herrera ","Gandarela "...};
16      if(archivo == NULL){
17        perror("Error en la apertura del archivo");
18        return 1;
19      }else{
20        for(int i=0;i<datos;i++){
21          char nombreCompl[40];
22          numero = rand() % 50;
23          strcpy(nombreCompl, nombres[numero+2]);
24          strcat(nombreCompl,apellidos[numero+10]);
25          strcat(nombreCompl,apellidos[numero-4]);
26          strcat(nombreCompl,"\n");
27          fputs(nombreCompl,archivo);
28        }
29      }
30      confirmacion = 1;
31      MPI_Send(&confirmacion,1,MPI_INT,0,tag,
      MPI_COMM_WORLD);
32    }
33
34    if(pid==0){
35      suma = 0;
36      for(int i =1; i<=size-1;i++){
37        MPI_Recv(&confirmacion, 1,MPI_INT, i, tag,
      MPI_COMM_WORLD, &info);
38        suma = suma + confirmacion;
39        if(suma==(size-1)){
```

```
        tiempo_fin = MPI_Wtime();
        tiempo_total = tiempo_fin - tiempo_inicio;
        printf("El tiempo de ejecuci n es: %f\n",
    tiempo_total);
      }
    }
  }
  fclose(archivo);
  MPI_Finalize();
  return 0;
}
```

```
Terminal
p6sd
karina@karina-VirtualBox:~/Descargas/p6sd$ mpicc nombresParalelo.c -o
 nombres
karina@karina-VirtualBox:~/Descargas/p6sd$ mpirun -np 3 ./nombres
El tiempo de ejecución es: 0.001706
karina@karina-VirtualBox:~/Descargas/p6sd$
```

Generación de ejecutables y archivo csv



Nota: Los tiempos mostrados en las capturas de pantalla pueden variar respecto a la tabla de comparación.

VII. COMPARACIÓN Y MÉTRICAS.

A. *Métricas en secuencial*

El tiempo de ejecución que tardó el algoritmo de forma secuencial es de: ***0.0533 [s]***

B. *Métricas en paralelo.*

Runtime

TABLE I

MÉTRICA DE RENDIMIENTO PARA *Runtime*.

Número de procesadores	Runtime (s)
2	0.0671
3	0.0730
4	0.1026
5	0.1196
6	0.1535
7	0.1444
8	0.1740
9	0.1346
10	0.1836
100	0.6678

El runtime aumenta conforme los procesadores aumentan.

Cost Factor

TABLE II

MÉTRICA DE RENDIMIENTO PARA *Costfactor*.

Número de procesadores	Costfactor (s)
2	0.1342
3	0.219
4	0.4104
5	0.598
6	0.921
7	1.0108
8	1.392
9	1.2114
10	1.836
100	66.78

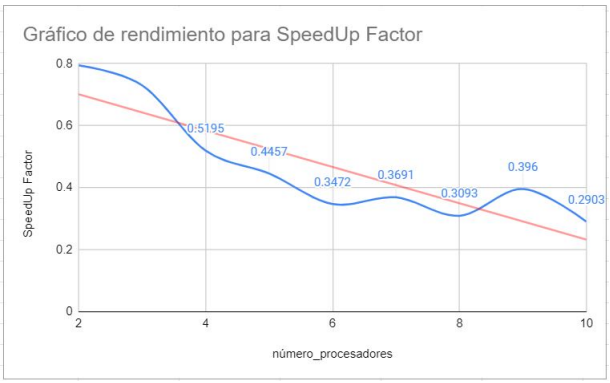
El costfactor aumenta conforme los procesadores aumentan.

TABLE III

MÉTRICA DE RENDIMIENTO PARA *Speedup Factor*.

Número de procesadores	Speedup Factor (s)
2	0.7943
3	0.7301
4	0.5195
5	0.4457
6	0.3472
7	0.3691
8	0.3063
9	0.3960
10	0.2903
100	0.0798

Speedup Factor



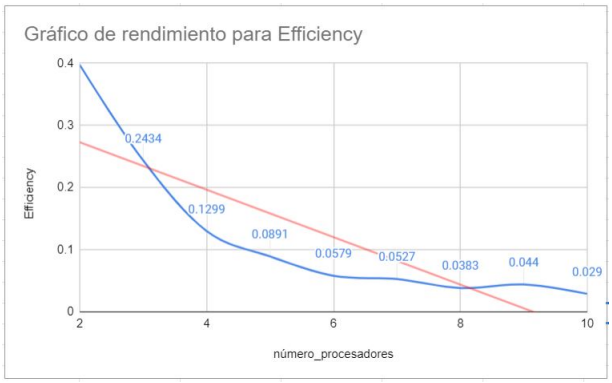
El speedup disminuye conforme los procesadores aumentan.

Efficiency

TABLE IV

MÉTRICA DE RENDIMIENTO PARA *Efficiency*.

Número de procesadores	Efficiency (s)
2	0.3972
3	0.2434
4	0.1299
5	0.0891
6	0.0579
7	0.0527
8	0.0383
9	0.0440
10	0.0290
100	0.0008



El efficency disminuye conforme los procesadores aumentan.

VIII. CONCLUSIONES

Se cumplieron los objetivos de la práctica debido a que se implementó correctamente el patrón arquitectónico *Estrella*, sin embargo, en la práctica se observó que en ocasiones la implementación de un programa en paralelo no suele ser lo más conveniente, ya que, como se observó en las métricas de rendimiento, aumenta linealmente el tiempo de ejecución conforme aumenta el número de procesadores ejecutando las tareas. Observando la diferencia de tiempo entre la ejecución secuencial y la paralela; suele suponerse que es mejor tener un programa en paralelo que genere 20000 nombres, sin embargo, esto no es válido en este caso debido a que el tiempo de ejecución en secuencial es menor que el runtime en paralelo, aun cuando el número de procesadores en paralelo es el mínimo.

También podemos observar en la métrica de rendimiento *Efficency* que a mayor número de procesadores ejecutando las tareas menor es la eficiencia del programa.

Por lo que concluimos que es vital analizar la solución de cada problema que se nos plantee, debido a que en muchas ocasiones paralelizar un programa no es lo más conveniente, ya que el tiempo de comunicación entre los procesos y la división de tareas puede resultar menos conveniente que tener un programa secuencial en el que un solo procesador se encargue de todo.

REFERENCES

[1] Ayala,J.A.(29 marzo 2020) Práctica 6:Generador aleatorio de nombres. Sistemas Distribuidos, 1.

[2] Reynoso, C. (marzo 2004) Estilos y Patrones en la estrategia de arquitectura de Microsoft. Buenos Aires, Argentina, 73.