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

- Coroutines : Coroutine refers to procedures which allow multiple entry and exit points using suspend and resume functionality. Execution in a coroutine begins where it was suspended and values of local variables of procedure is retained using a structure. lwCs are similar to them as they also allow multiple entry and exit points and retain execution state of threads which had executed in them. Also like coroutines, lwC also do not use stack to implement this functionality.
- FD is a handle inside process address space. Unix uses FD to identify all files, devices or we can say resources using FDs and so they are chosen to identify lwC to keep the generality. But system calls like open, read, write on FDs which are refering to lwC, will result in an error.
- Unix implement two core Sandboxes:
    - Process Level: One process cannot access the address space of another process.
    - User Id Level: A Unix process is Owned by a particular userid.
- One example where we have snapshot and rollback useful is for eg, when we use public computer for browsing. The Clean up program could have bugs, so rollback is much more secure and efficient.
- In terms of security, snapshot and rollback can provide various services like:
    - No data leakage
    - Integrity (snapshot is taken at a point where program is in consistent state, so whenever some data lost or data modification happens we can just rollback and rerun the instructions)
    - It does not help in ensuring availability(DOS).
- There are two kinds of servers, event driven (like nginx) or multithreaded (like apache).
- Multi-threaded Server :
    - Every time a new connection comes, it is handled by a separate thread which could be created in advance. OS scheduler performs thread ordering.

    - Each connection can also be handled by a separate process which can be pre forked. But in this case overhead is of process scheduling and IPC which is more than the case when threads were used. But this method provides more session isolation.
- Single Threaded - Event driven Server :
    - It minimizers per connection state and maintains all states in a single thread.
    - Also since there is only one thread, OS scheduler does not come into picture.
    - These give high performance but are difficult to write.
- ==========
- Can we create root lwC again? No, we can not create a root lwC but we can create its exact copy with same resources, same VM mappings etc. Root is just a name given to one lwC which is present in a process by default when it is created and is identified by a well known file descriptor.
- Counterpart of lwC in current operating system is that if a process has permission to read, open and write a sensitive file (like “/etc/passwd” in linux) then it can restrict its child process to open or write that file. It’s just that in this new abstract there is no child process but access is restricted for threads which can not switch to privileged lwC.
- Are lwCs created before calling lwRestrict also restricted to by this call? No, since permissions are associated with file descriptor tables and lwCs created before have already copied fd table and hence permissions, so they can’t be restricted.
- How will parent lwC have reference to a new buffer created by child lwC using COW to deallocate in future? Parent lwC will not have reference to any private buffer of child lwC. It is not permitted to deallocate it.
- What if some memory is mapped in child at overlay address? That memory will be unmapped.
- On switching to a new lwC execution begins on line lwCreate, where will execution resume on switching to it after 1st time? It will begin at line where thread running in child lwC left its execution i.e. on line where it called lwSwitch.
- What impact does this abstraction has on security i.e how it affects TCB (Trusted Computing Base)?
    - Removal from TCB : Before implementing lwCs, main function was in TCB. But with lwC, except for the first few lines (where new lwC are created and we assign them privileges), main function is excluded from TCB.

    - Addition to TCB : All the framework code that is added to implement lwC APIs is added to TCB. But adding this code to TCB is better than adding application code to TCB as this code is written by experts and tested many times.
- To provide access enforcement, there are two approaches:
    - One is to test every call to API which could cause unwanted result before the call is made i.e. inline checks.
    - Other is to check when trouble happens like exceptions, page faults, trap etc.
- lwSyscall adds a new facility to pose as a different process by making system call on behalf of thread running in different lwC.
- Where will child return in reference monitor as it did not call lwSwitch? It did not call lwSwitch explicitly but that call was made by sandboxing mechanism of Operating System on its behalf when a privileged system call is made. This thread will resume from there in sandboxing mechanism of OS.
- What if child explicitly calls lwSwitch to reference monitor lwC? We can either restrict this by not passing lwC fd of reference monitor to child lwCs or reference monitor can just return some error and switch back to caller lwC.
- Pattern of reference monitor is similar to that of kernel-application process relationship where kernel performs various checks on parameters of application process when it tries to perform privileged operations using traps.
- We can stop threads in some particular lwCs to perform exit system call. But even that will not give immunity against DoS attack as thread can simply run in infinite loop and never call lwSwitch.
- Even SIGALRM can not be used to prevent DOS attacks as they are non attributable signals in this scheme which are delivered to root lwC when a thread switches to it. So if malicious thread never switches to root lwC is will not run handler for SIGALRM and we can not stop this thread.
- Non Attributable signals are those which can not be associated with a particular instruction like SIGALRM, SIGKILL, unlike SIGFPE which can be associated to a instruction and hence is attributable. No process can itself specify that which signals are attributable and which are not.
- We can also implement reference monitors using strace but in in-process reference monitors implemented using lwC, scheduling overheads go away. Also IPC is faster in

lwC because call to create mapping of shared memory among processes are costlier operation. Thus, lwC are simpler and faster.

- Number of PCID on Free BSD : 12bits(4096)
- Hyper-threading : It refers to running more than one (say two) threads on same core. Such a core has two copies of registers and some shared functionalities so that two thread can run parallelly on same core. This is disabled during lwC experiments because it is hard to model and sometimes can reduce performance.
- Speed-step : It refers to increasing frequency of CPU for sometime to enhance its speed at cost of more heat generation and power usage.
- lwC unlike processes don't protect against DOS attack but provides rest of the things including privilege separation, sensitive data isolation and execution state.