Mainframe Event Handling
Mainframe Systems Team
Topics

- Mainframe Overview
- Definition
- Description
- Event handler
- Mainframe Events
- Goals
- Mainframe Consoles and Message Suppression
- Events: viewing, dashboard, details, help, flow
- Case
# BNP Paribas Fortis Mainframe Environment Overview

2 Datacenters: 4 Sysplex Environments: A, B, C, T

<table>
<thead>
<tr>
<th>Environment</th>
<th>Site A (AB building)</th>
<th>Site X (Haren)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td>SYA1</td>
<td>SYA2</td>
</tr>
<tr>
<td></td>
<td>SYA3</td>
<td>SYA4</td>
</tr>
<tr>
<td></td>
<td>SYA5 (TSM)</td>
<td>SYA6 (TSM)</td>
</tr>
<tr>
<td></td>
<td>SYA7 (AGI)</td>
<td>SYA8 (AGI)</td>
</tr>
<tr>
<td>QA</td>
<td>SYB2</td>
<td>SYB1</td>
</tr>
<tr>
<td></td>
<td>SYB7 (AGI)</td>
<td>SYB8 (AGI)</td>
</tr>
<tr>
<td>Development</td>
<td>SYC1</td>
<td>SYC2</td>
</tr>
<tr>
<td></td>
<td>SYC7 (AGI)</td>
<td></td>
</tr>
<tr>
<td>Labo/Test</td>
<td>SYT1</td>
<td>SYT2</td>
</tr>
</tbody>
</table>
Physical overview

Reason of the 3rd z196:
Server Time Protocol is the mandatory time coordination protocol used by System z mainframes. Time coordination is a critical function of parallel sysplex. 3 servers must each be assigned a different role to protect STP optimally and thus maximize continuity of operations.

This 3rd server is an extension to the existing configuration and will act as an external coupling facility server. It will add 4x 1 External CF LPARs the existing 4x 2 Internal CF LPARs. The major advantage will be the reduction of zOS CPU cost associated to the elimination of structure duplexing.
In computing, an event is an action or occurrence detected by the program that may be handled by the program. Typically events are handled synchronously with the program flow, that is, the program has one or more dedicated places where events are handled. Typical sources of events include the user (who presses a key on the keyboard, in other words, through a keystroke). Another source is a hardware device such as a timer. Any program can trigger its own custom set of events as well, e.g. to communicate the completion of a task. A computer program that changes its behaviour in response to events is said to be event-driven, often with the goal of being interactive.
Event driven systems are typically used when there is some asynchronous external activity that needs to be handled by a program. For example, a user who presses a button on their mouse. The outside activity causes the event (it fires), some outside hardware and or software will collect data about the event, and when the program signals that it is ready to accept an event, the event will be dispatched to the event handler software that will deal with it.

Events are typically used in user interfaces, where actions in the outside world (mouse clicks, window-resizing, keyboard presses, messages from other programs, etc.) are handled by the program as a series of events. Programs written for many windowing environments consist predominantly of event handlers.

Events can also be used at instruction set level, where they complement interrupts. Compared to interrupts, events are normally handled synchronously: the program explicitly waits for an event to be serviced (typically by calling an instruction that dispatches the next event), whereas an interrupt can demand service at any time.
Event handler

- In computer programming, an event handler is an asynchronous callback subroutine that handles inputs received in a program (ex. a listener in Java).
  
  For example, mouse movements and clicks are interpreted as menu selections. The events initially originate from actions on the operating system level, such as interrupts generated by hardware devices, software interrupt instructions, or state changes in polling.

- Event notification is a term used in conjunction with communications software for linking applications that generate small messages (the "events") to applications that monitor the associated conditions and may take actions triggered by events.
Mainframe Events

API - Application Program Interface
ARM - Automatic Restart Management
CMD - Command
DOM - Delete-operator-message
EOJ - End-of-job
EOM - End-of-memory
EOS - End-of-step
GLV - Global variable
MSG - Message
REQ - End user request
SCR - Screen
SEC - Security
TLM - Time limit-exceeding
TOD - Time-of-day
USS - UNIX System Services
Mainframe Events

• Application Program Interface
  An API event occurs when an application program calls the API interface. Typically, the application that calls the API is a system service provider program, such as a tape library manager, or a network control program. When these programs detect an event that needs attention, they can initiate automation rules by calling the API.

• Automatic Restart Management
  An ARM event occurs when the z/OS Automatic Restart Manager tries to restart an ARM-registered job or started task after an unexpected termination. The restart may occur on the same system or on another system in the Sysplex if the termination was due to a complete system failure.
Mainframe Events

- **Command**
  A CMD event occurs when any z/OS or subsystem command is issued on the system.

