Light-Weight Contexts: An OS Abstraction for Safety and Performance

James Litton, *University of Maryland, College Park and Max Planck Institute for Software Systems (MPI-SWS);* Anjo Vahldiek-Oberwagner, Eslam Elnikety, and Deepak Garg, *Max Planck Institute for Software Systems (MPI-SWS);* Bhattacharjee, *University of Maryland, College Park;* Peter Druschel, *Max Planck Institute f*Bobbyor *Software Systems (MPI-SWS)*

Introduction

- New OS Abstraction – light-weight contexts (*lwC*s)
- Light-Weight Contexts (LWC) is an operating system (OS) abstraction used for managing isolated contexts within a single process.
- Why *lwC*s
  - existing methods for session isolation are often slower than *lwC*s.
  - *lwC*-supported sensitive data compartments have negligible overhead on production servers.

Advantages

- *lwC*s enable a range of new in-process capabilities, such as
  - fast roll-back
  - protection rings (by credential restriction)
  - session isolation
  - Efficiency
  - Ease of use
  - protected compartments (using VM and resource mappings).

How does it help with security?

- It helps us achieve security with the following properties
  - Isolation
  - Sandboxing
  - Resource Control
  - Reduced Attack Surface
  - Improved Code Quality

Related Work

- Some other solutions include
  - Shreds [9] – however Lcw are fully independent of threads, require no compiler support, and rely on page-based hardware protection only. *lwC*s also provide protection rings and snapshots, which shreds do not.
  - Dune [4] – however it has higher overhead cost due to TLB misses and kernel calls.
  - Software fault isolation (SFI) [29] and NaCl [35] – *lwC*s instead allow fine-grained control over memory, file descriptors and other process credentials, and provide snapshots as part of an OS abstraction.
  - A few more including SpaceJMP [12], Corey [6], however they don’t provide OS Snapshots and in-process isolation.

Why LcWs?

- While related work is being continued LcWs provide the following advantages
  - Lcw are fully independent of threads
  - Provide protection rings (Security) and snapshots
  - Low or negligible overhead cost
  - Fine-grained control over memory,
  - In-process isolation

Creating LcWs

- Starts with creating an LcW.
- The lwCreate call creates a new (child) *lwC* in the cur- rent process.
- the child *lwC*'s initial state is an identical copy of the calling (parent) *lwC*'s state, except for its descriptor.
- By default, the new *lwC* gets a private copy of the calling *lwC*'s state at the time of the call
- Shared memory regions in the calling *lwC* are shared with the new *lwC*.
- The implementation does not stop other threads exe- cuting in the parent *lwC* during an lwCreate.

