MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

ÍNDICE DE LA PRESENTACIÓN

MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

1. **Medidas a tomar:** Las necesarias para medir el tiempo de respuesta y la productividad del servidor

Método:

Tiempo de respuesta → Capturando eventos { Tiempo antes de la petición / Tiempo después de la respuesta

Productividad → Contando las transacciones realizadas y dividiéndolas por la duración de la medición

```
Usuario()
{     ← Instrucción de captura del tiempo de inicio
      (después de crear el socket, antes de connect)
   Petición();
      ← Instrucción de captura del tiempo de finalización
   Reflexión();  NumPets++;        (después de closesocket)
}
```

2. Tecnología de medición de tiempos

Muchos computadores disponen de un contador de alta resolución y el sistema operativo dispone de funciones que permiten consultarlo

En Wintel → "Contador de prestaciones de alta resolución"
(high-resolution performance counter ó high resolution timer)

Dos problemas a resolver:

1.- Contar ciclos o eventos:

QueryPerformanceCounter — **Para obtener el valor del contador**

El contador de alta resolución contiene 0 al arrancar el computador y se incrementa continuamente durante el funcionamiento del computador

2.- Obtener la equivalencia entre los ciclos contados y tiempo:

QueryPerformanceFrequency

Para obtener la frecuencia de funcionamiento del contador **(en cuentas / segundo)**

MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

Las funciones trabajan con variables de 64 bits LARGE_INTEGER, definidas:

```
typedef union _LARGE_INTEGER
{   struct
    { DWORD LowPart;
      LONG    HighPart;
    }
    LONGLONG QuadPart;
} LARGE_INTEGER, *PLARGE_INTEGER;
```

Para obtener la frecuencia de funcionamiento del contador (en cuentas / segundo)

BOOL QueryPerformanceFrequency(LARGE_INTEGER *lpFrequency); // Frecuencia del contador

La frecuencia depende del sistema (procesador + chipset de la placa base)
Devuelve TRUE (!=0) si el sistema dispone del contador de alta resolución y FALSE (==0) si no dispone
***lpFrequency* contendrá el número de ciclos que equivalen a un segundo**

Para obtener el valor del contador

BOOL QueryPerformanceCounter(LARGE_INTEGER *lpPerformanceCount); // Valor del contador

Devuelve TRUE (!=0) si tiene éxito y FALSE (==0) si falla
***lpPerformanceCount* contendrá el valor del contador de ciclos hasta el momento**

Ejemplo de medición de tiempos

```
main(void)
{ BOOL Error;
  LARGE_INTEGER CuentasPorSeg, IniConta, FinConta, DifConta;
  unsigned long TiempoTranscurrido;
  int i,j,k=0;

  // Obtener la frecuencia de trabajo del contador
  Error = QueryPerformanceFrequency(&CuentasPorSeg);
  if(Error == 0)
  { printf("No se dispone de un contador de alta resolucion\n");
    exit(-1);
  }
  printf("\nFrecuencia del contador en cuentas por segundo ...\n");
  printf("\n  %I64d dec = %016I64X hex = ( %08X %08X )\n\n",
    CuentasPorSeg.QuadPart, CuentasPorSeg.QuadPart,
    CuentasPorSeg.HighPart, CuentasPorSeg.LowPart);

// Consumir tiempo en un bucle
  QueryPerformanceCounter(&IniConta);
  for(i=0; i<2000; i++)
    for(j=0; j<100000; j++) k++; k--;
  QueryPerformanceCounter(&FinConta);
```

