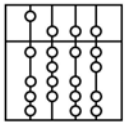


INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN
Informatik VI: Robotics and Embedded Systems
Dr. Gerhard Schrott

Prozeßrechner-Praktikum
Echtzeitsysteme

Einführung in VxWorks

Aufgaben

H. Hoffmann, J. Reiböck, G. Schrott

19. April 2004

1. VxWorks

VxWorks ist ein Echtzeitbetriebssystem der Firma Wind River Systems (www.windriver.com), das für den Einsatz in einer verteilten zeitkritischen Anwendungen entworfen wurde. Ihm liegt der Host-Target-Ansatz zu Grunde: Der Entwicklungsrechner läuft unter Windows XP und die Applikation wird auf den Targetrechner geladen und unter VxWorks ausgeführt.

So kann auf dem Host die Entwicklung der Software (Codierung, Compilierung, Debugging, Simulation ...) in einer komfortablen Umgebung durchgeführt werden, da für die Erledigung dieser Aufgaben kein Echtzeitbetriebssystem notwendig ist. Das Ausführen und Testen der Software erfolgt jedoch auf dem Targetrechner unter VxWorks mit Echtzeitbedingungen.

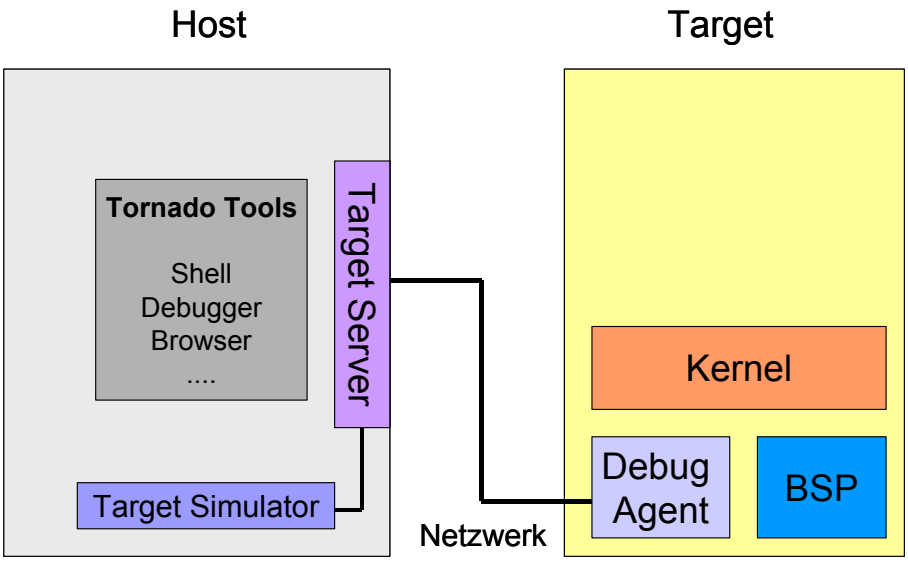


Abb.1: Die Architektur des Systems

VxWorks bietet über eine Shell auf dem Hostrechner interaktiven Zugriff auf den Targetrechner. So können die Programme auf den Targetrechner übertragen und schließlich ausgeführt werden. Auf dem Host lässt sich auch ein Targetrechner simulieren (mit VxSim), z.B. für die Einarbeitung und erste Versuche mit Tornado/VxWorks oder falls noch kein Targetrechner vorhanden sein sollte.

Das Herz des Laufzeitsystems von VxWorks ist der „wind"-Microkernel. Er ist skalierbar, man kann also selbst die Komponenten des Kernes je nach persönlichem Bedarf zusammenstellen. Der Kernel stellt Funktionen zur Verfügung wie z.B. Speichermanagement, eine Multitasking-Umgebung, Zeitdienste, Intertask Kommunikations- und Synchronisations-Mechanismen.

2. Tornado

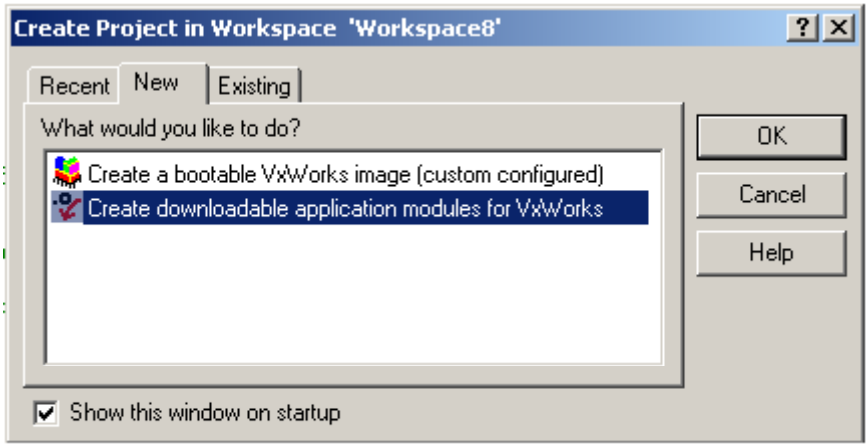
Das Cross-Entwicklungspaket Tornado ist speziell auf die Anforderungen der Softwareentwicklung für Echtzeitsysteme zugeschnitten und erleichtert so die Entwicklung von Programmen. Tornado läuft in Windows eingebettet auf dem Hostrechner.
Tornado bietet einige Tools und Funktionen, die zur Entwicklung von Echtzeitanwendungen hilfreich sind. Diese laufen hauptsächlich auf dem Hostrechner ab, so ist man unabhängig von den Ressourcen des Targetrechners.
Alle Handbücher zu Tornado und Vxworks sind im Verzeichnis Tornado/docs oder in Tornado selbst über den Menüpunkt *Help* verfügbar.

2.1. *Projekte und Anwendungen*

Alle Arbeiten mit Anwendungen in VxWorks finden im Kontext von Projekten statt. Ihre Verwendung und weitere Funktionen und Begriffe werden in diesem Abschnitt erklärt.

2.1.1. Projekte:

Alle Anwendungen für VxWorks werden in Projekten erstellt, wobei jedes Projekt sein eigenes Verzeichnis benötigt. Ein Projekt enthält den Quellcode, build settings etc, die verwendet werden, um herunterladbare oder bootbare Anwendungen zu erstellen.
Ein Projekt legt man an, indem man auf *File>new Project* klickt und in dem sich öffnenden Fenster auswählt, ob man ein herunterladbares oder bootbares VxWorks-Image erstellen möchte.



Im nächsten Schritt gibt man noch einen Namen und den Speicherort für das Projekt an und optional eine Beschreibung des Projektes. Außerdem muss man noch definieren, zu welchem Workspace man das Projekt hinzufügen möchte.

