TEC | Tecnológico de Costa Rica

Proyecto 1. Documento de diseño.
Protocolos de coherencia en sistemas multiprocesador

José Daniel Montoya Salazar
2016089548

Arquitectura de Computadores II
Profesor: Luis Barboza Artavia

Grupo 02

Entrega: 23 de octubre

II Semestre
2020

Índice.

Lista de requerimientos del sistema.

Requerimientos funcionales.

- Sistema en general.
    - Diseñar una plataforma de simulación en software de un sistema multiprocesador con una memoria compartida.
    - Implementar un protocolo de coherencia MOESI basado en monitoreo.
    - Cada procesador debe de generar instrucciones de manera aleatoria utilizando una distribución probabilística formal.
    - Los tipos de instrucciones a generar son *read*, *write* y *calc*.
    - En todo momento debe de visualizarse la acción que realiza cada núcleo, así como el bloque de memoria que está accediendo.
    - Se debe de implementar un sistema temporal modificable según:
        - Número de ciclos a ejecutar continuamente.
        - Ejecución continua.
        - Paso a paso.
    - Cuando el sistema esté en pausa, el usuario debe ser capaz de agregar una instrucción en un procesador específico y ejecutarse en el siguiente ciclo.

- Memoria principal.
    - Actualizar el contenido compartido de los bloques en escrituras.

- Caché.
    - Realizar el diseño con un único nivel de caché L1 local en los procesadores.
    - Las memorias caché son mapeadas de forma asociativa two-way.
    - Deben de contar con un sistema de coherencia.

- Interfaz gráfica.
    - El sistema debe visualizarse por medio de una interfaz gráfica con el fin de observar el comportamiento de cada uno de los procesadores al generar las instrucciones.
    - En todo momento debe de permitir la visualización de:
        - Los 16 bloques de memoria.
        - Los 4 bloques de cada procesador en todo momento.
        - Última instrucción generada, así como la última generada para cada procesador.

Requerimientos no funcionales.

- Sistema en general.
    - El sistema debe de contar con cuatro procesadores que funcionen de manera paralela.
    - Cada procesador será identificado por su número único.
    - Las instrucciones *calc* no requieren de memoria.
    - Los accesos a memoria deben de simular el fenómeno de pared de memoria.
    - Las instrucciones deben de contar con:
        - Identificador del procesador que lanza la instrucción.

        - Tipo de operación (*read*, *write*, *calc*).
        - En el caso de *write* se debe indicar la dirección de memoria (binario) y el dato a escribir (hexadecimal), respectivamente.
    - Una vez obtenida la instrucción, la dirección del bloque de memoria también será asignada de manera aleatoria con una distribución formal.
    - Se debe de poder pausar el sistema temporal.

- Memoria principal.
    - Será única y compartida a través de un único bus.
    - Cuenta con 16 bloques de 16 bits cada uno y deben ser representados en hexadecimal..
    - Debe de contar con una política de escritura establecida previamente.
    - Al iniciar la aplicación, el contenido de los bloques de memoria deben ser 0.

- Caché.
    - Las caché deben de empezar "frías".
    - El sistema de coherencia deberá de controlar la lectura del bloque correspondiente desde la memoria principal, y la escritura hacia las demás cachés.
    - Debe de contener:
        - Número de bloque.
        - Estado de coherencia según MOESI.
        - Dirección de memoria.
        - Datos de 16 bits en hexadecimal.

- Interfaz gráfica.
    - La temporización elegida debe permitir la correcta visualización de todas las acciones generadas.
    - Sólo se podrá agregar una instrucción en modo pausa y se verá reflejada luego de ejecutar la instrucción actual.

Elaboración de opciones de solución al problema.

Snoopy dentro de los procesadores.

En esta solución el monitoreo se realiza a través de una comunicación entre los procesadores como se muestra en la figura 1. Esto a nivel de software utiliza únicamente 6 hilos (4 procesadores, un reloj y la interfaz gráfica), de modo que se reducen bastante los costos de comunicación. Además, el procesador que lanza la instrucción es el encargado de revisar el monitoreo en el resto de cachés para asignar los estados correspondientes, esto aumenta las posibilidades de que al final del ciclo de reloj se hayan logrado completar todas las tareas necesarias. En esta solución se utiliza una política de escritura *write-through* de modo que el uso en el bus es bastante intensivo, sin embargo simplifica mucho la lógica de los estados. El reloj a implementar es una variable de tipo entero compartida que se incrementará cada cierto tiempo y se pretende utilizar el lenguaje de programación *Python* pues brinda herramientas de programación orientada a objetos y manejo de hilos útiles.

