TOPPERS-ETロボコンセミナー
（MINDSTORMS-NXT:TOPPERS/JSP版）

TOPPERSプロジェクト
教育ワーキング・グループ

TOPPERS

本教材の利用条件

NEXCESS基礎コース02
組込みソフトウェア開発技術の基礎

Copyright (C) 2006-2007 by 名古屋大学 組込みソフトウェア技術者人材養成プログラム
Copyright (C) 2006-2007 by本田晋也, 高田広章

上記著作権者は，以下の(1)～(4)の条件を満たす場合に限り，本コンテンツ（本コンテンツを改変・翻訳したものを含む．以下同じ）を使用・複製・改変・翻訳・再配布（以下，利用と呼ぶ）することを無償で許諾する．

(1) この枠内の著作権表記等が，そのままの形でコンテンツ中に含まれていること．
(2) 本コンテンツを再配布する場合には，再配布の形態等を，以下のウェブサイトから報告すること．
http://www.nces.is.nagoya-u.ac.jp/NEXCESS/REPORT/
(3) 本コンテンツを改変・翻訳する場合には，コンテンツを改変・翻訳した旨の記述を，コンテンツ中に含めること．また，改変・翻訳者の著作権表記等は，この枠内の著作権表記等とは別に行うこと．
(4) 本コンテンツの利用により直接的または間接的に生じるいかなる損害からも，上記著作権者を免責すること．

※ 本コンテンツの一部は，文部科学省 科学技術振興調整費により，名古屋大学 組込みソフトウェア技術者人材養成プログラム（NEXCESS）の一環として作成しました．
※ 本コンテンツ中に記載されている商品名やサービス名などは，各社の商標または登録商標です．

TOPPERS

本ドキュメントに関して

1. 著作権に関しての表記

〈TOPPERS-ETロボコンセミナー(MINDSTORMS-NXT版)〉

Copyright (C) 2006-2009 by 名古屋大学 組込みソフトウェア技術者人材養成プログラム
Copyright (C) 2006-2009 by 本田晋也、高田広章　名古屋大学
Copyright (C) 2008-2010 by 竹内良輔（株)リコー

上記著作権者は、以下の (1)～(3) の条件を満たす場合に限り、本ドキュメント（本ドキュメントを改変したものを含む。以下同じ）を使用・複製・改変・再配布（以下、利用と呼ぶ）することを無償で許諾する。
(1) 本ドキュメントを利用する場合には、上記の著作権表示、この利用条件および以下の無保証規定が、そのままの形でドキュメント中に含まれていること。
(2) 本ドキュメントを改変する場合には、ドキュメントを改変した旨の記述を、改変後のドキュメント中に含めること。ただし、改変後のドキュメントがTOPPERSプロジェクト指定の開発成果物である場合には、この限りではない。
(3) 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも、上記著作権者およびTOPPERSプロジェクトを免責すること。また、本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも、上記著作権者およびTOPPERSプロジェクトを免責すること。
本ドキュメントは、無保証で提供されているものである。上記著作権者およびTOPPERSプロジェクトは、本ドキュメントに関して、特定の使用目的に対する適合性も含めて、いかなる保障もしない。また、本ドキュメントの利用により直接的または間接的に生じたいかなる損害に関しても、その責任を負わない。

2. 本ドキュメントに関するご意見・ご提言・ご感想・ご質問等がありましたら、TOPPERSプロジェクト事務局までE-Mailにてご連絡ください。

3. 本ドキュメントの内容は、内容の改善や適正化の目的で予告無く改定することがあります。

本ドキュメントでは、Microsoft社のClip Art Galleryコンテンツを使用しています。

TRONは"The Real-time Operating system Nucleus"の略称です。ITRONは"Industrial TRON"の略称です。
μITRONは"Micro Industrial TRON"の略称です。TOPPERS/JSPはToyohashi Open Platform for Embedded Real-Time System/Just Standard Profile Kernelの略称です。」

本ドキュメント中の商品名及び商標名は、各社の商標または登録商標です。

TOPPERS

スケジュール

- 1日目

1. 開発環境とプラットフォーム	1.0時間
2. ITRONの仕様について学ぶ	1.5時間
3. TOPPERS/JSPの導入	1.5時間
4. 走行体サンプルプログラム	1.0時間
5. まとめ	0.5時間

TOPPERS

概要

リアルタイムOSの基礎及び、
TOPPERS/JSPカーネルとプラットフォームを学ぶ

- 開発環境とプラットフォーム
  - NXT用のクロス開発環境について
  - アプリ（モデル）とプラットフォームについて
- ITRON仕様について学ぶ
  - μITRON仕様と歴史
  - TOPPERS/JSPについて
- TOPPERS/JSPの導入
  - SAMPLE1/SAMPLEの導入、Bluetoothについて
- プラットフォームについて
  - デバイスドライバとECROBOTについて

