Assignment 1 and 2: Heartbeat Implementation.

- Overview
- Implementation Snapshot
- Technology Constraints
- Quality attribute scenario
- Readme

Assignment 3: Performance Implementation

- Quality attribute scenario
- Readme

Assignment 1 and 2: Heartbeat Implementation.

Overview

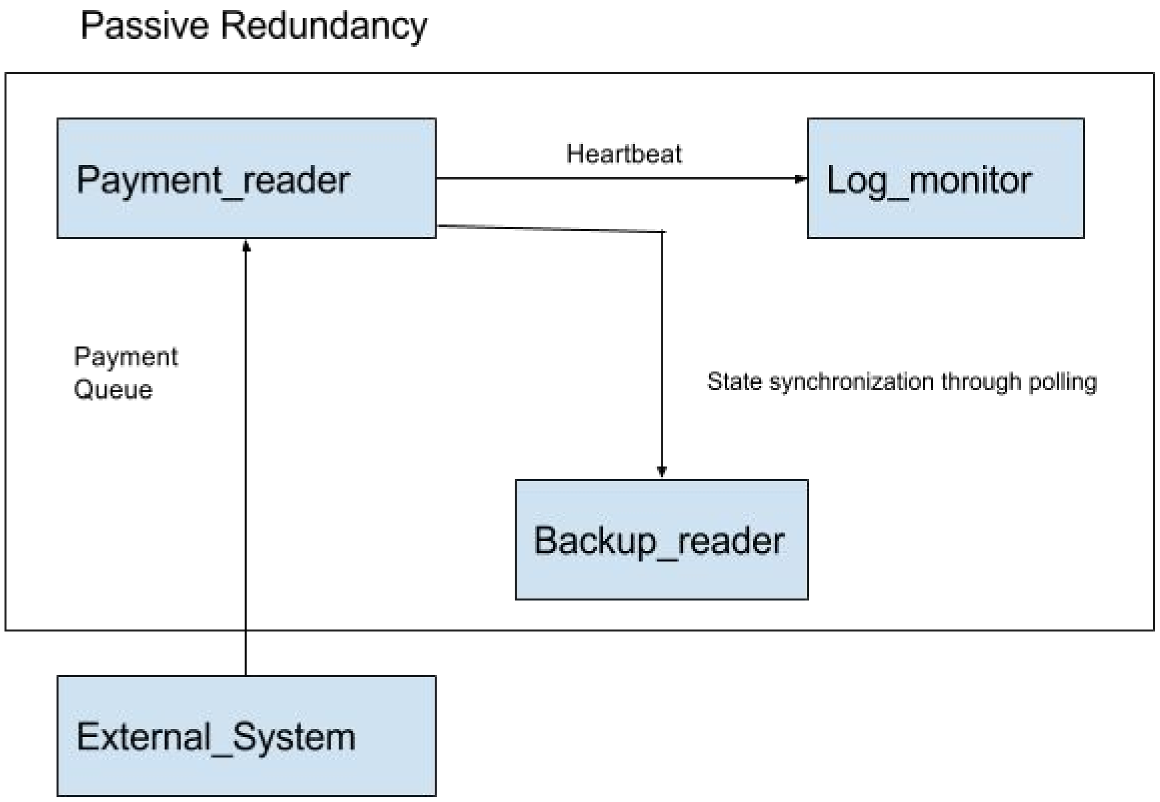

Figure 1

1. **Domain:** Banking domain
2. Overview of the critical process that is being modelled:
    - **External system:** Sends payments to the reader.
    - **Payment_reader:** This process has two threads, one thread is responsible for processing the payments and the second thread is responsible for sending heartbeats to the log_monitor
    - **Log_monitor:** This process is responsible for ensuring that the processes that it monitors are functioning correctly. This process triggers an alert to the ticketing system whenever heartbeats are skipped. The heartbeat timelines and the corresponding severity of the error messages logged are given below:

        20 seconds → Warning message is logged
        40 seconds → Major message is logged
        60 seconds → Critical message is logged

3. Non-deterministic failure in this process which makes it crash.
   In determining a non-deterministic process which stops the heartbeat, we would be setting up a flag field which would kill the thread running the heartbeat alive messages. This flag field

would be set and unset by an external process based on random function generator. The random function generator would set and unset flag based on the random values generated every 5 seconds. Further this can be extended to be based on the testing the desired availability percentage.

4. The tactics that we are planning to implement are:
   Passive redundancy - For passive redundancy, we want the system to be updated for every "x" time interval. The queue object would be passed to the other process. The payment reader module updated a queue object whenever there is a request for transaction. The queue object is static and hence multiple threads try to update it, there is a synchronisation which maintains the order of processing on First Come First Serve basis. The queue object would contain the transaction information as well as its state. After every "x" interval in the system , we would be passing the queue object which holds the information of the transaction as well as the state information, to the passive redundant system. So when the availability of the system is critical, we can still process the transactions by shifting to the redundant system.

