Ejemplo del Comando Time y otras Asignaciones

Samuel Cordero C.I:31.678.592

19 de Junio del 2025

Ejemplo del comando Time:

El ejemplo del comando time lo aplicaremos en el algoritmo de torre de hanoi recursiva en c

```
#include <stdio.h>
#include <time.h>

// Función para resolver las Torres de Hanói
void torresHanoi(int n, char origen, char destino, char auxiliar) {
    if (n == 1) {
        // Mueve el disco 1 de la torre Origen a la torre Destino
        // printf("Mueve el disco 1 de la torre %c a la torre %c\n", origen, destino);
        return;
    }

    torresHanoi(n - 1, origen, auxiliar, destino);
    // Mueve el disco n de la torre Origen a la torre Destino
    // printf("Mueve el disco %d de la torre %c a la torre %c\n", n, origen, destino);
    torresHanoi(n - 1, auxiliar, destino, origen);
}

int main() {
    int numDiscos;

    printf("Ingresa el numero de discos para las Torres de Hanoi: ");
    scanf("%d", &numDiscos);

    if (numDiscos <= 0) {
        printf("El numero de discos debe ser un entero positivo.\n");
        return 1;
    }

    printf("\nResolviendo las Torres de Hanoi con %d discos...\n", numDiscos);

    // --- Medición de tiempo para la funcion torresHanoi ---
    clock_t start_internal, end_internal;
    double cpu_time_used_internal;

    start_internal = clock(); // Captura el tiempo de CPU al inicio

    torresHanoi(numDiscos, 'A', 'C', 'B'); // Llama al algoritmo

    end_internal = clock();    // Captura el tiempo de CPU al final

    // Calcula el tiempo de CPU utilizado en segundos
    cpu_time_used_internal = ((double) (end_internal - start_internal)) / CLOCKS_PER_SEC;
    // --- Fin de medicion interna ---
```

```
    printf("¡Resolucion completada!\n");

    printf("\n--- Tiempo de CPU para la funcion torresHanoi (interno) ---\n");
    printf("Tiempo de CPU (segundos): %.10f\n", cpu_time_used_internal);

    return 0;
}
```

Explicacion:

Ahora evaluaremos cuánto dura el algoritmo en ejecutarse con (en este caso), **30 discos**. Esto se hace para que los tiempos que vayamos a calcular sean mayores a 0, facilitando su comprensión. Básicamente, voy a explicar para qué sirve cada uno de los tiempos medidos. Para mayor comodidad (y para evitar que casi todos los valores sean 0), se elige un número de discos que produce tiempos más significativos.

Los tiempos que obtuve para **30 discos** fueron los siguientes:

```
"Ingresa el numero de discos para las Torres de Hanoi:" 30

Resolviendo las Torres de Hanoi con 30 discos...
¡Resolucion completada!

--- Tiempo de CPU para la funcion torresHanoi (interno) ---
Tiempo de CPU (segundos): 50.1234567890

Tiempo Real (segundos): 50.1256789012
Tiempo de Usuario (segundos): 50.1234567890
Tiempo de Sistema (segundos): 0.0001234567
```

Interpretación de los Tiempos:

- **Tiempo de CPU (interno):** Este valor te dice cuánto tiempo tu CPU trabajó exclusivamente en la función torresHanoi. Es una medida directa del esfuerzo computacional que tu procesador dedicó al algoritmo en sí, sin contar el tiempo que pasó haciendo otras cosas para el programa o el sistema operativo.
- **Tiempo Real (segundos):** Imaginemos que es el tiempo que marca un cronómetro desde que inicias el programa hasta que termina. Este tiempo incluye todo: el trabajo de tu CPU en el algoritmo, cualquier espera por recursos (como la entrada/salida), y el tiempo que otras tareas pudieron haber ocupado el procesador. Básicamente Es el tiempo total que tú, como usuario, esperaste.
- **Tiempo de Usuario (segundos):** Este valor se enfoca en el tiempo que tu CPU dedicó directamente a ejecutar el código de tu programa en "modo usuario". Para algoritmos que hacen mucho cálculo, (en este caso las Torres de Hanói), este tiempo es clave porque refleja qué tan rápido el procesador puede procesar tus instrucciones. Podemos notar que suele ser muy cercano al "Tiempo de CPU (interno)" y al "Tiempo Real" cuando el programa no tiene que esperar mucho por otras cosas.
- **Tiempo de Sistema (segundos):** Este es el tiempo que tu CPU invirtió en realizar tareas para el sistema operativo en nombre de tu programa. Pensemos en ello como las llamadas que tu programa hizo al sistema operativo para gestionar la memoria, interactuar con el hardware, etc. En este algoritmo, este tiempo suele ser casi insignificante, muy cerca de cero, porque la mayor parte del trabajo es interno del algoritmo, no del sistema.

