**1. Entrada/Salida (E/S): Conceptos de dispositivos de E/S y su interacción con la memoria y CPU**

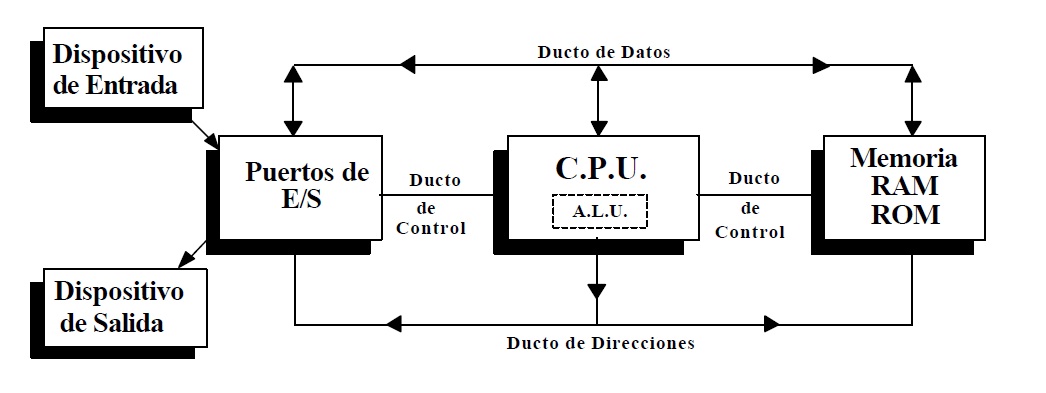


Figure 1: Diagrama general de arquitectura de computadora

La E/S (entrada/salida) en los sistemas de cómputo es la capa crucial en la conexión de los periféricos con el centro del sistema: la CPU y la memoria. Estos mecanismos no sólo consisten en la transferencia de bits, sino también en políticas de sincronización, coherencia de datos y de manejo de errores del sistema, la cual depende mucho del rendimiento final del sistema también. En los sistemas de computación de alto rendimiento tal como los sistemas de computación científica y el almacenamiento a gran escala, instrumentar y hacer visibles las rutas de E/S estima como una medida primaria para encontrar cuellos de botella y comportamientos no esperados. Neuwirth et al. (2025) tratan de plantear un marco para poder hacer la E/S explicable en HPC, subrayando la importancia de trazar flujos de datos y de hacer visibles las métricas de desempeño como los principales instrumentos de optimización de la pila completa [DOI:10.1145/3733723.3733741].

**1.1 Arquitectura general de E/S y módulos**

Los módulos de E/S son los intermediarios entre los dispositivos y la memoria/CPU, llevando a cabo el controlador, la cola, las políticas de buffering, etc. La interacción de estos módulos con la jerarquía de memoria implica, decisiones sobre el tipo de mapeo (MMIO contra PMIO), la coherencia entre caches y dispositivos y la política de estructuras de colas para I/O. Xu, et al (2024) explican cómo el uso de pmem y mecanismos de caching en tránsito puede reconfigurar el diseño de módulos de E/S, afectando las latencias y la durabilidad de las operaciones de bloque mediante el uso de caches intermedias a la CPU [DOI:10.1016/j.sysarc.2024.103109].

Tabla 1.1: Clasificación de dispositivos de E/S

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de Dispositivo** | **Ejemplos** | **Velocidad de Transferencia** | **Función Principal** |
| Entrada | Teclado, Mouse | Baja (kbps) | Captura de datos |
| Salida | Monitor, Impresora | Media (Mbps) | Presentación de información |
| Entrada/Salida | Disco Duro, Pantallas Táctiles | Alta (Gbps) | Lectura y escritura de datos |

