# Checkpoint - Restore Process

# CRIU

- Checkpoint Restore in Userspace

- Command line utility which stops a process and dumps its memory which can be restored on the same machine or any other machine.

# How CRIU Works

- The checkpoint process takes most of the information it needs from the /proc fs
  - Files descriptors information (via **/proc/$pid/fd** and **/proc/$pid/fdinfo**).
  - Pipes parameters.
  - Memory maps (via **/proc/$pid/maps** and **/proc/$pid/map_files/**).

# How CRIU Works - Checkpointing

- Dumper walks through the process tree by collecting threads from /proc/$pid/task directory and children from /proc/$pid/task/children to collect children recursively
- While walking the tree, it attaches itself to each process using ptrace and then stops it using PTRACE_SIEZE

# How CRIU Works - Checkpointing

For each process-

- VMAs areas are parsed from **/proc/$pid/smaps** and mapped files are read from **/proc/$pid/map_files** links.
- File descriptor numbers are read via **/proc/$pid/fd**
- Core parameters of a task (such as registers and friends) are being dumped via ptrace interface and parsing **/proc/$pid/stat** entry.

# How CRIU Works - Checkpointing

- Then CRIU injects a parasite code into a task via ptrace interface.
- First few bytes are injected for *mmap* syscall at CS:IP the task has at moment of seizing
- Then ptrace allow CRIU to run the injected syscall and enough memory is allocated for a parasite code chunk needed for dumping
- Then, the parasite code for dumping is copied into dumpee address space CS:IP is set to point to this parasite code.

# How CRIU Works - Checkpointing

- After dumping, ptrace is again used to cure the process of all the parasite code
- CRIU then detaches from the process.
- Process resumes/stops.

# How CRIU Works - Restoring

- On the top-level it consists of 4 steps

First step -

- CRIU reads in image files and finds out which processes share which resources.

- In later steps, these resources are restored by one process and inherited/obtained by all others.

# How CRIU Works - Restoring

## Second Step -

- CRIU calls fork() to recreate the process tree.

## Third Step -

CRIU restores all resources but

- memory mappings exact location
- timers
- credentials
- threads

On this stage CRIU opens files, prepares namespaces, maps (and fills with data) private memory areas(these maps are not in correct address), creates sockets, calls chdir() and chroot()

# How CRIU Works - Restoring

Fourth Step -

- Since criu morphs into the target process, it will have to unmap all its memory and put back the target one.
- CRIU's code will get over-mmaped by the process's mapping as it is restored and CRIU will seg-fault once the old code is unmapped.
- The CRIU restorer code is mapped into the appropriate hole in the target process's memory mappings so that it doesn't get over-mmaped by the process's mapping.
- Now the target process's memory, timers, credentials and threads are restored

# Dumping Files

- Stores the fd table of the process in a file which contains fd-id pair. Here id is the reference to the file in other images.
- Other images are -
  - reg-files.img for regular files, that are created by open() call
  - unixsk.img for unix sockets
  - pipes.img for pipes
  - inetsk for IP sockets (both TCP and UDP)
  - signalfd.img for signal fd

  In each of these images, info about File and Inode of respective file is preserved.

# C/R TCP Connection - TCP Repair

- A new option TCP_REPAIR has been added to the kernel
- Using this option puts a socket into a special mode,
- In this mode, any action performed on the socket does not result in anything defined by an appropriate protocol actions, but rather directly puts the socket into the final state.
- ex - Closing the socket will not send the FIN packet but put the socket in closed state.

# C/R TCP Connection - Checkpoint

- Put the socket into TCP_REPAIR mode
- Gather all the information about the TCP socket like address and port of the remote end, the sizes and contents of the send and receive queues, TCP sequence numbers, MSS etc.
- Close the socket. (No FIN packet is sent, so the conection is still active on the other side)
- During this process, no remote packet should enter the TCP recv queue otherwise the kernel will send a RST packet and reset the connection. This can be ensured by a netfilter rule.