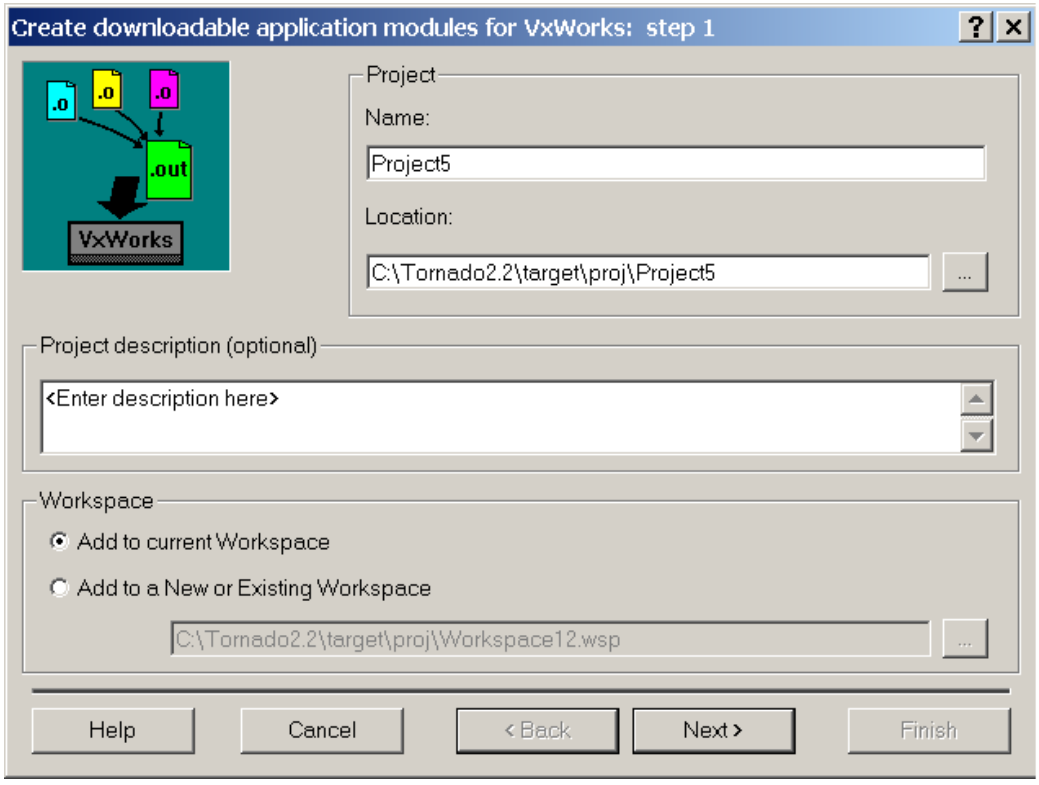


Abb.3 Erstellen eines Projektes, Schritt 2 (Beispiel downloadable application)

Im nächsten Schritt muss man noch angeben, auf welchem Board Support Package (BSP) bzw. auf welcher Toolchain (bei herunterladbaren Anwendungen) das Projekt basieren soll. Alternativ kann man das neue Projekt auf ein bereits bestehendes aufbauen lassen.

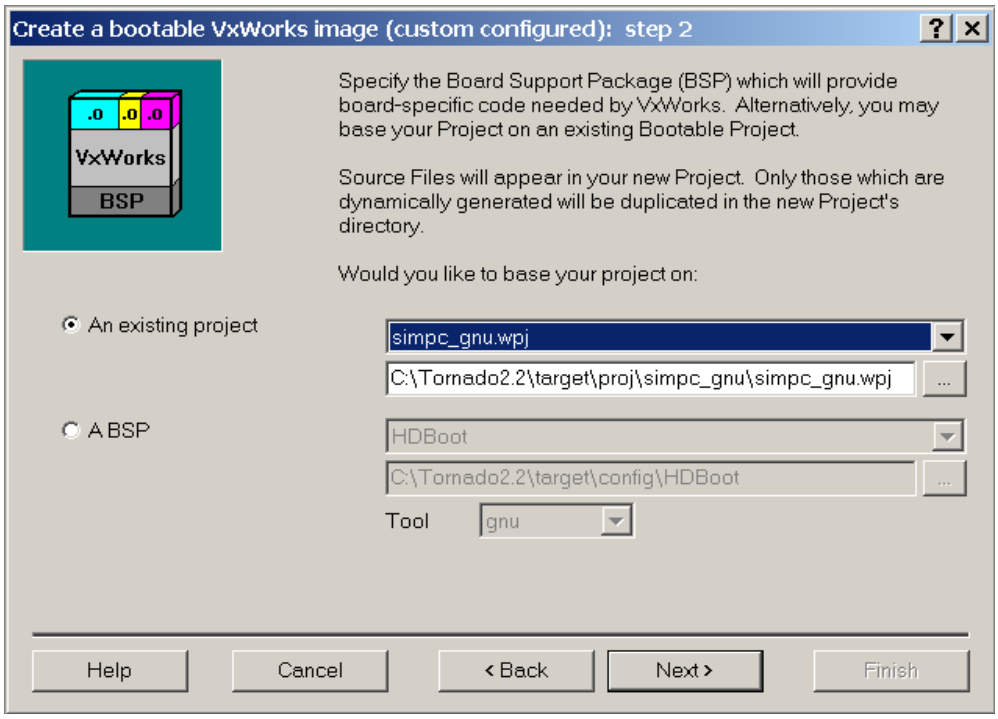
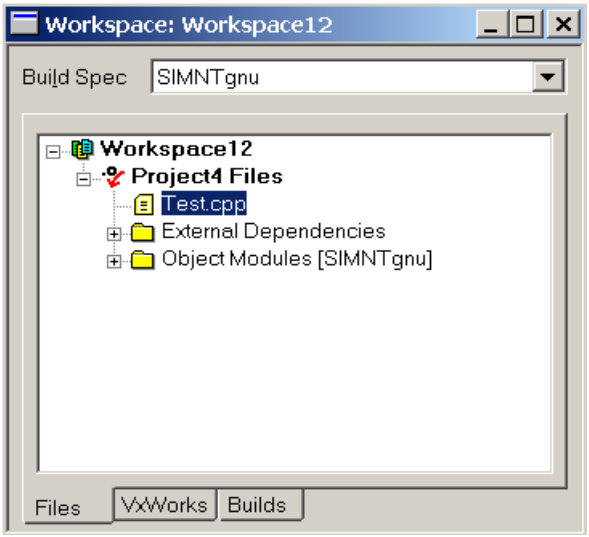


Abb.4: Erstellen eines Projektes, Schritt 3 (Beispiel bootable application)

2.1.2. Workspace:

Er ist eine Art logischer und graphischer Container für ein oder mehrere Projekte. Man kann z.B. alle Projekte, die inhaltlich miteinander zu tun haben, in einem Workspace ablegen (z.B. herunterladbare Anwendung, VxWorks Images etc).



2.1.3. Toolchain:

Dies ist eine Sammlung von Werkzeugen, um Anwendungen auf den speziellen Prozessor des Targets auszurichten. Wenn man z.B. den Simulator verwenden möchte, muss man als Toolchain beim Anlegen des herunterladbaren Projektes SIMNTgnu auswählen.

2.1.4. Downloadable Application:

Eine herunterladbare Anwendung besteht aus Objektmodulen, die auf VxWorks heruntergeladen und dynamisch gelinkt werden können. Sie können dann aus der Shell oder dem Debugger heraus gestartet werden.
Um eine downloadable application zu erstellen und auszuführen muss man:

1. Ein Projekt für eine downloadable application erstellen

   Im Menü File auf *new project* klicken und im Tab New den Eintrag 'Create downloadable application modules' auswählen. Die Felder für Name und Location ausfüllen und das Projekt entweder zu einem bereits existierenden Workspace oder einem neuen hinzufügen. Nun muss man noch auswählen, ob man das Projekt auf ein bereits existierendes Projekt oder auf eine BSP (bei einem Projekt für den Simulator muss man SIMNTgnu auswählen, sonst wählt man den passenden Eintrag für das Target, ganz allgemein PENTIUMgnu) aufbauen lassen möchte.

