[1 Trapper 3](#_Toc428217023)

[1.1 Consideraciones de diseño 3](#_Toc428217024)

[1.1.1 Overhead (Sobrecarga) 3](#_Toc428217025)

[1.1.2 Flujo de ejecución 3](#_Toc428217026)

[1.1.3 Generación óptima de traps 5](#_Toc428217027)

[1.2 Implementación 6](#_Toc428217028)

[1.2.1 Diagrama de flujo 7](#_Toc428217029)

[1.3 Arquitectura 9](#_Toc428217030)

[1.3.1 Ficheros fuente 9](#_Toc428217031)

[1.3.2 Ficheros de cabecera 10](#_Toc428217032)

[1.4 Mensajes 10](#_Toc428217033)

[1.4.1 Inicio de sesión 11](#_Toc428217034)

[1.4.2 Fin de sesión 11](#_Toc428217035)

[1.4.3 Módulo 11](#_Toc428217036)

[1.4.4 Párrafo 12](#_Toc428217037)

[1.4.5 Call 12](#_Toc428217038)

[1.4.6 Cobertura 12](#_Toc428217039)

[1.4.7 Ficheros 13](#_Toc428217040)

[1.4.8 Perform 13](#_Toc428217041)

[1.5 Instalación y configuración 13](#_Toc428217042)

[1.6 Manual de usuario 14](#_Toc428217043)

# Trapper

Este componente es el encargado:

1. Obtener los tiempos y consumos del sistema imputándoselos al componente correspondiente
2. Generar los mensajes adecuados con esa información
3. Enviar los mensajes generados al Collector

## Consideraciones de diseño

A la hora de implementar este componente hay que tener en cuenta las siguientes consideraciones:

### Overhead (Sobrecarga)

Dentro del flujo de ejecución del conjunto de programas asociados a una sesión cada componente monitorizado: Módulos, párrafos, sentencias CALL, … va a invocar a este componente, en principio, dos veces: a la entrada y a la salida de ese componente.

Esto implica que, respecto al módulo original, al módulo monitorizado se le incorporan otros bloques de proceso, no relevantes desde el punto de vista funcional pero significativos desde el punto de vista de la sobrecarga que la ejecución de estos bloques supone respecto a los tiempos de ejecución y los consumos de CPU.

Por ello, el código necesario para realizar estos controles tiene que ser lo mas eficiente posible en el sentido de requerir el menos número de instrucciones posibles y el menor consumo de recursos posibles.

Si descartamos el lenguaje ensamblador por su dependencia del procesador sobre el que se ejecuta el programa, lo cual limitaría la portabilidad del sistema y su dificultad a la hora de mantenerlo; la siguiente elección es desarrollar este componente en C.

### Flujo de ejecución

Una de las características del lenguaje COBOL es que este no permite la creación de hilos de ejecución por lo que el conjunto de programas asociados a una sesión se ejecutan siempre en un único procesador[[1]](#footnote-1) y sus sentencias se ejecutan secuencialmente

Esto implica que la ejecución normal de cualquier programa COBOL, y por extensión cualquier programa *monohilo*, se comporta como la ejecución de un conjunto de bloques de código de manera secuencial que pueden ser monitorizados, salvo que se produzca un salto incondicional en el flujo de ejecución, con una pila de tiempos en las que se pone en la cima el instante de inicio del proceso de un determinado bloque y cuando este bloque finaliza, se extrae de la pila el instante de inicio, obteniendo el consumo por una simple diferencia de tiempos.

Por ejemplo, dado un programa similar al siguiente:

000009 PROCEDURE DIVISION.

000010 INICIO.

000011 *OTRAS SENTENCIAS …*

000012 PERFORM PROCESO1.

000013 PERFORM PROCESO2.

000014 *OTRAS SENTENCIAS …*

000015 STOP RUN.

000017 PROCESO1.

000018 *OTRAS SENTENCIAS ….*

000019 PERFORM PROCESO11.

000020

000021 PROCESO11.

000022 *OTRAS SENTENCIAS …*

000023

000024 PROCESO2.

000025 *OTRAS SENTENCIAS …*

000026

Código 1: Captura de tiempos

Los puntos a monitorizar serian:

|  |  |
| --- | --- |
| 000010 | Inicio programa (Tiempo relativo = 0) |
| 000011 | Inicio párrafo INICIO |
| 000015 | Fin párrafo INICIO  Fin programa |
| 000017 | Inicio párrafo PROCESO1 |
| 000019 | Fin párrafo PROCESO1 |
| 000022 | Inicio PROCESO11 |
| 000023 | Fin PROCESO11 |
| 000025 | Inicio PROCESO2 |
| 000026 | Fin PROCESO3 |

