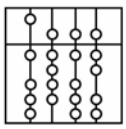


INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Informatik VI: Robotics and Embedded Systems
Dr. Gerhard Schrott

Prozessrechner-Praktikum
Echtzeitsysteme

Einführung in VxWorks

Aufgaben

H. Hoffmann, J. Reiböck, A. Kraus, G. Schrott

16. Oktober 2004

1. VxWorks

VxWorks ist ein Echtzeitbetriebssystem der Firma Wind River Systems (www.windriver.com), das für den Entwurf von verteilten zeitkritischen Anwendungen entwickelt wurde. Ihm liegt der Host-Target-Ansatz zu Grunde: Der Entwicklungsrechner läuft unter einem Windows-Betriebssystem und die echtzeitkritische Applikation wird auf einem Targetrechner ausgeführt.

Auf dem Host kann die Entwicklung der Software (Codierung, Compilierung, Debugging, Simulation, ...) in einer komfortablen Umgebung verrichtet werden. Für die Erledigung dieser Aufgaben ist kein Echtzeitbetriebssystem (VxWorks) notwendig. Das Ausführen und Testen der Software erfolgt jedoch auf dem Targetrechner unter Echtzeitbedingungen oder auf dem Simulator.

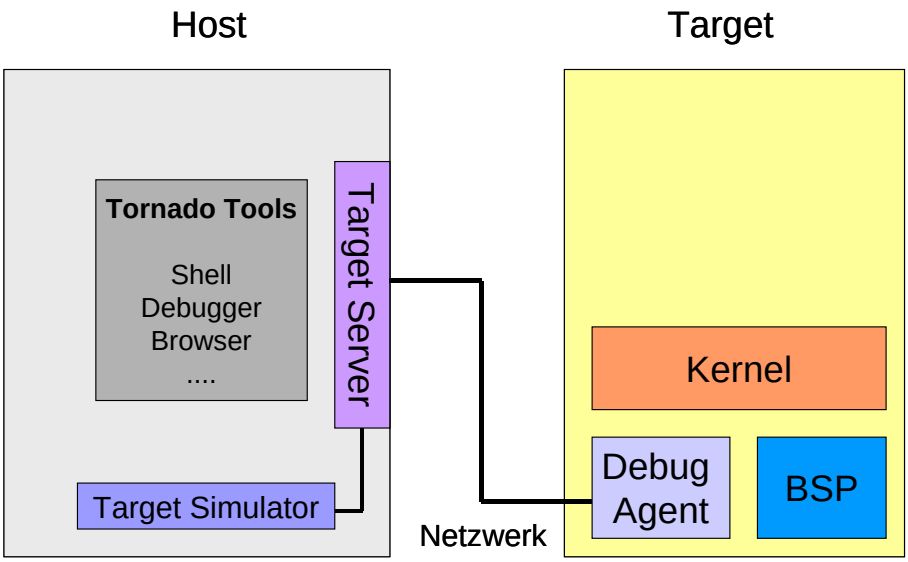


Abb.1 Grobüberblick über die Systemstruktur

Ein Hostrechner kann mit mehreren Targetrechnern verbunden sein. Mit Hilfe von Shells, die auf dem Hostrechner oder den Targetrechnern gestartet werden, kann ein Benutzer interaktiv Einfluss auf die echtzeitkritische VxWorks-Anwendung nehmen. Beispielsweise können durch Kommandos, welche über Shells abgesetzt werden, Anwendungen vom Host auf ein Target übertragen, dort ausgeführt und auch wieder vom Target gelöscht werden. Desweiteren lässt sich auf dem Host ein Targetrechner simulieren. Der so genannte Target-Simulator VxSim findet Verwendung, wenn hardwareunabhängige Teile der Echtzeitanwendung getestet werden sollen.

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

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 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. Workspace

Er ist eine Art logischer und graphischer Container für ein oder mehrere Projekte. Man sollte alle Projekte, die inhaltlich miteinander zu tun haben, in einem Workspace ablegen. Abb.2 zeigt den Workspace *Praktikum* auf dem Rechner atknoll41. Die einzelnen Projekte heißen *Aufzug Builds*, *Boot_Image Builds*, usw. .

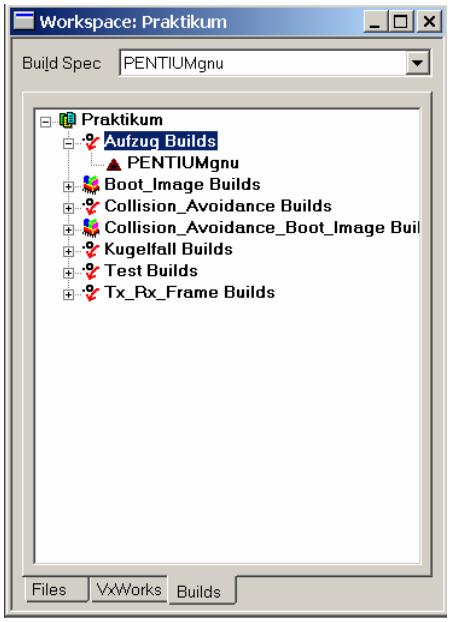


Abb.2 Workspace *Praktikum* auf dem Rechner atknoll41

2.1.2. Projekte

Jede Anwendung für VxWorks wird einem Projekt zugeordnet. Jedes Projekt benötigt sein eigenes Verzeichnis. Dort liegen der Quellcode, der auf mehrere Dateien verteilt sein kann, sowie Informationen zur Verwaltung des Projekts. Um ein neues Projekt ins Leben zu rufen, klickt man auf *File->new Project* und entscheidet anschliessend, ob eine herunterladbare Applikation oder bootbares Image erstellt werden soll (vgl. Abb.3). Im Zusammenhang mit den Praktikumsaufgaben sollten nur herunterladbare Projekte erstellt werden und niemals bootbare Images. Für die weiteren Beschreibungen im Manual wird nicht mehr zwischen Anwendung und Projekt unterschieden.

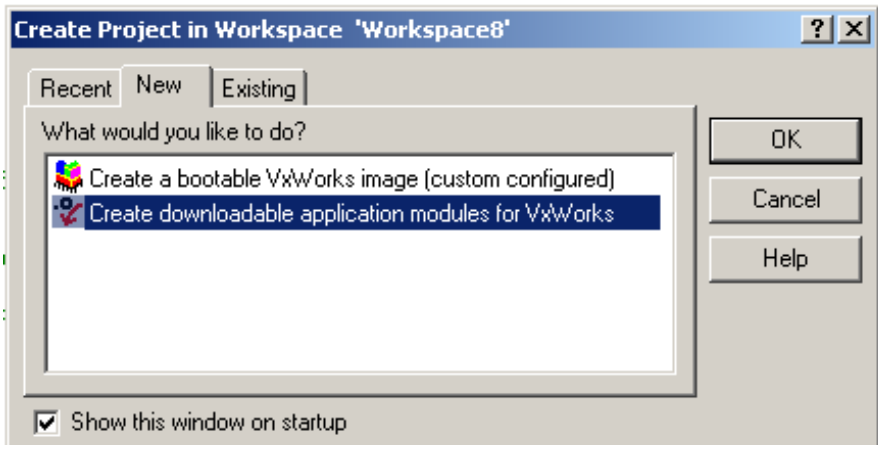


Abb.3 Entscheidung zwischen Boot-Image und Anwendung

Im nächsten Schritt gibt man einen Namen, den Speicherort und optional eine Beschreibung zum Projekts an. Es wird auch festgelegt, zu welchem Workspace das Projekt hinzufügt werden soll. Während des vorherigen Schrittes wurde automatisch ein neuer Workspace erstellt (vgl. Abb.4 – Workspace33.wsp). In diesem Schritt kann das Projekt nun einem anderen bereits existierenden Workspace hinzugefügt werden, indem dieser über die Schaltfläche  selektiert wird.

