Anbox on windows: Anbox WSL graphic bridging

Abstract

Anbox puts Android operating system into a container, abstract underlying Android system service into GNU/Linux system service. Existing Android application can thus be ran as a native application on GNU/Linux platform. One shortcoming of Anbox is that it uses Linux LXC container to achieve the goal of separating Android system from the host. It is the obstacle to make Anbox cross-platform since such infrastructure does not exist on windows or Macos. Fortunately, with the help of WSL, we have built a compatible layer to enable Anbox running on windows, and also bridge the graphic rendering command to realize graphic rendering using native host graphic library.

1. Introduction

The ability to run Android application across platform is crucial for development and testing purpose. It also meets demand of some users to run some Android application and games without an Android phone. Although most Android application is written in Java, a platform independent language, they use various native libraries that are exclusive on Android. For example, while directx are natively supported by most modern PC hardware, Android uses a subset of OpenGL computer graphic rendering application programming interface called OpenGL ES that is usually not supported on windows or macos. In order to use OpenGL ES, an additional compatibility layer must be added to translate OpenGL ES rendering command into native graphic rendering API such as Direct3D or OpenGL.

As we have managed to compile Anbox and a modified version of Android image provided by Anbox, we realized running Anbox on WSL2. However, this method is not without cost, as Anbox in this case uses existing infrastructure on a virtual machine such as lxc container and OpenGL rendering interface. As we want to utilize the rendering library on host to boost performance, a graphic rendering command bridging must be built to redirect the rendering commands from Android container to host and translate them into recognizable host library rendering API.

However, the structure of Anbox complicates the issue. The Anbox uses so-called session manager to act as a proxy to interact between user and Android container. More precisely, Anbox on one hand accepts graphic rendering commands from Android container and display them, on the other hand accept user input including keyboard, mouse etc and send them to Android container using designated stream protocol. In fact, Anbox supports 3 types of remote procedure calls. One of them is the usage of Unix domain socket, it is also the most widely used in the vanilla version of Anbox, both graphic rendering pipe and input redirection are realized using Unix domain socket. Another of them is TCP socket, it is only used for adb debugging purpose. Anbox also utilizes dbus as a meanings of initializing and runing a new application on Anbox Session manager. Since WSL2 only support tcp sockets as a mean of connection with the host, we converted all of those 3 types of connection into Tcp connection.

2. Architecture

Anbox consists of Anbox container manager, Anbox session manager and an Android container. Anbox is essentially and set of software that bridges connection between those component and provides an intuitive Android application view on the user end.

2.1 Anbox Session Manager

Anbox Session Manager consists a window manager for displaying running Android application, various connector to bridge with Anbox container manager and LXC container and also input manager to process input from user as well as a application manager to start Android application. When Anbox Session Manager starts running, it will set up a window manager to open a new window and set up input channel. Then it will create socket stream connectors to accept connection from Android container. Two of the important connector is QEMU connector for accepting graphic rendering commands and another is bridge connector to bridge user clipboard into LXC container. The input bridging is realized using the shared unix domain folder. More precisely, input manager will create virtual devices for input device such as keyboard, mouse and use connector and map them into unix domain socket corresponding to that device under a folder that will be mapped into LXC container.

In my origin implementation plan, ideally I want to move the Anbox Session Manager to host and keep the rest of Anbox component running inside WSL linux virual machine. However I meet some obstacles in implementing that plan. The Anbox Session Manager contains lots of components the couple tightly with Linux system that are not easy to be moved to windows host. For exa the application manager which is used to start Android application, uses dbus as means of inter-process connection which is not supported in windows.

To accommodate those issues, we have the Android session manager run as two parts, the Android session manager runs inside WSL is in charge of running the application manager and input manager as well as the bridge connector. It will also communicate with the container manager to start the

LXC container with the correct unix socket mapping. The Android session manager runs on host on the other hand is in charge of the qemu pipe through a Tcp connection since WSL does not support unix domain socket connection. It also contains the display manager for the display of Android application and input manager for accept the user input and send them to the input manager located inside WSL. The input manager inside WSL in this case is more like a input message forwarder to establish the communication with the Android container. Additionally since Android session manager on host can use Tcp connection for QEMU bridging. We use socat to relay the data between the tcp socket and the designated unix domain socket. The socat is ran on WSL and listen for any connection to the qemu pipe unix socket. And then it will established tcp connection with the Anbox session manager on host to enable the qemu byte stream between the Android container and session manager on windows. A illustration of the bridging structure is shown below.

