Markups made at meeting #19. Steve should consider generalizing this from threads to concurrent processes.

## 6.CGA Concurrency – Activation [CGA]

### 6.CGA.0 Terminology

Thread 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 Thread Activation: The creation and initiation of a thread by program initiation, an operating 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 concurrent process.

Note: Threads, in the context of this report, refer to a facility for multi-processing that is a language-feature rather than an operating system facility.

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

A vulnerability can occur if an attempt has been made to activate a thread, but a programming error or the lack of some resource 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 been activation has been successful. The unrecognized 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.

Although the term thread is used here and the context portrayed is that of shared memory threads executing as part of a process, everything documented applies equally to other variants of concurrency such as interrupt handlers being enabled by a process, processes being created on the same system using operating system routines, or processes created as a result of distributed messages sent over a network. The mitigation approaches will be similar to those listed in CGA.5 below, but the implications for standardization will be dependent on how much language support is provided for the programming of the concurrent system.

### 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 Systems"., 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

CWE 364 Signal Handler Race Condition

### 6.CGA.3 Mechanism of Failure

The context of the problem is that all threads except the main thread are 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. When the activation is “static”, resources have been preallocated, so activation failure because of a lack of resources will not occur. However errors may occur for reasons other than resource allocation and the results of an activation failure will be similar.

If the activating thread waits for each activated thread, then the activating thread will likely be notified of activation failures (if the particular construct or capability supports activation failure notification) and can be programmed to take alternate action. If notification occurs but alternate action is not programmed, then the program will execute erroneously. If the activating thread is loosely coupled with the activated threads, and the activating thread does not receive notification of a failure to activate, then it may wait indefinitely for the unactivated task to do its work, or may make wrong calculations because of incomplete data.

The single activation is a special case of activations of collections of threads simultaneously. This paradigm (activation of collections of threads) can be used in languages that parallelise calculations and create anonymous threads to execute each slice of data. In such situations the activating thread is unlikely to individually monitor each activated thread, so a failure of some to activate without explicit notification to the activating thread can result in erroneous calculations.

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 activated thread, or possibly causing errors or incorrect calculations.

### 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. In essence all traditional languages on fully functional operating systems (such as POSIX-compliant OS or Windows) can access the OS-provided 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 operating system command, library provided or language thread activation mechanisms.
* Handle errors and exceptions that occur on activation.
* Create explicit synchronization protocols, to ensure that all activations have occurred before beginning the parallel algorithm, if not provided by the language or by the threading subsystem.
* Use programming language provided features that couple 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:

* Consider including automatic synchronization of thread initiation as part of the concurrency model.