ArtCeleration
Project Experience

SAURABH JAGDHANE
ASU ID: 1209572595
(sjagdhan@asu.edu)

SANDESH SHETTY
ASU ID: 1209395990
(ssshett3@asu.edu)

Description:

1. Design a library framework and a service which can be exported to application developers for an Image editing application.
2. The application developer will build apps that can use this library framework to request multiple Image transforms.
3. The library is designed to provide following 5 image transform features:
   {"Gaussian Blur", "Neon edges", "Color Filter", "Sobel Edge Filter", "Motion Blur"}
4. The user can request for multiple image transforms by selecting an option from drop-down menu but the the processed image transform will be delivered in the order of request by the user.

Goals:

1. To design an efficient library framework and service which handles the user's request for any of the image transforms as mentioned in description above.
2. The library will be able to handle multiple requests without waiting for the previous image transform results.
3. The library framework will store user's requests using unique number generated within a library and library will queue all the requests in a FIFO (First in First out) manner
4. Library will offload all the image processing to the service. Service should act as a multithreaded server and can start multiple threads which will include the image processing logic.
5. Threads will inform the library once the processing is done and Shared Memory resources will be used to obtain results.
6. Threads will maintain the unique number and once done processing the respective thread data will be obtained from a queue. Library will check if the thread done processing is at the front of the queue, then only results will be displayed else it will keep waiting adhering to the FIFO mechanism.
7. Queue has to be maintained within a library acting as a FIFO and thread-safe (ConcurrentLinkedQueue).
8. The library service transactions act like a client-server architecture. To achieve modularity service is performed as an isolated process. The library service will communicate over a binder and shared memory.

Design:

Application components:-

1. **Artceleration Library (ArtLib.java)**

This is the library which implements all the methods required by the application: getTransformsArray, requestTransform, registerHandler and getTestsArray. The library acts a client and requests the service which acts as a server to process the transform requests made by the library. The library creates a new MemoryFile (equivalent of

AshMem in Java) every time the requestTransform method is invoked by the application. The requestTransform method in the library writes the Bitmap image to the MemFile and shares the file descriptor for this MemFile with the Service. The file descriptor is made Parcelable using getParcelableFileDescriptor method of MemoryFileUtil.java which is already provided to us. The library binds to the service and receives the instance of the Messenger object created in the service and we send the ParcelableFileDescriptor, type of transform requested, request number and size of the byte array corresponding to the Bitmap input. A global Messenger object is also created which points to a Handler class in library. Also, this Messenger object is set to the replyTo of the Message object sent to the Messenger of the service which the service uses to reply or communicate with the library. Now, when the library's handler receives a message from Service it reads the output bitmap using the file descriptor of the memory file shared by the service and starts a thread in which it continuously checks whether the request number of the head of the queue matches with the request number of the current one because only then we remove the head of the queue and invoke the onTransformProvessed method using the TransformHandler object registered with library by the application.

2. **Artceleration Service (ArtTransformService.java)**

The Artceleration Service does the work of processing each transform request by acting as a Server and creates a new thread for each new transform request it gets in its handler by the library which acts as a client. The service reads the MemoryFile for the input Bitmap using the file descriptor shared by the library. There are five classes associated with service which are threads corresponding to each of the transform type. The service passes the Messenger object of the library, input bitmap image object, request Number to map exactly to the request number in the queue when the output bitmap image is passed to the library and the memory file object which is created by the service for every new request.

. **Interface between application components:**

For communication between the library and the bounded service, Messenger instance of Service is returned in the onBind method of the Service. This Messenger object points to a handler class in service which has an overridden handleMessage to handle the requests made by the library. In this request message, the replyTo is set by the library to the Messenger instance created in it pointing to its own handler class which the service uses to reply back to the client (library) so that the library can read the output bitmap image. To write and read the Bitmap images we use a MemoryFile whose file descriptor is created and shared in both the library and service. For every transform request or transaction between the library and service a new MemoryFile is created.

Strategy:

Most of the time was invested in understanding MemoryFile and Binder transaction over remote processes. Along with this FIFO, Multithreading, Messenger IPC needs to be understood. So, Saurabh started off with creating MemoryFile and writing byte array of the image data into the MemoryFile. Also File Descriptor was passed using Messenger IPC and established a 2-way communication channel. Simultaneously Sandesh started off doing some research on thread-safe queue to be maintained with unique request numbers. Initially, Saurabh designed a service to be responsive to library but Sandesh introduced a multithreading for every transform request within a service for every image transform request to be handled in a separate thread.

No.	Task	Assignee	Deadline
1.	Library-service setup for a primitive data type	Saurabh	10-27-2016
2.	Writing raw image to Memory File	Saurabh	10-28-2016
3.	Thread safe FIFO queue	Sandesh	10-27-2016
4.	Two way Messenger IPC	Sandesh	10-31-2016
5.	Multithreading for every request to the service	Sandesh	11-03-2016
6.	Creation of classes for multiple image transforms	Saurabh	11-04-2016
7.	Handler in a library for response by multiple threads	Saurabh, Sandesh	11-06-2016

Challenges:

1. Two-way communication between Library and Service.

Solution: Using Messenger objects pointing to a handler in both the library and service. The service's messenger object instance is returned to library when the library binds to the service (server). The library (client) sends its Messenger object instance each time it requests a transform request from the service in the Message object. This is used by the service to reply back to the library. MemoryFile is used to share data between the two isolated processes.

2. Handling multiple requests for image transforms:

Solution: The service acts as a multithreaded server which creates a new thread for every transform request made by the library.

3. FIFO queue to maintain the order of requests

Solution: We have used ConcurrentLinkedQueue data structure as it is thread-safe which reduces the overhead of having to specifically synchronize thread operations on this global data structure. Also, we have request numbers to assign numbers to each request which is a static variable in our library class so that it can be mapped with the request number in queue when the processed image is returned to the library from service.

Improvements/ Security Flaws:

1. We will improve the processing time for multiple transform request by using multithreading.
2. Improving the multiple threads handling for performing image transforms in parallel.
3. Reducing the busy waiting for the threads before popping out of the queue.