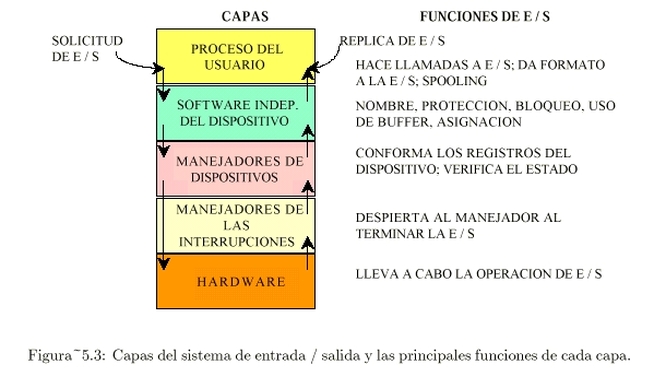


Figure 1.1: Esquema Global de manejadores de dispositivos

**1.2. Técnicas de Entrada/Salida**

**1.2.1 E/S programada (Programmed I/O, PIO): fundamentos y coste**

La E/S programada (PIO) supone que la CPU ejecuta instrucciones explícitas para mover datos hacia/desde los registros del dispositivo. Aunque conceptualmente simple y fácil de implementar, PIO somete a la CPU a ciclos de espera (busy-waiting) y consume recursos valiosos, lo que la vuelve ineficiente en cargas con alto ancho de banda. Investigaciones recientes muestran que, para dispositivos extremadamente rápidos y en arquitecturas con optimizaciones de caché, PIO revisitado con técnicas especializadas puede ofrecer latencias competitivas en ciertos escenarios de cola pequeña; sin embargo, el coste de CPU sigue siendo un factor limitante (Lee et al., 2022) [DOI:10.1016/j.sysarc.2021.102338].

Tabla 1.2.1: Ventajas y Desventajas de la E/S Programada

|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| Fácil de implementar | Alto uso de CPU |
| Control directo sobre el hardware | Ineficiente para grandes volúmenes de datos |
| Bajo costo | Latencia alta |

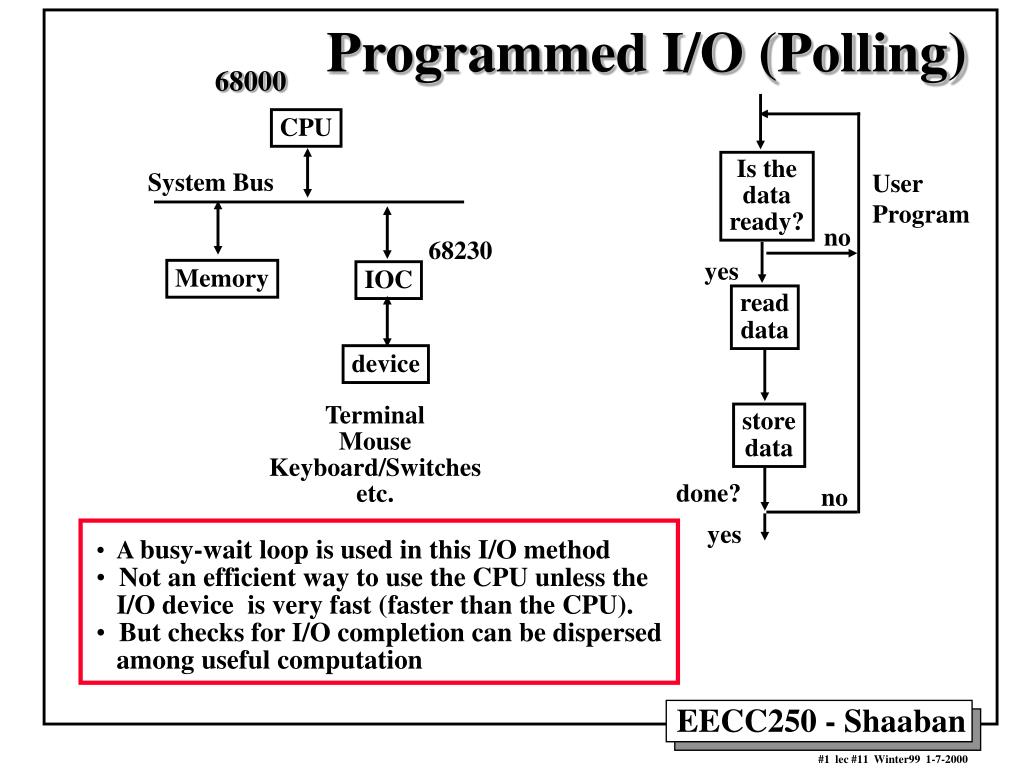


Figure 1.2.1: Ciclo de E/S programada (Polling)