開発環境とプラットフォーム

1. NXT用クロス開発環境
2. 拡張NXTファームウェアとNXTツール
3. アプリ（モデル）とプラットフォーム

クロス開発環境：ターゲットシステムとホストシステム

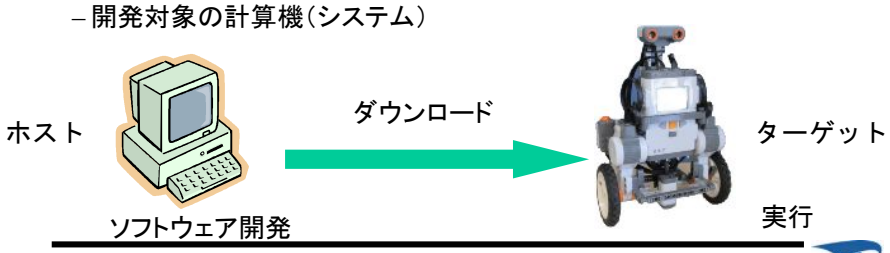
組込みシステムはクロス開発により開発する

クロス開発
- 開発環境（ホスト）と実行環境（ターゲット）が異なる開発形式
- 機械語も異なる

⇔セルフ開発

ホスト
- ソフトウェアを開発する（開発環境を実行する）計算機

ターゲット（ターゲットプロセッサ）NXTも組込みシステム
- 開発対象の計算機（システム）



組込みソフトウェア開発に使われるプログラミング言語

C言語
- ハードウェアを直接操作するプログラミングが可能であるため，組込みソフトウェア開発では，最も使われている

アセンブリ言語
- DSPなどの特殊なプロセッサで使われる場面が多い
- コンパイラが扱えない特殊命令を直接記述して，性能を出す

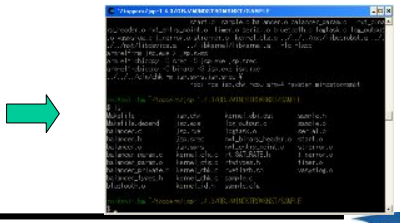
C++言語
- 利用は広がっているが，まだ限定的
- オーバーヘッドが大きい
- どのような実行コードになるか見えにくい

NXT用のプログラムを開発するための開発環境

- NXTはARM7プロセッサが使用されている
  - ARM用コンパイラとアセンブラが必要
  - GNUのARM用コンパイラを使用する
- GNUの実行にはUNIX環境が必要
  - Windows上で動作するUNIX環境：Cygwinを使用する
- NXT用の開発環境を構築するため必要なこと
  - CygwinとARM-GNUコンパイラのインストール
  - Cygwin(UNIX)の開発手法について学ぶ

DOS窓のようなCygwinのコマンド環境

ここでb-shell(bash)が実行できる



TOPPERS

開発環境：GCC

- GNUプロジェクトにより開発されているオープンソースのコンパイラ
- GCCは「GNU Compiler Collection」の略であり、名前が示すように多くの言語（C、C++、Objective-C、FORTRAN、Java、Ada）をサポート
- 多くの種類のプロセッサをサポート
  - Alpha、ARM、AVR、H8、IA64、M32R、M68K、MIPS、SH、SPARC、V850
- GCCはアセンブラやリンカとしてbinutilsを呼び出す
  - アセンブラ(gas)、リンカ(ld)、オブジェクトダンプ（objdump）
- デバッガとしてはGNUプロジェクトより同じくオープンソースのソフトウェアとしてgdbが提供されている

TOPPERS

開発環境 : Cygwin

- 一般的なGNU開発プログラムを含むUNIXのプログラムをWindows上で動作させるための環境
- Cygwinライブラリ（Cygwin.dll）によりUNIXのシステムコールを提供（バーチャルマシンではない）
- ほぼ全てがGPL/X11ライセンスのフリーソフトウェア
- Cygnus Solution社（現在はRed Hat社の一部）が開発
- インストールはCygwinのホームページからダウンロードできるインストーラを用いる
- インストール方法は書籍を参考のこと
  - Cygwin+CygwinJE-Windowsで動かすUNIX、佐藤竜一、アスキー
  - Cygwin—Windowsで使えるUNIX環境、川井 義治、米田 聡、ソフトバンクパブリッシング

TOPPERS

C言語のツールチェーン

C言語コードを実行コードに変換するためのツール郡

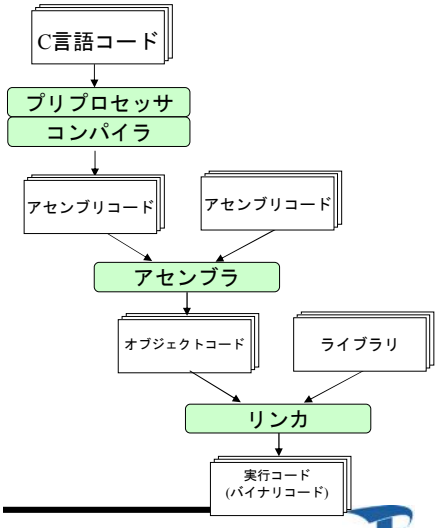
コンパイルドライバ
- 実行コードを生成するまでの一連の処理を実行
  - プリプロセッサ，コンパイラ，アセンブラ，リンカを呼び出す

プリプロセッサ
- #includeやマクロを展開

コンパイラ
- プリプロセスされたC言語コードをアセンブリコードへ変換

アセンブラ
- アセンブリコードをオブジェクトコード(機械語プログラム)に変換

リンカ
- 複数のオブジェクトコードとライブラリをリンクし実行コードを生成する



TOPPERS

Cygwinをインストールしよう1

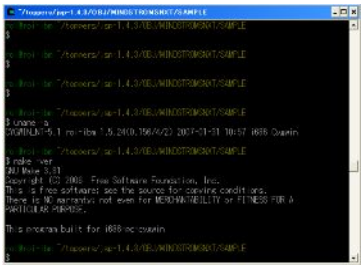
- CygwinのサイトからCygwin 1.5.x以降のバージョンをダウンロードし、Windows-XPまたはWindows-7にインストールします（makeのバージョンは3.81以降をお勧め）