2. Die Anwendung schreiben und zum Projekt hinzufügen

   Um eine neue Datei zu erstellen klickt man auf *File>new*, wählt den Typ der Datei und den Namen des Projektes, zu der sie gehören soll und gibt einen Namen und den Speicherort für die Datei an. Um eine bereits existierende Datei zu einem Projekt hinzuzufügen, rechts-klickt man auf das Workspace-Fenster und wählt *add Files* aus. Dann kann man die einzufügende Datei auswählen.

3. Das build- Kommando ausführen

   Im Menü build auf *build* oder *rebuild all* klicken oder mit rechts-klick auf das Projekt den Eintrag *Build All* oder *ReBuild All* auswählen. Unter dem Eintrag Builds im Workspace-Fenster kann man die Optionen für Builds einstellen, z.B. kann man ein Projekt für ein Target auch auf dem Simulator laufen lassen, indem man mit rechts-klick im Workspace-Fenster den Eintrag new Build auswählt und dann für Default Build Spec for to den Simulator (SIMNTgnu) auswählt.

4. Die Objektmodule auf das Target herunterladen

   Als erstes muss man einen Targetserver oder den TargetSimulator starten. Dann wählt man erst den Targetserver aus, klickt danach rechts in das Workspace-Fenster und wählt download *ProjektName.out* aus.

5. Die Anwendung aus der Shell oder dem Debugger starten

   Die Anwendung lässt sich jetzt aus der Shell (öffnen mit Klick auf  oder unter Tools>Shell) oder dem Debugger (öffnen mit Klick auf und starten der Kommandozeile) mit dem Namen der Mainfunktion starten. Diese Funktion muss nicht 'main' heißen, man kann sich beliebige Namen aussuchen. Jedoch sollte man, um Konflikte zu vermeiden, für jede main-Funktion einen anderen Namen nehmen (auch über die Projekte hinweg).

Die heruntergeladene Anwendung lässt sich mit rechtsklick auf das Projekt und Auswahl von *unload* wieder vom Target entfernen.

2.1.5. Starten eines Targetserver:

Im Menü Tools wählt man den Eintrag Target Server -> configure.. aus. Der Target Server Name ist beliebig, alle anderen Einstellungen kann man im Prinzip übernehmen, nur die IP-Adresse des Targets muss man noch angeben. Mit Klick auf Launch wird der Target Server nun gestartet.

2.2. Target Simulator

Der Target Simulator von VxWorks ist ein Port von VxWorks zu dem Hostsystem, der ein Target Betriebssystem simuliert. Man benötigt hierfür keinerlei Hardware für das Target. Der Simulator ist z. B. nützlich für erste Schritte mit Tornado und der Echtzeitentwicklung.
Mit dem Simulator lassen sich Anwendungen testen, die nicht von hardwarespezifischem Code oder der Target Hardware abhängen.
Man startet den Target Simulator, indem man auf Tools>Simulator klickt und den Standard-Simulator auswählt.

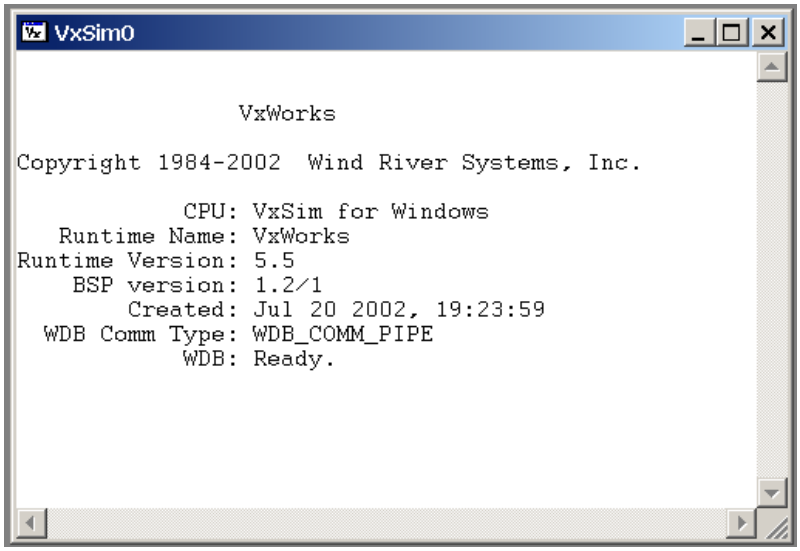


Abb.6: Der Target Simulator VxSim

2.3. Shell

Die Shell kann mit Klick auf das Symbol aufgerufen werden, wenn ein Target Server läuft. Sie wird auf dem Host ausgeführt und hat eine Doppelfunktion: Zum einen bietet sie Zugang zu allen Funktionalitäten von VxWorks, indem man einfach eine VxWorks-Routine in der Shell aufruft. Zum anderen kann sie gut zum Testen und Debuggen eingesetzt werden, da man Module einer Applikation interaktiv durch Aufrufen einer ihrer Routinen ausführen kann.
Man kann z.B. Tasks spawnen, lesen von und schreiben auf Targetdevices oder volle Kontrolle über das Target ausüben (z.B. das Target rebooten).
Hier ein Auszug der Funktionalität der Shell:

- Taskspezifische Breakpoints setzen
- Taskspezifisches singel-stepping
- Zugang zu Informationen über Tasks und das System
- Möglichkeit, Routinen des Benutzers aufzurufen

Die Shell liest einen Inputstream zeilenweise, parst und wertet jede Zeile aus und schickt das Ergebnis an einen Outputstream. Es wird die gleiche Syntax akzeptiert wie von einem C-Compiler (mit wenigen Abweichungen).