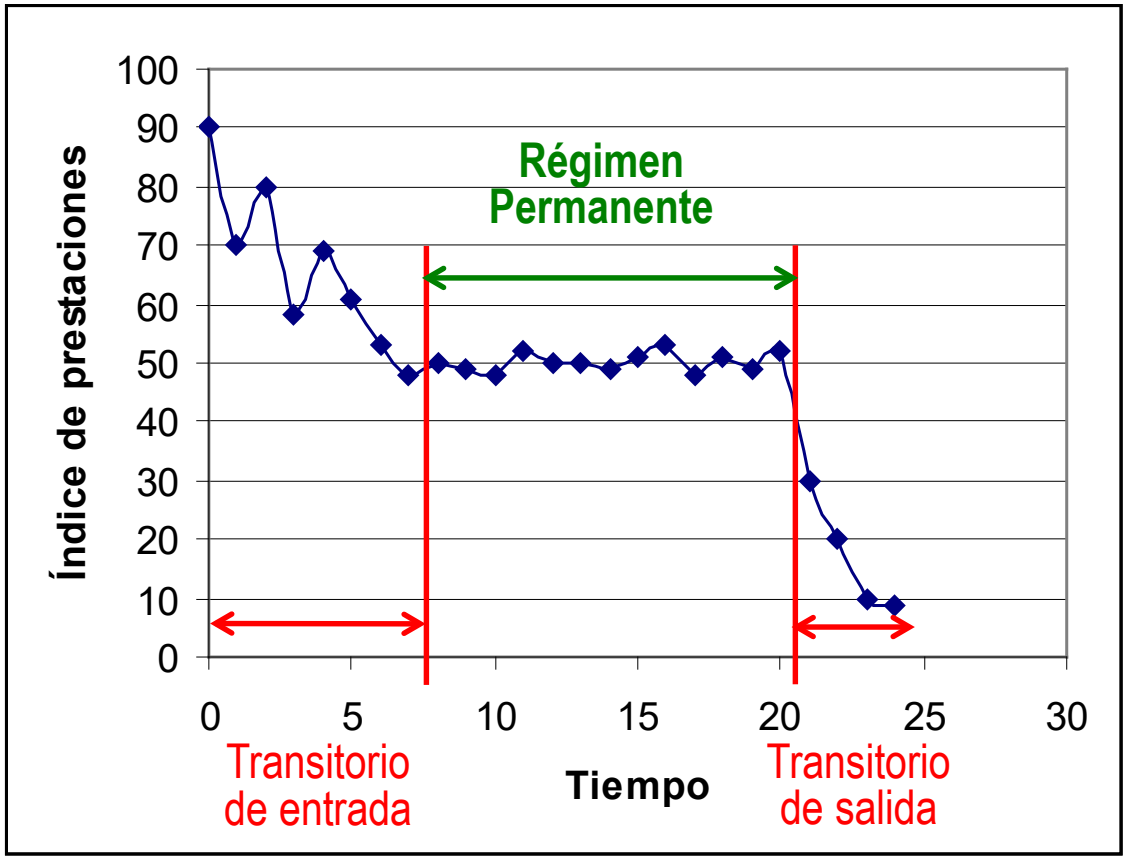
```
typedef union _LARGE_INTEGER
{  struct
   { DWORD LowPart;
     LONG    HighPart;
   }
   LONGLONG QuadPart;
} LARGE_INTEGER, *PLARGE_INTEGER;
```

Ejemplo de medición de tiempos (cont.)

Para pasar a milisegundos

```
// Calcular el tiempo transcurrido
DifConta.QuadPart = (FinConta.QuadPart - IniConta.QuadPart);
TiempoTranscurrido = (unsigned long)
        1000 *(DifConta.QuadPart / CuentasPorSeg.QuadPart);
printf("Tiempo transcurrido = %ld milisegundos ( %I64d - %I64d )\n\n",
  TiempoTranscurrido, FinConta.QuadPart, IniConta.QuadPart);


// Evaluar la intrusividad de la toma de tiempos
for(i=0; i<5; i++)
{ QueryPerformanceCounter(&IniConta);
  QueryPerformanceCounter(&FinConta);
  DifConta.QuadPart = FinConta.QuadPart - IniConta.QuadPart;
  TiempoTranscurrido = (unsigned long)
        ((1000000 * DifConta.QuadPart) / CuentasPorSeg.QuadPart);
  printf("Tiempo transcurrido = %I64d cuentas = %ld microsegundos\n",
    DifConta.QuadPart, TiempoTranscurrido);
}
} // main()
```