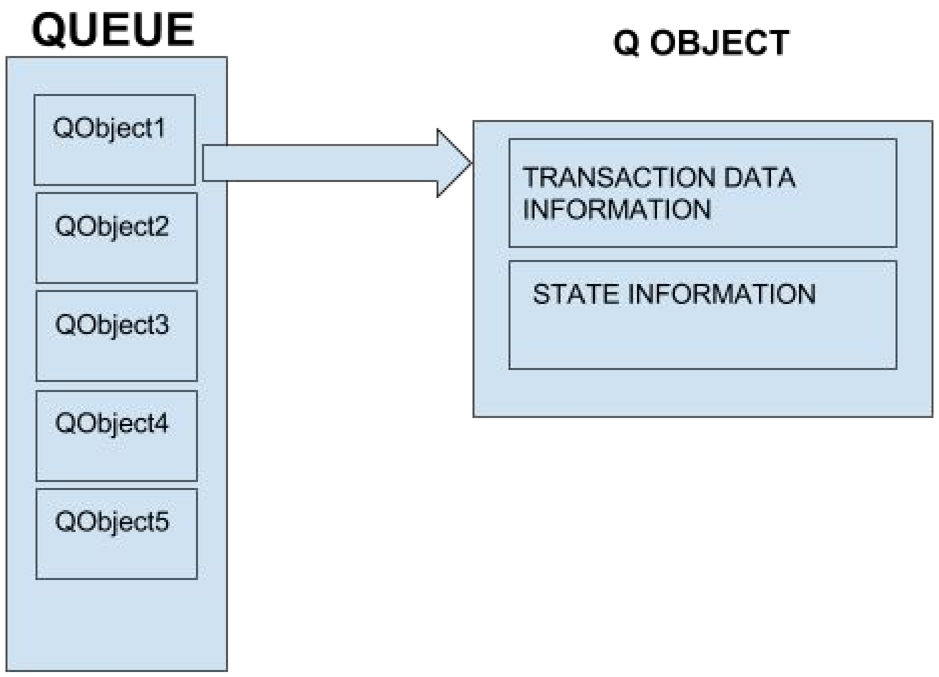

Figure 2

Implementation Snapshot

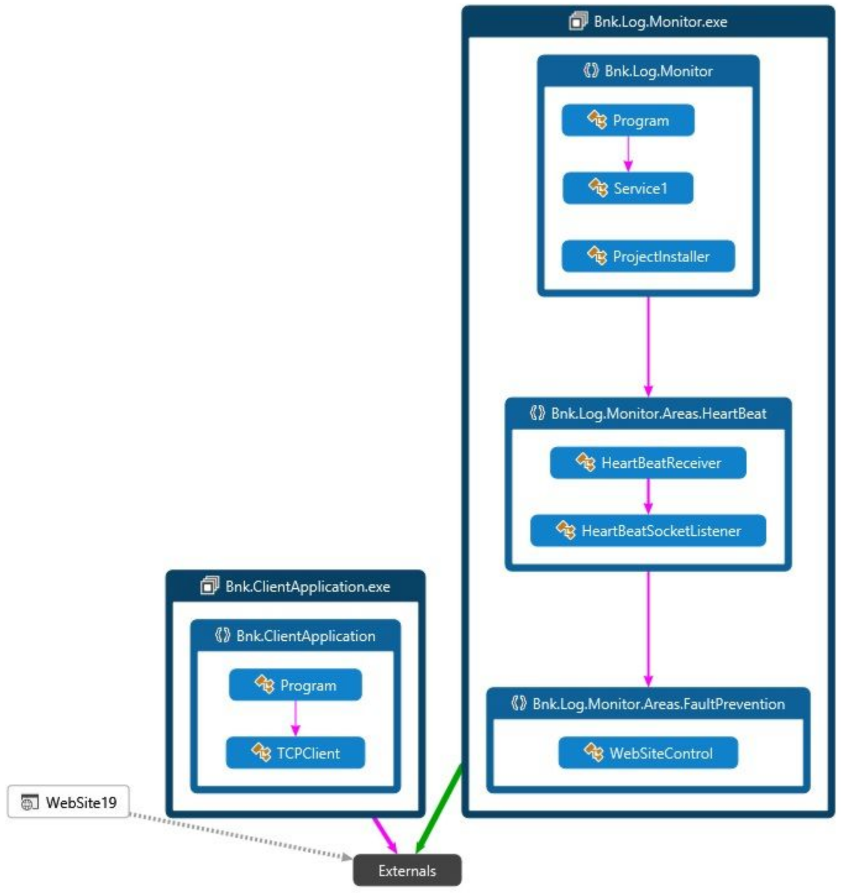

Figure 3

Technology Constraints

We are planning to implement the solution in C# using IPC and pipe objects.