  Cygwinサイト ⟹ http://www/cygwin.com/

  注意 トラブル回避のために、すでにCygwinをインストール済みの方は、バージョンの確認をお願いします

  Bash上で、uname –a(return)がCygwinのバージョン問い合わせ、
  make –ver(return)がmakeのバージョン表示です

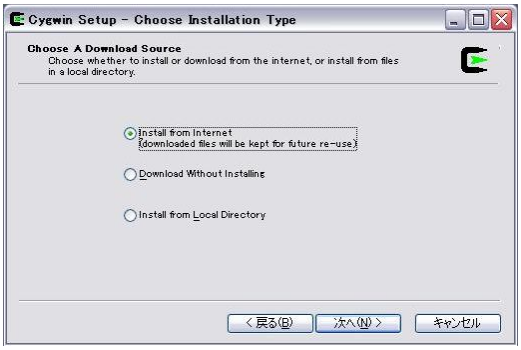
ここをクリック



TOPPERS

Cygwinをインストールしよう2

- setup.exeを起動して、インターネット経由のダウンロードインストールまたはダウンロード後インストールのどちらかを選択できます



TOPPERS

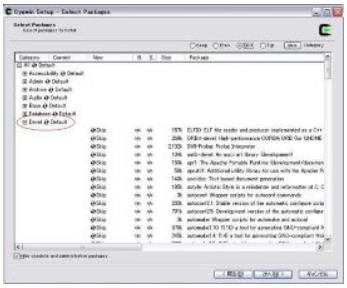
Cygwinをインストールしよう3

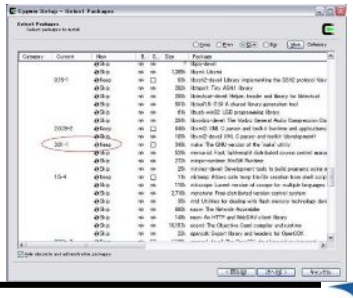
- マルチバイト文字およびスペースを含まないディレクトリにインストールします（例：C:¥cygwin）

TOPPERS

Cygwinをインストールしよう4

- makeのバージョンは3.81-1を選択します
- makeコマンドはmakeファイルの記述に従って、ビルドの手順を指定するコマンドです





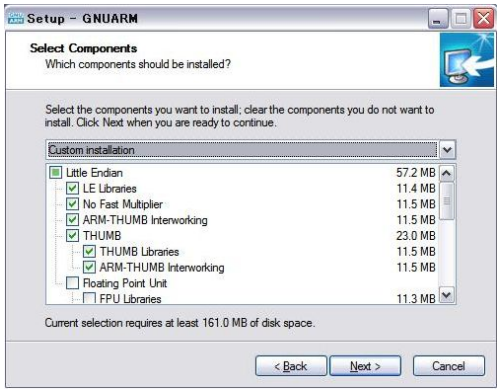
TOPPERS

ARM-GCCをインストールしよう1

- GCCのインストールは、GCCのソースコードをダウンロードして、インストールが可能ですが、手順が複雑なため、ここではET事務局で指定の手順のバイナリインストールを行います
- 指定の(bu-2.16.1 gcc-4.0.2-c-c++ nl-1.14.0 gi-6.4.exe)をダウンロードしてください

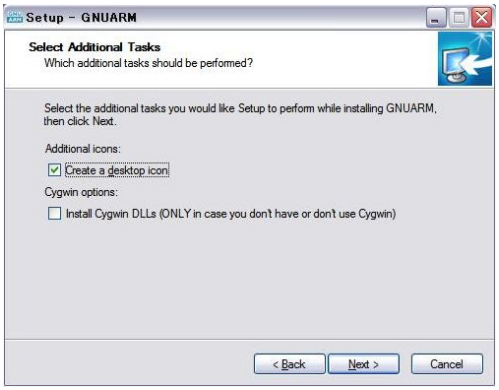
ARM-GCCをインストールしよう2

- インストールディレクトリはUINX風にC:¥cygwin¥usr¥localにインストールします
- NXTで使われているARM7(ATML AT91SAM7S256)にはlittle Endian, Floating Point Unitなし,THUMBコードの対応を行うために以下のダイアルログの設定としてください



ARM-GCCをインストールしよう3

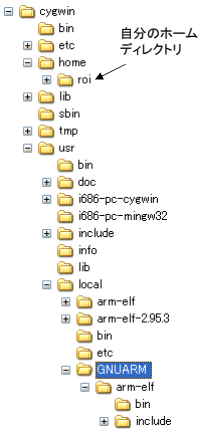
- Cygwinはインストール済みのため、”install Cygwin DLLs…”は選択しないでください
- インストール終了時に、GNU ARMインストールディレクトリに対するWindows環境変数(パス)登録を確認されますが、パスを登録する必要はありません



TOPPERS

ARM-GCCをインストールしよう4

- 自分のHOMEディレクトリの.bash_profileの内容を修正します
- .bash_profile中のexportコマンドにインストールしたARMGCCのコマンドパス(/usr/local/GNUARM/bin)を追加します
- Cygwinの起動後コンパイラ等のコマンドが使用できるようになります
  - arm-elf-gcc コンパイラ
  - arm-elf-ld　リンカ
  - arm-elf-ar　ライブラリアン




```
export PATH=${PATH}:/usr/local/GNUARM/arm-elf/bin
```