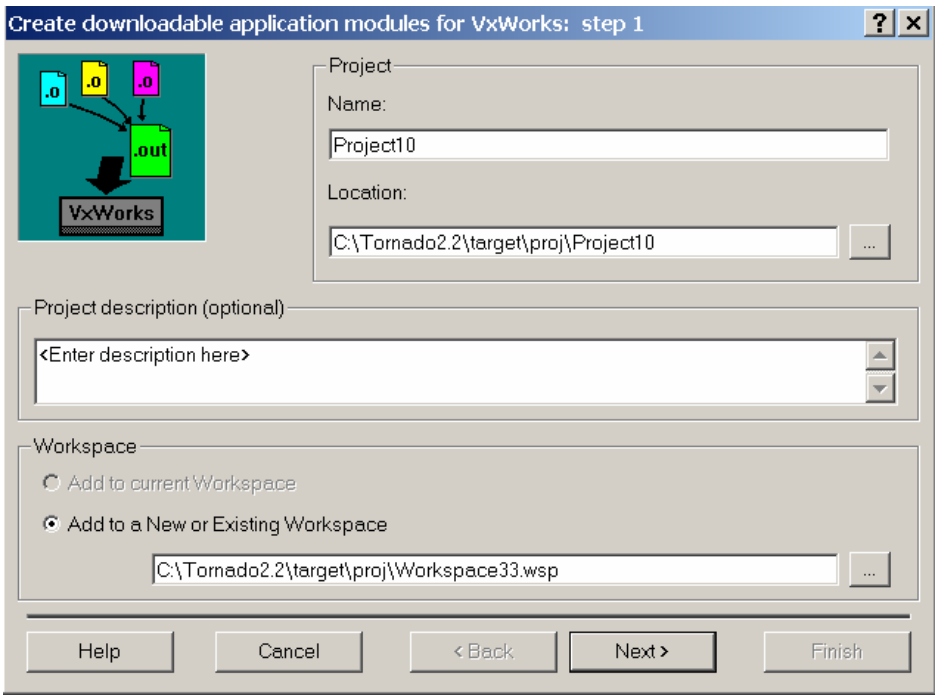


Abb.4 Angabe des Projektnamens und Projekt-Workspaces

Als Nächstes muss eine Toolchain (Erklärung zur Toolchain siehe Abschnitt 2.1.3.) angeben werden. Ein neues Projekt kann auf den bereits erprobten Toolchain-Einstellungen eines älteren Projektes aufbauen. In einem solchen Fall können die Einstellungen vom älteren Projekt einfach übernommen werden. Andererseits kann, wie in Abb.5 zu sehen ist, eine projektunabhängige Toolchain gewählt werden. Im Beispiel wird die projektunabhängige PENTIUMgnu-Toolchain ausgewählt, die im Praktikum für alle Targetrechner eingesetzt werden sollte. Eine andere für das Praktikum nützliche Toolchain ist die Simulator-Toolchain (SIMNTgnu), die ebenfalls projektunabhängig ausgewählt werden kann.

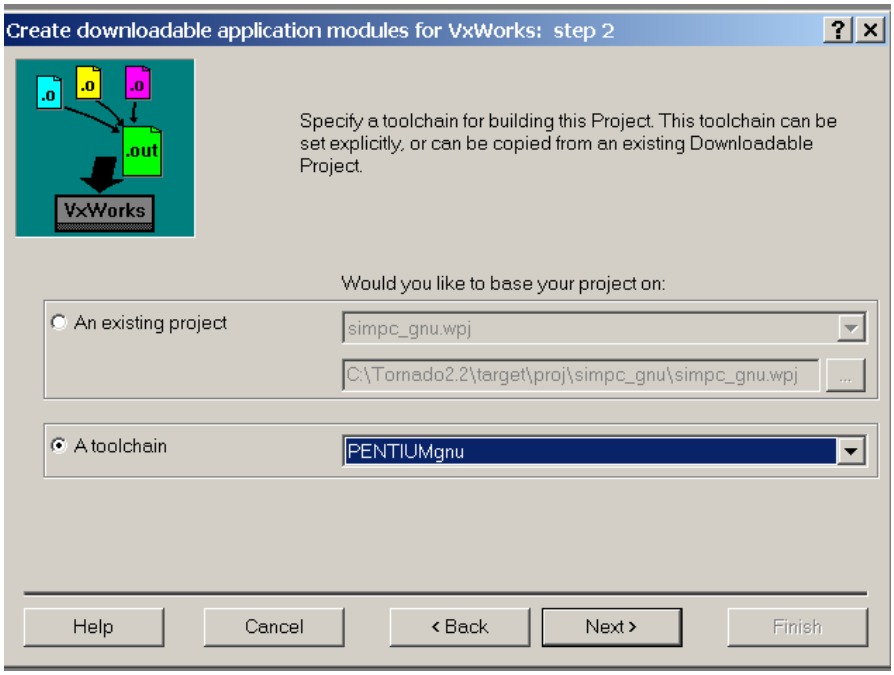


Abb.5 Wahl der Toolchain für ein Projekt

2.1.3. Toolchains für Targets und den Simulator

Eine Toolchain ist eine Sammlung von Werkzeugen. Durch diese Werkzeugsammlung können Anwendungen die spezielle Eigenschaften der Target-Hardware nutzen. Beispielweise werden durch die PENTIUM-Toolchain die speziellen Features eines Pentium-Prozessors zugänglich. Ab Aufgabe 4 sollte allerdings stets die PENTIUMgnu-Toolchain verwendet werden, weil das Boot-Image der Targetrechnern mit ihr kompiliert wurde. Weitere Toolchains wären die PENTIUM3gnu- und die PENTIUM4gnu-Toolchain.
Ein Simulator wird ebenfalls als Target behandelt, obwohl er, wie der Name schon vermuten lässt, lediglich Hardware simuliert. Der Simulator läuft auf jeweiligen Hostrechner. Durch die SIMNTgnu-Toolchain werden einer Anwendung die speziellen Features des Simulators zugänglich gemacht.

2.1.4. Host-Target-Verbindungen, das Starten eines Targetservers und des Simulators

Bevor ein Host mit einem Target verbunden werden kann, müssen die im Abschnitt 2.6. (Anbinden eines Hosts an die zentrale Tornado-Registry) angegebenen Schritte durchgeführt worden sein. Eine Verbindung zu einem Target sollte nur dann hergestellt werden, wenn das Target nicht schon von einer anderen Gruppe benutzt wird, also keine Verbindung mit dem Targetnamen im Kombinationsfeld der IDE auftaucht. In Abb.6 wird eine Verbindung zum Target Kugelfall hergestellt. Durch Auswahl des Menüpunktes *Tools->TargetServer* öffnet sich ein Menü, das jedem im Praktikumsraum vorhandenen Targetrechner enthält.

