CoCoA-5 - Feature #1399

CoCoA-5 interpreter: idle/busy indicator

21 Jan 2020 20:24 - John Abbott

Status:	Closed	**Start date:**	21 Jan 2020
Priority:	Normal	**Due date:**	
Assignee:	John Abbott	**% Done:**	100%
Category:	enhancing/improving	**Estimated time:**	7.30 hours
Target version:	CoCoA-5.3.0	**Spent time:**	7.15 hours

Description

Klaus Nguetsa is building a new GUI for ApCoCoA+CoCoA-5.

He would like to emulate the feature on the old C4 GUI where the screen indicated whether CoCoA was still computing, or just waiting for input.

How was this done in C4? And how can we do this in CoCoA-5?

Related issues:

Related to CoCoA-5 - Feature #299: GUI: message "running.." (close to output ...	**New**	**30 Jan 2013**
Related to CoCoA-5 - Bug #1438: ABORT for unterminated string literal	**Closed**	**05 Mar 2020**

History

#1 - 21 Jan 2020 20:36 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

I still have the C4 source code, but also have very little desire to delve into it to find out what was done for the old interface (using Qt?)

The main question seems to be: how is CoCoA-5 going to tell another program whether it is waiting for (top-level) input, or whether it is executing code?

- **(kludge)** make the interpreter print some invisible character just before it starts waiting for input, and a different invisible char just before it starts execution. This will fail if the user program happens to print out some of these invisible chars.
- **(OOB)** the status communication goes via another channel (and not stdout); this is OK provided that channel is used only in this way. A possibility is stderr, or maybe a socket, or even a file [must be careful that two CoCoAs running on the same machine use different files/sockets].
- might it be possible to use top? This could be not very portable.
- it could also be possible to send signals to another process (but how portable is that?)

How do other GUIs do this? It cannot be a new problem (even C4 did it).

#2 - 22 Jan 2020 09:58 - Anna Maria Bigatti

For cocoa5 it's easy: if there is no prompt, then it is busy.
That was not the case for cocoa-4, which had no prompt.

I know, this is a non-answer, but I think it is not so important.

#3 - 22 Jan 2020 15:01 - John Abbott

A problem with watching for the prompt is that a user program might print out a copy of the prompt inside some string.
Also I think it is necessary to signal both: "waiting for input" (i.e. the prompt), and "starting to execute command" (currently no explicit sign is given).

I'm not sure what happens if someone runs a program which uses StandardInput.
I'm not even sure what *should* happen in such a case.

#4 - 23 Jan 2020 15:05 - John Abbott

I looked through the CoCoA-4 code (which used a Qt interface).
The situation is different from Klaus's problem because the interpreter itself was modified to work with the Qt GUI.
The hope with CoCoA-5 was to build a GUI which did not require intrusive changes to the CoCoA-5 interpreter (or only minimal changes).

The is also the old Qt4 interface for CoCoA-5 which Ulrich has tried to resuscitate, apparently with some success (to be confirmed when I am in Genoa). Again this GUI has "intrusive" code in the interpreter (and I am not sure if there was an indication that CoCoA is busy).

#5 - 23 Jan 2020 17:32 - John Abbott

Perhaps the relevant function is Interpreter::readAndExecute, not sure though... (too busy with other stuff currently)

#6 - 24 Jan 2020 11:32 - John Abbott

I have just spoken to Kreuzer, and he showed me that the old ApCoCoA GUI would indicate either "Calculation in progress" or the CPU time of the last computation. This makes me strongly suspect that the old GUI also required modification to the C4 UI code (otherwise how/why would it indicate amount of CPU time consumed?

I think it is ambitious (*i.e.* risky) to hope to write the new UI part in C++ for CoCoA-5 in time for the COCOA School.
We could be lucky: perhaps copying the existing Qt-based code, and modifying that will suffice... but it is a gamble.

#7 - 24 Jan 2020 11:41 - John Abbott

I have just modified Interpreter.C so that it prints out [WAITING FOR INPUT] and [RUNNING].
A few quick tests suggest that it is working reasonably.

Not checked into CVS!