TOPPERS

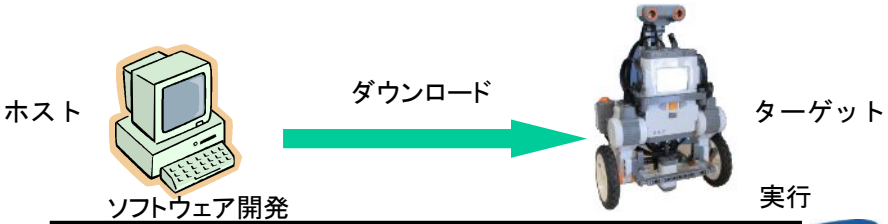
開発環境とプラットフォーム

1. NXT用クロス開発環境
2. 拡張NXTファームウェアとNXTツール
3. アプリ（モデル）とプラットフォーム

TOPPERS

ダウンロードツールのインストール

- MINDSTORMS NXTにプログラムをダウンロードするために パソコンにダウンロードツールを設定する
  - LEGO MINDSTORMS NXT Driver
  - NeXTTool
- MINDSTORMS NXTをダウンロード環境をインストールする
  - 拡張NXTファームウェア
  - NXT BIOS（ここでは説明しない）



TOPPERS

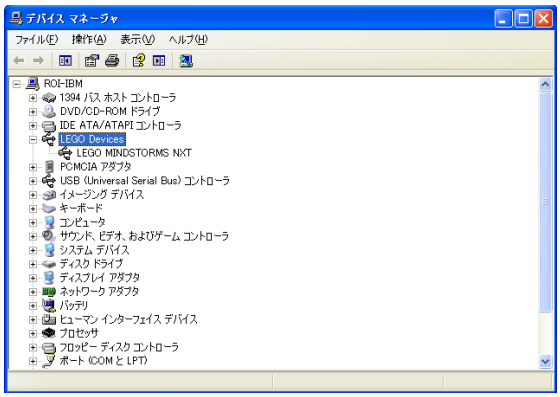
LEGO MINDSTORMS NXT Driverのインストール

- LEGO MINDSTORMS NXT DriverはLEGO社が提供しているNXTとのUSB通信ドライバーです
- LEGO標準のプログラミングソフトウェアをインストール済みの場合は、NXT Driverのインストールは不要です
- LEGO education school → NXT Programming → Software updateよりLEGO MINDSTORMS NXT Driver v1.02をダウンロードし、setup.exeを起動してインストールしてください

TOPPERS

MINDSTORMS NXT Driverの確認

- NXT Driverインストール後、NXTとパソコンをUSBをつなぐとデバイス マネージャにLEGO Driverが表示されることを確認します



TOPPERS

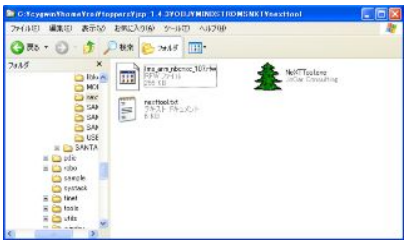
NeXTToolとファームウェアのダウンロード

- 拡張NXTファームウェアをインストールし、実行プログラムをダウンロードするためにNextToolをパソコンにダウンロードします
- NeXTToolはJohn Hansen氏が作成したNXTとの通信プログラムで、実行プログラム（*.rxeファイル）とファームウェア（*.rfwファイル）をNXTに書き込みます
- http://bricxcc.sourceforge.net/nexttool.zip（NextTool）をダウンロードしC:/cygwin/nexttoolに解凍します
- http://bricxcc.sourceforge.net/lms_arm_jch.zipをダウンロード解凍し、バージョン1.07の拡張NXTファームウェア（lms_arm_nbcnxc_107.rfw）をC:/cygwin/nexttoolフォルダにコピーします

TOPPERS

拡張NXTファームウェアのインストール1

- nexttoolフォルダを設定します
- 拡張NXTファームウェアをNXTにアップロードするために、NXTをファームウェアアップデートモードにする必要があります
- NXTの電源ONの状態でハードウェアリセットボタンを安全ピンなどで5秒以上押し続けます
- ファームウェアアップロードモードになるとクリック音が聞こえます



TOPPERS

拡張NXTファームウェアのインストール２

- Cygwinを起動しnexttoolディレクトリに移動します

```
cd /nexttool
```

- USBケーブルをNXTに接続し、NXT前面のオレンジ色のENTERボタンを押します
- 次のコマンドをCygwin上で入力し、拡張NXTファームウェアをNXTにインストールします

```
./NeXTTool.exe /COM=usb –firmware=lms_arm_nbcnxc_107.rfw
```

- インストールが終了すると、NXTの液晶画面が砂嵐画面のようになったり、ボタン操作ができなくなる場合があります。その場合、NXTのバッテリを外し、再び装着すると拡張NXTファームウェアが起動します

