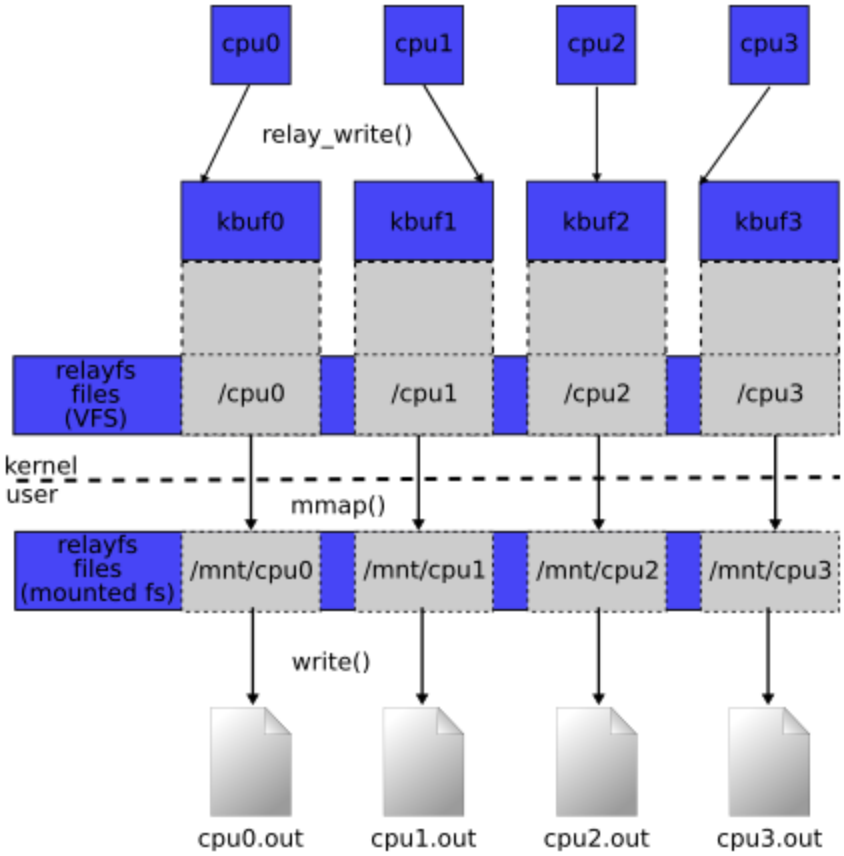
General Overview



This diagram shows each cpu logging data into their own respective kernel buffers using the relay_write() function found in the relay.h file. These buffers are represented on the debugfs that show up to the user side of the graph only when the file system is properly mounted. Then the kernel buffers can be mapped using mmap() from a user space application onto each relay file. As the data streams into the kernel buffer, the data can be written to the disk and analyzed for more information.

Relay and Debugfs

Relayfs are per-cpu kernel buffers that the kernel source code writes into. These buffers represent files to be mmap'ed by the user space program and are to be analyzed for kernel information. Each cpu writes into its own kernel buffer using the method relay_write(). Initialized with relay_open() in the main.c of the kernel files, data is being written and processed into the kernel buffers through the kmalloc and kfree functions in the slab.c file. Writing tracing functions into those methods allow the kernel to write into the buffer, using relay_write(), information about memory that is being allocated and deallocated by the kernel.

It is important to note that the old relay interface was called relayfs and has since changed its name to relay with the file system mounted along with the debugfs. When mounting the filesystem, the command has been changed to *mount -t debugfs debugfs /<directory>* with the -t flag on the mount command indicating that the object to be mounted is of a filesystem type and the directory indicating where to mount the actual filesystem.

The debugfs is initialized in the do_basic_setup() function of the main.c file of the kernel system "for lack of a better place." Once the debugfs is mounted, channels can be created along with the per cpu buffers. These files are named in relation to the number of online cpu's (i.e. cpu0 cpu1 cpu2 … cpuN) with one buffer per cpu. The relay_open command stores in parameters of the name of the channel, the parent parameter which is the root filesystem by default if no value is set, the sub buffer size, the number of sub buffers, and a reference to the relay channel client callback functions. The relay channel callback functions are a struct that contains the functions to create and remove a buffer file as well as sub buffer controls that monitor the files being created, destroyed, enabled, and the number of sub buffs and sub buff size.

The number of sub buffers and the size of each sub buffers is dependent on the application that is being run and the environment and scenario that it is being run in. However, it is dangerous to not have enough sub buffers since it is likely that you will spill into other memory that is not designated for the buffer and overwrite important data.

Example code on how to implement and use relay and debugfs on a linux kernel system can be found online.

Mmap()

It is important to provide a lower overhead mechanism for projecting kernel data to user space. There is a read() interface that is relatively easy to use but not very efficient. Mmap() is a better approach. After a channel and channel buffers per cpu are created using relay_open(), there must be method for the buffer stream to be read from the user space. This can be done by using mmap() and creating a system in which each channel buffer is associated with a file created by the host filesystem, debugfs, and then mapped into user space. In other words, mmap is used to map contents of a file descriptor into a memory segment that is located in the virtual address space of a calling process. Generally, the starting address for the map is specified in addr and length specifies the length of the mapping.

When creating a mmap, there a 3 different flags that can differentiate the type of mapping being done: MAP_FIXED, MAP_SHARED, and MAP_PRIVATE. MAP_FIXED forces the address to be fixed to the one specified thus different addresses cannot be selected. MAP_SHARED enables

mapping with all other processes that map this object. MAP_PRIVATE creates a private copy-on write mapping. Basically, the original file will not be affected when stores to the region are made. In our user space application that we created, we used MAP_PRIVATE because it was important it does not carry writes back to the buffer file being accessed.

Other parameters of mmap() include the prot integer that specifies whether the pages may be executed, read, written, or not accessed. In the user space application, the PROT_READ was passed into the mmap since the buffer can only be read by the user space and not written or executed. The file for the mmap to map is also passed into mmap with a specified offset at which to start the mapping of the file.

The example code of the kleak user space is almost identical to the objective this project outcome.

Sources

http://relayfs.sourceforge.net/index.html
https://www.kernel.org/doc/Documentation/filesystems/relay.txt
CS281 S15 lecture notes, VUSE