

Understanding Kerrighed Series

Checkpointing in Kerrighed

David Margery
PARIS Project
March 2004

Outline and Scope

- ◆ Scope:
 - ◆ Checkpoint and restart in Kerrighed today, or in a near future
 - ◆ Code that works in the main branch
 - ◆ Code that has work in a specific branch
- ◆ Outline
 - ◆ Our view of the problem
 - ◆ Mechanisms used in Kerrighed : *ghosts*
 - ◆ Policies
 - ◆ Where are process images stored ?
 - ◆ When do we checkpoint ?

Checkpoint and restart

- ◆ Extracting a process from kernel data-structures
- ◆ Saving a process
 - ◆ To memory
 - ◆ To disk
- ◆ Loading a process
 - ◆ Finding the process image
 - ◆ Creating the kernel data-structures
- ◆ Inserting a process in kernel data_structures
 - ◆ Creating a host process
- ◆ Managing saved processes

Kerrighed Processes and Ghosts

- ◆ The host operating system manages processes
 - ◆ Standard Linux processes
 - ◆ In 2.4.24 : one process per thread
 - ◆ Global pid (32 bits for binary compatibility)
- ◆ Kerrighed manages processes
 - ◆ All threads share
 - ◆ The same pid (the original Linux pid of the initial thread)
 - ◆ Other Kerrighed data structures
 - ◆ Those processes are implemented as Linux processes
- ◆ Ghosts enable virtualisation of Kerrighed processes over Linux processes

Ghosts

- ◆ Definition
 - ◆ *A ghost is a representation of a Kerrighed thread that is no longer dependant*
 - ◆ *On the Linux process that has hosted it*
 - ◆ *On the node that has hosted it*
 - ◆ *On the incarnation of Kerrighed where it was created*
- ◆ Uses
 - ◆ Thread creation (and more generally thread duplication)
 - ◆ Thread migration
 - ◆ Checkpointing

The Ghost Mechanism

- ◆ Adresses 3 problems
 - ◆ Common functions for data-structure parsing: share, for better code management, between:
 - ◆ Parse data-structures to create ghost
 - ◆ Parse ghost to create data-structures
 - ◆ Extraction and insertion of kernel objects
 - ◆ Most kernel object live in lists
 - ◆ Are allocated in specific allocation caches
 - ◆ Extraction and insertion of Linux processes
 - ◆ Need to ghost a process which is in a state that does not depend on the host kernel
 - ◆ Access to registers
 - ◆ No thread in kernel that use data-structures belonging to the process that cannot be extracted with the process (replay syscalls)

Extraction of Processes

- ◆ Now :on ret_from_schedule
 - ◆ Each task struct has a need_argorn flag (same mechanism as sigpending)
 - ◆ When about to access the processor
 - ◆ No ongoing syscall
 - ◆ Easy Access to registers
- ◆ Future
 - ◆ All runnable processes
 - ◆ No ongoing syscall
 - ◆ Need to find the registers (possible)
 - ◆ Processes blocked on Kerrighed managed syscalls
 - ◆ Replayable syscalls