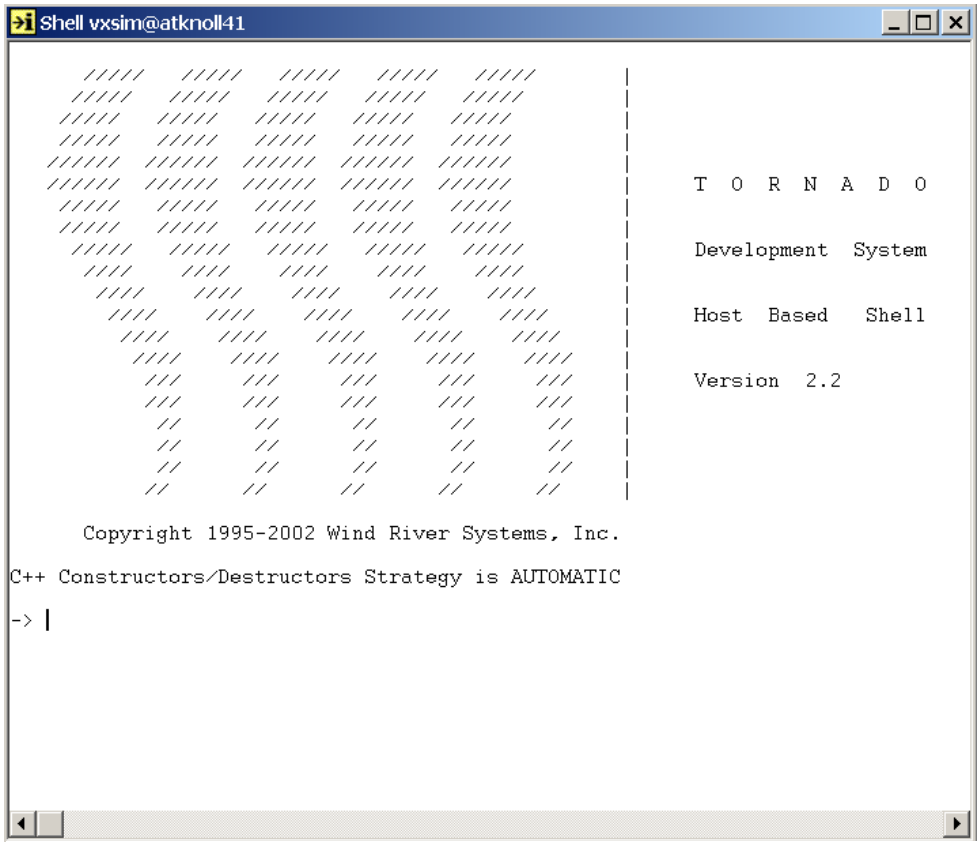


Abb.7: Eine Shell

2.4. Debugger

Tornado liefert ebenfalls einen Debugger, CrossWind. Die üblichen Debugging-Aktivitäten wie das Setzen von Breakpoints und die Kontrolle der Programmausführung können einfach durch Klicken ausgeführt werden.
Um den Debugger zu starten, wählt man erst das entsprechende Target aus. Dann klickt man auf das Symbol  .
Über Debug>Debug Windows kann man sich Fenster auswählen, die angezeigt werden sollen (z.B. watch).

Um Variablen oder Tasks zur Überwachung hinzuzufügen, rechts-klickt man auf die entsprechende Variable im Programmcode und wählt *add to watch* aus. Um Breakpoints zu setzen, rechts-klickt man auf die jeweilige Stelle im Code und wählt Breakpoint aus.
Das Programm lässt sich aus im Single-Step Modus ausführen (Taste F11).
Der Debugger enthält noch eine Menge weitere Funktionen, die im Tornado User's Guide dokumentiert sind (Kapitel 7).

2.5. Browser

Mit Hilfe des Browsers kann man den Zustand des Target überwachen (Er ist ähnlich wie die Shell, nur graphisch). Das Hauptfenster listet z.B. aktive Tasks und den Speicherverbrauch auf. Ebenfalls kann man Informationen sehen wie

- Detaillierte Informationen über Tasks,
  wie z.B. ihre Prioritäten, Nutzung der Register und andere Attribute
- Semaphore
- Message-Queues
- Watchdog Timer
- Nutzung des Stacks durch alle Tasks auf dem Target
- Nutzung der CPU des Target durch Tasks

Der Browser lässt sich, wenn ein Target ausgewählt ist, mit Klick auf  oder durch *Tools>Browser* starten. Alle angezeigten Informationen sind Snapshots. Sie können interaktiv mit Klick auf  aktualisiert werden.

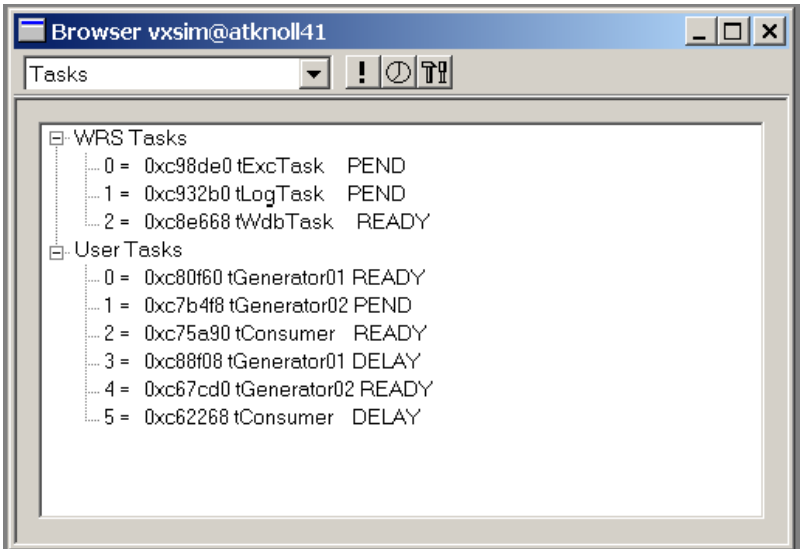


Abb.8: Der Browser

2.6. WindView

WindView ist ein dynamisches Visualisierungstool, das Informationen liefert über Kontextwechsel und die Ereignisse, die dazu geführt haben, und über sogenannte „instrumented objects“ wie z.B. Semaphor geben oder nehmen, senden oder empfangen von einer Messagequeue oder Signale. Für den Target Simulator wird eine integrierte Version von WindView mitgeliefert. Für andere Targets muss man dieses Tool gesondert erwerben.

WindView öffnet man über das Symbol  .

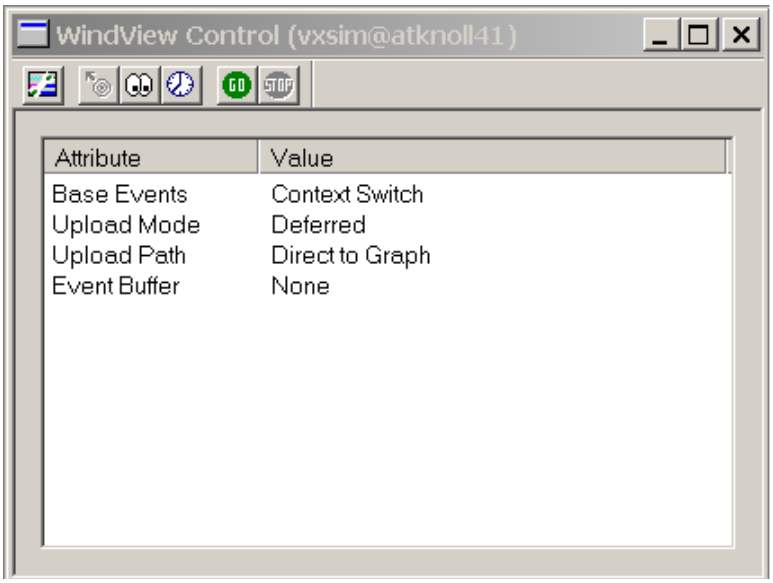
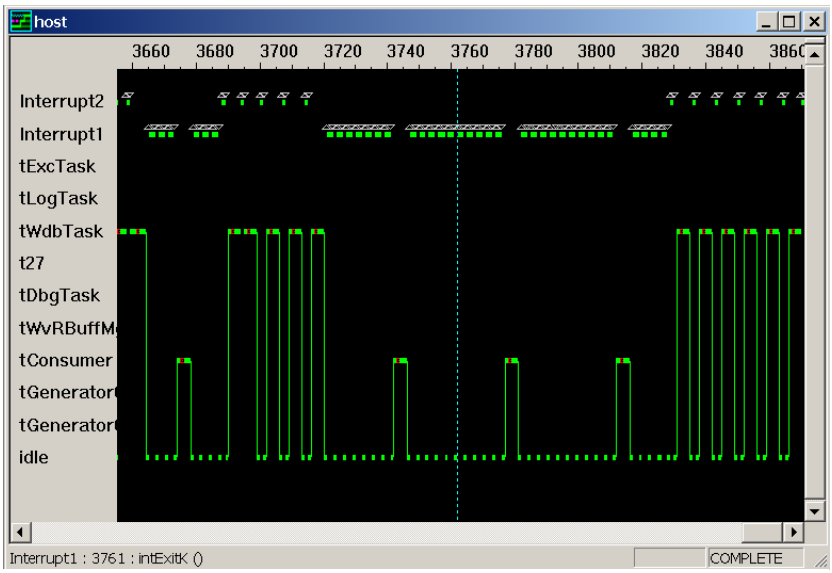