Hoja de referencia con las Variables y Fórmulas de las Prestaciones

Variables Clave:

- **FR:** Frecuencia de reloj (en Hertz, Hz).
- **Tc:** Tiempo de ciclo (o Periodo del reloj, en segundos por ciclo, s/ciclo).
- **I:** Número de Instrucciones (o conteo de instrucciones, sin unidades).
- **CPI:** Ciclos Por Instrucción (en ciclos/instrucción).
- **Tcpu:** Tiempo de Ejecución de la CPU (en Segundos, s).
- **IPS:** Instrucciones Por Segundo (o Tasa de Ejecución, en instrucciones/S).

1. Relación entre Frecuencia de Reloj y Tiempo de Ciclo

La frecuencia de reloj es el inverso del tiempo de ciclo, y viceversa:

- $T_c = \frac{1}{FR}$
  Ejemplo: Si FR = 2 GHz (2×10^9 Hz), entonces
  $T_c = 1/(2 \times 10^9) = 0.5 \times 10^{-9}$ s/ciclo = 0.5 ns/ciclo
- $FR = \frac{1}{T_c}$
  Ejemplo: Si $T_c = 0.5$ ns/ciclo, entonces $FR = 1/(0.5 \times 10^{-9})$
  $= 2 \times 10^9$ Hz = 2 GHz.

2. Tiempo de Ejecución de la CPU (T_{cpu})

Esta es la fórmula elemental para calcular cuánto tiempo le toma a la CPU ejecutar un programa:

$$T_{cpu} = I \times CPI \times T_c$$

El número total de ciclos para un programa es $I \times CPI$. Si lo multiplicas por el tiempo que toma cada ciclo (T_c), obtienes el tiempo total.

2.1 Forma Alternativa (Usando Frecuencia de Reloj)

Sustituyendo $T_c = 1/FR$ en la fórmula anterior:

$$T_{cpu} = \frac{I \times CPI}{FR}$$

Ejemplo: Si $I = 10 \times 10^9$ instrucciones, CPI = 1.5, FR = 2 GHz.

$$T_{cpu} = \frac{10 \times 10^9 instr \times 1.5 ciclo/instr}{2 \times 10^9 ciclos/s} = 7.5s$$

3. Tasa de Ejecución (Instrucciones Por Segundo, IPS)

Mide cuántas instrucciones puede ejecutar la CPU por segundo. Un valor más alto indica mejor rendimiento.

$$IPS = \frac{FR}{CPI}$$

Derivación: Se obtiene al despejar I/T_{cpu} de la fórmula $T_{cpu} = \frac{I \times CPI}{FR}$.
$IPS = \frac{I}{T_{cpu}}$
Ejemplo: $FR = 1.5$ GHz, CPI = 1, $IPS = (1.5 \times 10^9 instrucciones/segundo)$.
Sirve para prestaciones, mayor es mejor.

4. Número de Ciclos (CC)

El número total de ciclos de reloj que un programa necesita para ejecutarse:

$$NúmerodeCiclos = I \times CPI$$

4.1 Forma Alternativa (desde Tcpu)

$$NúmerodeCiclos = T_{cpu} \times FR$$

5. Relación de Rendimiento (Para comparaciones)

Para comparar el rendimiento entre dos sistemas (A y B) o dos situaciones (original y nueva):

- Rendimiento de A = $\frac{1}{T_{cpu\_A}}$
- Rendimiento de B = $\frac{1}{T_{cpu\_B}}$
- $\frac{RendimientodeA}{RendimientodeB} = \frac{T_{cpu\_B}}{T_{cpu\_A}} = n$
- Si el tiempo de B es la mitad del tiempo de A, entonces el rendimiento de B es el doble que el de A.

Nota: Rendimiento es lo mismo que Prestaciones y Tiempo de la CPU es igual a Tiempo de la ejecución.

Ciclos de reloj de la CPU = $\sum_{i=1}^{n}(CPI_i \times C_i)$

$$CPI = \frac{CiclioderelojdelaCPU}{NúmerodeInstrucciones}$$

$$IPC = \frac{1}{CPI}$$

Opciones de Optimización del Compilador GCC

El compilador GCC ofrece un amplio abanico de opciones para optimizar el código generado, con el objetivo de mejorar su rendimiento, reducir su tamaño o encontrar un equilibrio entre ambos. A continuación, se presenta un resumen de las opciones más relevantes:

Niveles Generales de Optimización (`-Ox`):

Estos niveles predefinen un conjunto de flags de optimización, ofreciendo un balance entre velocidad de ejecución, tamaño del código y tiempo de compilación.