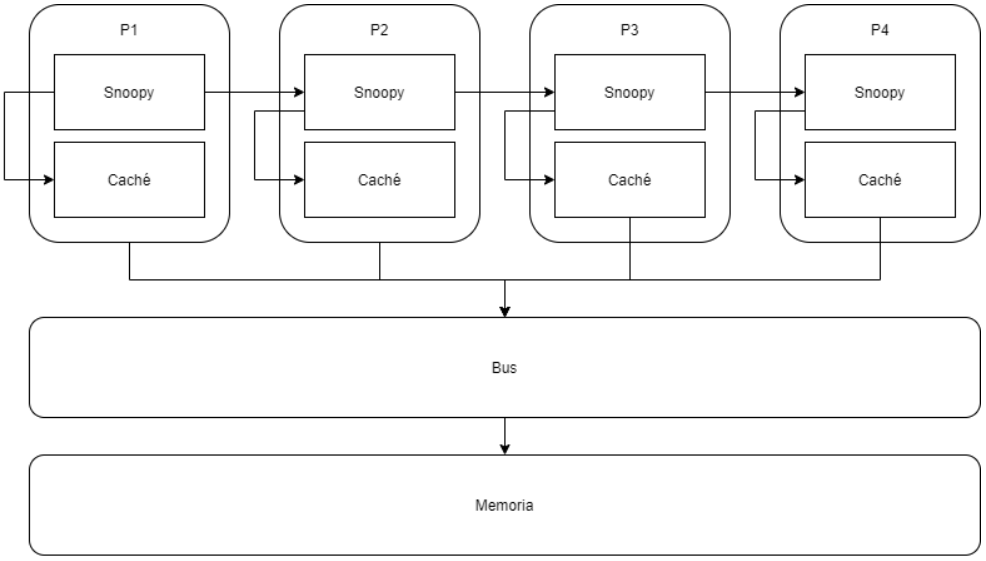

Figura 1. Diagrama de la solución 1 al problema.

Snoopy fuera de los procesadores.

En esta solución la política de escritura también será *write-through*, esto con reducir la complejidad de los estados del protocolo MOESI. En este caso el monitoreo se realiza en el bus, tal como lo dicta la teoría, sin embargo esto aumenta a 10 la cantidad de hilos necesarios (4 procesadores, 4 snoopy tags, un reloj y la interfaz gráfica), de modo que es un diseño ambicioso para ejecutar en una computadora de hogar o personal estándar. El ciclo de reloj debería de ser largo para garantizar que todos los hilos pudieran procesar los mensajes enviados a través del bus y los costos de comunicación entre los procesadores aumentan, además de la presión sobre los recursos compartidos como el bus. Aún así esta solución garantiza una aproximación bastante real al estudiado en clase. El reloj a implementar es una variable de tipo entero compartida que se incrementará cada cierto tiempo y se pretende utilizar el lenguaje de programación *Python* pues brinda herramientas de programación orientada a objetos y manejo de hilos útiles.