**1.2.2. E/S mediante interrupciones: mecanismo y complejidad**

La E/S con interrupciones evita que la CPU tenga que hacer reinicio continuo porque los dispositivos informan de forma activa cuando ocurre un evento relevante. La carga se encuentra, pues, en la gestión de la interrupción, del contexto del guardado/restaurado y de los ISRs. En virtualización o en entornos de alta densidad de dispositivos la latencia por la conmutación de contexto ha sido interesante; Seo et al. (2024) evidencian cómo la E/S polled de Linux, que necesitan una larga ruta de E/S, evitan la penalización de rendimiento en usos de interrupción, pues las optimizaciones del núcleo que eliminan pasos innecesarios en la ruta de E/S [DOI:10.1145/3655038.3665944].

Tabla 1.2.2: Comparación entre E/S Programada y E/S con Interrupciones

|  |  |  |
| --- | --- | --- |
| **Característica** | **E/S Programada** | **E/S con Interrupciones** |
| Uso de CPU | Alto | Bajo |
| Tiempo de respuesta | Lento | Rápido |
| Complejidad | Baja | Media |
| Ejemplos de uso | Teclado básico | Tarjeta de red |

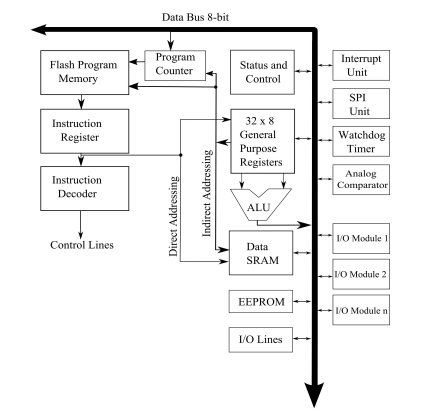
****

Figure 1.2.2: Manejo de interrupciones

**1.2.3. Comparativa práctica: PIO vs Interrupciones**

La comparación de PIO y E/S por interrupciones se ha de tener en cuenta la latencia, el uso de la CPU y la predecibilidad; en el PIO se tendrá una menor latencia en transferencias de tamaño muy pequeño, que se puede prever si es capaz de dedicarse la CPU a ello, en cambio, las interrupciones escalan mejor en entornos en multiproceso en los que la CPU tiene que atender múltiples tareas. A pesar de ello, las implementaciones del kernel han mejorado de forma que se acercan a lo mejor de los dos mundos mediante técnicas híbridas (polling con backoff). Los estudios empíricos y benchmarks muestran que la elección del mecanismo apropiado depende de la carga, de la topología de la memoria y de las capacidades del dispositivo (Lee et al., 2022) [DOI:10.1016/j.sysarc.2021.102338].

**1.2.4. Acceso Directo a Memoria (DMA): concepto y flujo**

El acceso directo a memoria (DMA) delega la transferencia de bloques entre dispositivo y memoria a un controlador especializado, liberando a la CPU y reduciendo overhead. El controlador DMA programa direcciones, conteos y señales de ciclo de bus para efectuar transferencias en el bus de memoria. El uso del DMA mejora la eficiencia en transferencias grandes y en operaciones de E/S continuas; además, su integración con controladores modernos permite operaciones scatter‑gather y minimiza copia de datos. Li et al. (2024) realizan un estudio exhaustivo de DMA en chip y presentan optimizaciones latencia‑orientadas que permiten reducir el umbral de tamaño de I/O a partir del cual el DMA es ventajoso [DOI:10.1145/3656477].

