21.3.3 Projekt – Prozess-Steuerung und Prozess-Daten

In den folgenden Kapiteln 21.3.3 bis 21.3.6 werden Ihnen Projekte vorgestellt, bei denen Sie lernen werden, wie man die Instruktionen SHELL und EXEC vorrangig bei *interaktiven* Konsolen-Programmen erfolgreich einsetzen kann.

Sicher ist Ihnen aufgefallen, dass bisher keine weiteren Möglichkeiten präsentiert wurden, wie Sie Prozess-Daten lesen oder Eingaben an einen aktiven Prozess permanent übergeben:

- Es existieren zwar Events die Ihnen signalisieren, dass Daten verfügbar sind, aber Sie können diese Daten weder direkt lesen noch Daten an den Prozess senden.
- Diese Aufgaben werden von einem Stream-Objekt übernommen, das bereits im Kapitel 15.2 vorgestellt wurde, da die Klasse *Process* von der Klasse *Stream* erbt.
- Daher können Sie ein Process-Objekt wie einen Stream behandeln und dessen Eingabe- und Ausgabefunktionen wie zum Beispiel Write, Print oder Read einsetzen.

Es wird der Einsatz der Event-Handler

- *ProcessEventName_Read()*,
- *ProcessEventName_Kill() und*
- *ProcessEventName_Error(sError as String)*

beschrieben und es werden Prozeduren vorgestellt, mit denen Sie Eingaben an einen Prozess übergeben respektive Eingaben (Daten oder Befehle) in die Standardeingabe eines Prozesses schreiben. Das Kernstück der präsentierten Prozeduren werden diese Anweisungen sein:

- Print #ProzessVariable Eingabe_String oder
- Write #ProzessVariable Eingabe_String.

21.3.3.1 Projekt Prozessdaten – 'Basic Calculator'

In diesem Projekt geht es darum, die Basis für eine graphische Benutzeroberfläche (GUI) für das interaktive Konsolen-Programm 'bc' zu entwickeln.

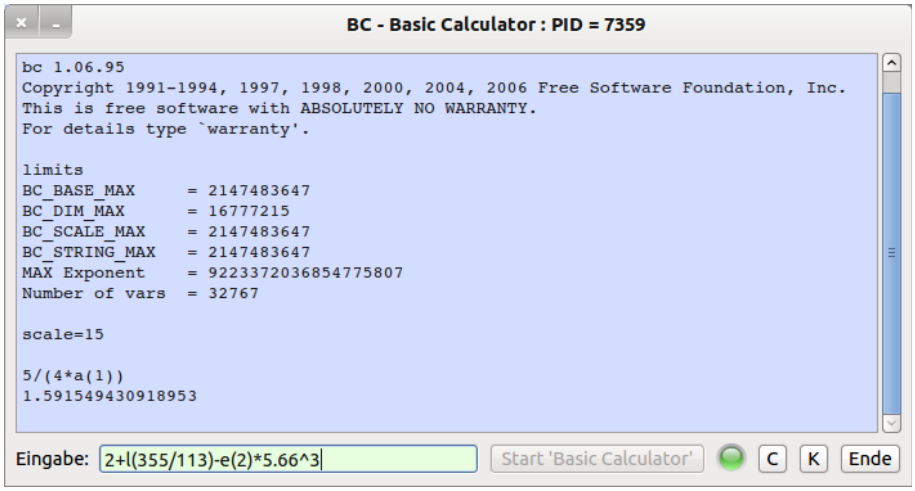

Abbildung 21.3.3.1.1: Entwurf GUI für das Programm 'bc' – Basic Calculator

Es gibt mindestens einen guten Grund, sich mit dem Programm 'bc' näher zu befassen:

```
hans@linux:~$ echo $(( 22/(7+4) ))
2
hans@linux:~$ echo $(( 22/(7+3) ))
2
hans@linux:~$ echo "scale=15; 0.77*2+e(3.44)" | bc -il
scale=15; 0.77*2+e(3.44)
32.726958168309462
hans@linux:~$
```

Im Gegensatz zur Integer-Arithmetik einer Konsole können Sie mit dem Programm 'bc' u.a. mit Fließkommazahlen rechnen, ausgewählte mathematische Funktionen nutzen und auch selbst definierte Konstanten und Funktionen speichern und verwenden.