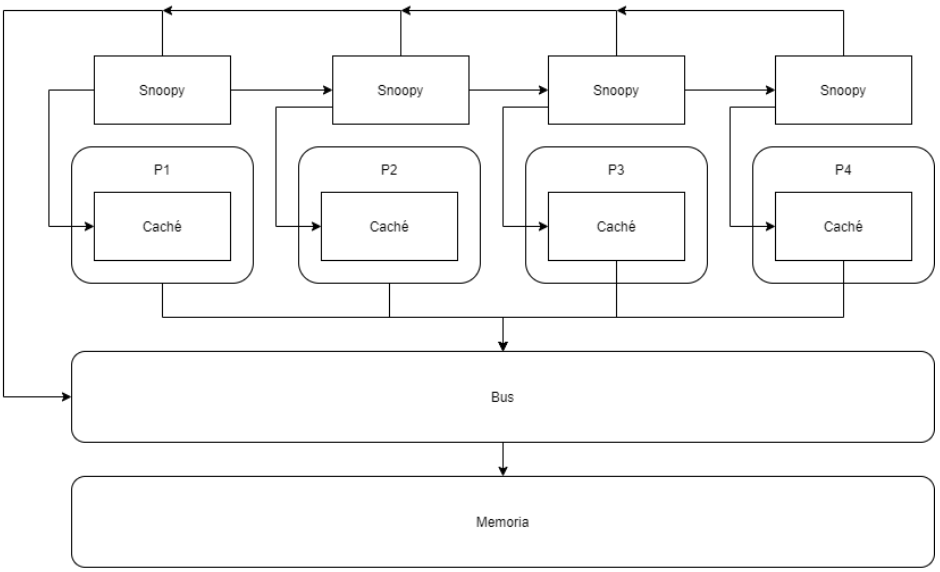

Figura 2. Diagrama de la solución 2 al problema.

Comparación de opciones de solución.

Snoopy dentro	Snoopy fuera
Monitoreo a través de una comunicación entre procesadores, de la que se encarga el procesador dueño de la instrucción.	Monitoreo en el bus por parte de todos los *snoopy tags*.
Menores costos computacionales, pues se utilizan solo 6 hilos.	Mayores costos computacionales pues se utilizan 10 hilos.
Menor presión en los recursos compartidos.	Mayor presión en los recursos compartidos.
Aproximación menos apegada a la teoría.	Corresponde a la teoría.
Ciclos de reloj más cortos por lo que la ejecución será más rápida.	Ciclos de reloj más largos, necesarios para garantizar la correcta lectura de señales por parte de todos los hilos.
Utiliza *write-through.*	Utiliza *write-through.*
Uso de *Python* como lenguaje de programación para construir el sistema.	Uso de *Python* como lenguaje de programación para construir el sistema.

Selección de la propuesta final.

Se opta por la opción 1, pues se considera que corresponde con la propuesta más acertada en cuanto a tiempo de desarrollo y recursos de hardware disponibles (computadora en la que se debe desarrollar el sistema). Ya que esta solución disminuye los costos computacionales de utilizar un *busy-waiting* en los recursos compartidos y además es la que brinda las mayores posibilidades de cumplir con los requerimientos de interfaz gráfica planteados, pues así la interfaz deberá de competir menos por los recursos, en comparación a la solución 2. Sin embargo, se reconoce que no es la solución más fiel a la teoría, pero esto no se considera un factor relevante pues gracias a los ciclos de reloj artificiales a implementar, igual se percibirá el fenómeno de la pared de memoria y la reducción de este efecto que ocasiona contar con cachés.

Implementación del diseño.

Descripción del protocolo diseñado.

El uso de *write-through* como política de escritura, implica que se puede prescindir de los estados de *Modify* e *Invalid* pues la memoria y las caché están siempre actualizadas. Esto también quiere decir que, el estado de *Owned* debe de asignarse a alguna caché en el mismo instante en que se detecta otra lectura o escritura de la etiqueta en cuestión, sin embargo como se mencionó anteriormente, los costo de comunicación son mucho mayores. Así, en la figura 3 se muestran las transiciones permitidas por el sistema.

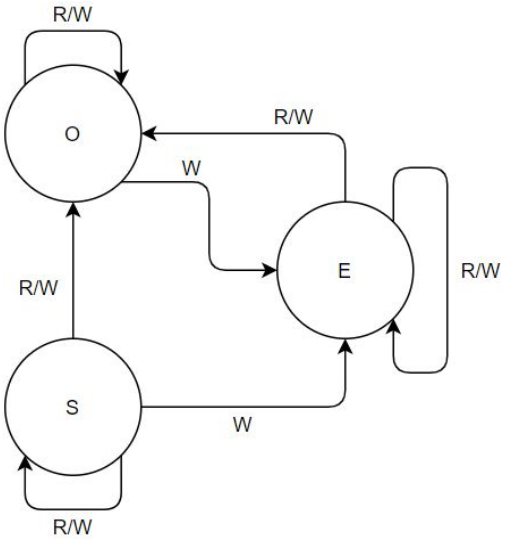

Figura 3. Transiciones permitidas en el protocolo de coherencia de caché implementado.

De modo que cuando se realiza una lectura/escritura de un dato, este se carga en caché, y si nadie más lo tiene, se le asigna el estado de *Exclusive*. Cuando otro procesador intenta hacer una lectura/escritura de este dato, la caché con el estado *Exclusive* es la encargada de brindar el mismo, así este pasa al estado de *Owned* y la caché solicitante a *Shared.* Si se sobreescribe el dato en la caché *Owned*, la caché *Shared* pasa a *Exclusive* y viceversa, a no ser que haya tres caché en estado *Shared* en este caso el estado *Owned* se otorga aleatoriamente a alguna de ellas.