2.2 Anbox Container Manager

Anbox container manager is another important component of Anbox, it is in charge of set up the LXC container for Android. Especially it will set up binder and ashmem, two crucial Android service for Android inter-process communication and memory mapping. It will also use unix domain socket to accept connection from Anbox container manager in order to start container or stop container according to incoming transmission. In this case the Anbox container manager will form a connection with Anbox Session Manager located inside WSL to utilize existing unix domain socket.

2.3 QEMU Pipe

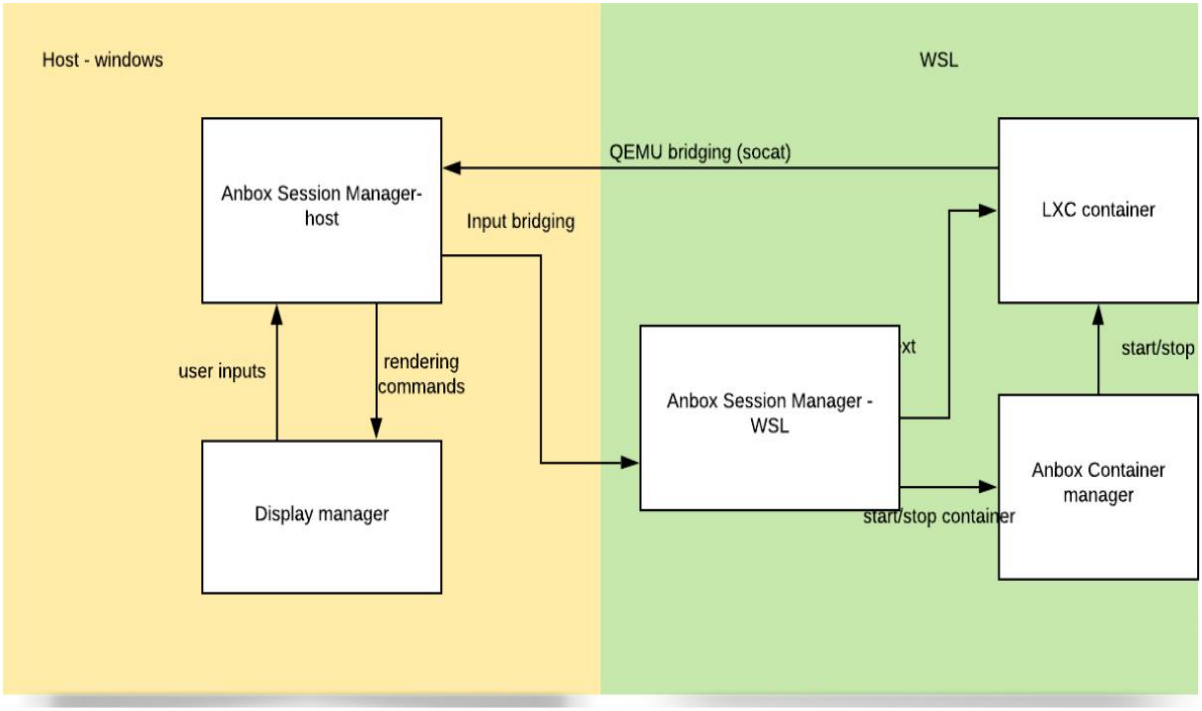
The concept of QEMU pipe is originally introduced in android emugl as a part of Android emulator. The idea of the use of QEMU pipe is to separate the emulator or container process from the graphic rendering process. When guest egl system Library is used, the guest kernel will transmit the graphic rendering command via a protocol stream to the graphic renderer which will decode and dispatch the graphic rendering commands to host translator libraries. Although there are multiple such egl translation library, we use angle in this project. Angle is almost native graphic layer engine developed by google that translates opengl es API call to windows direct3D API. It is currently under active development and it is widely used by chrome and firefox as well as application that ported from Android.

2.4 Linux API on windows

Another major obstacle to bringing Anbox Session Manager to windows is the Anbox's dependency on linux. Although the linux LXC container's function is not needed to run a Anbox Session manager, there are some components that depend on linux utility to function such as timeUtil and management API to executable path, even the Android emugl library need some modification to work on windows. Those API are rewritten to work under existing windows API.