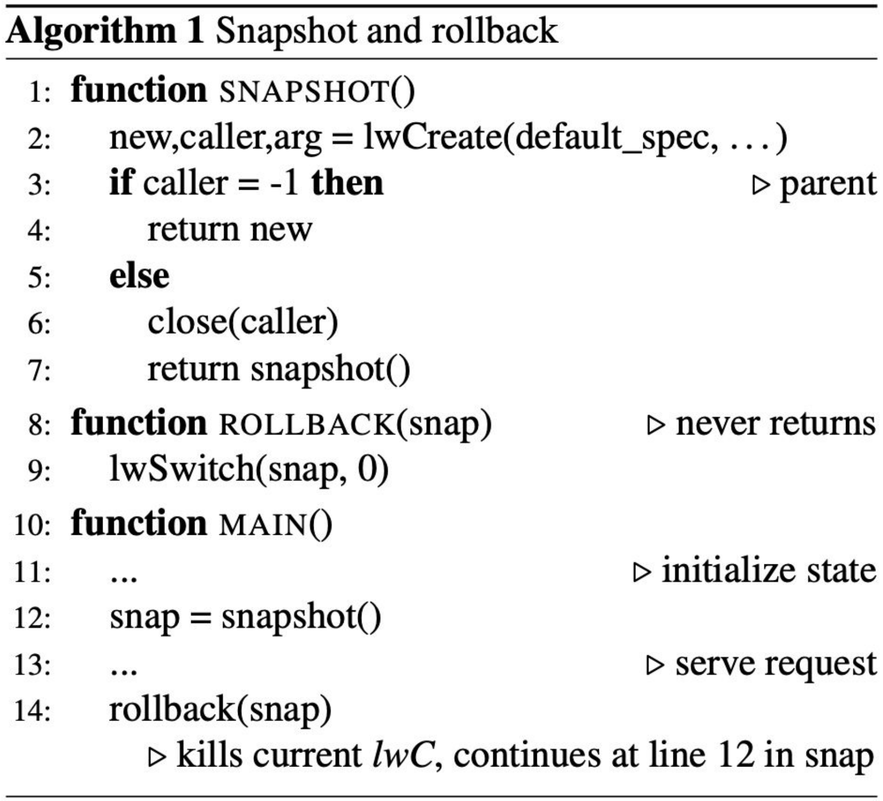
Switching between *lwCs*

- The lwSwitch operation switches the calling thread to the *lwC* with descriptor *target*, passing args as parameters.
- lwSwitch retains the state of the calling thread in the present *lwC*.

How does it achieve Isolation?

- *lwC*s do not have access to the state of each others' memory, file descriptors, and capabilities unless explicitly shared, they can provide strong isola- tion and privilege separation within a process.
- *lwC*s can reliably prevent accidental leakage of private information across user sessions, isolate authentication credentials and other secrets
- An application that wishes to limit information flow across *lwC*s should create *lwC*s without the LWC_SHARESIGNALS option (the default).

Snapshot and rollback



Algorithm 1 Snapshot and rollback

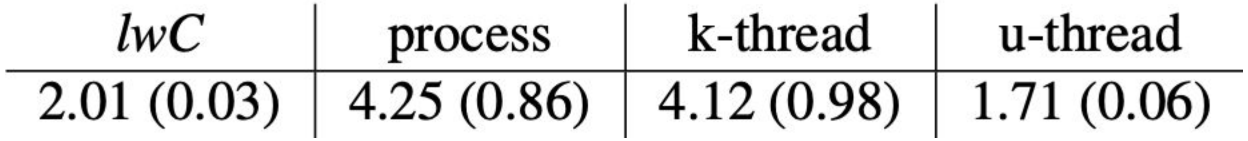
```
 1: function SNAPSHOT()
 2:     new,caller,arg = lwCreate(default_spec, . . . )
 3:     if caller = -1 then                          ▷ parent
 4:         return new
 5:     else
 6:         close(caller)
 7:         return snapshot()
 8: function ROLLBACK(snap)                   ▷ never returns
 9:     lwSwitch(snap, 0)
10: function MAIN()
11:     ...                                ▷ initialize state
12:     snap = snapshot()
13:     ...                                  ▷ serve request
14:     rollback(snap)
          ▷ kills current lwC, continues at line 12 in snap
```

lwC Implementation

- Like a process, each *lwC* has a file table, virtual memory space, and credentials associated with it.
- Memory
  - lwCreate replicates the vmspace associated with the parent lwC in ex- actly the same manner as fork.
- File Table
  - By default, during a call to lwCreate all file descriptors are copied into the *lwC* file table in the same manner as fork except that any associated file de- scriptor overlay rights are copied as well
- Permissions and Overlays
  - An executing lwC interacts with another lwC within a process by either switching to it or by overlaying (some of) that lwC's resources.

Evaluation

- The table compares the time to execute a lwSwitch



lwC	process	k-thread	u-thread
2.01 (0.03)	4.25 (0.86)	4.12 (0.98)	1.71 (0.06)

Conclusion

- *lwCs* provide isolation and privilege separation among program components within a process
- Also provides fast OS-level snapshots and coroutine style control transfer among contexts
- Provides fast roll-back

Reference

- ABADI, M., BUDIU, M., ERLINGSSON, U., AND LIGATTI, J. Control-flow integrity. In *Proceedings of the 12th ACM Confer- ence on Computer and Communications Security (CCS)* (2005), pp. 340–353.
- [2] AVIRAM, A., WENG, S.-C., HU, S., AND FORD, B. Efficient system-enforced deterministic parallelism. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Im- plementation* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 193–206.
- BANGA, G., DRUSCHEL, P., AND MOGUL, J. C. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation* (Berkeley, CA, USA, 1999), OSDI '99, USENIX Association, pp. 45–58.