拡張NXTファームウェア上でのRTOSの実行

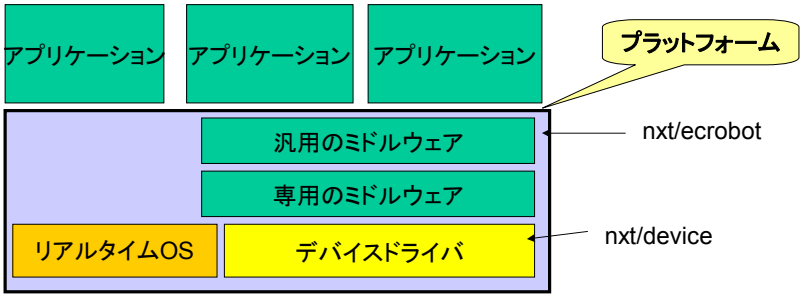
- ARM-GCCでビルドしたプログラムはアプリケーションとしてダウンロードし、プログラムを選択して実行します
- 起動したプログラム（RTOS）は、NXTハードウェアの制御を拡張NXTファームウェアからのっとり実行します
  - そのため、拡張NXTファームウェアの起動前の状態には戻れません
- 実行を抜けるには、電源をオフするか、起動したリスタート（RTOSを起動しなおす）しなおします
  - しかし、リスタートの場合、データ領域（書き換え可能なデータ変数）は初期状態に戻りませんので、プログラムの記載によっては正しくリスタートしない場合がありますので注意が必要です

開発環境とプラットフォーム

1. NXT用クロス開発環境
2. 拡張NXTファームウェアとNXTツール
3. アプリ(モデル)とプラットフォーム

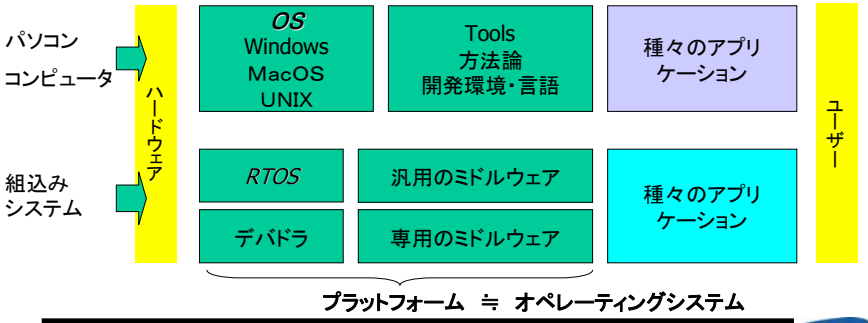
組込みプラットフォーム

- 中規模な組込みシステムでは、アプリケーション(モデル)とプラットフォームを分離して構築します
- プラットフォームは一般的にはいくつかの部品から構成されます
  - リアルタイムOS(リアルタイムカーネル)
  - デバイスドライバ
  - ミドルウェア



パソコンのOSとプラットフォームの違い

- リアルタイムOS（リアルタイムカーネル）は概念レベルではCPUをオブジェクト化したプログラムです。実際の実装では時間と開発環境も実装対象となります
- パソコン用のオペレーティングシステム（OS）はパソコンのもつ全てのハードウェアを抽象化したインターフェイスをアプリケーションに提供します



TOPPERS

プラットフォームを構築する意味

- プラットフォーム化の意味
  - 複雑なハードウェアの制御関数レベルで隠蔽する
  - 共通なハードウェアを同時実行しても、問題なく動作するように安全に排他制御する
  - ソフトウェア部品（ミドルウェア）の導入により、効率的に機能アップが可能

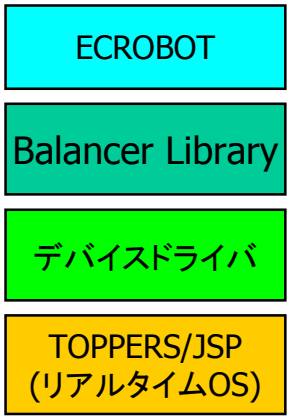
全体としてのメリット

- アプリケーション機能拡張、機能変更を容易にする
- 分業化による生産効率の向上
- 複雑な部分をプラットフォームに隠蔽することにより、アプリケーション（モデル）の簡素化できる

TOPPERS

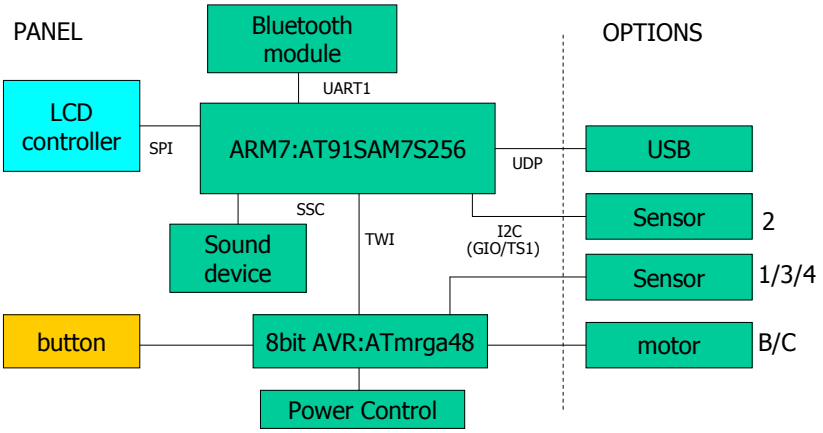
MINDSTORMS NXTのプラットフォーム

- NXTのプラットフォームはECROBOT/Balancer Library/デバイスドライバ/リアルタイムOSで構成される
- ECROBOT
  - アプリケーションへのアーキテクチャの提示
  - 共通メニューの表示
  - 電源、キーの管理
- Balancer Library
  - 移動情報をモータ情報に変換
- デバイスドライバ
  - ハードウェアの管理
- リアルタイムOS（TOPPERS/JSP）
  - CPU、時間のオブジェクト化（マルチタスク機能、タスク間通信等）
  - 開発環境の初期化、デバッグ環境