- **Delete-operator-message**
  A DOM event occurs when any z/OS component issues a DOM macro to remove a highlighted message from an MCS console; for example, a tape mount message gets internally DOMed when the mount is satisfied.

- **End-of-job**
  EOJ events occur when a task such as a batch job terminates.

- **End-of-memory**
  EOM events occur when any address space such as a TSO user or started task terminates.
Mainframe Events

• End-of-step
  An EOS event occurs when a step terminates in a job or started task.

• Global variable
  A global variable event occurs when the value of an Rexx global variable changes.

• Message
  A message event occurs when a system component sends a message to a console or to a system log. Following types of messages are known:
  • z/OS
  • IMS
  • CICS
  • JES2 or JES3
  • WTOs (write-to-operator), WTORs (write-to-operator-with-reply), and WTLs (write-to-log) generated by an application
  • Log file directed I/O
Mainframe Events

- **End user request**
  A REQ event is triggered on demand by any end user.

- **Screen**
  A SCR event occurs when the screen or state of an virtual terminal changes.

- **Security**
  A SEC event occurs when access to a protected function or feature is made.

- **Time limit-exceeding**
  A TLM event occurs when a job or task exceeds the processor time limit imposed by the system, either by default or by the TIME JCL parameter on the JOB or execute statement. A TLM event also occurs if a non-exempt job exceeds the maximum continuous wait time specified in the SMF parameters for the system.
Mainframe Events

- Time-of-day
  A time event occurs at a specified time or date or after a specified time interval.

- UNIX System Services
  A USS event occurs at the arrival of a USS syslogd message.

Note: in some installations the number and types of events may vary depending on used softwares and customizations.
WTORs during shutdown and IPL

- Large elapsed times are nearly all due to waiting for an operator to respond to a WTOR.
- As operators no longer closely monitor the system, waiting to reply to WTORs can lead to significant delays (>30 minutes is not unusual)
- Operators often do not know what to reply to uncommon WTORs
- Reply delays can affect all systems
Consoles
### Console messages: Route Codes

<table>
<thead>
<tr>
<th>String</th>
<th>Value</th>
<th>String</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSTRUCTN</td>
<td>1</td>
<td>PGMRINFO</td>
<td>11</td>
</tr>
<tr>
<td>MSTRINFO</td>
<td>2</td>
<td>EMULATOR</td>
<td>12</td>
</tr>
<tr>
<td>TAPEPOOL</td>
<td>3</td>
<td>Customer Reserved</td>
<td>13-20</td>
</tr>
<tr>
<td>DASDPOOL</td>
<td>4</td>
<td>Subsystem Reserved</td>
<td>21-28</td>
</tr>
<tr>
<td>TAPELIB</td>
<td>5</td>
<td>IBM Reserved</td>
<td>29-41</td>
</tr>
<tr>
<td>DISKLIB</td>
<td>6</td>
<td>Gen info JES2/JES3</td>
<td>42</td>
</tr>
<tr>
<td>UR</td>
<td>7</td>
<td>JES2/JES3 Reserved</td>
<td>43-64</td>
</tr>
<tr>
<td>TP</td>
<td>8</td>
<td>Processor Reserved</td>
<td>65-96</td>
</tr>
<tr>
<td>SECURITY</td>
<td>9</td>
<td>Device Reserved</td>
<td>97-128</td>
</tr>
<tr>
<td>SYSERROR</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Console messages: Descriptor Codes

<table>
<thead>
<tr>
<th>String</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSFAIL</td>
<td>1</td>
</tr>
<tr>
<td>IMEDACTN</td>
<td>2</td>
</tr>
<tr>
<td>EVENACTN</td>
<td>3</td>
</tr>
<tr>
<td>SYSSTAT</td>
<td>4</td>
</tr>
<tr>
<td>IMEDCMD</td>
<td>5</td>
</tr>
<tr>
<td>JOBSTAT</td>
<td>6</td>
</tr>
<tr>
<td>APPLPRGM</td>
<td>7</td>
</tr>
<tr>
<td>OOLMSG</td>
<td>8</td>
</tr>
</tbody>
</table>
Message Suppression

As we all seek for the highest message suppression rate, the effort can rapidly become disproportioned as maintenance of those rules lead to errors, redundancy, overhead and headache. In short: a nightmare.