Das Programm *Basic Calculator* ist ein geeigneter Kandidat für die Demonstration des Einsatzes der Instruktionen SHELL und EXEC, weil gezeigt werden kann wie man

(A) einen Prozess startet,
(B) Daten aus dem gestarteten Prozess liest und anzeigt,
(C) Fehlermeldungen ausliest, auswertet und anzeigt,
(D) Daten in die Standard-Eingabe des Prozesses schreibt,
(E) das Programm 'bc' regulär mit dem Kommando *quit* beendet und damit auch den Prozess,
(F) den gestarteten Prozess mit der Methode *hProcess.Kill* beendet, wenn das externe Programm 'bc' *nicht regulär* beendet wurde, weil das Gambas-Programm beendet wird und die Wirkung der Methode *hProcess.Kill* auswertet und anzeigt.

Der Quelltext wird vollständig angezeigt und in ausgewählten Abschnitten (A) bis (F) kommentiert:

```
' Gambas class file

Private $hProcess As Process
Private sProgrammName As String = "bc"
Private sStartParameter As String = "-i -l"

Public Sub Form_Open()
  FMain.Center
  FMain.Resizable = False
  txaOutput.Clear
  btnProcessKill.Enabled = False
  txbEingabe.ReadOnly = True
  SetLEDColor("orange")
End ' Form_Open()
```

(A)
Das externe Programm wird durch die Instruktion SHELL … gestartet. Der damit gestartete Prozess wird der Prozess-Variablen *$hProcess* zugewiesen und der Event-Name wird auf *myBCProcess* frei festgelegt.

```
Public Sub BCProcessStart()
  Dim sCommand As String

  txaOutput.SetFocus
  txaOutput.Clear
  txaOutput.Foreground = Color.Black
  $hProcess = Null

  sCommand = sProgrammName & " " & sStartParameter
  $hProcess = Shell sCommand For Read Write As "myBCProcess"

  FMain.Text = "BC - Basic Calculator : PID = " & $hProcess.Id
  btnBCProcessStart.Enabled = False
  btnProcessKill.Enabled = True
  txbEingabe.ReadOnly = False
  SetLEDColor("green")

End '  BCProcessStart()
```

(B)
Daten können aus dem Prozess im Event-Handler *myBCProcess_Read()* ausgelesen werden:

```
Public Sub myBCProcess_Read()
  Dim sPuffer As String

  txaOutput.Insert(gb.NewLine)
  sPuffer = ""
  Read #$hProcess, sPuffer, Lof($hProcess)
  txaOutput.Insert(sPuffer)
End ' myBCProcess_Read()
```

Die folgenden Alternativen sind für den blau markierten Text möglich:

Alternative1:
```
Read #$hProcess, sPuffer, -256
```

Alternative 2:
```
While Not Eof($hProcess)
  Read #$hProcess, sPuffer, Lof($hProcess)
Wend
```

Alternative 3:
```
Read #Last, sPuffer, Lof(Last)
```

Das folgenden Instruktionen sind keine Alternative, obgleich sie oft angeführt wird, denn es werden nicht *alle* ankommenden Daten auf der Standard-Ausgabe des Prozesses *vollständig* ausgelesen:

```
    Line Input #Last, sPuffer
    Line Input #$hProcess, sPuffer
```