Abb.9: Das Tool WindView

Mit Klick auf  startet man das die Datensammlung. Ein paar Sekunden warten, dann auf den Update-Button  klicken, um den Status der Datensammlung zu aktualisieren, wieder kurz warten und dann auf Stop  klicken. Jetzt den Upload-Button )benutzen, um die Daten auf den Host zu laden. Über die Zoom-Buttons  lässt sich der Auszug dann vergrößern oder verkleinern.



3. Aufgaben

3.1.1. Übersicht über die Aufgaben

0. Einführung in C unter VxWorks
1. Multitasking und Interprozesskommunikation
2. Zyklisch ablaufende Prozesse
3. Erzeuger-Verbraucher-Problem
4. Kugelfall-Versuch
5. Aufzugssteuerung
6. Fertigungsanlage

3.1.2. Erfolgreiche Teilnahme am Praktikum

Folgende Leistungen sind Voraussetzung für den Erwerb des Praktikumscheins:

a) Die Aufgaben 0,1,2,3 und 4 von ADA müssen bearbeitet werden.
b) Eine Steuerung für die Aufgaben 5 oder 6 soll erstellt werden, wobei die Lösung zur Aufgabe 6 im Team implementiert werden kann.
c) Anwesenheit an allen Praktikumsnachmittagen
d) Vorführung der Lösungen
e) Mündliches Kolloquium (gruppenweise) am Ende des Praktikums
f) Eine Woche vor Praktikumsende ist pro Gruppe eine Ausarbeitung abzugeben, die für jede der bearbeiteten Aufgaben folgendes enthalten muß:
   - Auf DIN A4-Format zugeschnittenes Programmlisting
   - Programmdokumentation

3.2. Aufgabe 0: Einführung in C unter VxWorks

3.2.1. Ziel

C ist eine sehr einfache Programmiersprache die durch die Verwendung verschiedener Bibliotheken erweitert werden kann. Die folgende Aufgabe soll Ihnen dabei helfen sich in C einzuarbeiten und die speziellen Eigenarten von C unter VxWorks kennenzulernen.

3.2.2. Aufgabe

Verbessern Sie das folgende in Ihrem Homeverzeichnis enthaltene, fehlerhafte C-Programm und führen Sie es am Simulator sowie am Targetrechner aus. Der Ablauf von Tasks unter VxWorks hängt von Prioritäten ab: testen Sie das Zusammenspiel der Tasks mit verschiedenen Prioritäten. Geben Sie das korrigierte Programmlisting und die kommentierte Ausgabe im DIN A4-Format geheftet ab.

3.2.3. Implementierungshinweise

Empfehlenswert ist die Lektüre von Kernighans & Ritchies „The C Programming Language“.
Kleine Feinheiten von C Programmen unter VxWorks sind:

- spezielle Libraries: immer eingebunden werden vxWorks.h und taskLib.h
- keine main Methode notwendig. Es läßt sich von der Shell aus jede Funktion anspringen. Das liegt daran, dass VxWorks keinen Unterschied zwischen Systemcalls und C Subfunktionsaufrufen kennt.
- Der doppelte Slash // wird nicht als Kommentarzeichen erkannt.

```
#include <vxWorks.h>
#include <taskLib.h>
#include <stdio.h>

#define STACK_SIZE 10000

double tid_Main, tid_Task1, tid_Task2;
void Task1 (void);
void Task2 (void);

STATUS main (void)
{
    tid_Main = taskIdSelf();
    printf("Hauptprogramm gestartet!\n");
    tid_Task1 = taskStart ("tProcess1", 200, 0, STACK_SIZE,
                           (FUNCPTR)Task1,0,0,0,0,0,0,0,0,0,0);
    taskPause(tid_Main);
    tid_Task2 = taskStart ("tProcess1", 300, 0, STACK_SIZE,
                            Task2,0,0,0,0,0,0,0,0,0,0
    printf("Hauptprogramm beendet!\n");
}

int Task1 (void)
{
    printf("Task1 gestartet!\n‟);
    string str = ("Hello World!");
    for( i = 0; i < 10; i++){
        printf("%s\n", str);
    } else {
        printf("Goodbye!\n");
    }
    printf("Task1 beendet!\n");
    taskResume(tid_Main);
}

int Task2 (void)
{
     printf("Task2 gestartet!\n");
     int x, y;
     for( x =  0; x >= 0; x++){
        y /= sin(x);
     }
     taskSuspend(tid_Task1);
     printf("Task2 beendet!\n");
}
```

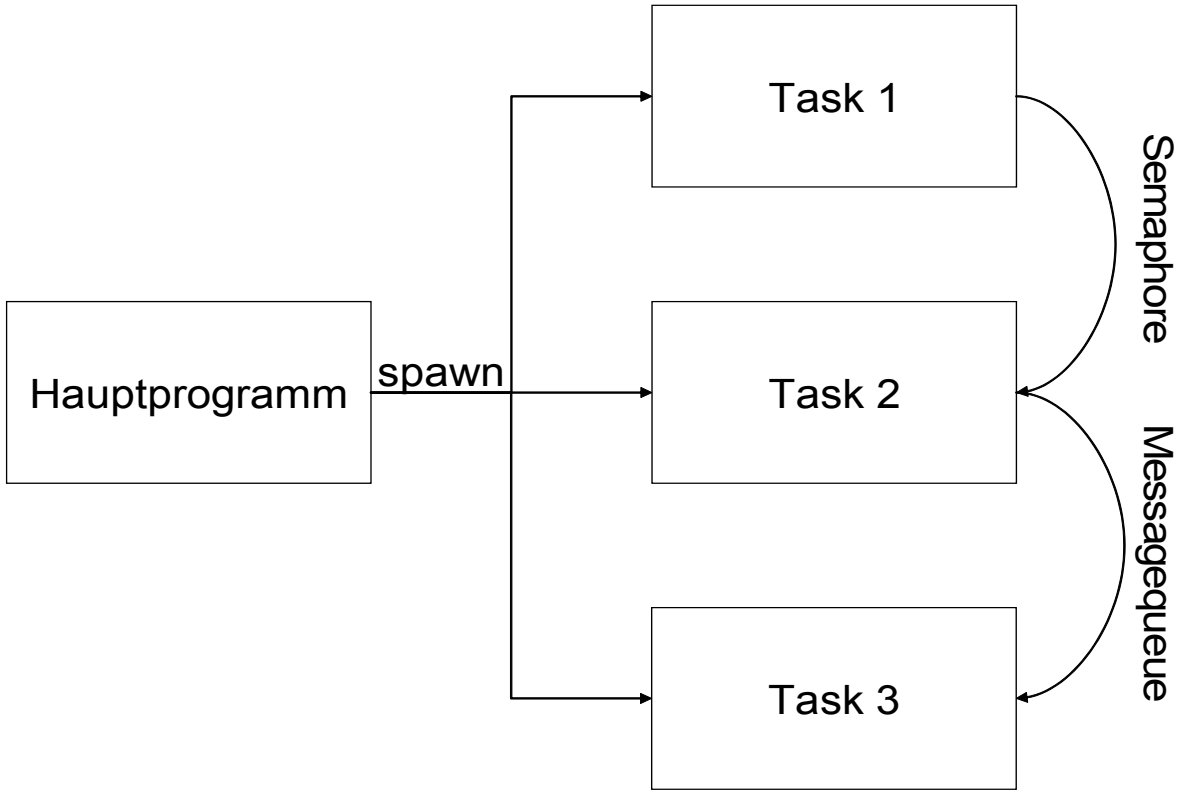
3.3. Aufgabe 1: Multitasking und Interprozesskommunikation