Diagrama de bloques del modelo sistema multiprocesador.

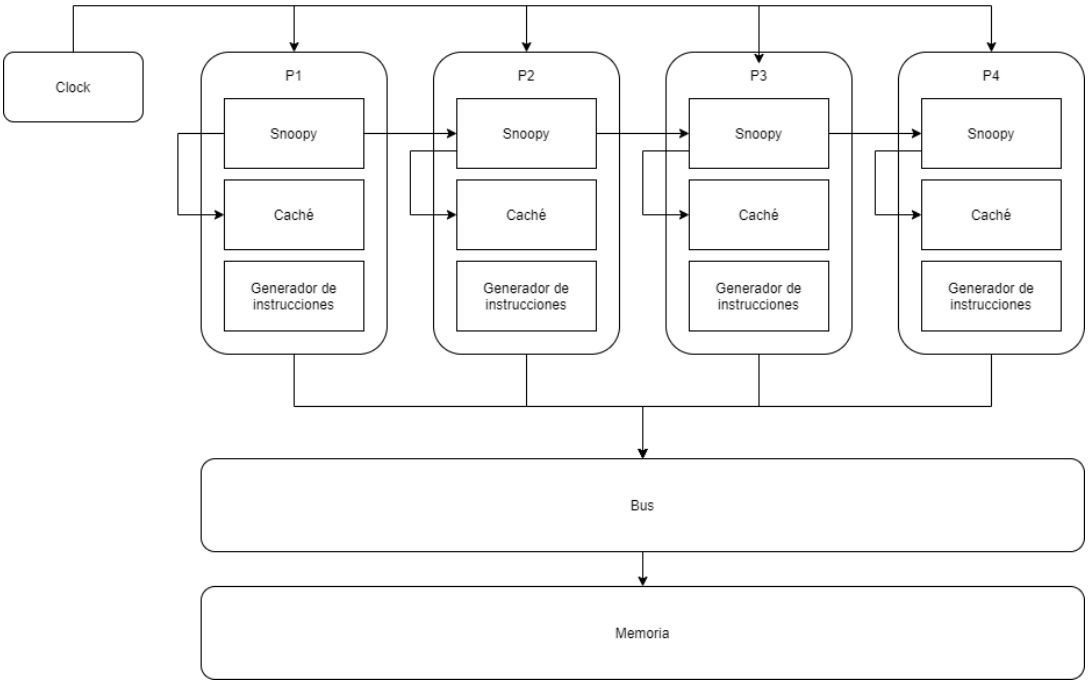

Figura 4. Diagrama de bloques del modelo sistema multiprocesador.

Diagrama de bloques del computador.

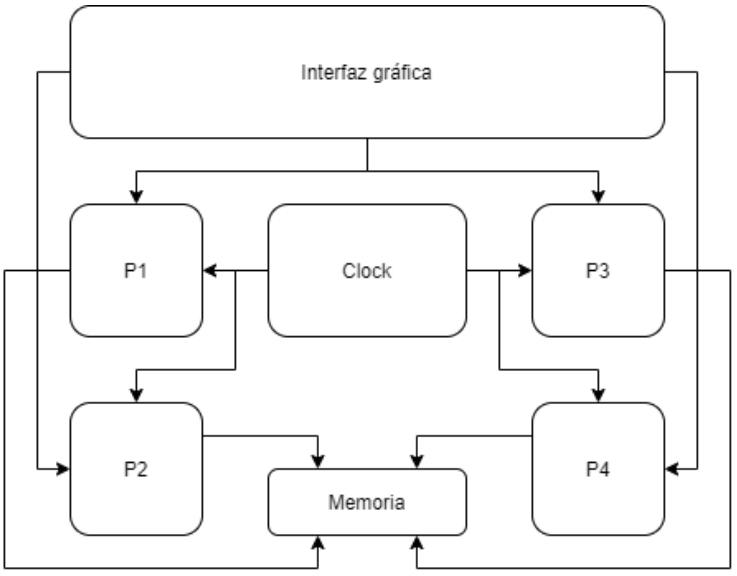

Figura 5. Diagrama de bloques del computador.