Y considerando el flujo de ejecución mostrado y asumiendo que cada bloque consume una unidad de tiempo la pila de tiempos tendría un aspecto similar al mostrado en la siguiente figura:



Diagrama 1: Ejemplo de pila de monitorización

Lo cual da los siguientes valores de consumos de tiempos:

| **Tiempo** | **Acción** | **Timer** | **Consumo** |
| --- | --- | --- | --- |
| 0 | Push Proceso | 0 |  |
|  | Push INICIO | 0 |  |
| 1 | Push PROCESO1 | 1 |  |
| 2 | Push PROCESO11 | 2 |  |
| 3 | Pop PROCESO11 | 3 | 3 – 2 = 1 |
| 4 | Pop PROCESO1 | 4 | 4 – 2 = 2 |
|  | Push PROCESO2 | 4 |  |
| 5 | Pop PROCESO2 | 5 | 5 – 4 = 1 |
| 6 | Push Inicio | 6 | 6 – 0 = 6 |
|  | Push Proceso | 6 | 6 – 0 = 6 |

### Generación óptima de traps

Otro aspecto relevante a considerar a la hora de la implementación de este componente, directamente relacionado con la sobrecarga del proceso inherente a la captura de tiempos y generación de mensajes es la optimización de los mensajes que se deben enviar al *Collector* de manera que se mantenga en todo momento una información fiable y suficientemente detallada para poder extraer conclusiones de las mismas.

De acuerdo con la estructura de pila descrita anteriormente, una primera aproximación sería generar un mensaje cada vez que se saca un elemento de la pila (Este sería el modo de proceso definido como **DETAILED**)

Sin embargo, si observamos el siguiente código, utilizado como ejemplo para efectuar una ordenación de datos por el método de la burbuja:

PROCEDURE DIVISION.

INICIO.

CALL CARGA USING DATA-SORT.

PERFORM ORDENAR.

CALL DESCARGA USING DATA-SORT.

STOP RUN.

ORDENAR.

COMPUTE MAX = ITEMS

PERFORM BURBUJA VARYING MAX FROM MAX BY -1 UNTIL MAX = 1.

BURBUJA.

MOVE 1 TO IDX

PERFORM COMPARAR VARYING IDX FROM 1 BY 1 UNTIL IDX = MAX.

COMPARAR.

IF ORDEN(IDX) > ORDEN (IDX + 1)

MOVE ORDEN(IDX + 1) TO AUX

MOVE ORDEN(IDX) TO ORDEN(IDX + 1)

MOVE AUX TO ORDEN(IDX)

END-IF.

Código 2: Ordenación método de la burbuja

Suponiendo un conjunto de entradas de *n* = 1.000 elementos, observamos que al párrafo BURBUJA se ejecutará *n – 1* veces; es decir, *k = n – 1 = 999* y este, a su vez, invocará al párrafo COMPARAR *k – 1* veces cada vez; es decir, la primera llamadas a BURBUJA invocará a COMPARAR 999 veces, la segunda 998 veces, la tercera 997, etc. Hasta que se esté comparando el último elemento, lo cual, para el ejemplo presentado supone que COMPARAR se ejecutará de acuerdo con una progresión aritmética de 1 a 999 elementos con una diferencia de 1.

Parece lógico entonces considerar que la generación de más de 500.000 mensajes para un simple proceso de ordenación no es óptimo ni necesario (salvo en casos especiales) y que, sin pérdida de información se puede generar, al final de la ejecución del módulo, un mensaje por cada bloque de código monitorizado, que contenga los valores acumulados y la cuenta de las veces que se ha ejecutado.

Este modo de proceso es el que se define como **MODULE**.

Extendiendo la lógica descrita a las rutinas utilizadas; supongamos que la rutina CARGA, encargada de cargar los datos en memoria, en lugar de ser invocada una única vez se ejecuta de acuerdo con el siguiente pseudocódigo[[2]](#footnote-2):

CALL CARGA USING ABRIR-FICHERO

MIENTRAS HAYA-REGISTROS

CALL CARGA USING LEER-REGISTRO

ADD REGISTRO TO TABLA

FIN MIENTRAS

CALL CARGA USING CERRAR-FICHERO

Se produce la misma explosión de generación de mensajes y la consiguiente pérdida de rendimiento. Con lo cual se puede extender la filosofía descrita anteriormente y esperar al final de la sesión para enviar los acumulados tanto de los bloques como de los módulos ejecutados.