3. Transitorios de arranque y terminación

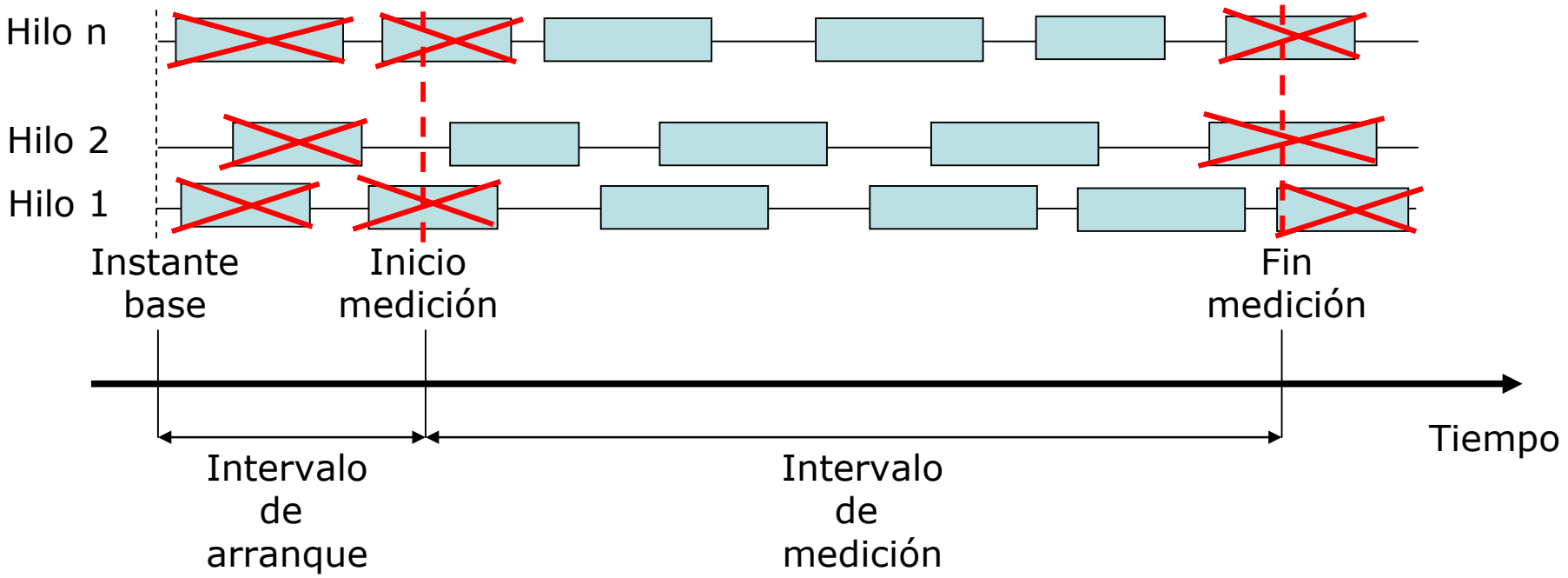


4. Control del intervalo de medición

Cada hilo (usuario) NO lanzará un número fijo de peticiones

Cada hilo (usuario) debe lanzar peticiones sin parar hasta el instante de "Fin de medición"

Se definen 2 intervalos de tiempo que el analista debe controlar { Arranque, Medición }

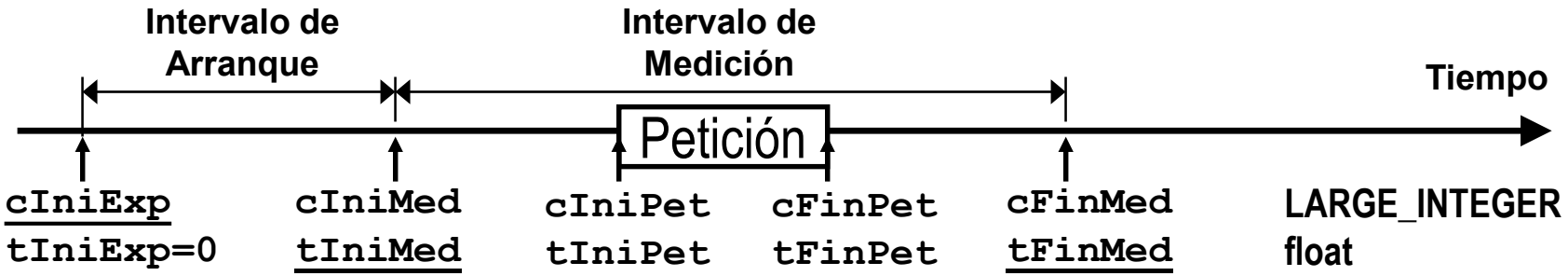


¡Las transacciones que no estén completamente dentro del intervalo de medición no se usan!