Tabla 1.2.4: Comparación entre E/S con Interrupciones y DMA

|  |  |  |
| --- | --- | --- |
| **Característica** | **E/S con Interrupciones** | **DMA** |
| Intervención de CPU | Moderada | Mínima |
| Velocidad de Transferencia | Media | Alta |
| Complejidad del hardware | Media | Alta |
| Uso típico | Periféricos de velocidad media | Disco duro, tarjetas gráficas |

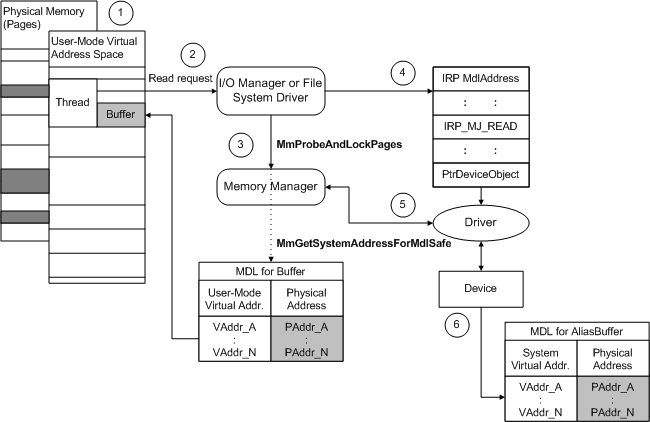


Figure 1.2.2: Diagrama de flujo DMA mostrando transferencia directa dispositivo

**1.2.5. Diseño de controladores y descriptor rings para DMA**

La estructura basada en descriptores y anillos en los controladores DMA resulta vital para la utilización eficiente de los sistemas de transferencia y ordenación de la transferencia. Una reestructuración contemporánea de estos anillos podría suprimir las latencias de cola y mejorar el control del acceso en los mecanismos de integración, lo que permitiría contemplar mayor flexibilidad entre la ordenación y la validación de las transferencias. Artículos recientes (arXiv:2409.08141) proponen una serie de alternativas de rediseño que permiten los descriptores DMA para optimizar la concurrencia, reducir la carga de gestión y facilitar su integración con SmartNICs y aceleradores de red (arXiv:2409.08141).

**1.3. Problemas y técnicas avanzadas en E/S**

**1.3.1. Problemas de coherencia: DCA y accesos a caché**

El acceso temporal directo a caché, DCA por sus siglas en inglés, y técnicas análogas permiten a dispositivos cargar los datos directamente en las cachés de la CPU y así reducir la latencia de acceso para aplicaciones, por otra parte, sensibles al tiempo de acceso. Sin embargo, DCA introduce retos con respecto a la coherencia y contención de las líneas de caché; Wang et al. (2022), tras estudiar a fondo el rendimiento y las implicaciones de DCA en multicore y redes de host evidencian que la ventaja de latencia se puede ver contrarrestada por la contención de caché y el impacto negativo en el rendimiento global de no manejarse de la forma adecuada [DOI:10.1145/3547353.3522662].

**1.3.2. E/S en entornos virtualizados y nubes IaaS**

La virtualización añade capas entre el dispositivo físico y la aplicación: hipervisores, backends y el guest OS. Este apilamiento incrementa la complejidad de manejar E/S eficiente y segura. VPRI (Guo et al., 2024) presenta un co‑diseño software‑hardware para manejar fallos de página de I/O (IOPFs), reduciendo la latencia al integrar mecanismos de resolución colaborativa entre dispositivo e hipervisor, lo que es clave para cargas de IaaS con acceso remoto a memoria y dispositivos virtualizados [DOI:10.1145/3694715.3695957].

**1.3.3. E/S y almacenamiento persistente: pmem y caching en tránsito**

La llegada de la memoria persistente byte-adressable como arquitectura de nueva generación supuso nuevas configuraciones, ya que las operaciones de bloques tradicionales pueden ser trasladadas a pmem especializadas con tablas de traducción y capas de cache. Xu et al. (2024) informan que la cache en el tránsito (I/O transit caching, almacenamiento en el tránsito de E/S, cercano al procesador) permite una menor latencia y mejora durabilidad, junto a la descripcón de software que dirige la eviicción junto la sincronización a la memoria persistente para mantener la consistencia [DOI:10.1016/j.sysarc.2024.103109].