Este modo de proceso es el que se define como **GLOBAL**.

A modo de resumen, para el supuesto planteado de un proceso de ordenación de 1.000 elementos donde la carga de datos se realiza en otro módulo que es invocado una vez para recuperar cada registro se muestra a continuación el volumen de mensajes generados para cada una de las opciones:

| **Mensaje** | **DETAILED** | **MODULE** | **GLOBAL** |
| --- | --- | --- | --- |
| Sesion | 2 | 2 | 2 |
| Modulo | 1.004 | 1.004 | 3 |
| Call | 1.003 | 2 | 2 |
| Parrafos | 502.005 | 1.007 | 8 |
| **Total** | **504.014** | **1.015** | **15** |

### Generación óptima de mensajes

## Implementación

La librería es llamada, por cada componente monitorizado dos veces: Una cuando entra en el componente y otra cuando sale.

Para realizar el proceso, la librería se apoya en tres estructuras de datos:

* Una pila donde se almacenan los tiempos de inicio de cada componente
* Una pila donde se almacena los módulos activos en el árbol de llamadas
* Una lista donde se almacena la información de cada modulo



Y dos grupos de mensajes:

* Mensajes de inicio (bit menos significativo a 0)
* Mensajes de fin (bit menos significativo a 1)

### Diagrama de flujo

El diagrama de flujo de alto nivel del proceso



Diagrama 2: Flujo SDPTRAPB



## Arquitectura



### Ficheros fuente

| **Modulo** | **Descripción** |
| --- | --- |
|  |  |
| SDPTRAPB | Punto de entrada a la librería.  Se encarga de gestionar el flujo del proceso |
|  |  |
| amqsdp | Gestiona el envío de mensajes |
|  |  |
| cob2c | Convierte la COPY COBOL a una estructura SDP y viceversa |
|  |  |
| md5 | Genera una firma digital MD5 |
|  |  |
| sha256 | Genera una firma digital SHA-256 en función del procesador |
|  |  |
| sha256be | Genera una firma digital SHA-256 en arquitecturas BigEndian |
|  |  |
| sha256le | Genera una firma digital SHA-256 en arquitecturas LittleEndian |
|  |  |
| messages | Gestiona el almacenamiento de los Timers y la generación de mensajes |
|  |  |
| modulos | Gestiona la lista de módulos |
|  |  |
| pila | Gestiona la creación de pilas |
|  |  |
| salw32 | Abstrae las peculiaridades del Sistema Windows |
|  |  |
| timerWin | Obtiene los consumos del sistema en sistemas Windows |
|  |  |
| timerLnx | Obtiene los consumos del sistema en sistemas Linux |
|  |  |

### Ficheros de cabecera

De manera general, los ficheros de cabecera cuyo nombre coincide con un fichero de código, definen las funciones públicas de ese módulo.

El resto de ficheros de cabecera se describen a continuación:

| **Modulo** | **Descripción** |
| --- | --- |
|  |  |
| cmqc.h | Fichero de cabecera del sistema de mensajería MQ Series |
|  |  |
| global.h | Define las variables de uso global dentro del sistema. |
|  |  |
| jggsal.h | Abstrae el sistema de las características específicas de un Sistema Operativo concreto.  Redefine funciones de uso común e incluye las librerías especificas de ese Sistema Operativo |
|  |  |
| sdpConfig.h | Establece las constantes y macros que definen el comportamiento global del sistema |
|  |  |
| sdpTypes.h | Define las estructuras de datos usadas en la librería |
|  |  |
| timer.h | Incluye el módulo de cálculo de Timers adecuado para la plataforma |
|  |  |
| trapCodes.h | Define los códigos de los mensajes.  Estos valores se deben corresponder con sus equivalentes en los otros componentes del sistema |

### Dependencias externas

### Estructuras de datos

## Mensajes

### Inicio de sesión

Este mensaje se envía en el momento que se genera independientemente del estado del buffer de mensajes.

Marca el inicio de una sesión.

|  |  |  |
| --- | --- | --- |
| **Orden** | **Campo** | **Descripción** |
| 1 | Tipo | Identificador del mensaje. Valor: **0** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión  Valor: **0** |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Principal | Firma digital del módulo principal |
| 5 | Usuario | Identificador del usuario |
| 6 | Timestamp | Timestamp del sistema |
| 7 | Elapsed | Tiempo consumido desde que se invocó el programa |
| 8 | CPU | Tiempo de CPU consumido desde que se invocó el programa |

### Fin de sesión

Este mensaje se envía cuando finaliza el módulo principal; es decir, cuando se han vaciado las pilas.