NOTE the changes are inside Interpreter::run. The clue was to mimic what was implemented for the Qt interface (code which is inside CPP blocks visible only when C5IDE is set). Inserted clog<<"[WAITING]" around line 2866; inserted clog<<"[RUNNING]" around line 2900.

#8 - 24 Jan 2020 15:53 - John Abbott

Here is what I get currently:

```
clog:[RUNNING]
clog:[WAITING FOR INPUT]
>>> n := factorial(10000000);
clog:[RUNNING]
clog:[TIME TAKEN: 2.5]
clog:[WAITING FOR INPUT]
>>>
```

Not checked into CVS!!

#9 - 04 Feb 2020 13:35 - John Abbott

- % Done changed from 10 to 20

I have just spoken to Klaus.
JAA will try to modify the interpreter so that it send the "status messages" to a specified socket; the socket will be specified by a command-line argument (hopefully), which will be supplied by the GUI upon start up.

#10 - 13 Feb 2020 15:36 - John Abbott

- Assignee set to John Abbott

- % Done changed from 20 to 60

A command line flag for the CoCoAInterpreter is now available, but it does not yet open a socket. This will be fixed at the next meeting with Klaus (I hope).

#11 - 19 Feb 2020 11:29 - John Abbott

Perhaps see also issue #1236 to see how I implemented sockets...

#12 - 21 Feb 2020 14:16 - John Abbott

- % Done changed from 60 to 80

I have just had a meeting with Klaus:

- KISS: we do not use sockets at the moment, but instead CoCoAInterpreter sends status messages to a specified file
- command line argument is now called **--export-status** an expects to be followed by the full path of the file to write to
- the status messages are simplified: **R** means running, and **W** means waiting for input; W is followed by the cputime of the last command

An example of the status outputs is:

```
W0.1
R
W0
R
W25.8
```

If a computation is interrupted, a **W** followed the cpu-time-so-far is sent as status message.
source and **SourceRegion** are regarded as single commands: so a single **R** and single **W** status message are produced.

#13 - 21 Feb 2020 14:18 - John Abbott

I have given Klaus a (minimal) executable to use for developing on his computer.
A full executable gave a linker error related to libgmpxx; I suppose my version used dynamic linking for the system installed version...

#14 - 02 Mar 2020 22:02 - John Abbott

- Related to Feature #299: GUI: message "running.." (close to output window?) added

#15 - 04 Mar 2020 17:00 - John Abbott

My proposed solution does not work quite as I would like.

With the input **if n = factorial(10^7) then println "equal"; endif** the log produces **R** and then **W2.5**.
BUT the same input with a final semicolon produces a different log, namely R then W2.5 **and then** R and W0.

I see two possible ways around this:

1. after a compound command, if the next symbol is just "semicolon" then silently consume it
2. do not print R and W lines in the log if the command is void

Approach 2 seems easier to me; and would also handle the case of accidentally putting too many semicolons after a command.

#16 - 04 Mar 2020 22:32 - John Abbott

I have changed the main eval loop so that EmptyStatements are skipped over without being executed (since they do nothing anyway).
At the moment Skip is still executed, but I can easily change that.

It seems to work as desired: *i.e.* it does not issue R and W0 log output.

#17 - 04 Mar 2020 22:32 - John Abbott

- Status changed from In Progress to Feedback

- % Done changed from 80 to 90

#18 - 06 Mar 2020 12:19 - John Abbott

- Related to Bug #1438: ABORT for unterminated string literal added

#19 - 06 Mar 2020 12:23 - John Abbott

- Estimated time set to 7.30 h

My testing suggests that this is OK now.

As I wrote in #1438, I now doubt whether it was useful to try to handle "empty statements" (*i.e.* superfluous semicolons) specially, so that no status log messages were produced. The example I gave in #1438 was:

```
n := factorial(10^7); m := 1;
```

This will produce two status log mesgs: the first for the long computation, but then immediately "supplanted" by the log for the short computation (*i.e.* constant 1).
If the GUI is to produce useful info about time taken, then it needs to indicate the time for all commands together (*e.g.* as would happen if they are sent via SourceRegion).

#20 - 06 Mar 2020 12:24 - John Abbott

- Status changed from Feedback to Closed

- % Done changed from 90 to 100