Quality attribute scenario

Scenario	Source	Stimulus	Environment	Response	Response measure
Availability: When the process under consideration crashes, heartbeat	Internal	Process crash	Normal mode	The system logs a critical error and initiates the recovery	The recovery process should start and handle the processing

sender stops sending the heartbeat. The heartbeat receiver waits for pre configured time of 60 seconds. Once the 60 second mark is reached, a critical error is logged and the recovery is initiated				process	within 10 seconds.
Availability: When the process under consideration crashes, heartbeat sender stops sending the heartbeat. The heartbeat receiver waits for pre configured time of 40 seconds. Once the 40 second mark is reached, a critical error is logged and the recovery is initiated	Internal	Process crash	Normal mode	The system logs a major error and continues to wait for the process	The system should wait for the process to return to normal operation.

Readme

The below are the list of deliverables needed to setup the environment and test the implementation of heat tactic explained above:

- **Windows_Features.JPG** - Windows feature snapshot - The features that needs to be enabled are highlighted in blue box.
- **Unhandled Microsoft .NET exception.JPG** - This error has no impact, If the snapshot error is encountered, click on “No, cancel debugging”
- **Submission.zip -** This folder contains the projects

Steps:

1. Enable the features highlighted in the Windows_Features.jpg and restart the system
2. Unzip the contents of the zip file Submission.zip.
3. Solution1 contains the website, the log monitor and the test client application to test the windows service.
4. Bnk.Log.Monitor contains the log monitor and the test client application to test the windows service.
5. Bnk.Log.Monitor project
   a. Open the project and rebuild it.
   b. Open “VS2012 ARM Cross Tools Command Prompt” as an Administrator
   c. Browse to Bnk.Log.Monitor\Bnk.Log.Monitor\bin\Debug and run the below command

InstallUtil Bnk.Log.Monitor.exe

   d. Run services.msc and start the "monitor Service"

6. The Target Framework for all the application and the website is 4.5. Hence you need to register 4.5 Framework in IIS.
   a. Open command prompt as an administrator
   b. Goto C:\Windows\Microsoft.NET\Framework\v4.0.30319
   c. Execute the below command

      aspnet_regiis.exe -i
7. Inetmgr
   a. Run inetmgr
   b. Right click on the sites on the left pane and select "Add Websites" and
   c. Fill the below given details:
      i. Site name - architecture11
      ii. Physical Path - C:\inetpub\wwwroot\architecture11
      iii. Port - 8050
      iv. Uncheck the "Start Website immediately" and click "Ok"
   d. Create another website by repeating Step b
   e. Fill the below given details:
      i. Site name - architecture12
      ii. Physical Path - C:\inetpub\wwwroot\architecture12
      iii. Port - 8060
      iv. Uncheck the "Start Website immediately" and click "Ok"

   Also ensure that none of the websites are running currently.
8. Solution1 project
   a. Open the project and rebuild the solution. While Building the website if there are warnings for referencing missing DLLs, then find the path to folder "Microsoft Web Tools".
      i. Usually for 64 bit Operating System - **C:\Program Files (x86)\Microsoft Web Tools**
      ii. For 32 bit Operating System - **C:\Program Files\Microsoft Web Tools**
      iii. And then correct the path of dll.refresh file displayed while double clicking on the warnings.
   b. In solution manager, right click WebSite19 and select "Publish Web App"
   c. Select architecture11 from the dropdown and click on "Publish"
   d. Repeat step b and Select architecture12 from the dropdown and click on "Publish"
9. Attach the monitor to the project by selecting keyboard"CLT+ALT+P"
   a. In the window that pops up select "Show all users" checkbox at the bottom.
   b. Select Bnk.Log.Monitor.exe process and click on Attach
   c. View in the output window of visual studio about the process log
10. Starting the website - architecture11
    a. Run inetmgr
    b. In the window that opens, click on sites to dropdown the list of websites that have been added.
    c. Rightclick on "architecture11" → Manage Website → Start
    d. Browse the website http://localhost:8050 to initiate the heartbeat messages
11. Observe in the output window mentioned in step 9 for Alive messages which occurs every 19 seconds. There is random failure invocation that stops the heartbeat from being sent. This triggers the alerting mechanism after an interval of 20 seconds. After 40 seconds it will generate a warning messages and after 60 seconds a critical alert will be logged and the Website rolls over to architecture12. This website can be accessed via http://localhost:8060. The probability of failure of heartbeat has been set at 50%.

Reference: http://www.codeproject.com/Articles/5733/A-TCP-IP-Server-written-in-C

Prerequisite:
Windows 8 or 8.1
Visual Studio 2015