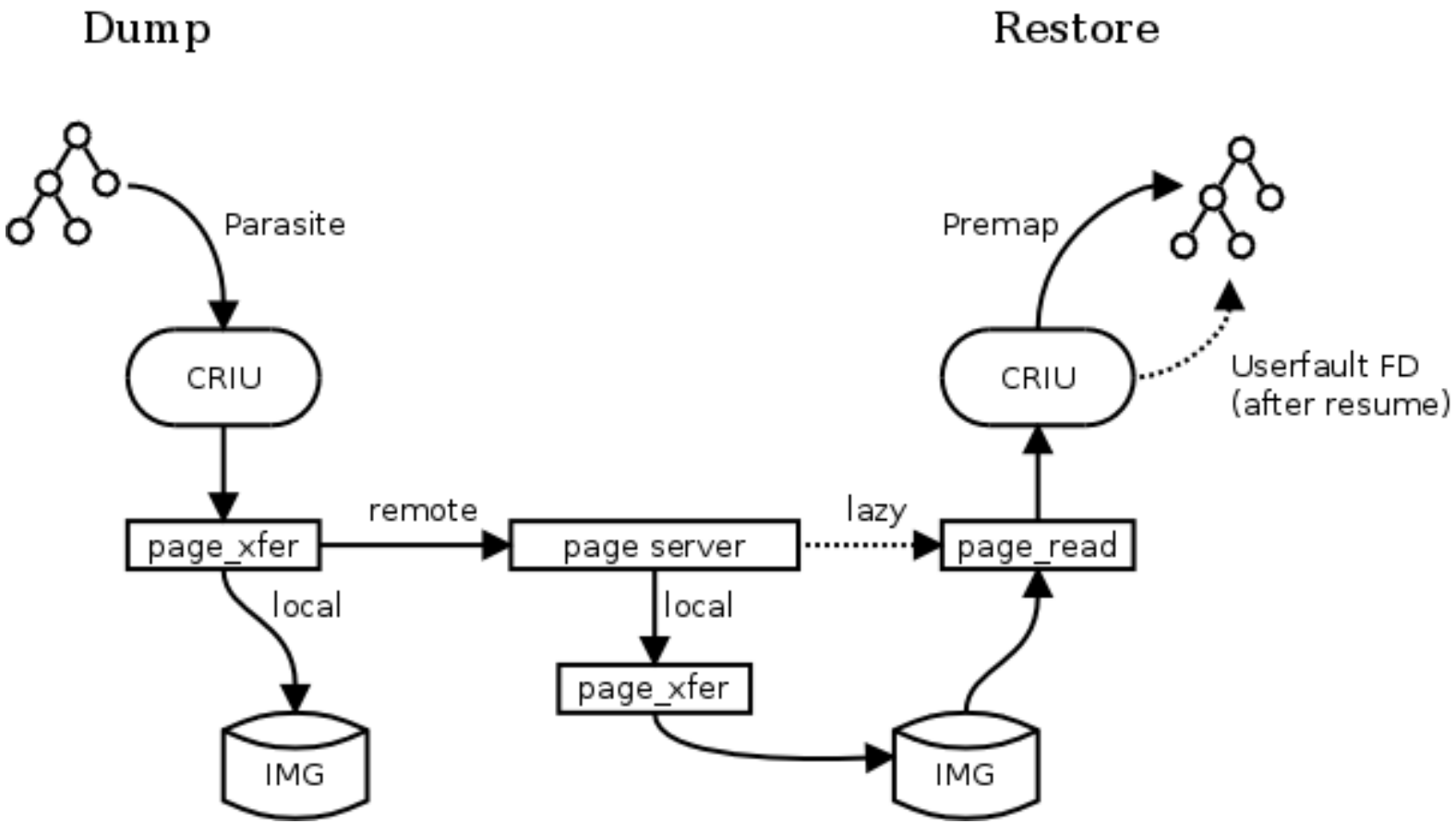
# C/R TCP Connection - Restoring

- Create a new socket in put it in TCP_REPAIR mode
- The socket is bound to the proper port number; a number of the usual checks for port numbers are suspended when the socket is in repair mode.
- Restore the queues, Sequence numbers and other parameters.
- Call connect() on the socket (this again puts it directly into established state without any checks or packets send)
- Take the socket out of the repair mode

# Live Migration

- For migration across machines, the files accessed by processes should be present on both the machines. This can be achieved through NFS or by using rsync to copy files from one box to another.
- Dump the process on one machine, copy the dumped files to the other machine and restore it on the other machine.
- A separated project P.Haul on top of CRIU to address this.

# Live Migration - Disk-less




- Lazy migration not yet supported. Page server copies the pages from one RAM to another

# Iterative Migration

- Take regular dumps of the process without stopping the process after dump and copy it to the other machine.
- Each successive dump will contain memory changed after previous dump.
- Similar to pre-copy approach in VM migration.

# Migrating Docker Container

- CRIU can checkpoint and restore a process tree running inside a Docker container.
- But the restored tree will not become a child of Docker and, from Docker's point of view, the container's state will remain "Exited".
- So, native support from Docker is needed for Docker daemon to maintain parent-child relationship and to correctly keep track of container states
- Work in progress

# DEMO

- CRIU requires some options enabled while compiling the kernel which are not enabled in most kernels by default.
- So, kernel needs to recompiled with to use options to use CRIU.

# BLCR

- Stands for Berkeley Lab Checkpoint/Restart
- It is an Open Source, system-level checkpointer designed with High Performance Computing (HPC) applications in mind, in particular MPI.
- Supports checkpointing a single process (including multithreaded)
- Compatible with LAM MPI

- Doesn’t support migration of TCP sockets.

- But application using LAM MPI can be migrated as it’s communication interface has been modified to interact with BLCR and handle migrated processes.