3.3.1. Ziel

Gerade bei Programmen die auf einem echtzeitfähigen Betriebssystem ausgeführt werden, ist es wichtig jederzeit über den Zeitverlauf einzelner Tasks und deren Zusammenspiel mit anderen, parallel laufenden Tasks Bescheid zu wissen. Das Ziel dieser Aufgabe ist es aufzuzeigen, wie Tasks untereinander kommunizieren können.

3.3.2. Aufgabe

Schreiben Sie ein Programm bei dem 3 Tasks parallel gestartet werden. Während die erste Task sofort mit der Ausführung (ggf. nur ein Delay) beginnen darf, wartet die zweite Task auf einen Semaphor, der zunächst von Task 1 freigegeben werden muss. Die dritte Task wiederum wartet bis sie von der Messagequeue eine von Task 2 an sie gerichtet Nachricht mit Inhalt "Task 3 start" erhält. Um den zeitlichen Ablauf nachvollziehen zu können bietet es sich an, die einzelnen Schritte der Tasks mit Kommentaren auf die Konsole zu versehen. Geben Sie Ihr kommentiertes Programmlisting zusammen mit der Programmausgabe im DIN A4-Format geheftet ab.



3.3.3. Implementierungshinweise

Um die nachfolgende Aufgabe lösen zu können müssen Sie Tasks, einen Semaphor und eine Messagequeue erzeugen und deren zeitlichen Verlauf nachvollziehen. Eine Warnung vorweg: von einigen der benutzten Funktionen und Prinzipien gibt es POSIX-Varianten. Diese haben nur bedingt etwas mit den Konzepten unter VxWorks zu tun, es ist mehr als wahrscheinlich dass sie, falls sie funktionieren, nicht korrekt ablaufen. Benutzen Sie die VxWorks Funktionen!

Das Erzeugen & Starten einer neuen Task:

`Int taskSpawn(`	Der Rückgabewert ist die ID der erzeugten Task
`Name,`	Name der Task, ASCII String, frei wählbar
`Priorität,`	Priorität der Task, Integer, 0-255
`Optionen,`	Eigenschaften der Task, Word, meist 0x00
`Stack,`	Größe des Stack, Integer
`Funktion,`	Zeiger auf die aufzurufende Funktion, FUNCPTR
`Arg1,`	10 Übergabeparameter an die Funktion, Integer
`…,`	
`Arg10)`	

Pausieren einer Task:

`STATUS taskDelay(`	Der Rückgabewert ist OK oder ERROR
`Ticks)`	Ticks (1/60 Sekunden) die gewartet werden soll

Anhalten einer Task:

`STATUS taskSuspend(`	Der Rückgabewert ist OK oder ERROR
`ID)`	ID der Task die angehalten werden soll

Fortsetzen einer Task:

`STATUS taskResume(`	Der Rückgabewert ist OK oder ERROR
`ID)`	ID der Task die fortgesetzt werden soll

Löschen einer Task:

`STATUS taskDelete(`	Der Rückgabewert ist OK oder ERROR
`ID)`	ID der Task die gelöscht werden soll

VxWorks kennt verschiedene Semaphoren: den binären Semaphor, den zählenden Semaphor, die Mutex und die Shared Memory Mutex. All diese Semaphoren können nach Einbinden der `semLib.h` Bibliothek benutzt werden. Interessant ist hier ein einfacher binärer Semaphor, der folgendermaßen benutzt wird:

Semaphore initialisieren:

`SEM_ID semBCreate(`	Der Rückgabewert ist die ID (der Bezeichner)
`Option,`	Art des Semaphor: SEM_Q_FIFO oder SEM_Q_PRIORITY
`Status)`	Anfangswert: SEM_FULL oder SEM_EMPTY

Semaphore löschen:

`STATUS semBDelete(`	Der Rückgabewert ist OK oder ERROR
`ID)`	ID des zu löschenden Semaphor

Semaphore nehmen:

`STATUS semTake(`	Der Rückgabewert ist OK oder ERROR
`SEM_ID,`	Name des Semaphor (von semBCreate)
`Timeout)`	Wartezeit auf den Semaphor in Ticks (1/60 Sekunden); oder WAIT_FOREVER; oder NO_WAIT

Semaphore geben:

`STATUS semGive(`	Der Rückgabewert ist OK oder ERROR
`SEM_ID)`	Name des Semaphor (von semBCreate)

Eine weitere Möglichkeit der Interprozesskomunikation ergibt sich durch die Benutzung von sogenannten Messagequeues. Vor der Benutzung muss die Bibliothek `msgQLib.h` eingebunden werden.

Messagequeue initialisieren:

`MSG_Q_ID msgQCreate(`	Der Rückgabewert ist der Bezeichner der Queue
`Größe,`	Anzahl der Nachrichten die gefasst werden können, Int
`Länge,`	Maximale Nachrichtenlänge in Bytes, Int
`Art)`	Art der Queue: MSG_Q_FIFO oder MSG_Q_PRIORITY

Messagequeue löschen:

`STATUS msgQDelete(`	Der Rückgabewert ist OK oder ERROR
`ID)`	Bezeichner der zu löschenden Queue

Nachricht senden:

`STATUS msgQSend(`	Der Rückgabewert ist OK oder ERROR
`ID,`	Zielqueue der Nachricht, MSG_Q_ID
`Nachricht,`	Nachricht, Char[]
`Länge,`	Nachrichtenlänge, Uint
`Timeout,`	Ticks die auf Zustellung gewartet werden soll
`Priorität)`	MSG_PRI_NORMAL oder MSG_PRI_URGENT

Nachricht empfangen:

