## 6.CGS Concurrency – Premature Termination [CGS]

### 6.CGS.0 Terminology

Thread Abort: A request to stop and shut down a thread immediately.

Note: The request is asynchronous if from another thread, or synchronous if from the thread itself. The effect of the abort request (e.g. whether it is treated as an exception) and its immediacy (i.e., how long the thread may continue to execute before it is shut down) depend on language-specific rules. Immediate shutdown minimizes latency but may leave shared data structures in a corrupted state.

Termination Directing Thread: The thread (including the OS) that requests the abort of one or more threads.

Thread Termination: The completion and orderly shutdown of a thread, where the thread is permitted to make data objects consistent, return any heap-acquired storage, and notify any dependent threads that it is terminating.

Note: 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 finalization of the local objects of the thread;
* Waiting for any threads that may depend on the thread to terminate;
* finalization of any state associated with dependent threads;
* Notification of outer scopes that finalization is complete, including possible notification of the activating task;
* 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.

Master of a Thread: Any thread which must wait for the terminated thread before it can take further execution steps (including termination of itself).

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

When a thread is working cooperatively with other threads and terminates prematurely for whatever reason but unknown to other threads, then the portion of the interaction protocol between the terminating thread and other threads is damaged. This may result in:

* indefinite blocking of the other threads as they wait for the terminated thread if the interaction protocol was synchronous;
* other threads receiving wrong or incomplete results if the interaction was asynchronous; or
* deadlock if all other threads were depending upon the terminated thread for some aspect of their computation before continuing.

### 6.CGS.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.CGS.3 Mechanism of Failure

If a thread terminates prematurely, threads that depend upon services from the terminated thread (in the sense of waiting exclusively for a specific action before continuing) may wait forever since held locks may be left in a locked state resulting in waiting threads never being released or messages or events expected from the terminated thread will never be received.

If a thread depends on the terminating thread and receives notification of termination, but the dependent thread ignores the termination notification, then a protocol failure will occur in the dependent thread. For asynchronous termination events, an unexpected event may cause immediate transfer of control from the execution place of dependent thread to another (possible unknown), resulting in corrupted objects or resources; or may cause termination in the master thread, and an expected propagation of failures.

These conditions can result in

* premature shutdown of the system;
* corruption or arbitrary execution of code;
* livelock;
* deadlock;

depending upon how other threads handle the termination errors.

If the thread termination is the result of an abort and the abort is immediate, there is nothing that can be done within the aborted thread to prepare data for return to master tasks, except possibly the management thread (or operating system) notifies other threads that the event occurred. If the aborted thread was holding resources or performing active updates when aborted, then any direct access by other threads to such locks, resources or memory may result in corruption of those threads or of the complete system, up to and including arbitrary code execution.

## 6.CGS.4 Applicable Language Characteristics

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

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

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

* Use concurrency mechanisms that are known to be robust.
* At appropriate times use mechanisms of the language or system to determine that necessary threads are still operating. Such mechanisms may be direct communication, runtime-level checks, explicit dependency relationships, or progress counters in shared communication code to verify progress.
* Handle events and exceptions from termination.
* Provide manager threads to monitor progress and to collect and recover from improper terminations or abortions of threads.

### 6.CGS.6 Implications for Standardization

In future standardization activities, the following items should be considered:

* Provide a mechanism to preclude the abort of a thread from another thread during critical pieces of code. Some languages (e.g. Ada or Real-Time Java) provide a notion of an abort-deferred region.
* Provide a mechanism to signal another thread (or an entity that can be queried by other threads) when a thread terminates.
* Provide a mechanism that, within critical pieces of code, defers the delivery of asynchronous exceptions or asynchronous transfers of control.