CS730: Topics in OS **Semester 2021-22-II**

Project Report

Paras Mittal parasm@iitk.ac.in
Dev Chauhan devgiri@iitk.ac.in
Somu Prajapati somupra@iitk.ac.in

1 Introduction

Our aim was to develop a kernel module for checkpointing a single process into persistent storage so that it can be restored later. We chose linux kernel for our implementation due to its thorough documentation and well maintained source code.

2 Implementation

2.1 Persisting Kernel Data Structures

After going through the POSIX state of a process in the linux kernel, we categorised the data structures into the following categories that are handled differently while copying at the time of checkpoint and restore:

1. **Recursive DSes**
   - List heads
   - Red-Black Trees
2. **Posix locks**
   - Mutex
   - Semaphore
   - Spinlock
3. **Arch-dependent DSes**
   - mm_context_t
4. **Compound Data Structures**
   - VMA structs
   - Anonymous VMAs
   - mm_struct (...)

Recursive data structures like red-black trees (struct rb_node) which do not contain any specific information or pointers to other structs, do not need to be persisted and restored and can simply be created at the time of restoring a process (e.g. rbnodes can be created anew, associated with vm structs and inserted into the rbtree root present in mm_structs.). While complex recursive structures like vm_area_struct and anon_vma need to be persisted completely while following pointers(deep copy).
Mutex and Semaphores contain wait queues that need to be copied recursively. For a single threaded process locks do not need to be copied and can be initialised at restore time.
All other data structures are composed of one or more of these DSes.

2.2 Restoring the checkpointed process

2.2.1 How to create new process?

Creating a new process happens only once in the kernel at the time of creating the `init` process, all other processes are created from existing processes using `clone`.

As creating process from scratch is a very rare event in kernel itself (e.g. happens only once, which is also a not trivial condition), what values to initialize for new process that will be created, assigning pid, updating global structures (e.g. global linked list of `mm_structs` are maintained, various slab caches used for different structures, etc), etc, are difficult questions to answer using linux kernel documentation and source code.

Creating a new process, just from the information about it rather than forking it is unlike anything that `clone` has to offer. Closest relation that we can have is if we have our checkpointed information available as a process. In that case, restoring the process is like cloning a process who has its state in persistent memory. However, we can not call something like `clone` from a persisted process. As we want our new process to be well established and clone of persisted process, we use the already running process `crp restore`, and copy process information as done in `do_fork`. Note that here we need to properly provide a copy of each and every data structure (e.g. rb trees, linked lists, pointers to other structs, etc.)

2.2.2 Restoring the Kernel State

As mentioned in section 2.2.1, for updating kernel state of new process, we load the checkpointed structures (e.g. list of `vm_area_struct`, `mm_struct` information)

Major hurdle / frequent cause of faults in restoring the kernel state we found was the inherent and strong assumption in the POSIX design of kernel that every structure is available in the memory pointed by a pointer, as explained in section 2.1.

2.2.3 Arch dependent register restoring/checkpointing

Restoring `pt_regs` is not enough for some special registers like `FS`, `GS`, control registers, etc. Such CPU states are stored in `task_struct->thread` which is architecture dependent and requires architecture specific handling for restoring these registers. e.g. `x86_fsbase_write_cpu` has been used to restore %fs register while for other registers it sufficies to write them to `pt_regs`, which will be restored when process is scheduled.

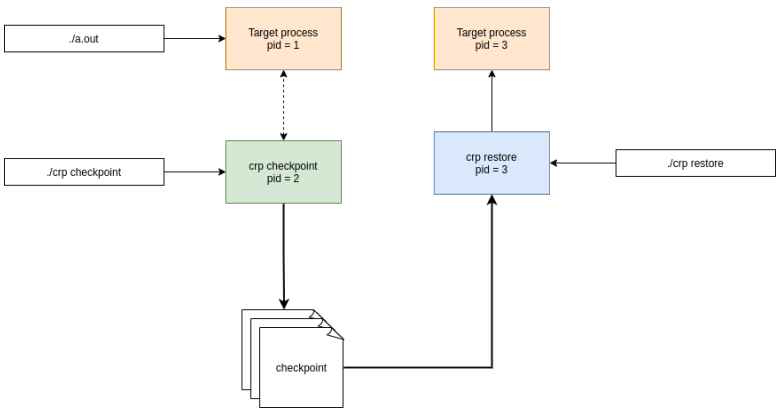

Figure 1: Checkpoint and Restore mechanism

3 Source code

github.com/dev-chauhan/crp.git

4 Contribution

Dev	ideation, CPU state restore, mm restore, user facing library and base driver, kallsyms, reports and presentations, testing other methods for restore
Paras	ideation, VMA restore, rbtree restore, process kernel state inspection(virtual memory), reports and presentations, testing other methods for restore
Somu	Reading about POSIX kernel structures important for checkpoint/restore similar to Aurora