**1.3.4. Técnicas híbridas de polling y sondeo eficiente**

Las técnicas híbridas de polling funcionan con sondeo activo y utilizan mecanismos de notificación para disminuir tanto la latencia como el uso de la CPU; por ejemplo, una de estas técnicas mantiene ventanas cortas de sondeo activo en las que se chequea el dispositivo de forma constante, y si no responde, se cambia a modo de sondeo con notificación por interrupción. Lee et al. (2022) realizan un conjunto de esquemas que tienen como objetivo minimizar ciclos de CPU asociados al polling potencialmente a costa de la latencia, implementar nuevos métodos que se adapten según el patrón de carga en antiguos dispositivos de almacenamiento ultrarrápidos, pero sin sacrificar la latencia [DOI:10.1016/j.sysarc.2021.102338].

**1.3.5. On‑chip DMA y movimiento de datos en sistemas NVM**

El rápido movimiento de datos entre DRAM, NVM y dispositivos se convierte en un cuello de botella emergente; reutilización de DMA on‑chip y tuneo según el perfil de acceso se encargan de acelerar transferencias no dependientes exclusivamente de la CPU. Fastmove (Li et al., 2024) presenta un análisis del diseño de DMA on-chip y muestra ganancias en rendimiento para sistemas de almacenamiento basados en NVM, optimizando arbitraje, tamaño de la ráfaga, y rutas para el acceso que minimizan latencias pico. [DOI:10.1145/3656477].

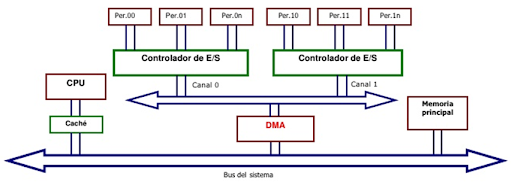


Figure 1.3.5: DMA

**1.3.6. Seguridad y aislamiento en DMA: D‑Box y mitigaciones**

El DMA, por su capacidad de acceso directo a la memoria, presenta riesgos si periféricos maliciosos o comprometidos pueden leer/escribir regiones sensibles. D‑Box (arXiv:2201.05199) propone técnicas para asegurar canales DMA en sistemas embebidos mediante validación de descriptores, control de acceso y mecanismos criptográficos ligeros que preservan rendimiento sin abrir vectores de ataque que comprometan la integridad de la memoria [arXiv:2201.05199].

**1.3.7. SmartNICs y preprocesamiento en red con DMA**

El impulso que se está dando hacia SmartNICs y funciones in‑network es el que ha llevado a plantear algunos diseños donde la memoria del host y la del SmartNIC están interconectadas para permitir accesos eficientes. El trabajo In‑Network Preprocessing & DMA (arXiv:2501.12032) estudia cómo delegar tareas de preprocesamiento en la NIC y utilizar DMA para mover los resultados, con la ventaja de reducir la carga impuesta a la CPU y las latencias de extremo a extremo en aplicaciones de red y de almacenamiento [arXiv:2501.12032].

**1.3.8. Acceso a memoria en red y puentes SmartNIC‑Host**

Además del preprocesado, es posible implementar arquitecturas en las que el SmartNIC comparte parte del espacio de la memoria direccionable para poder disponerse de memoria para acelerar tareas distribuidas. Farooqi et al. (2025) realizan un estudio comparando las diferentes técnicas de acceso a memoria entre SmartNIC y host, ofreciendo mediciones claras en cuanto a la latencia, el ancho de banda y el coste de coherencia que permiten guiar el diseño de soluciones híbridas [arXiv:2507.04001].

**1.3.9. Procesamiento en memoria (PIM) y DaPPA**

El paradigma PIM reduce el movimiento de datos llevando computación hacia donde residen los datos. DaPPA (Oliveira et al., 2023) presenta un framework para facilitar la programación de arquitecturas PIM (p. ej. UPMEM), automatizando particionamiento, gestión de memoria y recolección de resultados, lo que simplifica la adopción de PIM para aplicaciones data‑parallel y reduce la latencia derivada del traslado DRAM↔CPU [arXiv:2310.10168].