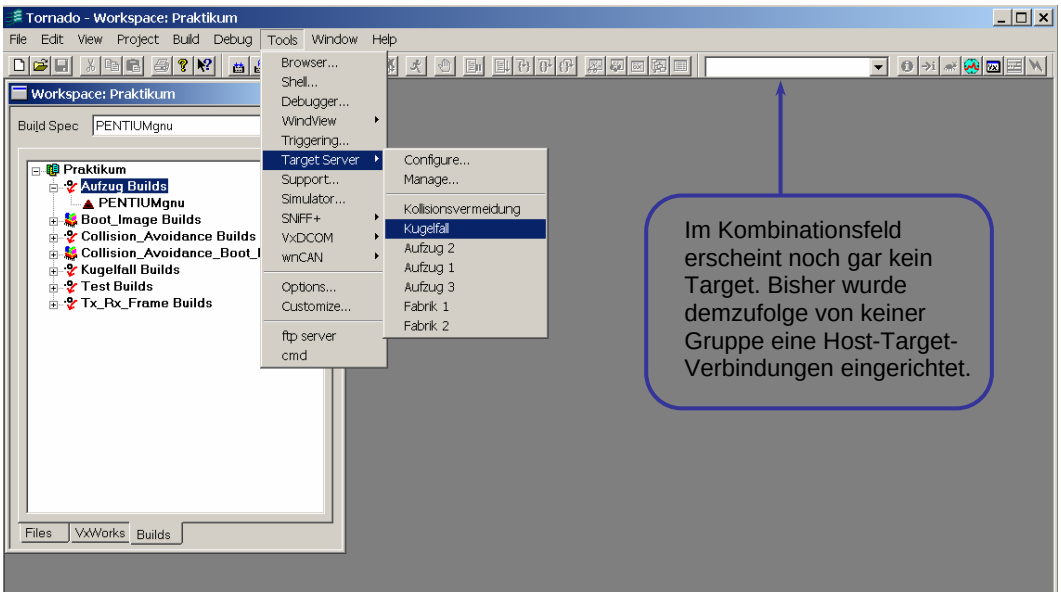


Abb.6 Verbindungsaufbau zum Target Kugelfall

Nachdem das Target Kugelfall mit der linken Maustaste bestätigt wurde, wird auf dem Hostrechner ein sogenannter Targetserver gestartet, über den die Verbindung zum Target kontrolliert wird. Einen laufenden Targetserver erkennt man an einer kleinen roten Zielscheibe rechts unten in der Windows-Symbolleiste.

Ein Host kann mit mehreren Targets verbunden sein, aber verwendet in der Regel nur einen Simulator, der auf dem Hostrechner gestartet wurde. Bei Betrieb des Simulators kann keine target-spezifische Hardware verwendet werden. Der Simulator ist nützlich, wenn spezielle Zielhardware noch nicht zur Verfügung steht und trotzdem erste hardwareunabhängige Anwendungs-komponenten getestet werden sollen. Im Praktikum wird der Simulator für das Erlernen der ersten Schritte mit Tornado und VxWorks eingesetzt. Die Aufgaben 0 bis 3 können aber genauso auf einem Target zum Laufen gebracht werden.
Der Simulators wird durch Klicken auf das Symbol gestartet. Es öffnet sich ein Fenster wie in Abb.7.

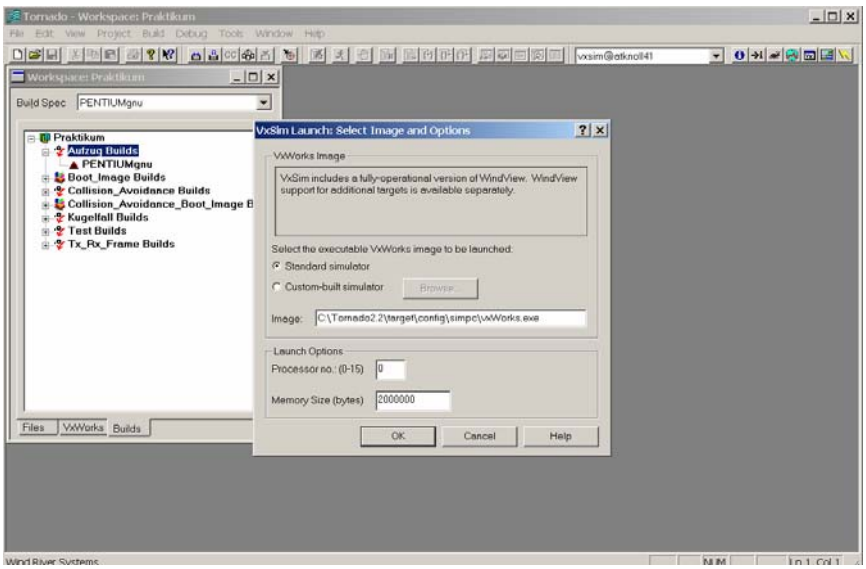


Abb.7 erster Dialog beim Starten des Simulators

Das Fenster wird mit OK bestätigt; ebenso das folgende Fenster von Abb.8.

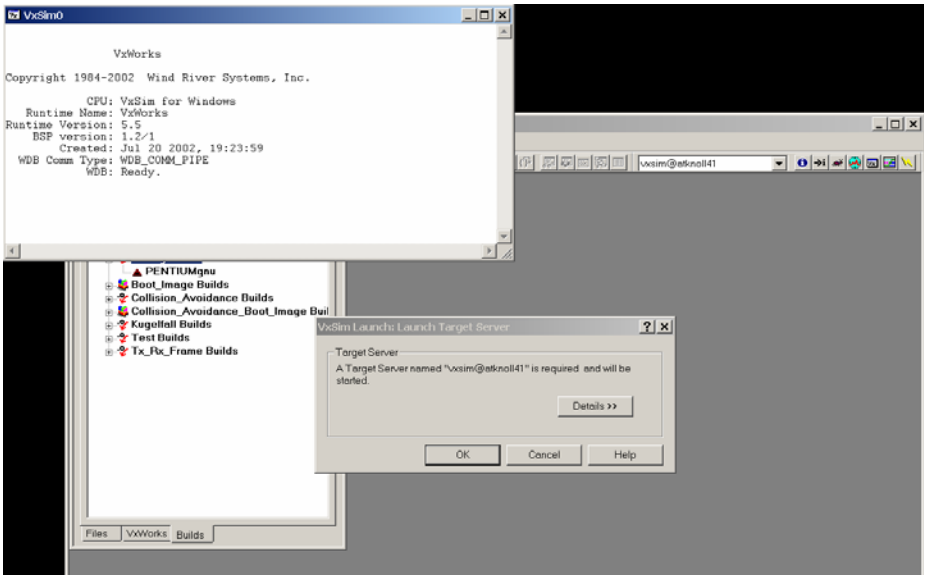


Abb.8 zweiter Dialog beim Starten des Simulators

In Abb.8 ist links oben bereits das Simulatorfenster zu sehen. Ein gestarteter Simulator besitzt ebenfalls einen Targetserver (!), der die Verbindung zwischen Tornado und Simulator kontrolliert. Zu erkennen ist er ebenfalls an einer kleinen roten Zielscheibe recht unten in der Windows-Symbolleiste.

2.1.5. Downloadable Applications

Nach einem Kalt- oder Warmstart fordern alle Targetrechner selbständig ein Boot-Image vom Rechner atknoll41 an. Nachdem das Boot-Image auf die Targets transferiert wurde, kann eine (herunterladbare) Anwendung ins Image integriert werden. Auf den Targets liegt dann quasi ein großes Image, inklusive der heruntergeladenen Anwendung. Die Anwendung lässt sich nun durch Eingabe der Anwendungsstartroutine in eine Shell oder über den Debugger starten.

Hinweise zum Erstellen, Kompilieren, Herunterladen und Starten einer Anwendung

- Dateien einer Anwendung hinzufügen:
  Um eine neue Datei zu erstellen, klickt man auf *File->new*, wählt den Typ der Datei, den Namen des Projektes, zu dem 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 im sich öffnenden Menü den Punkt *add Files*. Dann kann die hinzuzufügende Datei ausgewählt werden.