Diseño del software.

- **Diagramas.**

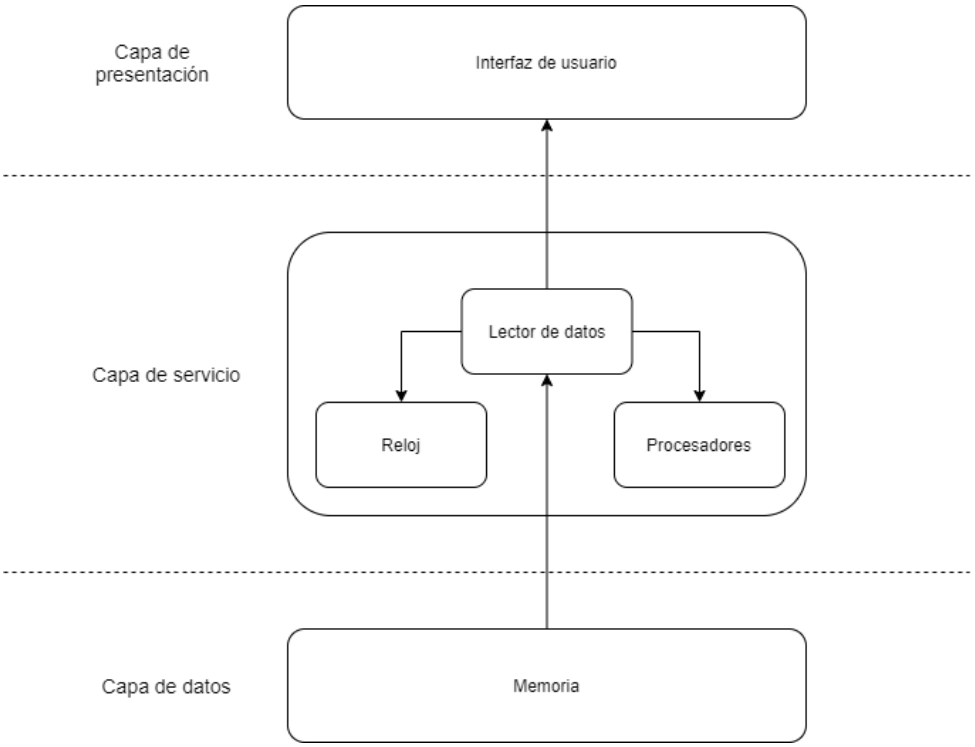

Figura 6. Diagrama de arquitectura del software construido.

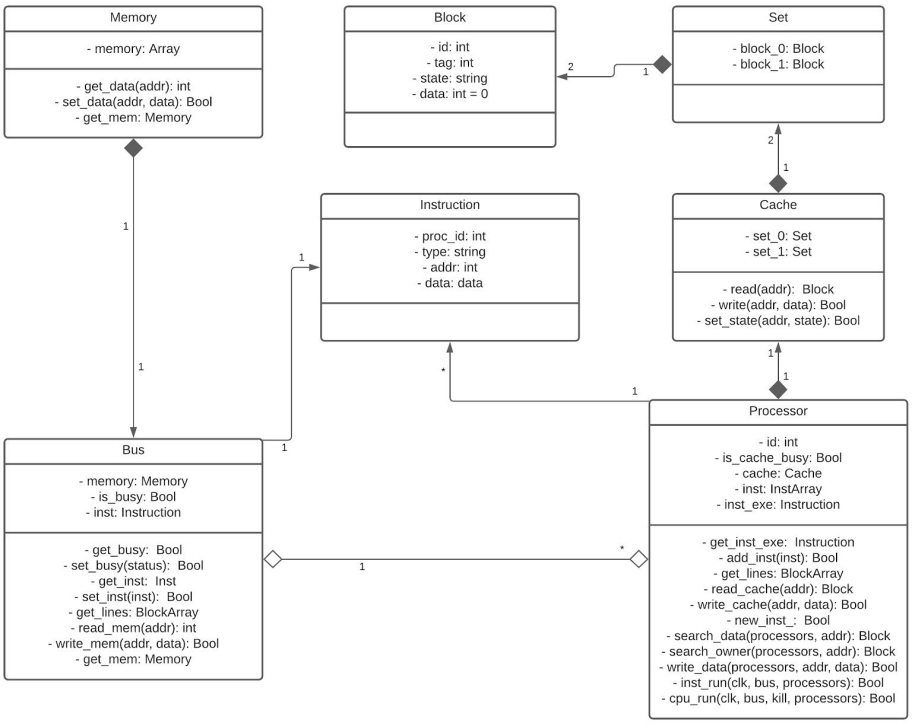

Figura 7. Diagrama de clases del software construido.