MINDSTORMS NXTのハードウェア構成

- NXTはARM7にて主制御を行っています（以下は、想像図）



MINDSTORMS NXT用TOPPERS/JSPの実装

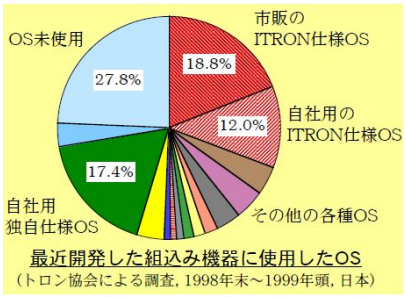
- MINDSTORMS NXT用のTOPPERS/JSPは、バージョン1.4.3のarmv4/at91sam7s（GNU-ARMコンパイラ用）に対して、以下の修正を行い実装しました
  - RXE実行対応（拡張ファームウェア上の実行形式）
  - THUMBコード対応
  - TIMERをTCからPITに変更
  - シリアルデバイスをUARTからBluetoothモジュールに変更
- Bluetoothの初期化後、コネクションを行うためにsystaskディレクトリの以下のファイルを追加修正した
  - bluetooth.c/logtask.c

ITRONの基礎知識

1. ITRON仕様とは
2. リアルタイムOSの基礎
3. ITRON仕様の機能概要
4. TOPPERS/JSPカーネル

ITRON仕様とは

- トロンプロジェクトにおいて標準化してきた組込みシステム向けのリアルタイムカーネルとそれに関連する仕様
- 誰でも自由に実装できるオープンな仕様
  - それに準拠して実装されたOSはオープンとは限らない
- 産学共同の標準化活動の成果
- 多くのプロセッサ上に実装され、国内では業界標準に



最近開発した組込み機器に使用したOS
（トロン協会による調査，1998年末～1999年頭，日本）

トロンプロジェクトとは

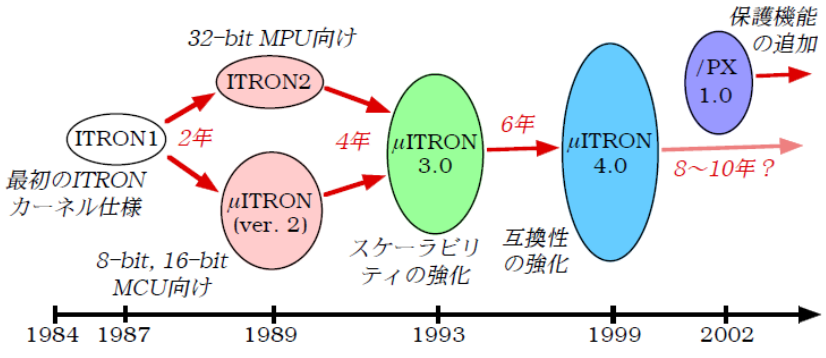
- ユビキタスコンピューティング社会における基盤技術の構築を目指す産学共同のプロジェクト
  - プロジェクトリーダ：坂村健（東京大学教授）
  - プロジェクトの開始：1984年
- “オープンアーキテクチャ”

各方面への取り込み（主なもの）

- ITRON（組込みシステム向けOS）
- BTRON（パソコン/ワークステーション向けOS）
- CTRON（通信/サーバ向けOS）
- トロン仕様マイクロプロセッサ
- トロンHMI（Muman-Machine Interface）
- T-Engine（組込みシステム向け開発プラットフォーム）
- ユビキタスIDとeTRON

ITRONカーネル仕様の歴史と現状

- プロジェクト開始から約20年間に、4世代のITRONカーネル仕様を策定・公開
- μITRON4.0仕様は、現世代のリアルタイムOS技術の範囲では、極めて完成度の高い仕様に



ITRON仕様の特徴

- OSの小型軽量化が可能
  - ワンチップマイコンにも適用可能
- 仕様の理解が容易
  - 技術者教育のための標準化の側面を重視
- 完全にオープンな標準仕様
  - ロイヤリティなしで実装することができる
- 多種多様なプロセッサ用に実装できる/されている
  - 8bitワンチップマイコンから32bit RISCマイコンまで
  - 異なるプロセッサへの移行が容易に
- 多くの機器で使用実績がある
  - 組込みシステム分野で最も広く使われているOS仕様
- 多くのメーカ/ベンダがサポート

ITRON仕様カーネルの開発状況・利用状況

- ITRON仕様カーネルの開発状況
  - 多くのITRON仕様準拠のRTOSが開発・販売されている
  - 自社用に実装した例も多数
  - いくつかのオープンソースの実装
- ITRON仕様カーネルの主な適応機器例

AV機器、家電、個人用情報機器、娯楽/教育機器
テレビ、ビデオ、デジタルカメラ、STB、オーディオ、電子レンジ、炊飯器
PDA、電子手帳、カーナビ、ゲーム機、電子楽器