`Int msgQReceive(`	Der Rückgabewert ist der Länge des empfangenen Buffers
`ID,`	Quellqueue der Nachricht, MSG_Q_ID
`Nachricht,`	Puffer für die Nachricht, Char[]
`Länge,`	Nachrichtenlänge, Uint
`Timeout)`	Ticks die auf den Empfang gewartet werden soll

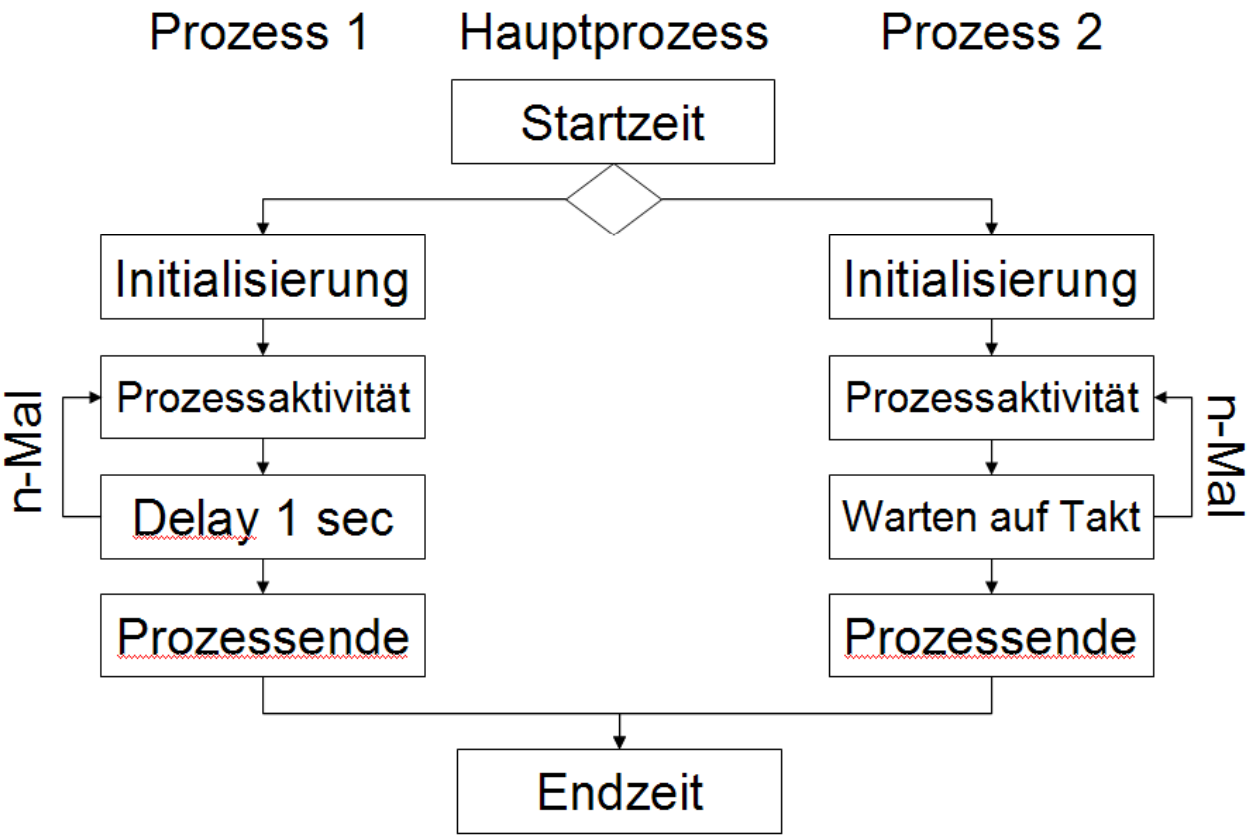
3.4. Aufgabe 2: Zyklisch ablaufende Prozesse

3.4.1. Ziel

In diese Aufgabe soll das unterschiedliche Zeitverhalten zyklischer Prozesse gezeigt werden.

3.4.2. Aufgabe

Programmieren Sie nach dem angegebenen Muster je einen zyklisch ablaufenden Prozess. Der erste Prozess soll nach seiner Prozessaktivität 1 Sekunde warten und dann neu starten; der zweite Prozess hingegen soll im Takt von 1 Sekunde neu gestartet werden. Geben Sie nach jeder Prozessaktivität die Zeit aus.
Um den Takt zu realisieren verwenden Sie einen Watchdog-Timer und schreiben einen kleinen Timer-Listener der bei Auslösen des Timers einen Semaphor (auf den Prozess 2 wartet) freigibt und den Timer neu startet.
Um ein unverfälschtes Ergebnis zu erhalten müssen die Prozesse sequentiell ablaufen. Geben Sie Ihr kommentiertes Programmlisting zusammen mit der Programmausgabe und der Interpretation des unterschiedliche Zeitverhaltens der beiden Prozesse im DIN A4-Format geheftet ab.



3.4.3. Implementierungshinweise

Wählen Sie als Prozessaktivität die Berechnung von `x = sin(x*x/x);` wobei der initiale Wert von x gleich PI (auf 5 Stellen genau) sein soll. Führen Sie diese Berechnung ½ Million mal durch.

```
x = PI;  /* PI muss definiert werden! */
for( index = 0; index <= 500000; index++){
    x = sin(x*x/x);
}
```

Da der Sinus in C nicht verfügbar ist muss die Bibliothek `math.h` in das Programm eingebunden werden. Gleiches gilt für die Methoden zur Zeitmessung, hierfür ist die Bibliothek `time.h` nötig. Um die Zeiten messen zu können müssen zunächst zwei Structs vom Typ timespec spezifiziert werden:

```
struct timespec start, stop;
```

Um die aktuelle Zeit zu erhalten wird die Funktion clock_gettime mit den Parametern CLOCK_REALTIME und einer Referenz auf eine der beiden timespecs aufgerufen:

```
clock_gettime(CLOCK_REALTIME, &start);
```

Der Unterschied zweier Zeiten in Sekunden lässt sich nun folgendermassen berechnen:

```
elapsed  = (stop.tv_sec - start.tv_sec);
elapsed += (double)(stop.tv_nsec - start.tv_nsec);
elapsed /= (double)BILLION; /* BILLION = 1000000000 */
```

Die Variable elapsed ist hierbei vom Typ Double, mit BILLION ist ein Makro gemeint, welches hier den Wert 1000000000 einsetzt.

Der Watchdog-Timer in VxWorks ist eine Softwarelösung, damit verschiedene Prozesse gemeinsam auf eine Hardware-Clock zugreifen können. Da Watchdog Timer auf Interruptebene ablaufen (das heißt: keine Prioritäten, kein Scheduling etc.) sollte der Code der Timeout-Methode so kurz wie möglich gehalten werden: kurzum, bauen Sie einen kleinen Timer-Listener! Um Watchdogs unter VxWorks benutzen zu können, binden Sie bitte die Library `wdLib.h` ein. Auch hier gilt wieder: Die POSIX-Namensvettern haben nur bedingt etwas mit diesen Konzepten zu tun. Benutzen Sie die VxWorks Varianten!