- **Descripción del algoritmo planteado.**
  - En esta sección se muestra el diagrama de flujo del algoritmo por el cual se ejecutan instrucciones. Esto se realiza para un procesador pero se infiere que es el mismo algoritmo en cada uno de los procesadores pues el sistema está implementado con *threads*.

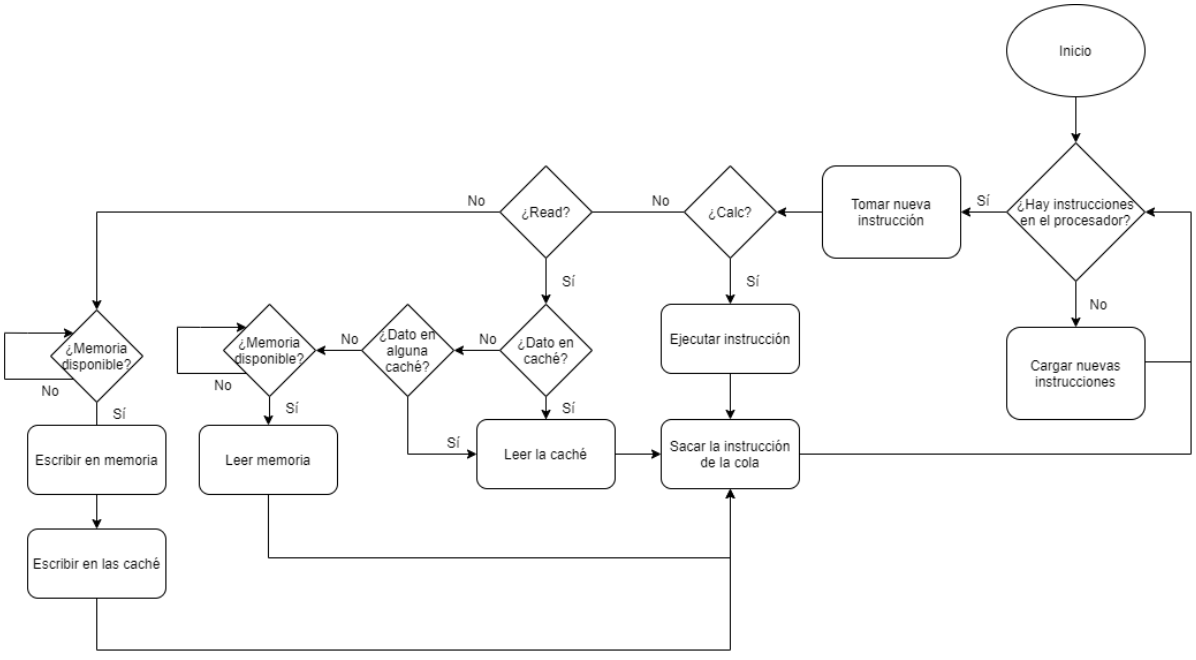

Figura 8. Diagrama de flujo del algoritmo para ejecutar instrucciones en un procesador.

- **Distribución de probabilidad implementada.**
    - Para cumplir este requerimiento se utilizó la distribución binomial, la cual es una distribución de probabilidad discreta que cuenta el número de éxitos en una secuencia de *n* ensayos con dos únicos resultados posibles (éxito ó fracaso). Además estos ensayos deben de ser independientes entre sí, con una probabilidad fija *p* de ocurrencia del éxito [1]. Se optó por esta distribución pues se determinó empíricamente que tiene el menor tiempo de ejecución promedio en la implementación brindada por la biblioteca *SciPy* de la que se referencia la documentación en [2].

Referencias bibliográficas.

[1] H. Alvarado, & C. Batanero. (2007). *Dificultades de comprensión de la aproximación normal a la distribución binomial*. Números. Revista de Didáctica de las Matemáticas, 67, 1-7.

[2] SciPyOrg. (2020). *SciPy documentation*. [Online] Recuperado de: https://docs.scipy.org/doc/scipy/reference/stats.html