- Das build- Kommando ausführen
  Im Menü *build* den Menüpunkt *build* oder *rebuild all* wählen oder nach Rechtsklick auf das Projekt den Eintrag *Build All* oder *ReBuild All* auswählen. Optionen für Builds sind nach Auswahl der Registerkarte *Builds* (Registerfeld unten Mitte im Workspace-Fenster) einstellbar. Beispielsweise kann man ein Projekt, dass letztendlich auf einem Pentium-Target laufen soll, zu Testzwecken für den Simulator kompilieren lassen. Dazu wählt man nach Rechtsklick auf das entsprechende Projekt im Registerfeld *Builds* den Eintrag *new Build* und selektiert beim Label *Default Build Spec for to* den Simulator (SIMNTgnu). Anschließend muss das Projekt noch neu kompiliert werden.

- Objektmodule auf ein Target herunterladen
  Um eine Anwendung auf ein Target zu laden, muss mindestens ein Targetserver oder der Simulator ausgewählt sein.

  Erkennbar ist eine solche Auswahl an einem blauem Hintergrund im Kombinationsfeld der Tornado IDE. Eine Möglichkeit, um eine kompilierte Anwendung auf ein Target zu laden, ist nach Rechts-Klick auf das herunterzuladende Projekt den Menüpunkt *download ProjektName.out* auszuwählen. Der Download-Vorgang startet. Nach erfolgreichem Download wird keine Rückmeldung ausgegeben.

- Eine Anwendung über die Shell oder den Debugger starten
  Eine auf ein Target oder den Simulator geladene Anwendung lässt sich über die Shell (Öffnen einer Shell zum aktiven Target durch Klick auf das Symbol ) oder mit dem Debugger (Öffnen durch Klick auf das Symbol ) und Eingabe der Haupt- bzw. Startroutine aufrufen. Die Startroutine muss nicht zwangsläufig 'main' heißen, sondern kann einen beliebigen Namen tragen. Um Namenskonflikte zu vermeiden, sollten Startroutinen verschiedener Anwendungen unterschiedliche Namen tragen, denn beide Anwendungen können zur gleichen Zeit auf demselben Target vorhanden sein (z.B. bei wechselseitiger Ausführung der Anwendungen).

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

2.2. Shells

Wenn ein Target Server läuft, kann eine Host-Shell durch Klick auf das Symbol aufgerufen werden. Es existieren neben Host-Shells auch Target-Shells. Eine Target-Shell wird automatisch beim Verbinden zu einem Target geöffnet. Es wird empfohlen, jegliche Shell-Kommunikation über die Host-Shell abzuwickeln, weil deren Funktionsumfang größer ist und das Target von unnötigen Aufgaben entlastet wird. Shells bieten Zugang zu vielen Funktionalitäten des Betriebssystems. So können über sie global deklarierten Routinen aufgerufen werden, man kann sie zum Testen und Debuggen eingesetzen und natürlich ist der gesamte Funktionsumfang von VxWorks über Shells im Textmodus aufrufbar. Über Shells lassen sich Tasks absetzen (spawn) oder es lässt sich ein Neustart des Targets (reboot) initiieren, womit immer auch ein unberührtes Image (ohne Anwendungsdaten) auf das Target kommt.

Die Shell liest einen Inputstream zeilenweise, parst jede Zeile, wertet sie aus und schickt das Ergebnis an einen Outputstream. Ein besonderes Highlight der Shell-Kommunikation ist, dass interaktiv Zeilen in C/C++ Syntax ausgewertet werden können. Bis auf wenige Abweichungen wird dieselbe Syntax, wie sie von einem C/C++-Compiler her bekannt ist, akzeptiert. Das System verarbeitet diese Anweisungen exakt zu dem gleichen Ergebnis, dass mit einer Anwendung, die ins Boot-Images integriert wurde, erreicht wird. Die Möglichkeit, Anweisungen interpretativ zu testen, erspart das Kompilieren und Herunterladen.

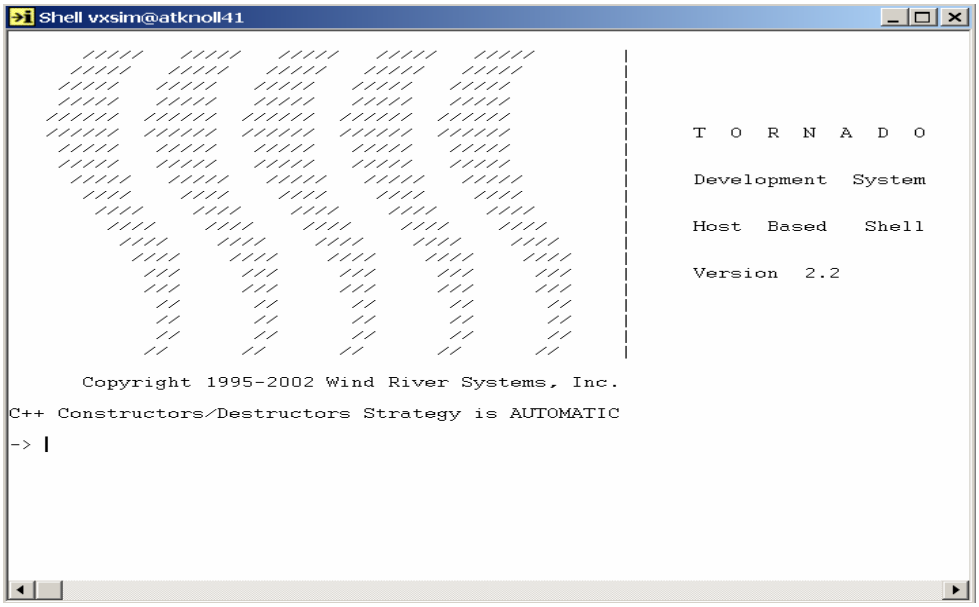


Abb.9 Host-Shell für die Kommunikation mit dem Simulator

2.3 Der Debugger

Nachdem der Debugger durch Klick auf das Symbol ausgewählt wurde, sind weitere Schritte zum Starten der Anwendung durchzuführen. Aus dem Menü *Debug* muss der Menüpunkt *Run* ausgewählt werden. In das sich öffnenden Fester *Run Task*, dass in Abb.10 zu sehen ist, müssen die Startroutine der Anwendung sowie eventuelle Übergabeparameter an die Anwendung eingegeben werden.

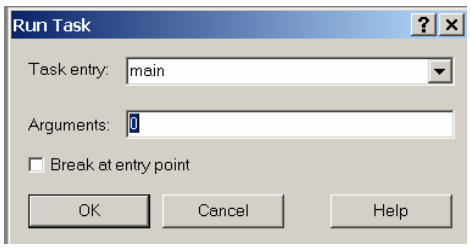


Abb.10 Angabe der Startroutine und der Übergabeparameter beim Debugging

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. Eine Anwendung lässt sich auch im Single-Step-Modus ausführen, z.B. mit der Taste F11. Der Debugger enthält noch eine Menge weitere Funktionen, die im Tornado User's Guide Kapitel 7 dokumentiert sind.

2.4. Browser

Mit Hilfe des Browsers kann man den Zustand eines Target überwachen. Mit dem Browser sind Informationen, die über eine Shells im Textmodus erhalten werden, graphisch darstellbar. Im Hauptfenster werden bsw. aktive Tasks und der Speicherverbrauch aufgelistet. Man erhält Informationen zu

- Taskprioritäten und Registernutzung,
- Semaphoren,
- Message-Queues,
- Watchdog Timern,
- Nutzung der Taskstacks,
- Nutzung der Target-CPU durch Anwendungstasks.

Der Browser lässt sich, nachdem ein Target ausgewählt wurde, mit Klick auf oder durch die Menüauswahl *Tools->Browser* starten. Alle angezeigten Informationen sind Snapshots. Sie können interaktiv durch Klick auf ! aktualisiert werden. Abb. 11 zeigt das Browerfenster, dass sich nach Aktivierung des Browsers öffnet.

