Process Program Questions

Q: A class or a module?
A: Because this problem is to create a resuable utility that does not save state, it is a module problem.

Q: Error handling? Robustness? Security? Are any of these required?
A:
Error handling:

Treat all errors as extraordinary cases, as we really shouldn't be encountering any in our program.

Robustness:

We will sanitize the inputs to prevent the user from tampering with the system.

Security:
No security is needed. We spawn a thread to sleep and print a message, both well understood, researched system commands, any security issues with these system calls should be known beforehand. If we need to override security to do this, there are other more concerning problems besides our code.

Q: What components of the Ruby exception hierarchy are applicable to this problem? Illustrate your answer.
A:
TimeoutError: Thread running the timer may timeout in the code block if the timer is set to too long of a timeout.
ArgumentError: Incorrect arguments may pass through the shell and attmpt to be used to initialize and operate the timer.
ThreadError: Could occur if an attempt is made to modify the thread calling the timer before the thread has been initialized.

Q: Is Module Errno useful in this problem? Illustrate your answer.
A: Module Errno is useful for this problem. Since the timer itself will be initialized in C, it is important that any errors that are occur are handed up the higher application level such that the error can be understood and reacted to in a better context.
For example, if the Errno 'ENOMEM' was received this would notify the program that the host has
ran out of memory. By being able to receive this Errno the application would be able to more elegantly recover this by disallowing the user from spawning additional threads (as well as notifying them) until enough processes have closed such that the user can safely send more commands.

Q: Describe the article at:
http://today.java.net/pub/a/today/2006/04/06/exception-handling-antipatterns.html
Convince the marker that these Anti-patterns don't exist in your solution.

The article details on common bad habits of programmers when throwing and handling exceptions.

These anti-patterns don't exist in our solution because we would be handling individual exceptions as extraordinary runtime problems and logging them.

Q: Do they exist in your Shell solution?

They do not exist in our Shell solution. We only plan to catch specific exceptions, and log them.

Q: How can I make the timing accurate? What time resolution should I be looking at, remember real-time systems? Time formats?
A: Timing can be made accurate by going to as fine a resolution as possible.
For real time systems, the smaller the resolution the better until a resolution is met such that time errors at that scale override the accuracy.
Time formats depend on the desired accuracy for the system. A real time system requires high accuracy (accuracy down to milli or sometimes microsecond). Hence, the time format may not be in the scale of years, months, days, or even hours. It is more likely that a format of minutes down to microseconds or milliseconds is more appropriate. For example, consider the case of brakes in a car. This situation is an example of a real-time system. When the user presses the brakes, they are not expecting the car to begin slowing down within the scale of minutes but instead the scale of microseconds. The driver would want the car stop as soon as possible.

Q: Does 'C' have better facilities for this problem than Ruby? (Big hint!)
A: C has much better facilities for this problem than ruby. Both functions support timers to the scale of nsec. However, since C is much lower to the OS than Ruby, C would be the best choice. Since a developer can directly all assembly from C, this gives a developer much greater control over getting highly accurate time values in the nsec range. Ruby (which cannot directly call Assembly) is able to provide time values down to nsec but these values are highly inaccurate compared to a well-tuned piece of C code.

Q: What should be user controllable? Can we trust the user?
A: The only components that should be user controllable are the timer length and the timer message. We cannot trust the user. Even these two inputs should remain tightly controlled such that the user remains in a sandbox environment. The message needs to be checked that is a valid message (invalid characters or an empty message). The delay time needs to be checked to ensure that it remains

within a reasonable limit (time specified is not an unreasonably large value).
The user needs to be presented with the bare minimum required for functionality such there is less of a chance of invalid inputs that could possibly corrupt the internal program state.