Watchdog initialisieren:

`WDOG_ID wdCreate( void )`	Der Rückgabewert ist der Bezeichner des Watchdogs oder NULL

Watchdog löschen:

`STATUS wdDelete(`	Der Rückgabewert ist OK oder ERROR
`wdID)`	Bezeichner des Watchdogs, WDOG_ID

Watchdog starten & einhängen:

`STATUS wdStart(`	Der Rückgabewert ist OK oder ERROR
`wdID,`	Bezeichner des Watchdogs, WDOG_ID
`Delay,`	Verzögerung in Ticks, int
`Routine,`	Funktion die bei Timeout aufgerufen werden soll, FUNCPTR
`Parameter)`	Watchdog Parameter, int, meist 0

Watchdog unterbrechen:

`STATUS wdCancel(`	Der Rückgabewert ist OK oder ERROR
`wdID)`	Bezeichner des Watchdogs, WDOG_ID

Bei der Implementierung ist zu beachten, dass der Watchdog nur einmal abläuft. Benötigt man einen zyklischen Ablauf so muss der Timer durch die beim Timeout aufgerufene Funktion neu gestartet werden. Die zwei Aufgaben des TimerListeners sind also somit klar: zum einen den Watchdog neu starten, zum anderen eine Semaphore freigeben die Task 2 zu starten erlaubt. Task 2 bei Timeout starten ist eine schlechte Idee, siehe oben: Watchdogs arbeiten auf Interruptebene, also Finger weg von allem was blockiert (z.B. `read`, `semTake`, `msgQReceive` und `printf`)!

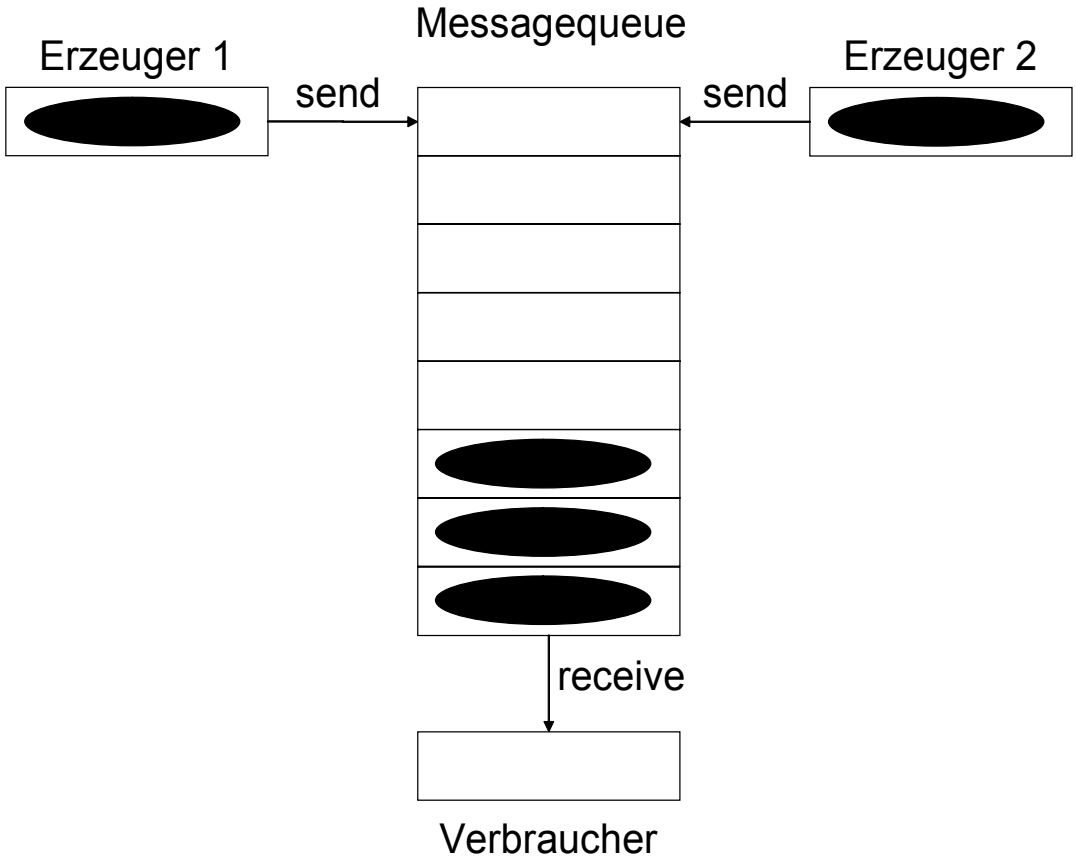
3.5. Aufgabe 3: Erzeuger-Verbraucher Problem

3.5.1. Ziel

Prozesse und deren Kommunikation sind oftmals sehr komplex. Ähnlich wie z.B. Anfragen an einen Webserver kann die Kommunikation einen Prozess "überfordern". Diese Aufgabe soll vermitteln, wie in VxWorks mit Puffern, den Messagequeues verfahren wird.

3.5.2. Aufgabe

Starten Sie zwei Erzeugerprozesse (der gleichen Funktion) die wechselseitig je 15 Nachrichten alle 0.5 Sekunden in die Messagequeue schicken. Das Fassungsvermögen der Queue sei auf 8 Einträge beschränkt. Starten Sie einen Verbraucher, der die Nachrichten alle 1,2 Sekunden abruft. Um den Verlauf nachvollziehen zu können ist es sinnvoll, die diversen Schritte der einzelnen Prozesse auf der Konsole auszugeben. Geben Sie das kommentierte Programmlisting inklusive Ausgaben der Tasks im DIN A4-Format geheftet ab.



3.5.3. Implementierungshinweise

Die zwei Erzeugerprozesse sollen Prozesse derselben Funktion sein, das heißt Sie müssen zweimal hintereinander die Funktion taskSpawn() ausführen. Lassen Sie jeden Erzeuger seinen Namen (oder einfach „Erzeuger 1“ bzw. „Erzeuger 2“) und die Nummer der gesendeten Nachricht in die Nachricht selbst schreiben. Sie finden sehr einfach heraus um welchen Erzeuger es sich handelt, wenn Sie die aktuelle ID Ihrer Task mit der global Verfügbaren ID der Erzeugertasks vergleichen.

Die eigene TaskID ermitteln:

`int taskIdSelf(void)`	Der Rückgabewert ist die eigene TaskID

Den Namen der Task (beim Spawning vergeben) ermitteln:

`char *taskName(tID)`	Der Rückgabewert ist ein Zeiger auf den Namen (String) ID der Task, deren Namen man wissen möchte.

Die Nachricht für den Messagebuffer können Sie wie folgt aus der Nummer der aktuellen Nachricht (N_ID) und dem Namen der Task (werbinich) zusammensetzen:

```
sprintf(msgBuff, "Nachricht %i von %s", N_ID, werbinich);
```

Geben Sie entsprechend Fehler- & Erfolgsmeldungen aus wenn das Senden in die Queue oder das Empfangen von der Queue funktioniert oder nicht funktioniert hat. Als Erfolgsmeldung des Verbrauchers geben Sie die empfangene Nachricht aus.
Zu guter letzt beenden Sie bitte den Verbraucher wenn keine weiteren Nachrichten mehr in der Queue sind.

Anzahl der Nachrichten in der Queue ermitteln:

`int taskIdSelf(`	Der Rückgabewert ist die Anzahl der Nachrichten
`msgQID)`	Bezeichner der Messagequeue