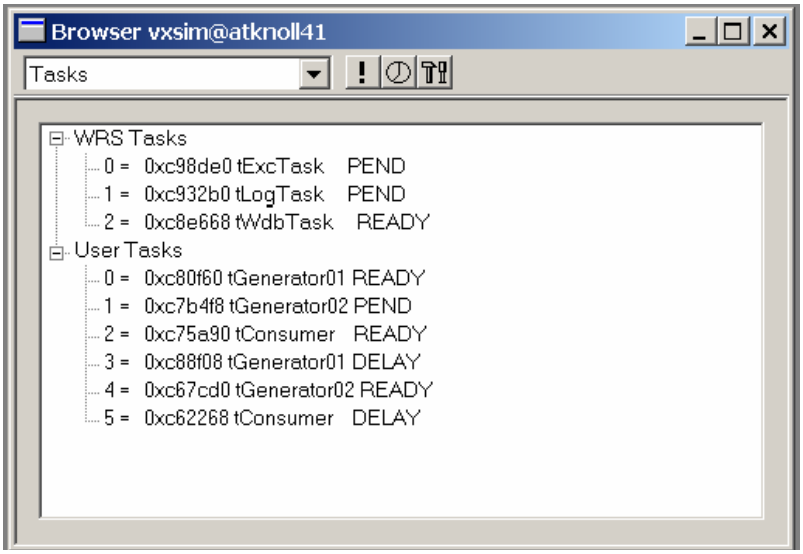


Abb.11 Das Browserfenster

2.5. WindView

WindView ist ein dynamisches Visualisierungstool, mit dem man beispielsweise Informationen über Ereignisse, die zu Kontextwechseln führten, übersichtlich darstellen kann. Mit Hilfe vom WindView und sogenannten „instrumented objects“ kann das Geben und Nehmen von Semaphoren oder das Senden und Empfangen mit Messagequeues benutzerfreundlich beobachtet werden. 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  .

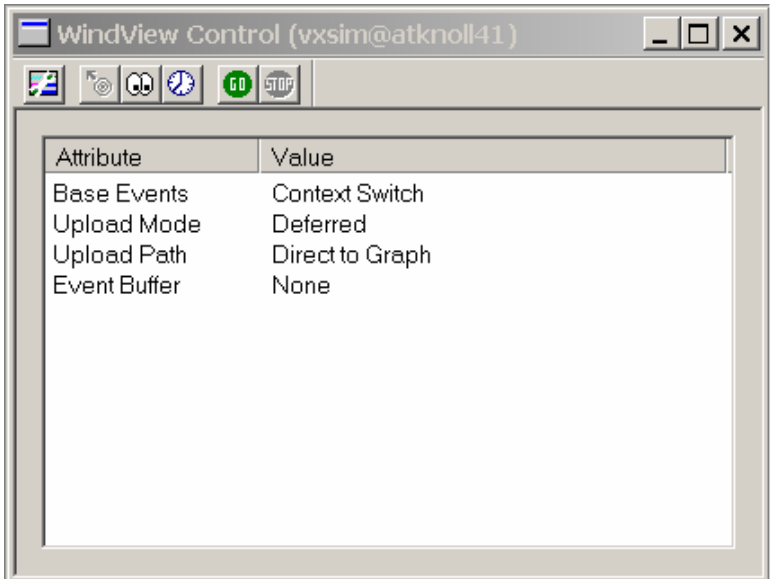


Abb.12 Das WindView-Fenster

Mit Klick auf  startet man das eine 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-Buttor  benutzen, um die Daten auf den Host zu laden. Über die Zoom-Buttons  lässt sich der Auszug dann vergrößern oder verkleinern.

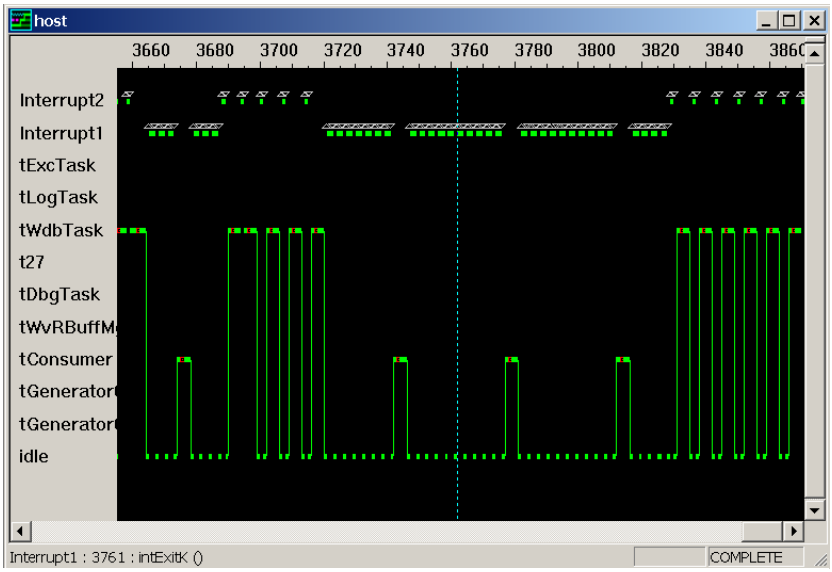


Abb.13 Beispiel für WindView

2.6. Zentrale Tornado-Registry (Targetrechner-Zugriffsverwaltung)

Spätestens ab Aufgabe 4 muss eine Verbindung zu einem der Targetrechner hergestellt werden. Führen Sie dazu im Vorfeld die unter 2.6.2. aufgelisteten Schritte durch oder bitten Sie die Praktikumsaufsicht, die nötigen Einrichtungsschritte für Sie durchzuführen.

2.6.1. Nutzung der zentralen Tornado-Registry

Um auszuschließen, dass von mehreren Hostrechnern zur gleichen Zeit auf einen gemeinsamen Targetrechner zugegriffen werden kann, findet eine zentrale Zugriffskontrolle Verwendung. Der zentrale Praktikumsrechner atknoll41 vermittelt zwischen Hosts und Targets. Eine gültige Host-Target-Verbindung wird nur aufgebaut, wenn der Rechner atknoll41 in seiner Tornado-Registry keinen Host vorfindet, der das gewünschte Target bereits benutzt. Erst nachdem ein Host seine Verbindung zum Target gelöst hat, kann ein anderer Host sich zu diesem Target verbinden.

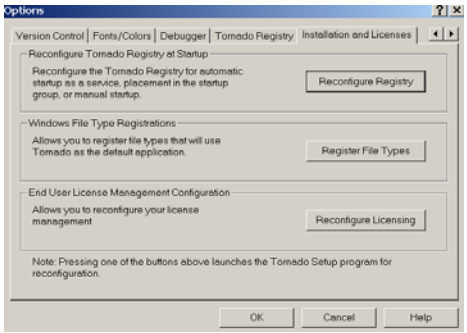
Der Benutzer erkennt eine bereits bestehende Verbindung mit Hilfe eines Kombinationsfelds der Tornado-IDE. Im Kombinationsfeld der eingeblendeten Standardsymbolleiste erscheint jede Verbindung mit Hostnamen gefolgt von einem @ und dem Targetnamen (zwei Beispiele: atknoll67@Kugelfall, atknoll68@Aufzug2). Falls ein weiterer Benutzer versucht sich zu einem belegten Target zu verbinden, wird er in der sich automatisch öffnenden Target-Shell keinen Prompt sehen. Die Targetverbindung ist damit nicht nutzbar und muss wieder beendet werden. Hierzu kann der betreffende Targetserver über sein Symbol in der unteren Bildschirmleiste (rechts) ausgewählt werden. Allgemein sollte man folgendes beachten: Verbindungsaufbau nur, wenn keine Gruppe das Target bereits belegt! Es ist möglich, jede bestehende Verbindung aus der Standardsymbolleiste auszuwählen und dafür eine Hostshell zu öffnen. Hierüber können dann leider auch Kommandos zu Targets anderer Gruppen abgesetzt werden. Öffnen Sie deshalb bitte nur Hostshells zu "Ihrem" Target!

