Revolutionizing PC Classroom with FreeBSD

Chun-Cheng Yeh
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
leafoliages@gmail.com

***Abstract*—This paper presents a redesign of PC classroom infrastructure using FreeBSD and its native hypervisor, Bhyve. The proposed system addresses limitations of current Proxmox VE based virtual machine deployments by enabling direct hardware control and improved system flexibility. Key contributions include a user-friendly guest-switching mechanism, an accelerated deployment system utilizing ZFS incremental backups, and an integrated screen sharing solution for enhanced classroom interactivity.**

***Index Terms*—FreeBSD, Bhyve, ZFS, PC classroom, virtualization, screen sharing**

I. Introduction

The PC classrooms administered by NYCU CSIT uses Proxmox VE with VMs tailored to meet the requirements of each course. However, the design of such a system limits direct physical control over the underlying hardware and consequently reduces system flexibility. This project seeks to redesign the PC classroom infrastructure by adopting FreeBSD and its native hypervisor, Bhyve [1], thereby enabling instructors and students to fully leverage the capabilities of a more open educational computing environment.

The PC classroom provides Windows and Linux systems by default for course use. However, some courses require customized environments. To conveniently switch between operating systems or course environments, we use virtual machines (VMs). These environments are installed as virtual machines on Proxmox VE and are switched when classes change. The graphics card and USB controller are passed through to the guest system, allowing it to function as if it were running on a physical computer. As a result, Proxmox VE hosts can only be managed over the network.

To deploy a guest environment to every computer, we first create a prototype virtual machine on one Proxmox VE host and export it as an image. This image is then distributed to each computer via BitTorrent, after which the guest system is restored from the image.

One disadvantage of a passthrough-based VM system is the inconvenience of switching between virtual machines. If a class needs to use multiple VMs simultaneously, passthrough restricts access to the host and other guest systems. Therefore, the first objective of this project is to develop a guest-switching mechanism with a user interface that allows users to select and launch the desired virtual machine.

The image deployment process can also be improved. Since an image may be several gigabytes in size, distributing it to hundreds of computers can take several hours. Thus, the second objective is to design an accelerated deployment method for customized guest environments to improve overall efficiency.

Finally, the third objective is to design and integrate a screen-sharing solution for instructors to enhance classroom interactivity and instructional effectiveness.

II. Development

A. VM Switching Mechanism

Instead of passing the graphics card through to the guests, the guest-switching mechanism is implemented using VNC. Bhyve provides native support for VNC by exposing a VNC port that allows access to the guest. A VNC viewer is then used to display the guest's screen by connecting to the exposed VNC port. In this project, tigerVNC [2] is used. A menu application is developed to manage the switching process: it starts the selected guest and opens the VNC viewer to display its screen. The VNC viewer application is modified so that it can be exited with a single hotkey binding. Native support from Bhyve and its ability to enable fast switching are the two main reasons VNC is chosen as the display mechanism. In addition, it enables access to remote guests, which will be discussed later in this paper.

The menu application is implemented using PyQt5 [3] and can be launched simply with `python app.py`. The left panel lists all guests installed on the host. This application is integrated with `vm-bhyve` [4], and uses the command

```
vm list | grep -v CPU | awk '{ print $1 }'
```

to retrieve the names of all guests. The right panel displays basic information about the selected guest along with a screenshot of its current state. This is achieved using `asyncvnc` and its screenshot function. After establishing a VNC connection, the screenshot function returns a pixel array representing the captured screen. The pixel array can then be converted into an image and saved using `PIL`. To display the screenshot, a `QPixmap` object is constructed from the saved image and inserted into a `QLabel` using the `setPixmap` method. Figure 1 shows the layout of the menu application.

```
<QLabel object>.setPixmap(
  QtGui.QPixmap("shot.png"))
```

Since the menu application is not executed with root privileges, commands that require elevated permissions, such as `vm` and `zfs`, must be run with proper privilege escalation. Using `sudo` is not suitable here, as it would require manual password