パソコン周辺機器/OA機器
プリンタ、スキャナ、ディスクドライブ、DVDドライブ、コピー、FAX

通信機器
留守番電話機、携帯電話、ATMスイッチ、放送機器、無線設備、人工衛星

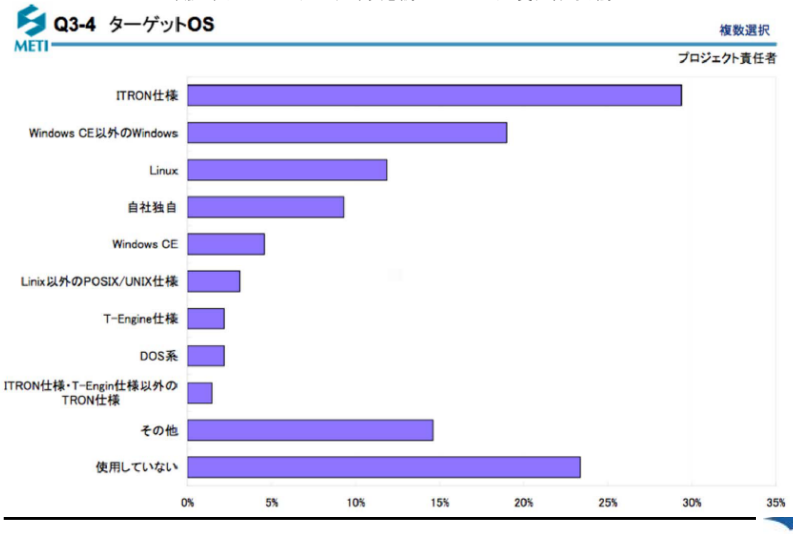
運輸機器、工業制御/FA機器/設備機器、その他
自動車、プラント制御、工業用ロボット、自動販売機、医療用機器

TOPPERS

RTOSの調査結果

経済産業省 Copyright ©2006 Ministry of Economy, Trade and Industry All Rights Reserved

2006年版 組込みソフトウェア産業実態調査:プロジェクト責任者向け調査



TOPPERS

ITRONの機能概要

1. ITRON仕様とは
2. リアルタイムOSの基礎
3. ITRON仕様の機能概要
4. TOPPERS/JSPカーネル

リアルタイムOS(RTOS)とは？

- 文字通りリアルタイムシステム構築のためのOS
- 具体的には,次のような特徴を持つOS(これらの特徴をすべて持っているとは限らない)
  - (1) リアルタイムシステム向けの機能を持つ
    - プリエンプティブな優先度ベーススケジューリング
    - 優先度継承や優先度上限プロトコルのサポートなど
  - (2) 予測可能性を持つ
    - OSの各サービス時間があらかじめわかっている
    - ただし,予測可能性にもいろいろなレベルがある
  - (3) 時間制約を管理 ← 一部の研究ベースのRTOS
    - OSが各処理の時間制約を考慮してスケジューリング
  - (4) 高速に応答(制御対象に対して十分に)

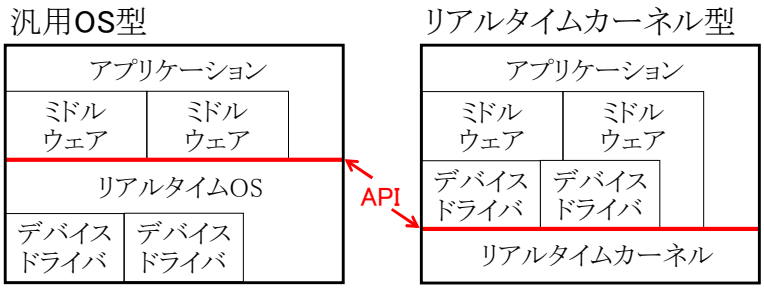
OSの機能面から見た組込みシステムの特性

- OSが必ずサポートしなければならないI/O装置はない
  - 多くの組込みシステムが共通に持つI/O装置が無い
    例）ストレージデバイスを持たない組込みシステムも多い
  - 複数のアプリケーションで同一のI/O装置を共有する状況は少ない
- 保護のための機能は必須ではない
  - 組込みソフトウェアは，組込み対象の機器を制御することのみを目的に設計される
  - 組込みソフトウェアは，機器に固定されている
    →デバッグが終われば，アプリケーションソフトウェアは信頼できるという前提が成り立つ
  - ! 最近，汎用システムと近い性質を持つ組込みシステムも多くなってきた（典型例は携帯電話．カーナビもその傾向）

RTOSとリアルタイムカーネル

- リアルタイムカーネルとは
  - （元々は）RTOSの中心になるモジュールの意味
  - どのようなシステムにも共通する資源を扱う
    - プロセッサ，メモリ，タイマ，…
    - 操作に時間のかかる資源を扱うモジュールは，カーネルの上で実現した方がスマートという理由も
  - 保護機能を持たない場合が多い
  - リアルタイムモニタ，リアルタイムエグゼクティブと呼ばれることもある
- RTOSとの関係
  - リアルタイムカーネル相当の機能しか必要としない場合には「リアルタイムカーネル＝RTOS」
  - 汎用のOSとはかなり違うものなので，リアルタイムカーネルと呼んで区別する

アーキテクチャによるRTOSの分類（中間的なものもある）



- OSの機能は豊富
- サイズは比較的大きい
- 一般に応答時間は遅い
- デバイスドライバは別のAPIで作成

- カーネルの機能は限定
- カーネルサイズは小さい
- 一般に応答時間は早い
- デバイスドライバとアプリケーションは同じAPI

！以下では，主にリアルタイムカーネル型を想定

リアルタイムカーネル/RTOSの主な機能

- リアルタイムカーネルの機能
  - マルチタスク機能 → プロセッサの仮想化
  - タスク間通信・同期機能
  - 時間同期/管理 → タイマの仮想化
  - メモリ管理 → メモリの仮想化
  - 割込み管理/処理，例外管理/処理，システム管理など
  - 保護機能 … 保護機能を持ったRTOSも増えつつある
- 汎用OS型のRTOSのその他の機能
  - 入出力管理／デバイス管理機能
  - ファイル管理機能（ファイルシステム）
  - 通信・ネットワーク機能（プロトコルスタック）
  - ユーザインタフェース機能（GUIなど）
  - プログラム管理／ローディング機能　などなど