5. Mecanismo de captura de datos

Tenemos dos alternativas de referirnos a los eventos del experimento:

- Ciclos.- obtenidos de las funciones QueryPerformanceCounter, c…..
- Tiempos.- transformando los ciclos a tiempos, t……

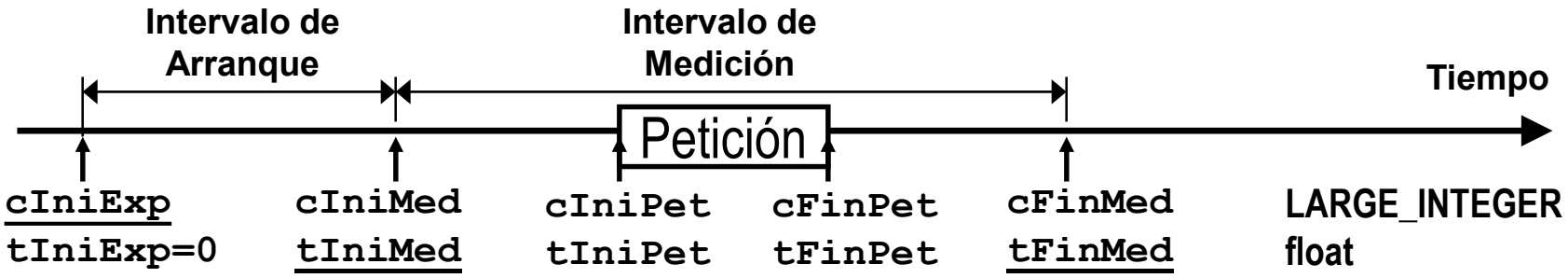


Al inicio del experimento los valores conocidos son los subrayados:

- `tIniMed` y `tFinMed` se obtienen a partir de parámetros que se le pasan al programa
- `cIniExp` se obtiene realizando una llamada al contador al inicio

Teniendo en cuenta la relación entre ciclos y tiempos podemos usar todo ciclos o todo tiempos, según interese

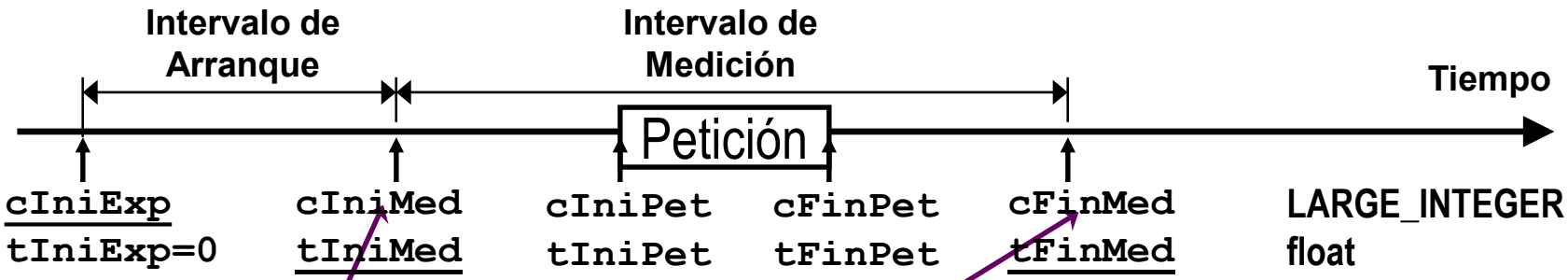
MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN




```
main()  {
   QueryPerformanceFrequency(&cPorSeg); // LARGE_INTEGER
   cPorMseg = (float) cPorSeg.LowPart / (float) 1000;  // float

   QueryPerformanceCounter(&cIniExp);

   tIniMed = 0 + IntervArranque;
   tFinMed = 0 + IntervArranque + IntervMedicion;

   LANZA USUARIOS;
   ……..
}
```

Petición válida **SI**:
tIniPet >= tIniMed y
tFinPet <= tFinMed

O alternativamente: Trabajando exclusivamente con ciclos




```
main() {
   QueryPerformanceFrequency(&cPorSeg); // LARGE_INTEGER
   cPorMseg = (float) cPorSeg.LowPart / (float) 1000;  // float
   QueryPerformanceCounter(&cIniExp);
   tIniMed = 0 + IntervArranque;
   tFinMed = 0 + IntervArranque + IntervMedicion;
   cIniMed.QuadPart = cIniExp.QuadPart +
   cPorSeg.LowPart × IntervArranque
   cFinMed.QuadPart = cIniExp.QuadPart +
   cPorSeg.LowPart × (IntervArranque + IntervMedicion)
   LANZA USUARIOS;
}
```

Petición válida **SI**:
cIniPet >= cIniMed y
cFinPet <= cFinMed
Facilita comparaciones

MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

Función para convertir ciclos a milisegundos:

```
// Variable global calculada a partir de QueryPerformanceFrequency

 float cPorMseg;

// Función que devuelve los milisegundos transcurridos

double MST (LARGE_INTEGER cIni, LARGE_INTEGER cFin) {
  LARGE_INTEGER Dif;
  float MiliSeg;

  Dif.QuadPart = cFin.QuadPart - cIni.QuadPart;
  MiliSeg = (float) Dif. QuadPart / cPorMseg;

  return (MiliSeg);
}
```

Tipos de variables para almacenar los datos:

Son posibles diferentes alternativas (Matrices o Array de estructuras):

```
// Las variables dependen de alternativa de instrumentación elegida
float TiempoDeRespuesta[MAXUSUARIOS][MAXPETICIONES];
float TiempoIniPeticion[MAXUSUARIOS][MAXPETICIONES];
float TiempoFinPeticion[MAXUSUARIOS][MAXPETICIONES];
unsigned long ciclosIniPeticion[MAXUSUARIOS][MAXPETICIONES];
unsigned long ciclosFinPeticion[MAXUSUARIOS][MAXPETICIONES];
```

Alternativas 1 ó 2

Alt. 3

Matrices

```
struct datos {
        int contPet;
        float TiempoDeRespuesta[MAXPETICIONES];
        unsigned long ciclosIniPeticion[MAXPETICIONES];
        unsigned long ciclosFinPeticion[MAXPETICIONES];
}; datos datoHilo[MAXUSUARIOS];
```

Array de estructuras

MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

Instrumentación.- **Alternativa 1**, trabajando con tiempos:

```
usuario()
{ ……….
 do
  { QueryPerformanceCounter(&cIniPet);
        PETICIÓN();
    QueryPerformanceCounter(&cFinPet);
    tIniPet = MST(cIniExp,cIniPet);  // Para reducir intrusividad
    tFinPet = MST(cIniExp,cFinPet);
    ……..
     if ( (tIniPet >= tIniMed) && (tFinPet <= tFinMed) ) { // Petición válida
       TpoResp[Usu][Pet]=tFinPet–tIniPet; // Se guarda Tpo. Respuesta
       TpoIni[Usu][Pet]=tIniPet; TpoFin[Usu][Pet]=tFinPet // Guarda tiempos
       Pet[Usu]++; // Contar petición
       if (Pet[Usu] >= MAXPETICIONES) exit(); // ERROR DESBORDAMIENTO
     }
    ……..
  } while(tIniPet <= tFinMed);
}
```

Variables locales

Variables de almacenamiento de tipo matriz

Variable global

Permite la representación gráfica de la evolución del Tpo. Respuesta con el tiempo

Instrumentación.- **Alternativa 2**, mezclando ciclos y tiempos:

```
usuario()
{…….
 do
  { QueryPerformanceCounter(&cIniPet);
       PETICIÓN();
    QueryPerformanceCounter(&cFinPet);
     ……..
    if ( (cIniPet >= cIniMed) && (cFinPet <= cFinMed) ) // Petición válida
     {  TiempoIniPetición[Usu][Pet] = MST(cIniExp,cIniPet);
        TiempoFinPeticion[Usu][Pet] = MST(cIniExp,cFinPet); // Anotar datos
        Pet[Usu]++;
       if (Pet[Usu] >= MAXPETICIONES) exit(); // ERROR
     }
   ……..
  } while(cIniPet <= cFinMed);
}
```

Variables locales

Variables globales

Permite la representación gráfica de la evolución del Tpo. Respuesta con el tiempo

MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

Instrumentación.- **Alternativa 3**, trabajando solo con ciclos:

```
usuario()
LARGE_INTEGER Difciclos;
{ ……
  do
  { QueryPerformanceCounter(&cIniPet);
        PETICIÓN();
    QueryPerformanceCounter(&cFinPet);
    ……..
      if ( (cIniPet >= cIniMed) && (cFinPet <= cFinMed) ) // Petición válida
      {  Difciclos.QuadPart = cIniPet.QuadPart – cIniExp.QuadPart;
         ciclosIniPetición[Usu][Pet] = Difciclos.LowPart;
         Difciclos.QuadPart = cFinPet.QuadPart – cIniExp.QuadPart;
         ciclosFinPeticion[Usu][Pet] = Difciclos.LowPart;  // Anotar datos
         Pet[Usu]++;
         if (Pet[Usu] >= MAXPETICIONES) exit(); // ERROR
      }
    ……..
  } while(cIniPet <= cFinMed);
}
```