Die Instruktion LINE INPUT sollte bei Prozessen nur mit Bedacht eingesetzt werden. LINE INPUT liefert Ausgaben des Prozesses in Einheiten von "Zeilen", d.h. empfangene Daten bleiben für das Gambas-Programm noch solange unsichtbar, bis die Zeile beendet wird. Aber nicht jedes Programm beendet wichtige Daten durch ein NewLine-Zeichen und deshalb können diese Informationen im Gambas-Programm nicht ausgelesen werden. So sendet zum Beispiel die Shell einen Befehlsprompt um zu signalisieren, dass sie bereit ist ein neues Kommando zu empfangen. Dieser Prompt wird in der Regel nicht durch ein NewLine beendet, weshalb das Programm – wenn es LINE INPUT verwendet – diesen Prompt nicht lesen kann. Die implizierte Bereitschaft der Shell weiterzuarbeiten geht verloren.

(C)
Das Event *Process_Error(...)* hat eine Besonderheit: Man liest nicht *direkt* von der Standard-Fehlerausgabe des Prozesses, sondern bekommt die Daten bereits im Parameter des Event-Handlers mit *Process_Error(sErrorMessage As String)*, wenn Daten an der Standard-Fehlerausgabe des Prozesses vorliegen. Ob Sie die Fehlermeldung ungefiltert ausgeben oder noch eine Fehlerbehandlungsroutine vorsehen, um in Abhängigkeit vom angezeigten Fehler den Programmablauf zu beeinflussen, wird sicher durch die zu lösende Aufgabe bestimmt:

```
Public Sub myBCProcess_Error(sFehler As String)
  txaOutput.Clear
  txaOutput.Insert(gb.NewLine)
  txaOutput.Insert("PROZESS-FEHLER!" & gb.NewLine)
  txaOutput.Insert(gb.NewLine)
  txaOutput.Insert(sFehler)
End ' myBCProcess_Error(..)
```

(D)
Wenn Eingaben des Benutzers an den Prozess übergeben werden sollen, dann müssen Sie das in einer eigenen Prozedur realisieren. Das Programm 'bc' als Konsolen-Programm erwartet arithmetische Ausdrücke und spezielle Kommandos in einer Zeile auf der Standard-Eingabe des gestarteten Prozesses. Diese Eingaben werden im Haupt-Programm über eine Eingabebox bereitgestellt:

```
Public Sub txbEingabe_Activate()
  If txbEingabe.Text = "" Then Return
  txbEingabe.Text = Trim(txbEingabe.Text)
  WriteToMyBCProcess(txbEingabe.Text)
  txbEingabe.Clear
End ' txbEingabe_Activate()
```

Sie müssen sich in einer *eigenen Prozedur* darum kümmern, die Eingaben für das externe Programm 'bc' in die Standardeingabe des gestarteten Prozesses zu schreiben. Da Sie ein Process-Objekt wie einen Stream behandeln können, setzen Sie dessen Ausgabefunktionen *Print* oder *Write* ein, nachdem geprüft wurde, ob ein Prozess-Objekt existiert und ob der Prozess aktiv ist:

```
Public Sub WriteToMyBCProcess(sInput As String)
   If $hProcess Then
      If $hProcess.State = $hProcess.Running Then
```

```
        Print #$hProcess, sInput
      Endif ' Process Running ?
   Endif ' Process started?
End ' WriteToMBCyProcess(...)
```

(E)
Wenn Sie das Kommando 'quit' eingeben, dann wird das externe Programm *Basic Calculator* regulär beendet und das Prozess-Objekt zerstört, nachdem zuvor der Prozess beendet wurde.