**1.4. Relación con tendencias y otros paradigmas**

**1.4.1. Técnicas de memoria en LLMs y tendencias futuras**

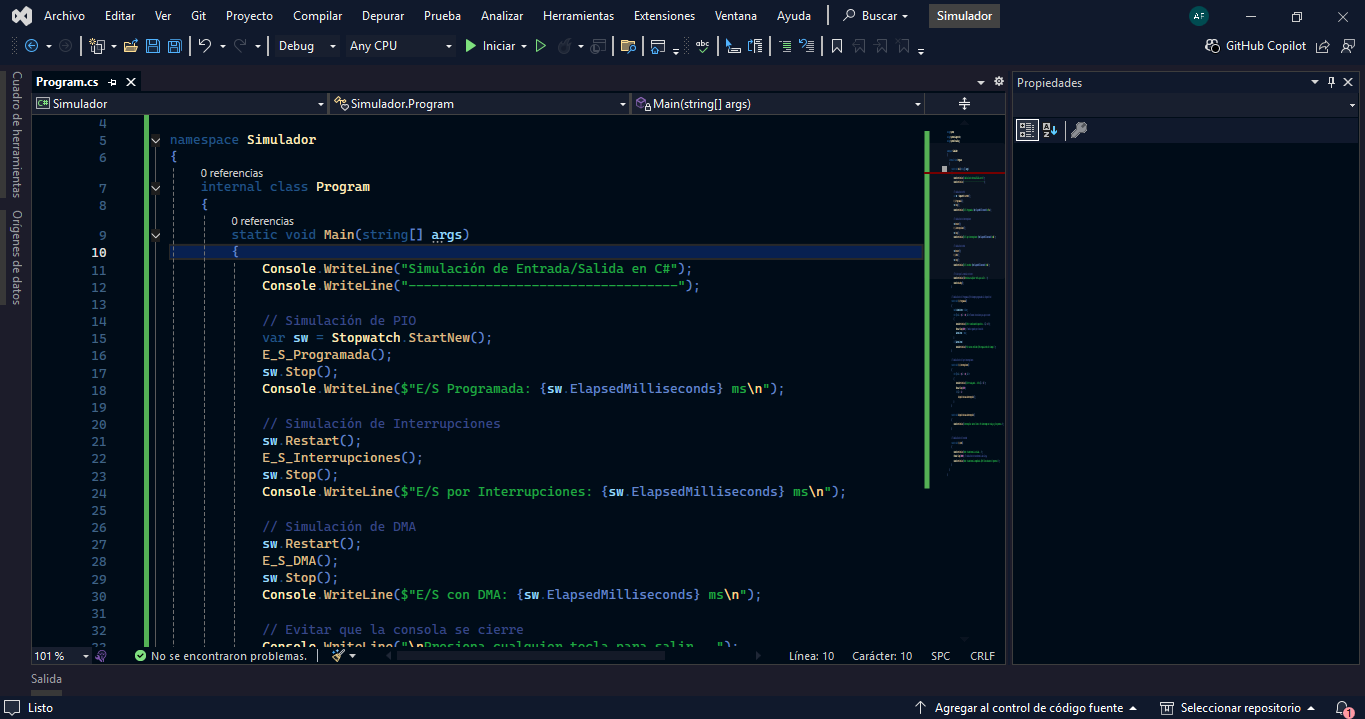
Si bien la E/S tradicional hace énfasis en el hardware y los dispositivos, la gestión de memoria asociada a los LLMs (Modelos de Lenguaje masivos) nos podría dejar vislumbrar la existencia de alternativas interesantes: memoria jerárquica, caché semántica o políticas de retención dinámica son ejemplos de soluciones de E/S que podrían priorizar la tratativa de datos relevantes. Trabajos recientes sobre memoria cognitiva en LLMs y sus encuestas de técnicas (Shan et al., 2025; Wu et al., 2025) hacen hincapié en expresar, indexar y recuperar estado de manera eficiente, lo que puede encontrarse en paralelo con la organización de buffers, caches y políticas de prefetching en sistemas de E/S [arXiv:2504.02441; arXiv:2504.15965].