2.5 Input Processing

In order to interact with user to provide app usability, Anbox uses SDL's cross platform input API. As SDL is compatible to windows, it is used for our modified Anbox Session Manager to capture user inputs including keyboard input and touch/pointer input. Hoever, as anbox uses main thread to create a sdl window and uses another thread to wait for input event, the application windows is not responding



because windows does not support sdl to use thread to wait for window event. The issue is worked around by placing a loop at the main thread to wait for SDL events.

3 Evaluation plan

To prove the porting is successful on windows, we will need to test various application function correctly on our modified version of Anbox. Andoird application like tumbler, flipboard that can run on unmodified version of Anbox should still run correctly on our windows version of Anbox session manager. In addition, to measure the performance difference, we will run Passmark to measure the graphic rendering efficiency as compared to unmodified version of Anbox.

4 Experiment

We use Visual Studio to set up environment on windows and then run host side anbox session manager. Then the socat is ran on WSL, after the socket relay is set up, Anbox Session Manager on WSL is started with a single window mode. We have tested using angle as a translation library on windows to translate opengl es API into directx11 API. However, because of compatibility issue, the program crashes at the process of rendering graphic commands. It can be attributed to opengl translator library behaviors differently on different platform as the Android System image is compiled and ran on a linux virtual machine and the graphic rendering part of anbox is ran on windows. Then we took the advise provided by Android emulator guide and compiled the opengl es translation library(egl) using swiftshader. The swiftshader library basically implements the opengl es API just using CPU. Without using GPU, it eliminates the unstable behavior of opengl es translator. The experiment results show no error in running the graphic rendering commands. However, the result is not being displayed on application window. This may be related to windows application window connection issue. In addition, the socat conversion and tcp socket connection adds significant overhead. The vanilla version of anbox ran on linux wsl takes around 10 seconds to initialize and display the application manager GUI. However the bridged version of anbox session manager takes around 5 minutes to finish executing commands to render the application manager GUI.

5 Related work

There are various version of implementation of running Android application on a desktop. For example, BlueStack is a emulator that provide Android gaming experience on desktop platform. Android-x86 is the project that port Android Open Source project to x86 platform that hosts different patches for Android x86 platform from open source community. Those software however all uses vitalization technology to provide an Android platform. Thus although those software did provide the ability to run on desktop platform, some vm overhead has been introduced. The Anbox, on the other hand, does not include an vm but a linux container to achieve the same goal since Android essentially uses a linux kernel. Such utilization of existing infrastructure reduce the redundancy of introduce a vm on linux platform but it brings application compatibility issues. For the image bridging part, most Android emulator utilizes the fact that opengl es translator library can also be installed on host-side. Thus the opengl es graphic rendering commands can be directly bridging into host image rendering translator library. If connection uses a qemu pipe on the same platform, it can significantly boost the graphic rendering efficiency. However, this approach may encounter compatibility issue when graphic rendering commands are bridged between different platform. The THINC virtual display architecture on the other hand intercept the graphic rendering commands and translate to a stateless client side graphic rendering commands called THINC Protocol Display Commands. Thus it gets the rid of the compatibility issues of different graphic rendering library or even the different graphic rendering library implementation of a same graphic.

6 Future work

Unfortunately after code migration, the anbox session manager cannot render the graphic on the application window even the graphic rendering commands are executed on the host and no errors return. The next step is apparently to figure out what is wrong with graphic display. In addition, the approach of using socat to convert unix socket connection to tcp connection does not prove promising in terms of efficiency. So alternative approaches to implement qemu pipe can be exploited to improve efficiency. For example, the android image code can be tweaked to directly establish a tcp connection with the anbox session manager on windows without using socat to relay the connection. Other than that, due to numerous compatibility issues encountered during building the project and debugging the project, the project progress is slower than expected and the input command bridging has not been implemented. In future work, those function will be implemented and the evaluation plan described above will also be executed.

7 Reference

Ricardo A. Baratto, Leonard N. Kim, and Jason Nieh, *THINC: A Virtual Display Architecture for ThinClient Computing*, SOSP'05, October 23–26, 2005

Anbox, https://anbox.io

Windows Subsystem for Linux 2, https://docs.microsoft.com/en-us/windows/wsl/wsl2-install