2.6.2. Anbinden eines Hosts an die zentrale Tornado-Registry

Die Zugriffskontrolle auf Targets funktioniert gruppen- und rechnerabhängig. Das bedeutet, beim Benutzen eines neuen Hostrechners, müssen die Einrichtungsschritte erneut durchgeführt werden. Eine einmal für eine Gruppe vollzogene Einrichtung an einem Rechner bleibt dauerhaft erhalten und kann jederzeit wieder genutzt werden. Folgende Schritte sind durchzuführen:

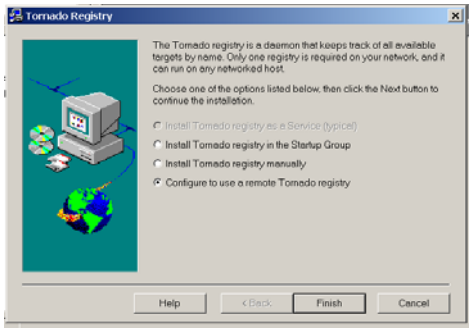
1. Tornado starten,

2. Menüpunkt: Tools->Options->Installation_and_Licenses auswählen,



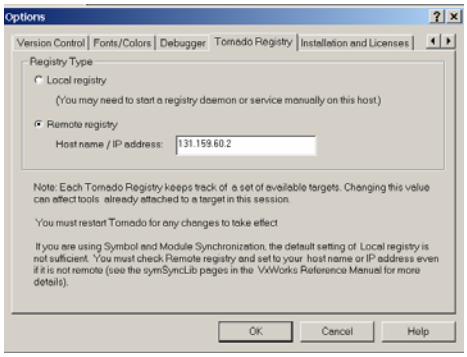
3. Button *Reconfigure_Registry* auswählen,

4. die Task *Setup* über die untere Bildschirmleiste fokussieren und anschließend auf *Next* klicken,



5. Option-Button *Configure to use a remote Tornado* registry aktivieren und auf *Finish* klicken,

6. Menüpunkt: Tools->Options->Tornado_Registry auswählen,



7. Option-Button *Remote registry* auswählen, in das Feld *Host name / IP address* 131.159.60.2 eintragen und mit *OK* bestätigen,

8. Tornado schließen,

9. aus dem Praktikumsverzeichnis auf atknoll41 die Datei *Targets.reg* aufrufen und die nächste Bildschirmmeldungen bestätigen,

10. Tornado erneut starten.

Nach erfolgreichem Einrichten der zentralen Tornado-Registy werden alle bestehenden

Host-Target-Verbindungen im Kombinationsfeld 

der Standardsymbolleiste der Tornado-IDE angezeigt.

2.7. Die Praktikumsaufgaben

2.7.1. 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

2.7.2. Erfolgreiche Teilnahme am Praktikum

Folgende Leistungen sind Voraussetzung für den Erwerb des Praktikumscheins:

a) Die Aufgaben 0, 1, 2, 3 und 4 müssen bearbeitet werden.

b) Eine Lösung 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** ist nötig.

d) Die Lösung zu Aufgabe 5, bzw. 6, ist vorzuführen.

e) Am Ende des Praktikums findet gruppenweise ein **mündliches Kolloquium** statt.

f) **Eine Woche vor Praktikumsende ist pro Gruppe eine Ausarbeitung abzugeben**, die für jede der bearbeiteten Aufgaben ein auf DIN A4-Format zugeschnittenes Programm-Listing und eine Dokumentation der einzelnen Programme enthält.

2.8. Aufgabe 0: Einführung in C unter VxWorks

2.8.1. Ziel

C ist eine weit verbreitete Programmiersprache, die sich durch spezielle Bibliotheken nahezu beliebig erweitern lässt. Die erste Aufgabe soll Ihnen helfen sich in C einzuarbeiten und soll Ihnen einige Möglichkeiten und Besonderheiten bei die C-Programmierung unter VxWorks nahebringen. Verwendung findet eine GNU-C-Version.

2.8.2. Aufgabe

Verbessern Sie das fehlerhafte C-Programm *aufgabe0.c*, das sich im Praktikumsverzeichnis des Rechners atknoll41 befindet. Arbeiten Sie mit einer Kopie des Programms *aufgabe0.c* in Ihrem Home-Verzeichnis. Führen Sie das korrigierte Programm am Simulator sowie am Targetrechner aus. Die Aufrufreihenfolge von Tasks hängt unter anderem von deren Priorität ab. Betrachten Sie deshalb bei den Aufgaben das Zusammenspiel der Tasks auch unter Berücksichtigung der jeweils vergebenen Prioritäten!

2.8.3. Implementierungshinweise

Empfehlenswert ist die Lektüre von Kernighans & Ritchies „The C Programming Language“. Besonderheiten für die C-Programmierung in Verbindung mit VxWorks sind, dass:

- spezielle Libraries immer eingebunden sein müssen (vxWorks.h und taskLib.h),
- die Startroutine nicht notwendigerweise *main* heißt,
- Anwendungs-Routinen von Shells aus aufrufbar sind (Voraussetzung: Routinen sind global und nicht mit Schlüsselwort LOCAL deklariert),
- der doppelte Slash // nicht als Kommentarzeichen erkannt wird.

Wie bei ANSI-C üblich, sind innerhalb eines Blockes keine Deklarationen nach Anweisungen erlaubt.

```
#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");
}
```

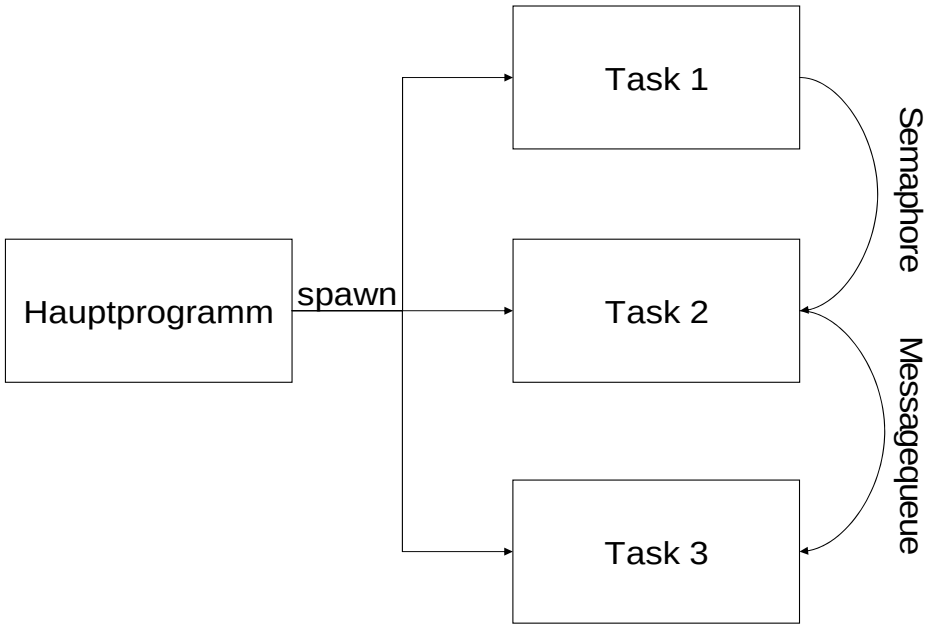
2.9. Aufgabe 1: Multitasking und Interprozesskommunikation

2.9.1. Ziel

Für Programme, die Realzeitbedingungen einhalten sollen, muss der Zeitbedarf einzelner Tasks bekannt sein. Außerdem muss man über den Datenaustausch zwischen (pseudo-)parallel ablaufenden Tasks Bescheid wissen. In Aufgabe 1 sollen grundlegende Methoden der Intertask-Kommunikation angewandt werden.

