## 6.CGT Concurrency – Directed Termination [CGT]

### 6.CGT.0 Terminology

Abort: The completion and shut down of a thread, where the thread is not permitted any execution after the command to abort has been received by the thread, or by the runtime services that control the thread. In particular, the thread will not be able to release any locks that it has explicitly acquired, and may not release any OS provided locks or data structures.

Abort deferred region: A section of code where a thread is permitted to ignore abort directives, usually because it is holding a system resource or the risk of corruption to the application is significant while the thread is manipulating certain resources.

Termination: The completion and orderly shutdown of a thread, where the thread is permitted to make data objects consistent, return any heap-acquired storage, notify any dependent threads that it is terminating, and finalise system resources dedicated to the thread. There are a number of steps in the termination of a thread as listed below, but depending upon the multithreading model, some of these steps may be combined, may be explicitly programmed, or may be missing.

* The termination of programmed execution of the thread, including termination of any synchronous communication;
* The finalisation of the local objects of the thread;
* Waiting for any threads that may depend on the thread to terminate;
* Finalisation of any state associated with dependent threads;
* Notification of outer scopes that finalisation is complete, including possible notification of the activating task; and
* Removal and cleanup of thread control blocks and any state accessible by the thread by possibly threads in outer scopes.

Terminated Thread: The thread that is being halted from any further execution.Termination Directing Thread: The thread (including the OS) that requests the abort or termination of one or more threads.

### 6.CGT.1 Description of Application Vulnerability

This discussion is associated with the effects of unmet termination directives. For a discussion of premature termination, see CGT Concurrency – Premature Termination.

When a thread is working cooperatively with other threads and is directed to terminate, there are a number of error situations that may occur that can lead to compromise of the system. The termination directing thread may request that one or more other threads abort or terminate, but the terminated thread(s) may not be in a state such that the termination can occur, may ignore the direction, or may take longer to abort or terminate then the application can tolerate.Late termination may cause a failure to meet deadlines, implying incomplete calculation, leading the application to deliver no results or incorrect results. Non-termination may cause deadlock, livelock, failure to release resources, and corrupted data abstractions. All of these may lead to failure of the application.

### 6.CGT.2 Cross References

Hoare C.A.R., "Communicating Sequential Processes", Prentice Hall, 1985

Holzmann G., "The SPIN Model Checker: Principles and Reference Manual"., Addison Wesley Professional. 2003

Larsen, Peterson, Wang, "Model Checking for Real-Time Systems"., Proceedings of the 10th International Conference on Fundamentals of Computation Theory, 1995

The Ravenscar Tasking Profile, specified in ISO/IEC 8652:1995 Ada with TC 1:2001 and AM 1:2007

CWE 364 Signal Handler Race Condition

### 6.CGT.3 Mechanism of Failure

The abort of a thread may not happen if a thread is in an abort-deferred region and does not leave that region (for whatever reason) after the abort directive is given. Similarly, if abort is implemented as an event sent to a thread and it is permitted to ignore such events, then the abort will not be obeyed.

The termination of a thread may not happen if the thread ignores the directive to terminate, or if the termination process raises exceptions that result in the thread not terminating.

The termination directing thread will be expecting the thread(s) to terminate (or abort) and may proceed on the expectation that the abort/termination occurred. If some of the threads that were directed to terminate do not terminate, then the overall application will not move on to the next phase of the application, or will produce erroneous results.Livelock, the freeze-up of a system because some threads start cycling indefinitely, or deadlock, the freeze-up of the system because some threads are blocked waiting for other threads (such as waiting for them to terminate) are distinct possibilities resulting from this vulnerability. Arbitrary execution of random code is also a distinct possibility from some kinds of termination errors, but arbitrary execution of known code is not likely since it is hard to determine where nonterminating threads will be in their execution when the terminating thread notification is delivered.

## 6.CGT.4 Applicable Language Characteristics

Languages that permit concurrency within the language, or support libraries and operating systems (such as POSIX-compliant OSs or Windows) that provide hooks for concurrency control.

## 6.CGT.5 Avoiding the Vulnerability or Mitigating its Effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

* Use a language that provides a complete concurrency mechanism.
* Use mechanisms of the language or system to determine that aborted threads or threads directed to terminate are indeed terminated. Such mechanisms may be direct communication, runtime-level checks, explicit dependency relationships, or progress counters in shared communication code to verify progressProgram fall-back handlers to report or recover from failure to terminate situations.

### 6.CGT.6 Implications for Standardisation

* In future standardisation activities, the following items should be considered: Provide a mechanism (either a language mechanism or a service call) to signal another thread (or an entity that can be queried by other threads) when a thread terminates.