Some numbers:

- Production Systems: > 6,000,000 Msg/day
- QA Systems: > 3,000,000 Msg/day
- Development Systems: > 2,000,000 Msg/day
- Test Systems: > 800,000 Msg/day

=> 82,600,000 Msg/week
=> 4,295,200,000 Msg/year
=> Management of Msg suppression is critical.
Message suppression: history

No Suppress:

- Operator needed in front of each console
  - Space: more displays needed (primary/backup)
  - More personnel required when adding systems
  - Tunneled view
  - Improper actions under stress
- Trouble in keeping up with the message rate
  - Faster systems
  - More applications
  - Miss/overlook important messages
  - Bursts which could lead to an outage
Suppress on demand

- Most centers started with MPF (MVS Parmlib member)
  - Select specific or generic message ID’s for suppression
  - Simple, no logical operations (like if-then-else)
  - Regular manual review needed
- Later by using Subsystem applications
  - Logical operations supported
  - Still based on message ID’s, requiring a lot of entries.
  - Maintenance increases with suppression set size.
  - Regular Syslog analysis required (applications added/removed).
Message suppression: MPF

- Suppression via MPF

MPF = Message Processing Facility

Controls message processing for an MVS system:
- Message presentation (color, intensity and highlighting)
- Suppression
- Retention (AMRF) and selection (for NetView)
- Message and Command processing exits
Message suppression: MPF Setup and Control

Setup:

• Through the MPFLSTxx Parmlib member, you can specify which messages the system is to suppress using the msgid parameter with the SUP option.

Control:

• Set MPF=xx
Message suppression: MPF Contents Sample

BROWSE   SYS1.PARMLIB(MPFLST00) - 01.00

Command ===>

/* JES2 MESSAGES
$HASP000,AUTO(NO),SUP(NO) OK (CMD ACCEPTED)
$HASP001,AUTO(NO)              TEXT VIA $DM
$HASP094,SUP(NO),USEREXIT(AORCD14) I/O ERROR ON LINE
$HASP100,AUTO(NO)             LOGS ON STCINRDR
$HASP102,USEREXIT(AORCD16) USER MSGS
$HASP103,USEREXIT(AORCD16) USER MSGS
$HASP110,USEREXIT(AORCD16) XXXXXX -- ILLEGAL JOB CARD
$HASP111,USEREXIT(AORCD16) INVALID /*ROUTE CARD
$HASP112,USEREXIT(AORCD16) INVALID /*JOBPARM CARD
$HASP113,USEREXIT(AORCD16) INVALID /*OUTPUT CARD
Message suppression: Solution

If you don’t want to suppress what you don’t want to see, show only what you want to show.

In other words, apply an Inverted MPF Logic
- Suppress all Msgs and show only the essential information
Message suppression: Exceptions

- Messages with standard IBM suffix
  - A(ction)
  - D(ecision)
  - E(ventual Action)
  - S(evere Error)
  - W(arning)

- WTO’s with action Routing Code
  - 1: Operator Action
  - 3: Tape Pool
  - 5: Tape Library
  - 7: Unit record Pool
  - 10: System/Error Maintenance
Message suppression: Exceptions

- WTO’s with action Descriptor Code
  - 1: System failure
  - 2: Immediate Action required
  - 3: Eventual Action required
  - 11: Critical Eventual Action required

- Command response Messages
Viewing events

- a technical view
- control center
- a business view
- all events
- number of OPEN events
Event Dashboard
Event Dashboard Details
Obtaining help on an event

- Some important event attributes
- Event groups in which the event is shown
- Link to CREMA for 1st level support
- Other useful links
1. Send event by starting a USS process.
2. The USS process executes the EIF command to create the event and uses the Event Integration Facility postzmsg command to send it to the EIF Probe.
3. The EIF Probe receives the event and applies its rules to map the event contents to the Netcool/OMNibus alerts.status table.
4. Netcool/OMNibus stores the event for use in its event management functions (display, correlating, trigger automation, etc).
Case: Automating Sysout Processing

Print 1,000 sysouts on high throughput printers (>400 pages/minute).
Operators issues manually a command to process each sysout.
Printer has a NPS interval between each sysout of 30 secs.
1000 x 30 = 30,000 / 3600 = 8.33h
Printer has no NPS when next sysout is preset.

Solution: capture $hass message event to trigger action for setting up next sysout to process
Main goals of event handling

- To capture specific events from z/OS operating system, applications and hardware.
- Reduce down and outage time as much as possible to a strict minimum (impact: client services, batch window, €)
- Get applications back up and running as quickly as possible after an IPL
- Support for hierarchical stop of all z/OS resources (not restricted to STCs)
- No or reduced operator intervention
- Hide complexity
- Relieve time consuming lookup of command syntax in manuals
- Inform operators of progress
- Faster hang detection & automated resolution
- Minimizing planned as well as unplanned outages
- Improving mainframe service levels
- Apply best practices
- Standardization:
  - Same setup and implementation everywhere (copy/paste)
  - Same code (vendor and user) everywhere
Questions