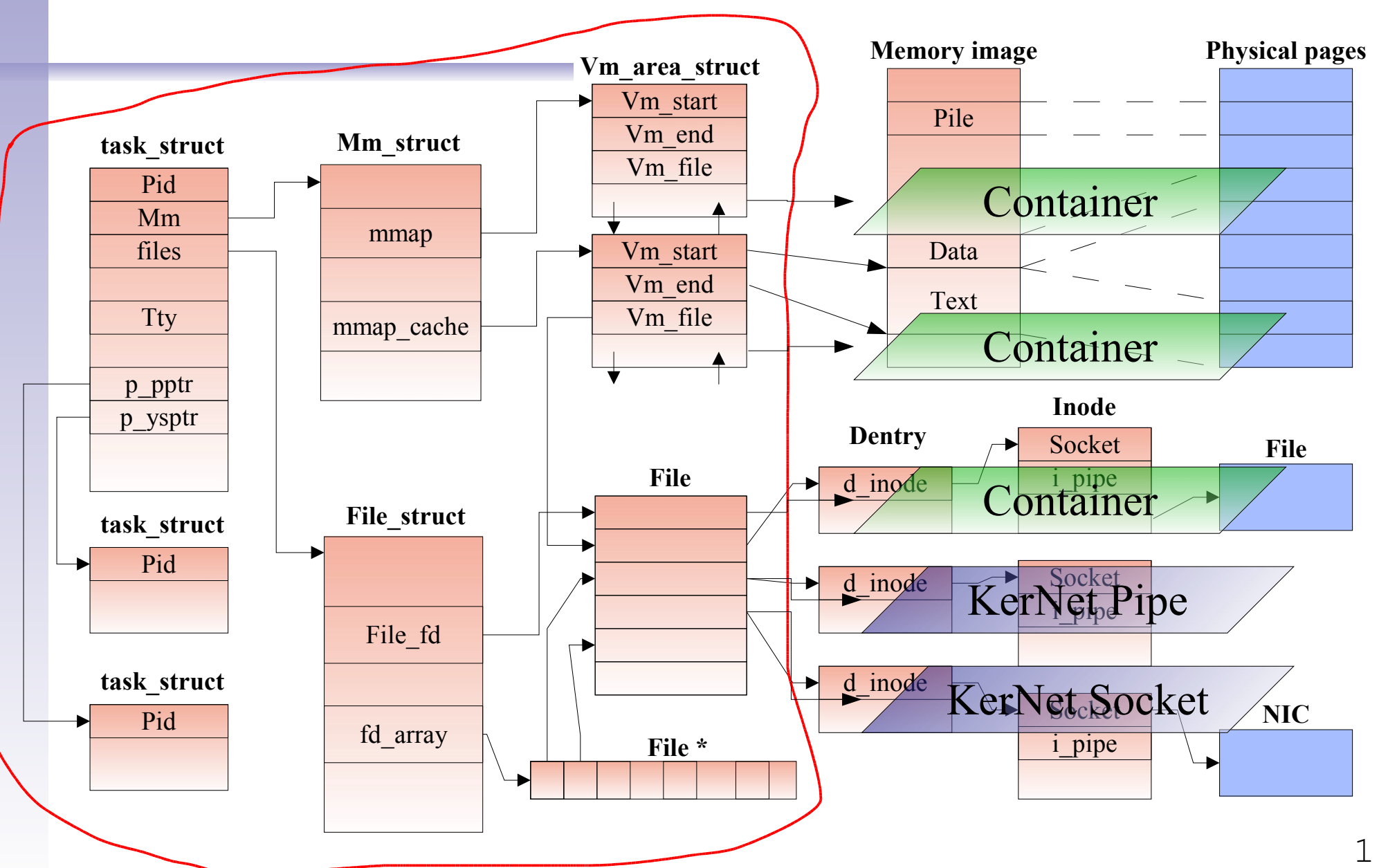
Insertion of Processes

- ◆ Using a modified `do_fork`, that can fork a specially built process image
 - ◆ When possible, we use the `CLONE_*` flags
- ◆ Restarting a process becomes:
 - ◆ Locate a ghost of the process
 - ◆ Building a “fake” process from it
 - ◆ Forking that process
 - ◆ Destroy the “fake” process

Common ghosting functions

- ◆ `make_ghost(ghost_t * ghost, void * data, size_t size)`
- ◆ The ghost parameter abstracts from :
 - ◆ The ongoing operation (Duplication, migration, checkpoint)
 - ◆ The storage media (distant node, distant disk)
- ◆ Write one function by kernel data-type that needs to be exported using `make_ghost`
 - ◆ Now : separate import and export for allocation reasons
 - ◆ Future : it is possible to change the interface of `make_ghost` to have one function (to manage allocation)
- ◆ Ghosting of Kerrighed data types often needs additional info found in the ghost parameter

The Ghost Frontier



Checkpointing Kerrighed Structures

- ◆ Containers (Shared Memory management)
 - ◆ Checkpoint to disk operational in a code branch (needs merge in the main branch), and is incremental
 - ◆ An Icare-like implementation still to be implemented
- ◆ Kernet Streams
 - ◆ Rather trivial when both ends are checkpointed
- ◆ Cluster-wide synchronisation primitives (Elrond)
 - ◆ Trivial when checkpointing all processes sharing the primitives

Parallel Ghost

- ◆ Now : a ghost is the image of a thread
 - ◆ Equivalent to a process if the program is not parallel
- ◆ Future : a (parallel) ghost is the image of a process
 - ◆ Multi-threaded process (Pthread)
 - ◆ Easy to identify the involved processes
 - ◆ Synchronous Ghosting of each thread is trivial
 - ◆ Ghosting of shared memory has worked
 - ◆ Multi-process program (MPI)
 - ◆ Using krg-rsh for remote execution, all processes belong to the same group : we can identify the involved processes
 - ◆ Ghosting of Kernet streams is trivial

Related processes and ghosts ?

- ◆ In the event of failure, what happens to related processes (wait4 returns ?)
- ◆ Reparenting of restarted processes ?
- ◆ Posix extensions needed ?

Policies

- ◆ Policies for checkpointing are not mature in Kerrighed
 - ◆ Checkpoint on request (by any process)
 - ◆ Checkpoint on local disk or in local memory
- ◆ Implementing better policies is not difficult
 - ◆ Now: policy is hard coded
 - ◆ Goal: replicate the global scheduler model
 - ◆ Write the infrastructure that will enable easy experimentation of policies
 - ◆ Path: identify and implement the infrastructure needed

Infrastructure for policy writing

- ◆ Failure detection (has worked)
- ◆ Atomic multicast
- ◆ Dependency tracking
- ◆ Storage management
 - ◆ Stable storage
 - ◆ Distributed storage
 - ◆ Fault tolerant storage
- ◆ Automatic checkpointing
 - ◆ Description of the policy used

Conclusion

- ◆ In Kerrighed, checkpoint and restart of sequential processes is working
- ◆ In Kerrighed, checkpoint and restart of parallel processes is near
- ◆ Better simple policies could be implemented
- ◆ Infrastructure for non-trivial policies is not present.