JOSnet:
JOS on Amazon's EC2 Cluster

Yanni Coroneos, Rob Sloan

Abstract:

To prove the efficacy of JOS by successfully running it on either a xen ParaVM or HVM

Motivation:

At the end of lab5 JOS was perfectly placed as a base operating system to further develop for specific applications. As a proof of concept, our aim is to tailor JOS to run on a distributed grid of VM's with Xen as the VMM. Future development could entail traditional grid computing concepts or a botnet. A botnet would be cool.

Goals:

Boot JOS on Xen and on an EC2 node

Process:

Since vm guests are not traditionally physically accessible, a suite of remote administration tools had to be developed. In the context of JOS, this consists of a

1. simple network stack,
2. file system administration programs,
3. file system imaging programs,
4. a bootable image of JOSnet

The network stack is the result of lab6, namely an e1000e driver with a an LwIP network server. Specifically, the driver uses a polling process for receive and transmit. The driver also has the ability to read the MAC address off of the e1000e. This is useful for packet filtering.

The JOS library of user programs was greatly extended with the addition of directories, unlink, an rsh server, and an shtml-enabled web server. The addition of unlink arose from a natural need of freeing disk space. The implementation simply consists of a user-level program that nicely asks the filesystem to zero out the contents of the inode corresponding to the target file or directory. The rsh server is quite elegant in its interface to sh: it duplicates the socket file descriptor to stdin and stdout and spawns /sh.

```
int service(int client)
{
   //dup stdin and stdout
   dup(client, 1);
   dup(client, 0);
```

```
        //spawn sh
        int id=0;
        id=spawnl("/sh", "sh", (char*)0);
        if (id<0)
        {
                        printf("spawn sh: %e\n", id);
          return -1;
        }
        wait(id);
        return 0;
      }
```

The purpose of the shtml web server is to enable high level observation of the JOSnet node. The shtml protocol is somewhat analogous to an http shell except the web admin gets to choose which commands are exposed and how. Upon receiving a GET request, the web server opens the file and then checks the first line for an shtml expression of the form where ‘string’ is a valid shell command.

```
"<exec ‘string’>
```

In order to deploy JOSnet quickly on a large number of nodes, the filesystem must be pre-baked with a certain layout and our administration binaries. The original build system for JOS included a primitive utility called fsformat which could add specific binaries to a single root directory. It had no support for sub-level directory creation. We have enhanced fsformat to duplicate an entire directory tree in the JOSnet filesystem. We accomplished this by recursively walking a provided root directory using standard POSIX libraries. With our new utility we pre-bake every JOSnet image with our administration binaries and a root path for our webserver.

In order to even boot, JOSnet has to be compatible with modern bootloaders. We chose to make it compatible with GRUB legacy because this is what xen uses. The boot process begins with a GRUB El-Torito stage thats jumps into the kernel entry point. From there, JOSnet is in control and boot normally.

An additional feature of JOSnet is its ramdisk. The entire filesystem gets put into a data segment of the userspace file system according to our linker scripts. Stubs were added to ide.c so that the underlying fileystem interface remained intact. The ramfs is selectable with a define in memlayout.h.

testing on QEMU:
Before doing anything on xen, we regularly test it QEMU because of its enhanced debugging abilities. QEMU can successfully run JOSnet with/without the ramfs.

testing on a Xen ParaVM guest:

This is our primary target because the majority of EC2 nodes run ParaVM guests. In order to get JOSnet to boot on a ParaVM we had to modify the bootloader to include a xen_guest header that describes JOSnet to the vmm

```
.section __xen_guest
  # Lol I'm totally linux
  .ascii "GUEST_OS=linux",
  .ascii ",GUEST_VER=2.4",
  .ascii ",XEN_VER=xen-3.0",
  .ascii ",VIRT_BASE=0x0",
  .ascii ",ELF_PADDR_OFFSET=0x0",
  .ascii ",HYPERCALL_PAGE=0x02",
  .ascii ",PAE=yes",
  .ascii ",LOADER=generic"
  .long   0

.section .xen_shared_info
  shared_info:

.section .xen_hypercall_page
  hypercall_page:
```

After successfully reading the header, xen would continue running JOSnet. Unfortunately, JOSnet only gets three instructions executed after the entry point before it crashes or xen kills it. We never figured out what was happening.

Testing on a xen HVM guest:

Xen HVM's are intended to more closely emulate real hardware, unlike a ParaVM where the kernel is supposed to know it's running in a VM. JOSnet works on a xen HVM instance with/without the ramfs.

Testing on Amazon EC2:

As of recently, Rob's EC2 account got hacked and Amazon has blocked access to him and everybody else. In theory, we can spawn an HVM with JOSnet but there's no guarantee that it will work. Maybe by the checkoff meeting we will have tried and debugged this.