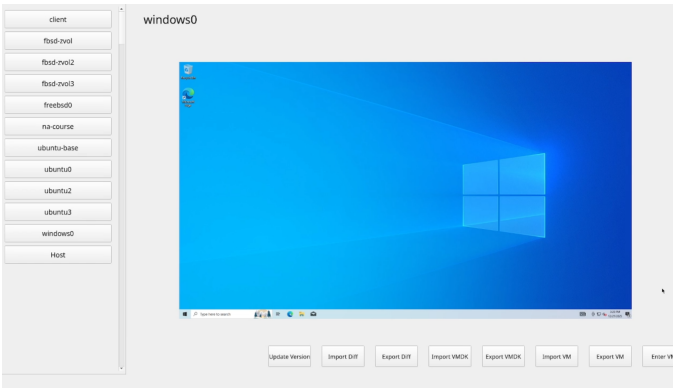

Fig. 1. Menu App

entry. Instead, the solution is to grant wheel and operator privileges through `mac_do` [5]. This can be configured using the following `sysctl` setting:

```
security.mac.do.rules=
  "uid=1001>uid=0,gid=0,+gid=0,+gid=5"
```

This configuration allows access to root-required binaries and devices. A limitation of this approach is the lack of a command whitelist; security could be further improved by restricting privilege escalation to only explicitly permitted commands.

B. Deployment Improvement

Guest machine disks are stored in ZFS zvolumes [6]. By utilizing ZFS incremental backup, modifications made to a guest between snapshots can be exported as a difference image. Instead of deploying the entire disk image, distributing the much smaller difference image significantly accelerates the deployment process.

The deployment scheme can be divided into three stages: preparation, customization, and deployment. In the preparation stage, base system guests such as Windows, Linux, or FreeBSD are installed and pre-deployed to all Bhyve host computers. In the customization stage, instructors obtain the base image published by the classroom administrator and apply their required customizations to the guest. Finally, in the deployment stage, the customized guest image is returned to the classroom administrator, who exports its difference image and distributes it to all computers. By merging the difference image with a copy of the pre-deployed base image, the customized guest required for each lecture can be restored. Figure 2 illustrates this process.

Four ZFS-related operations are involved in this deployment scheme: export guest, import guest, export diff, and import diff. As mentioned previously, this scheme requires the guest disk to be installed on a zvolume. Some ZFS-related operations are handled using the `pyzfs` Python library [7]. The "Export guest" operation exports a guest as a ZFS image using:

```
lzc.lzc_send(vm_base_snap.encode(),
  fromsnap=None, fd=f.fileno())
```

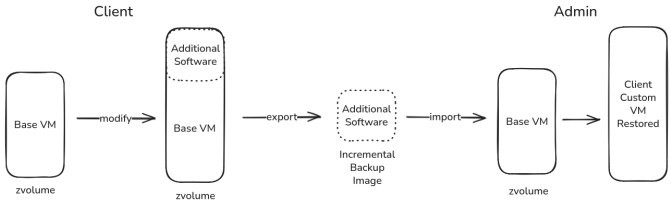

Fig. 2. Deployment with difference image

If the base snapshot named "snap0" does not exist, a snapshot with this name is created before exporting. It serves as the reference point for snapshot comparison to ensure that subsequent difference-image operations function correctly. The "Export guest" operation is typically used to export the base system image. The exported image can then be restored using "Import guest", which is implemented as:

```
cat <image_path> | zfs recv <target>
```

To generate a difference image, a new snapshot is created and compared with the base snapshot "snap0". The difference between these two snapshots can be exported using:

```
lzc.lzc_send(vm_new_snap.encode(),
  fromsnap=vm_base_snap.encode(),
  fd=f.fileno())
```

The difference image can be imported into the same guest (i.e., the same base system) on another computer from which it was originally exported. The "Import diff" function verifies that the target guest contains an identical base snapshot. Only when the base snapshot matches can the import succeed. A successful import updates the zvolume from the initial base state to the customized state. A failed import usually indicates that the target zvolume differs from the one from which the difference image was generated. Similar to the full import process, this function is implemented using `zfs recv`.