2.9.2. Aufgabe

Schreiben Sie ein Programm, das drei parallel ablaufende Tasks startet. Während die erste Task sofort mit der Ausführung – ggf. kurzes Delay – beginnt, wartet die zweite Task auf einen Semaphor, der von Task 1 freigegeben wird. Task 3 wartet solange, bis sie per Messagequeue von Task 2 die Nachricht “Task 3 start” erhält. Um den Programmablauf besser nachzuvollziehen zu können, ist es sinnvoll bestimmte Ausführungsetappen mit einer Meldung abzuschließen. Eine solche Meldung kann durch eine printf()-Anweisung initiiert werden.



2.9.2. Implementierungshinweise

In der Lösung zu Aufgabe 1 sollen drei Tasks, ein Semaphor und eine Messagequeue erzeugt werden. Achten Sie auf das zeitliche Zusammenspiel der Prozesse. Eine Warnung vorweg: für einige der im folgenden genannten Routinen existieren auch POSIX-Standards. Mit ihnen sind aus Kompatiblitätsgründen spezielle VxWorks-Features nicht nutzbar. Benutzen Sie deshalb keine POSIX-Definitionen, sondern nur die nicht einschränkenden VxWorks-Routinen!

Das Erzeugen und Starten einer neuen Task:

```
Int taskSpawn(
      Name,
      Priorität,
      Optionen,
      Stack,
      Funktion,
      Arg1,...,Arg10)
```

Rückgabewert ist die ID der erzeugten Task
Name der Task; ASCII-String; frei wählbar
Priorität der Task; Integer;
Es sind die Werte 0 (höchste Priorität) bis 255 zulässig.
Nur Werte zwischen 180 und 255 berwenden, weil höhere Prioritäten häufig für Systemprozesse genutzt werden.
Eigenschaften der Task; Word; meist 0x00
Größe des Stacks; Integer; Defaultwert ist 20000
Zeiger auf aufzurufende Funktion; FUNCPTR
10 Übergabeparameter an die Funktion; Integer

Pausieren einer Task:

```
STATUS taskDelay(
      Ticks)
```

Rückgabewert ist OK oder ERROR
Ticks (1/100 Sekunden), um welche die Task in der Abarbeitung verzögert wird.
Mit der Funktion sysClkRateGet() kann die Anzahl Ticks pro Sekunde ermittelt werden. sysClkRateGet() sollte immer den Wert 100 zurückliefern. Die kleinste für Anwendungen nutzbare Zeiteinheit ist dann 10ms.

Anhalten einer Task:

```
STATUS taskSuspend(
      ID)
```

Rückgabewert ist OK oder ERROR
ID der Task, die angehalten wird

Fortsetzen einer Task:

```
STATUS taskResume(
      ID)
```

Rückgabewert ist OK oder ERROR
ID der Task, die fortgesetzt wird

Löschen einer Task:

```
STATUS taskDelete(
      ID)
```

Rückgabewert ist OK oder ERROR
ID der Task, die gelöscht wird

VxWorks kennt verschiedene Semaphore: den binären Semaphor, den zählenden Semaphor und Mutex-Semaphore. Semaphore können nach Einbinden der semLib.h Bibliothek benutzt werden. Ein vielseitig einsetzbarer, effizient arbeitender und zudem konzeptionell einfacher Semaphor ist der binäre Semaphor. Ein binärer Semaphor kann mit folgenden Routinen verwaltet werden:

binären Semaphor initialisieren:

```
SEM_ID semBCreate(
      Option,
      Status)
```

Rückgabewert ist die ID (der Bezeichner)
Art der Warteschlange für blockierte Tasks
Anfangswert: SEM_FULL oder SEM_EMPTY

Semaphor löschen:

```
STATUS semDelete(
      ID)
```

Rückgabewert ist OK oder ERROR
ID des zu löschenden Semaphor

Semaphor nehmen (P-Operation):

`STATUS semTake(`	Rückgabewert ist OK oder ERROR
`SEM_ID,`	*Name des Semaphor (von semBCreate)*
`Timeout)`	*Wartezeit in Ticks (1/100 Sekunden); WAIT_FOREVER oder NO_WAIT*

Semaphor geben (V-Operation):

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

Eine weitere Möglichkeit zur Interprozesskommunikation sind Message-Queues. Um Message-Queues nutzen zu können, muss die Bibliothek msgQLib.h eingebunden werden.

Message-Queue initialisieren:

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

Message-Queue löschen:

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

Nachricht senden:

`STATUS msgQSend(`	Rückgabewert ist OK oder ERROR
`ID,`	*Zielqueue der Nachricht; MSG_Q_ID*
`Nachricht,`	*Nachricht; Char[]*
`Länge,`	*Nachrichtenlänge; UInt*
`Timeout,`	*Ticks, die auf Zustellung gewartet wird*
`Priorität)`	*Nachrichtenpriorität: MSG_PRI_NORMAL oder MSG_PRI_URGENT*

Nachricht empfangen:

`Int msgQReceive(`	Rückgabewert ist die Anzahl, der empfangenen Bytes
`ID,`	*Queue, aus der eine Nachricht entnommen wird; MSG_Q_ID*
`Nachricht,`	*Puffer für die Nachricht; Char[]*
`Länge,`	*Nachrichtenlänge; UInt*
`Timeout)`	*Ticks, die auf eine Nachricht aus der Queue gewartet wird (0 => kein Warten, -1 => ewiges Warten)*

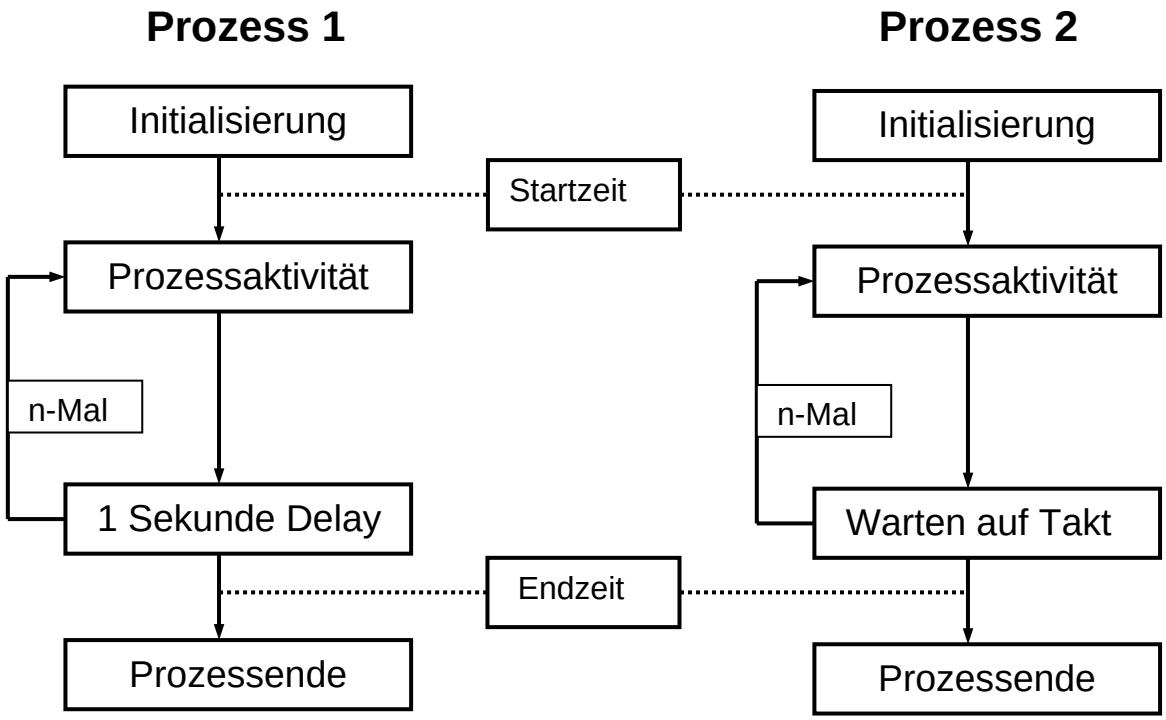
2.10. Aufgabe 2: zyklisch-ablaufende Prozesse

2.10.1. Ziel

Mit dieser Aufgabe soll das unterschiedliche Zeitverhalten zyklisch-aktivierter Prozesse verdeutlicht werden.