**Conclusión y recomendaciones de diseño**

El diseño de sistemas de E/S modernos exige un enfoque holístico: comprender la interacción entre dispositivos, caches, controladores DMA, SmartNICs y software de pila (kernel, hipervisor). La literatura reciente sugiere priorizar mecanismos que reduzcan movimiento innecesario de datos (PIM, on‑chip DMA), proteger los canales de DMA (D‑Box) y explotar arquitecturas de memoria persistente para cargas adecuadas (Caiti). Como recomendación práctica, los ingenieros deben caracterizar la carga (tamaños de I/O, latencia objetivo y patrones de acceso) y elegir el mecanismo (PIO, interrupciones, DMA, o híbridos) que ofrezca el mejor compromiso entre latencia, uso de CPU y seguridad. Para investigación futura, integrar modelos de memoria inspirados en LLMs con sistemas E/S podría abrir nuevos caminos para la priorización semántica de datos [DOI:10.1145/3656477].

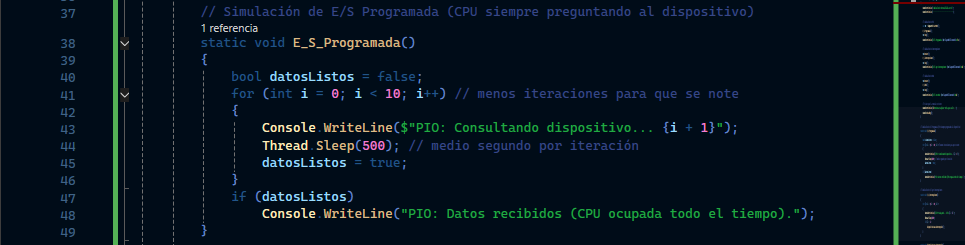
**Ejemplo:**

**Simulación de Entrada/Salida en C#**

****

En esta imagen se presenta el código de la simulación, en el apartado de anexos en el GitHub se encuentra el código junto a un video de la ejecución de este.

**1. Entrada/Salida Programada (PIO)**

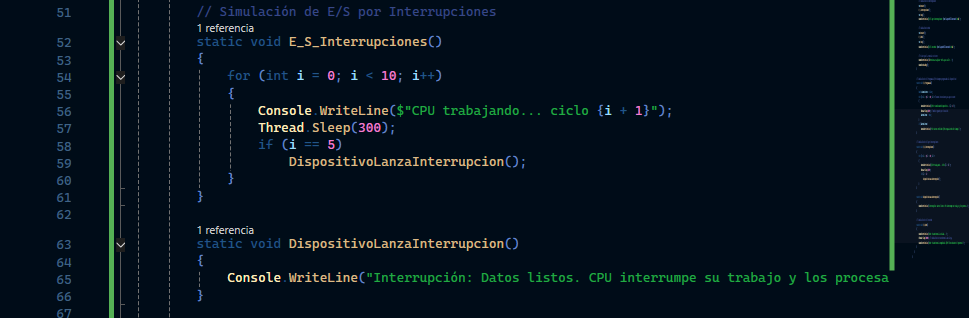
****

Lo que se simula en este apartado es la CPU, consulta repetidamente al dispositivo si los datos están preparados (*polling*).

**Se ejecuta:**

* Un bucle **For** de 10 iteraciones simula que la CPU pregunta repetidamente 10 veces.
* **Theread.Sleep(500)** pausa medio segundo cada vez, simulando la espera.
* **datosListos** se ponen en **true** para simular que eventualmente le llegan los datos.
* Al final, se imprime que recibie los datos.

**2. Entrada/Salida por Interrupciones**

****

Aquí la CPU hace otras tareas y solo se detiene cuando el dispositivo le “avisa” que los datos están listos.

**Se ejecuta:**

* El bucle que la CPU trabaja en otras cosas (Theread.Sleep(300) simula tiempo de trabajo).
* En el ciclo 6 (i== 5), se llama DispositivoLanzamietnointerrupcion(), que muestra un mensaje de que llegaron los datos.

**4. E/S con DMA**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En este apartado simula un controlador DMA que transfiere datos directamente entre memoria y el dispositivo, sin la intervención del CPU.

**Se ejecuta:**

* La impresión de transferencia
* **Theread.Sleep(2000)** que simula que el proceso tarda 2 segundos.
* La transferencia que termina y que la CPU estuvo libre mientras tanto.