Additional "Export vmdk" and "Import vmdk" operations are included to support exporting and importing guests in VMDK format. This feature allows guests to be customized using hypervisors such as VirtualBox. These operations are implemented as follows:

```
dd if=/dev/zvol/<vm_disk> \
  of=<image file> bs=1M
qemu-img convert -f raw -O vmdk \
  <image_file> <vmdk_file>
```

C. Screen Sharing

Screen sharing is implemented using VNC. Following the same logic as guest switching, the screen of a remote guest (usually the instructor's machine) can be accessed through the VNC port exposed on that machine. However, Bhyve's VNC port supports only a single client connection at a time. To allow the entire class to connect and view the instructor's screen simultaneously, VNC reflector [8] is placed in front of Bhyve's VNC port. It re-exports the VNC stream to another port, to

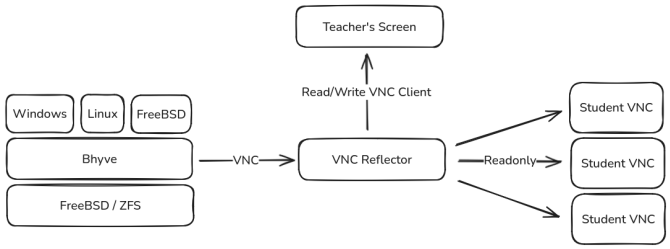

Fig. 3. Local and remote screen showing

which students can connect. A read-only policy for student (external) connections can be enforced by setting a password for read-write mode in VNC reflector. The following command starts a VNC reflector daemon that exposes port 6900, using the Bhyve VNC information provided in the `host_info` file and the password specified in the `passwd` file:

```
vncreflector -l 6900 -p <passwd> \
  <host_info>
```

To integrate local and remote guest screen display, the VNC client must connect to the ports exposed by VNC reflector rather than directly to Bhyve. One approach is to configure VNC reflector to expose only a single port and have the menu application keep track of the currently active guest, so that VNC reflector dynamically connects to the corresponding Bhyve VNC port. Another approach is to spawn a separate VNC reflector process with a distinct exposed port for each newly started guest; the new port can be assigned in a linear relationship to the original port. Since the latter approach is easier to manage, it would be adopted in this project. Figure 3 illustrates the overall VNC-based architecture.

Another issue arises in the integration of VNC reflector: incompatibility between the screenshot function and VNC reflector. Because the reflector distributes connections to multiple clients, it requires each client to connect with the "shared" parameter enabled. However, most VNC clients or libraries that support screenshot functionality do not implement the "shared" parameter, causing the screenshot feature to fail. This issue remains unresolved and may require patching third-party libraries, which is beyond the scope of this paper.

III. Extended Goals

A. Difference Image Version Control

In cases where a lecture requires a quick update of the guest—whether installing new software or reverting to last month's state—a version control system for difference images would be highly convenient. A central repository could be deployed to receive updated difference images from instructors and distribute the newly uploaded images to all students. As long as the instructor and students share the same guest with an identical base snapshot, the updated difference image can be applied immediately after it is downloaded.

IV. Summary

This project introduces three key improvements to the existing PC classroom system: label=0.

1) A locally managed environment-switching mechanism that offers greater flexibility compared to the centrally controlled configuration used in the current system.
2) A faster method for deploying customized environments, achieved by distributing only the incremental backup relative to the base image rather than the entire image.
3) A streamlined screen broadcasting solution seamlessly integrated into the system to facilitate classroom interaction and content sharing.

References

[1] FreeBSD Bhyve, https://wiki.freebsd.org/bhyve
[2] TigerVNC, https://tigervnc.org/
[3] Qt for Python, https://doc.qt.io/archives/qtforpython-5/
[4] vm-bhyve wiki, https://github.com/churchers/vm-bhyve/wiki
[5] mac_do(4) FreeBSD manual, https://man.freebsd.org/cgi/man.cgi?query=mac_do
[6] OpenZFS, https://openzfs.org/
[7] pyzfs documents, https://pyzfs.readthedocs.io/en/latest/
[8] vncreflector port, https://www.freshports.org/net/vncreflector