unsigned int
Dependiendo del tiempo puede desbordar el tipo de dato

MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

Instrumentación.- **Alternativa 3b**, trabajando solo con ciclos:

```
usuario()
{ ……
  do
  { QueryPerformanceCounter(&cIniPet);
        PETICIÓN();
    QueryPerformanceCounter(&cFinPet);
    ……..
      if ( (cIniPet >= cIniMed) && (cFinPet <= cFinMed) ) // Petición válida
      {// Anotar datos de inicio y fin de petición
         ciclosIniPetición[Usu][Pet] = cIniPet.QuadPart – cIniExp.QuadPart;
         ciclosFinPeticion[Usu][Pet] = cFinPet.QuadPart – cIniExp.QuadPart;
         Pet[Usu]++;
         if (Pet[Usu] >= MAXPETICIONES) exit(); // ERROR
      }
    ……..
  } while(cIniPet <= cFinMed);
}
```

Definidas de tipo **unsigned long**

6. Almacenamiento de los datos capturados

Método → 1) Almacenamiento en memoria durante el intervalo de medición
2) Volcado a disco al terminar el experimento de carga

(Reduce la intrusividad / Posible si no se generan muchos datos)

```
#define MAXUSUARIOS 150   // Dimensionado del almacenamiento
#define MAXPETICIONES 200 ← ≈ Duración Experimento / Tpo Reflexión
```

Son posibles diferentes alternativas (Matrices o Array de estructuras):

```
// Las variables dependen de alternativa de instrumentación elegida
float TiempoDeRespuesta[MAXUSUARIOS][MAXPETICIONES];
float TiempoIniPeticion[MAXUSUARIOS][MAXPETICIONES];
float TiempoFinPeticion[MAXUSUARIOS][MAXPETICIONES];
unsigned long ciclosIniPeticion[MAXUSUARIOS][MAXPETICIONES];
unsigned long ciclosFinPeticion[MAXUSUARIOS][MAXPETICIONES];
```

Alternativas 1 ó 2

Alt. 3

Matrices

```
struct datos {
        int contPet;
        float TiempoDeRespuesta[MAXPETICIONES];
        unsigned long ciclosIniPeticion[MAXPETICIONES];
        unsigned long ciclosFinPeticion[MAXPETICIONES];
}; datos datoHilo[MAXUSUARIOS];
```

Array de estructuras

MEDICIÓN DE SISTEMAS - INSTRUMENTACIÓN

Detalles a tener en cuenta:

Las variables de almacenamiento y control deben ser globales →
Todos los hilos pueden acceder a ellas

Cualquier otra estructura de datos es posible …

Es muy conveniente que no requiera mecanismos de sincronización entre los hilos (La aplicación instrumentada se ejecuta más lentamente modificando el comportamiento de la aplicación original)

Cada hilo escribirá en su array de la matriz, o en la estructura del array

7. Coordinación con el monitor de Windows

El monitor de Windows se usa para medir la utilización de recursos

- Hay que arrancarlo antes que el inyector
- Hay que pararlo después de que termine el inyector

Pero de todos los datos que ha capturado el monitor hay que usar SOLO LOS QUE CORRESPONDEN AL INTERVALO DE MEDICIÓN

El programa guardará la hora de inicio de la medición para posteriormente indicar cuales son las filas válidas del contador de rendimiento

El programa calculará guardará:

Hora inicio medición = hora_ini_exp + Intervalo arranque
Hora fin medición = hora_ini_exp + Intervalo arranque + Intervalo medición

Ejemplo de uso:

```
#include <time.h> // Fichero de inclusión necesario

main()
{   ……
    time_t hora_ini_exp, hora_inicio_medición, hora_fin_medicion;
    char cadena[26];
    ……
    time(&hora_ini_exp); // Devuelve segundos desde 01/01/1970
    hora_inicio_medicion = hora_ini_exp + IntervArranque; // en Seg
    hora_fin_medicion = hora_ini_exp + IntervArranque + IntervMedicion;
    ……
    LANZA USUARIOS y ESPERA QUE TERMINEN;
    ……
    ctime_s(cadena, sizeof(cadena), &hora_ini_exp)
    printf("\nInicio del experimento: %s\n", cadena);
    ……
}
```