6.CGA Concurrency - Activation

6.GA.0 Terminology

Activation : The creation and setup of a thread up to the point where it begins execution. Threads may depend upon one or more other threads to define its existence for objects to be accessed and to determine the duration time of execution.

Activated thread: The thread that is created and begins execution as a result of the activation.

Activating thread: The thread that exists first and makes the library calls or contains the language syntax that causes the Activated Thread to be activated. The Activating Thread may or may not wait for the Activated Thread to finish activation and may or may not check for errors if the activation fails. The Activating Thread may or may not be permitted to terminate until after the Activated Thread terminates.

Static Activation: The creation and initiation of a thread by program initiation, an oaperating system or runtime kernel, or by another thread as part of a declarative part of the thread before it begins execution. In static activation, a static analysis can determine exactly how many threads will be created and how much resource, in terms of memory, processors, cpu cycles, priority ranges and inter-thread communication structures, will be needed by the executing program before the program begins.

Dynamic Thread Activation: The creation and initiation of a thread by the another thread (including the main program) as an executable, repeatable command, statement or subprogram call.

Thread: A piece of code that can execute independently in time and memory space from other parts of the code. A Thread may be a separate program, an asynchronous artefact such as an interrupt, or a creation from the operating system or the language runtime.

6.CGA.1 Description of Application Vulnerability

A thread is activated by a static activation as part of its declarative region, but the lack of some resource or a programming error prevents the activation from completing. The activating thread may not have sufficient visibility or awareness into the execution of the activated thread to determine if it has indeed been activated. The unrecognised activation failure can cause a protocol failure in the activating thread or in other threads that rely upon some action by the unactivated thread. This may cause the other thread(s) to wait forever for some event from the unactivated thread, or may cause an unhandled event or exception in the other threads.

6.CGA.2 Cross References

Hoare A., "Communicating Sequential Processes", Prentice Hall, 1985
Holzmann G., "The SPIN Model Checker: Principles and Reference Manual"., Addison Wesley Professional. 2003

UPPAAL, available from www.uppaal.com,

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

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

6.CGA.3 Mechanism of Failure

All threads except the main thread must be activated by program steps of another thread. The activation of each thread requires that dedicated resources be created for that thread, such as a thread stack, thread attributes, and communication ports. If insufficient resources remain when the activation attempt is made, the activation will fail. Similarly, if there is a program error in the activated thread or if the activated thread detects an error that causes it to terminate before beginning its main work, then it may appear to have failed during activation.

If the activating thread is tightly coupled with the activating thread throughout the activation step, then then the activating

thread will likely be notified of the failure and can be programmed to take alternate action. If such alternate action is not programmed, then the program will execute erroneously. If the activated thread is loosely coupled with the activating thread, and the activating thread receives no notification of a failure to activate, then the activating thread may wait forever for the unactivated task to do its work, or may make wrong calculations because of incomplete data.

If the rest of the application is unaware that an activation has failed, an incorrect execution of the application algorithm may occur, such as deadlock of threads waiting for the activating thread.

Activation failures usually result in deadlock or livelock of the application, or cause errors and misbehaviours that may cause the user to lose trust in the application. It would be unlikely that an external attacker could take control of a system simply by causing activation failures; however when coupled with other vulnerabilities, activation failures could be used to change calculations or other attacks.

6.CGA.4 Applicable Language Characteristics

Languages that permit concurrency within the language, or that use support libraries and operating systems (such as POSIX or Windows) that provide concurrency control mechanisms.

6.CGA.5 Avoiding the Vulnerability or Mitigating its Effects

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

- Always check return codes on OS, library provided or language thread activation routines.
- Handle errors and exceptions that occur on activation if present in the language.
- Create explicit handshaking mechanisms for loosely coupled threaded systems, to ensure that all activations have occurred before beginning the parallel algorithm.
- Use programming language provided features that couples the activated thread with the activating thread to detect activation errors so that errors can be reported and recovery made.
- Use static activation in preference to dynamic activation so that static analysis can guarantee correct activation of threads.

6.CGA.6 Implications for Standardization

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

- Provide concurrency abstractions that support static analysis, such as one of the models that are known to have safe properties, such as CSP or the Ada Ravenscar tasking profile.