2.10.2. Aufgabe

Programmieren Sie je einen zyklisch-aktivierten Prozess nach dem im Bild angegebenen Muster. Der erste Prozess soll nach der Prozessaktivität eine Sekunde warten und dann erneut starten. Der zweite Prozess soll im Takt von einer Sekunde neu gestartet werden. Geben Sie nach jeder Prozessaktivität die insgesamt verbrauchte Zeit aus. Summieren Sie dazu die Zeitdifferenzen zwischen den Durchläufen auf. Somit stellt die Endzeit einen Vergleichswert für den Zeitbedarf der Prozesse dar.
Für die Erzeugung eines periodischen Takts empfiehlt sich die Verwendung eines Watchdog Timer. Schreiben Sie einen Timer-Listener, der bei Auslösen des Timers einen Semaphor freigibt, auf den Prozess 2 gewartet hat, und zudem den Timer neu startet.
Um ein unverfälschtes Ergebnis zu erhalten, müssen die Prozesse sequentiell ablaufen! Fügen Sie der Aufgabendokumentation eine Interpretation des unterschiedlichen Zeitverhaltens der beiden Prozesse bei.



2.10.3. Implementierungshinweise

Lassen Sie während der Prozessaktivität die Berechnung von y = sin(x) ausführen. Der initiale Wert von x soll gleich 1 sein. Führen Sie diese Berechnung N-mal durch, wobei N bei 100000 startet und pro Durchlauf um 100000 erhöht werden soll. Die Zahl n der Durchläufe sei 10.

```
double x; x = 1.0;
for (index = 0; index <= N; index++) { x = sin(x); }
```

Da der Sinus in C nicht verfügbar ist, muss die Bibliothek math.h in das Programm eingebunden werden. Ähnliches gilt für Prozeduren zur Zeitmessung. Hierfür ist die Bibliothek time.h nötig. Für die Zeitmessung müssen zwei Strukturen vom Typ struct timespec deklariert werden:

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

Um die aktuelle Systemzeit zu erhalten, wird die Prozedur clock_gettime() mit CLOCK_REALTIME als ersten Parameter und der Referenz auf eine der beiden Strukturen als zweiten Parameter aufgerufen:

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

Die Differenz zwischen Stop- und Startzeit lässt sich dann beispielsweise folgendermaßen berechnen.

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

Die Variable elapsed ist vom Typ Double. Mit BILLION ist ein parameterloses Makro gemeint, welches im Beispiel den Wert 1000000000 besitzt.

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-Prozedur 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. Die POSIX-Namensvettern haben nur bedingt etwas mit diesen Konzepten zu tun. Benutzen Sie die VxWorks Varianten!

Watchdog initialisieren:

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

Watchdog starten & einhängen:

`STATUS wdStart(`	Rückgabewert ist OK oder ERROR
`wdID,`	*Bezeichner des Watchdogs, WDOG_ID*
`Delay,`	*Verzögerung in Ticks (10 ms gleich 1 Tick), int*
`Routine,`	*Funktion, die bei Timeout aufgerufen werden soll, FUNCPTR*
`Parameter)`	*Watchdog Parameter, int, meist 0*

Watchdog unterbrechen:

```
STATUS wdCancel(
    wdID)
```

Rückgabewert ist OK oder ERROR
Bezeichner des Watchdogs, WDOG_ID

Watchdog löschen:

```
STATUS wdDelete(
    wdID)
```

Rückgabewert ist OK oder ERROR
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 Timer Listeners sind also somit klar: zum einen den Watchdog neu starten, zum anderen einen Semaphor freigeben, der es erlaubt Task 2 zu starten. Task 2 bei Timeout zu 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).

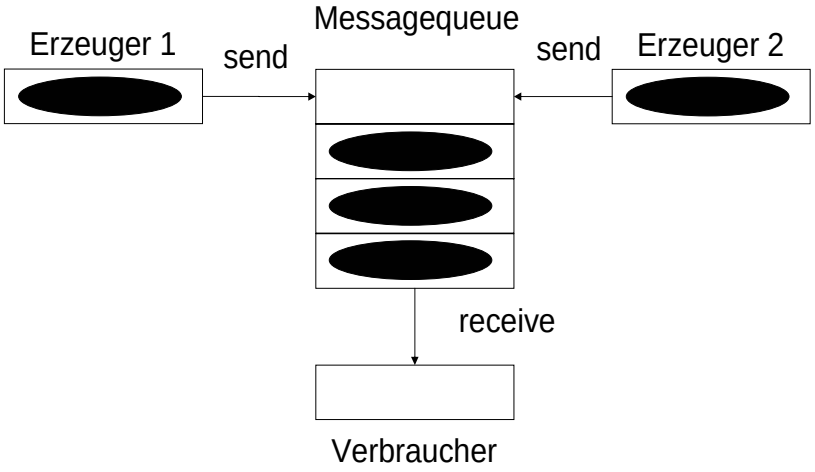
2.11. Aufgabe 3: Erzeuger-Verbraucher-Problem

2.11.1. Ziel

Prozesse und deren Kommunikation sind oftmals sehr komplex. Ähnlich wie zu viele Anfragen einen Webserver “überfordern” können, kann die Kommunikation einen Prozess “überfordern”. Diese Aufgabe soll vermitteln, wie in VxWorks mit Puffern, sogenannten Messagequeues, verfahren wird.

2.11.2. Aufgabe

Starten Sie zwei Erzeugerprozesse mit gleicher Funktionalität. Jeder Prozess soll pro Sekunden je eine Nachricht in eine Messagequeue legen bis schließlich von jedem Prozess insgesamt 10 Nachrichten verschickt wurden. Das Fassungsvermögen der Queue sei auf 4 Einträge beschränkt. Starten Sie einen Verbraucher, der die Nachrichten alle 1,5 Sekunden abruft. Um den Verlauf nachvollziehen zu können, ist es sinnvoll, die diversen Schritte der einzelnen Prozesse auf der Konsole auszugeben.



2.11.3. 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 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)
```

Rückgabewert ist die eigene TaskID.

Den Namen der Task (beim Spawning vergeben) ermitteln:

```
char *taskName(tID)
```

Rückgabewert ist ein Zeiger auf einen String,
der den Namen der Task mit Identifier tID enthält.

Die Nachricht für den Message-Puffer 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);
```

Als Erfolgsmeldung des Verbrauchers geben Sie die empfangene Nachricht aus. Zu guter Letzt beenden Sie bitte den Verbraucher über timeout, wenn keine weiteren Nachrichten mehr in der Queue sind.

Anzahl der Nachrichten in der Queue ermitteln:

```
int msgQNumMsgs(
      msgQID)
```

Rückgabewert ist die Anzahl der Nachrichten
in der Messagequeue.