En caso de que el programa acabe de manera inesperada, este mensaje no se enviará y se detectará el fallo del programa

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **1** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Principal | Firma digital del módulo principal |
| 5 | Usuario | Identificador del usuario |
| 6 | Timestamp | Timestamp del sistema |
| 7 | Elapsed | Tiempo consumido desde que se invocó el programa |
| 8 | CPU | Tiempo de CPU consumido desde que se invocó el programa |

### Módulo

Este mensaje se envía cuando finaliza la ejecución de un módulo

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **2** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Etiqueta | Nombre del módulo |
| 6 | Veces | Número de veces que se ha ejecutado |
| 7 | Elapsed | Tiempo total consumido |
| 8 | CPU | Tiempo total de CPU consumido |
| 9 | IntElapsed | Tiempo total consumido en el módulo |
| 10 | IntCPU | Tiempo total de CPU consumido en el módulo |

### Párrafo

Este mensaje se envía cuando finaliza la ejecución de un párrafo

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **4** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Etiqueta | Nombre del párrafo |
| 6 | Veces | Número de veces que se ha ejecutado. Valor: 1 |
| 7 | Elapsed | Tiempo total consumido |
| 8 | CPU | Tiempo total de CPU consumido |
| 9 | IntElapsed | Tiempo total consumido en el párrafo |
| 10 | IntCPU | Tiempo total de CPU consumido en el párrafo |

### Call

Este mensaje se envía cuando vuelve de una sentencia CALL

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **8** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Calling | Firma digital del módulo llamante |
| 5 | Called | Firma digital del módulo llamado |
| 6 | Etiqueta | Nombre del modulo llamado |
| 7 | Veces | Número de veces que se ha ejecutado |
| 7 | Elapsed | Tiempo total consumido |
| 8 | CPU | Tiempo total de CPU consumido |
| 9 | IntElapsed | Tiempo total consumido en el módulo |
| 10 | IntCPU | Tiempo total de CPU consumido en el módulo |

### Cobertura

Este mensaje se envía cuando una vez por módulo. Contiene los flags de los bloques de código que se han ejecutado

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **256** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Flags | Secuencia de ceros y unos indicando que bloques se han ejecutado |

### Ficheros

Este mensaje se envía cuando una vez por módulo. Contiene los contadores de los diferentes accesos que se han hecho a los posibles ficheros utilizados en ese modulo.

Cada fichero contempla 8 tipos de accesos (Apertura, Cierre, Lectura, Inserción, Actualización,Borrado,Lectura secuencial, Modificacion)

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **512** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Accesos | Secuencia con los contadores de los diferentes accesos realizados sobre cada uno de los ficheros separados por el carácter ‘X’ |

### Perform

Este mensaje se envía cuando una vez por módulo. En lugar de controlar el inicio y fin de un párrafo, se controla el inicio y el fin de la llamada a ese párrafo

| **Orden** | **Campo** | **Descripción** |
| --- | --- | --- |
| 1 | Tipo | Identificador del mensaje. Valor: **1024** |
| 2 | Orden | Número de secuencia del mensaje dentro de la sesión |
| 3 | Sesion | Firma digital de la sesión |
| 4 | Firma | Firma digital del módulo |
| 5 | Llamadas | Secuencia con los contadores del uso realizado de cada párrafo separados por el carácter ‘X’.  La identificación del párrafo se determina por su posición con respecto a la declaración de los párrafos identificados en el momento de compilación.  Ejemplo: El valor contenido antes de la primera ‘X’ (Primer párrafo) se corresponde con el párrafo identificado como 1 en la tabla de declaración de párrafos de esa versión de ese módulo |

## Instalación y configuración

## Manual de usuario

1. En determinadas arquitecturas es posible que un programa se vaya ejecutando en diferentes procesadores e incluso en diferentes máquinas a lo largo de su ciclo de vida, pero se mantiene la condición de que, en un instante dado, el proceso solo utiliza un procesador y su información de auditoria (consumo de CPU y de recursos) es consistente a lo largo de esas máquinas y procesadores. [↑](#footnote-ref-1)
2. Esta situación es típica en procesos de acceso a Bases de Datos [↑](#footnote-ref-2)