Das ausgelöste Ereignis *Kill* aktiviert den Event-Handler *myBCProcess_Kill()*, was Ihnen die Möglichkeit bietet, darauf angemessen zu reagieren:

```
Public Sub myBCProcess_Kill()
  txaOutput.Insert(gb.NewLine)
  txaOutput.Foreground = Color.Red
  txaOutput.Insert("Rückgabewert von '" & sProgrammName & "' = " & $hProcess.Value & Chr(10)

  Select Case $hProcess.State
  Case 0
    txaOutput.Insert("Prozess (PID= " & $hProcess.Id & ") normal beendet." & gb.NewLine)
  Case 1
    txaOutput.Insert("Prozess (PID= " & $hProcess.Id & ") gestoppt!" & gb.NewLine)
  Case 2
    txaOutput.Insert("Prozess (PID= " & $hProcess.Id & ") beendet! (SIGKILL)" & gb.NewLine)
  End Select ' $hProcess.State

  SetLEDColor("red")
  btnProcessKill.Enabled = False
  btnBCProcessStart.Enabled = True
End ' myBCProcess_Kill()
```

Sie erhalten im vorgestellten Projekt die folgende Ausgabe:

```
Rückgabewert vom Programm 'bc' =  0
Der Prozess mit der PID = 7411 wurde normal beendet.
```

(F)
Wenn Sie das Haupt-Programm schließen, dann sollten Sie vorher den gestarteten Prozess mit der Methode *ProzessVariablenName.Kill* beenden:

```
Public Sub btnProcessKill_Click()
  txaOutput.Clear
  If $hProcess Then $hProcess.Kill
  FMain.Text = "BC - Basic Calculator"
End ' btnProcessKill_Click()
```

Auch in diesem Fall erhalten Sie eine Ausgabe, die im Event-Handler *myBCProzcess_Kill()* generiert wurde:

```
Rückgabewert vom Programm 'bc' =  9
Der Prozess mit der PID = 7458 wurde beendet! (SIGKILL)
```

Der weitere Quelltext enthält keine Besonderheiten:

```
Public Sub btnClose_Click()
  FMain.Close
End ' btnClose_Click()

Public Sub btnTextAreaClear_Click()
  txaOutput.Clear
End ' btnTextAreaClear

Public Sub btnBCProcessStart_Click()
  BCProcessStart()
End ' btnBCProcessStart

Public Sub txaOutput_Change()
  txaOutput.Pos = Len(txaOutput.Text) ' ---> Sprung in die letzte Zeile
End ' txaOutput_Change()
```

```
Public Sub SetLEDColor(sLEDColor As String)
   PictureBox1.Picture = Picture["LED/led_" & sLEDColor & ".svg"]
End '  SetLEDColor(..)

Public Sub Form_Close()
  If $hProcess Then
     txaOutput.SetFocus
     txaOutput.Clear
     $hProcess.Kill
  Endif ' $hProcess ?
End ' Form_Close()
```

Hinweis:

Es bedeutet nicht notwendigerweise einen Fehler, wenn Daten auf der Standard-Fehlerausgabe ausgegeben werden, also das Error()-Event eines Prozesses aufgerufen wird. Das Programm "strace" zum Beispiel nimmt alle seine Ausgaben auf stderr vor, damit diese von den Ausgaben des aufgerufenen Programms durch Umleitung der Streams unterscheidbar sind. Wenn einen nur die Ausgabe von strace interessiert, dann kann man alles andere ignorieren (>/dev/null oder 2>strace.log).

21.3.3.2 Exkurs – Basic Calculator

Bevor Sie loslegen, um aus dem vorgestellten Ansatz für eine GUI für das Konsolenprogramm 'bc' Ihre Version zu entwickeln, hilft Ihnen ein Blick in die BC-Hilfen mit 'man bc' oder 'info bc'.

Hier werden interessante Quellen aufgeführt, um sich über die Besonderheiten und die Benutzung des Programms *Basic Calculator* zu informieren:

- http://openbook.galileocomputing.de/shell_programmierung/shell_004_001.htm
- http://spielwiese.la-evento.com/xelasblog/archives/21-Rechnen-in-der-Shell.html
- http://www.pro-linux.de/artikel/2/909/der-basic-calculator-bc.html
- http://www.linuxnix.com/2012/05/convert-binaryhex-oct-decimal-linuxunix.html
- http://www.linux-community.de/Internal/Artikel/Print-Artikel/LinuxUser/2004/04/Zu-Befehl-bash-bc/(printView)/true