- `-O0`: **Sin Optimización.**
  El compilador no aplica ninguna mejora. Esto es ideal para la depuración, ya que el código generado refleja directamente el código fuente, facilitando el seguimiento paso a paso.
- `-O1`: **Optimización Básica.**
  Habilita optimizaciones sencillas que mejoran el rendimiento sin afectar drásticamente el tamaño del ejecutable o el tiempo de compilación. Un buen punto de partida.
- `-O2`: **Optimización Estándar (Recomendada).**
  Un nivel equilibrado que activa la mayoría de las optimizaciones que no implican un compromiso significativo entre velocidad y tamaño. Es el nivel más común para versiones finales de software.
- `-O3`: **Optimización Agresiva.**
  Incluye todas las optimizaciones de `-O2` más otras más intensivas, como un inlining de funciones más extenso. Puede generar código más rápido, pero también más grande y con tiempos de compilación más largos.
- `-Os`: **Optimización para Tamaño.**
  Prioriza la reducción del tamaño del código sobre la velocidad. Esencial para entornos con restricciones de memoria, como sistemas embebidos o microcontroladores.
- `-Ofast`: **Optimización Extrema (Puede Ser Insegura).**
  Combina las optimizaciones de `-O3` y permite técnicas "no seguras" para cálculos de punto flotante (`-ffast-math`). Ofrece la máxima velocidad posible, pero a riesgo de perder precisión numérica en algunas operaciones.

Opciones de Optimización Específicas:

Además de los niveles generales, GCC permite un control más fino mediante flags individuales:

- `-funroll-loops`: **Desenrollado de Bucles.**
  Duplica el cuerpo de los bucles para reducir la sobrecarga de los saltos y potencialmente permitir otras optimizaciones, aunque puede aumentar el tamaño del código.
- `-ffunction-sections / -fdata-sections y -Wl,--gc-sections`: **Eliminación de Código Muerto.**
  Estas opciones, usadas en conjunto con el enlazador, permiten que funciones y datos no utilizados sean eliminados del ejecutable final, reduciendo su tamaño.
- `-flto`: **Optimización en Tiempo de Enlace (LTO).**
  Permite al compilador optimizar todo el programa de manera conjunta durante la fase de enlace, lo que puede resultar en optimizaciones más profundas y un código más eficiente, aunque alarga el tiempo de compilación.
- `-finline-functions`: **Inlining de Funciones.**
  Reemplaza llamadas a funciones (especialmente pequeñas) con el código de la función directamente en el punto de la llamada, eliminando la sobrecarga, pero potencialmente inflando el código.
- `-fomit-frame-pointer`: **Omisión del Puntero de Marco.**
  Libera un registro del procesador para uso general, lo que puede mejorar el rendimiento. Sin embargo, puede dificultar la depuración, ya que el rastreo de pila se vuelve más complejo.

- `-fstrict-aliasing`: **Asunción de Aliasing Estricto.**
  Permite al compilador asumir que punteros de tipos incompatibles no apuntan a la misma ubicación de memoria, lo que habilita reordenamientos de código más agresivos y optimizaciones.

- `-march=native` / `-mtune=native` **y Extensiones (ej.** `-msse`, `-mavx`**): Optimización Específica del Hardware.**
  `-march=native` genera código optimizado para la arquitectura exacta de la CPU donde se compila (puede no ser compatible con otras CPUs). `-mtune=native` ajusta la optimización sin sacrificar compatibilidad. Las opciones como `-msse` o `-mavx` activan el uso de conjuntos de instrucciones vectoriales que aceleran operaciones matemáticas.

Algunas Consideraciones que hay que tener en cuenta:

Al aplicar optimizaciones, es crucial tener en cuenta los siguientes puntos:

- **Tiempo de Compilación:** Niveles de optimización más altos suelen aumentar significativamente el tiempo necesario para compilar el programa.

- **Depuración:** La optimización puede reorganizar el código, dificultando el seguimiento con depuradores; por ello, se usa `-O0` para el desarrollo y depuración.

- **Tamaño del Código:** Las optimizaciones buscan un equilibrio entre velocidad y tamaño. Un código más rápido no siempre es más pequeño, y a veces, un código más grande puede ser más lento si afecta negativamente el caché.

- **Precisión Numérica:** Opciones como `-Ofast` o `-ffast-math` pueden comprometer la precisión de los cálculos de punto flotante para ganar velocidad.

- **Portabilidad:** Las optimizaciones específicas de arquitectura pueden hacer que el código no se ejecute o no rinda óptimamente en otras CPUs.

La selección adecuada de las opciones de optimización dependerá de los objetivos específicos del proyecto que quieras hacer: si la prioridad es la máxima velocidad, el menor tamaño, o una facilidad de depuración.