マルチタスク機能

- タスクとは？
  - プログラムの並行実行の単位
    - 1つのタスク中のプログラムは逐次的に実行される
    - 異なるタスクのプログラムは並行して実行される
  - プロセッサを抽象化・多重化したもの

- マルチタスク機能
  - 複数のタスクを疑似並列に実行するための機能
    - シングルプロセッサシステムでは，実際に同時に実行できるタスクは1つのみ
    - 複数のタスクが同時に実行しているかのように見せる

マルチタスクに関する用語の整理

- *ディスパッチ（タスクディスパッチ，タスク切換え）*
  - プロセッサが実行するタスクを切り換えること
  - ディスパッチを実現するOS内のモジュールがディスパッチャ
- *スケジューリング（タスクスケジューリング）*
  - どの時間にどのタスクを実行するかを決定すること
  - 多くのRTOSにおいては，次に実行するタスクを決定する処理
  - スケジューリングを実現するOS内のモジュールがスケジューラ（UNIX/Linuxのスケジューラは，スケジューリングに加えて，ディスパッチも行う）
- *スケジューリングアルゴリズム*
  - どのようにして次に実行するタスクを決定するか？

RTOSにおけるスケジューリング

- プリエンプティブな優先度ベーススケジューリング
  - ! ほとんどのRTOSで採用されているスケジューリング方式（RTOSによっては他の方式もサポートしている）
  - – *優先度ベーススケジューリング*
    - 最も優先度の高いタスクが実行される
    - 優先度の高いタスクが実行できなくなるまで，優先度の低いタスクは実行されない
    - 同一優先度タスク間では FCFS（First Come First Served）
  - – *プリエンプティブスケジューリング*
    - 優先度の高いタスクが実行可能になると，優先度の低いタスクが実行途中でも，タスク切換えが起こる
    - 実行可能になるきっかけは割込み
  - ! 汎用OSのスケジューリングとは大きく異なる

マルチタスクの必要性

- タスク分割の基本的な考え方
  - – 独立した処理の流れを独立したタスクに
    - 複数のサブシステムに対する処理
    - 別々の入出力装置/イベントに対する処理　など
- リアルタイムシステムにおけるタスク分割の意義
  - – *論理的な処理の順序と時間的な処理の順序を分離*
    - プログラムは論理的な処理順序で記述
    - RTOSは時間的な処理順序に従って実行する

  - – プログラムの保守性・再利用性の向上
  - – 外部で開発されたソフトウェア部品の導入を容易に

論理的な処理順序と時間的な処理順序の分離

- 例として次の処理を行う場合を考える
  - モータの制御処理を10ミリ秒周期で行う．1回の処理に最大5ミリ秒かかる
  - それと並行して，ビデオカメラで撮影した画像を認識する処理を行う．1回の処理に最大100ミリ秒かかる
- RTOS無しで実現するには…
  - モータの制御処理を，10ミリ秒周期で回るメインループで実行する
  - 画像認識プログラムを，5ミリ秒単位の20個の処理に分割し，メインループの中で1つずつ順に実行する

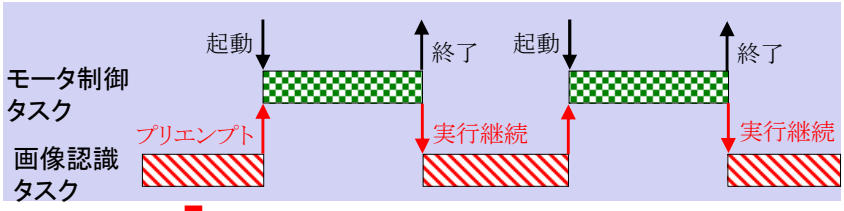
⇓

画像認識プログラムの保守性・再利用性が低下

TOPPERS

論理的な処理順序と時間的な処理順序の分離（続き）

- RTOS（マルチタスク機能）を用いると…
  - 2つの処理を別々のタスク（モータ制御タスクと画像認識タスク）で実現
  - モータ制御タスクの方に高い優先度を与える
  - モータ制御タスクは10ミリ秒周期で起動する



⇓

画像認識プログラムを分割する必要はなく，保守性・再利用性が向上

TOPPERS

論理的な処理順序と時間的な処理順序の分離（続き）

- 論理的な処理順序と時間的な処理順序の分離が本質
  - モータ制御処理と画像認識処理は，論理的には独立の処理
  - 時間的には，画像認識処理の途中にモータ制御処理を割り込ませないと間に合わない

  - プログラムは論理的な処理順序で（つまり両処理を独立して）記述し，それを時間的な処理順序で実行するのはRTOSに任せる

  - 時間制約を持ったプログラムの保守性・再利用性の向上
  - 外部で開発されたソフトウェア部品の導入を容易に
  - ！ ただし，この例の場合には，RTOSを使わずに，メインループと割込み処理で実現する方法もある

タスク状態

- 基本的なタスク状態
  - RTOSでは，タスクが疑似並列実行されている状況をアプリケーション開発者に明示することが必要

実行状態と実行可能状態
（RUNNING） （READY）

- 実行すべき処理が無い状態　休止状態（DORMANT）
  - 低優先度タスクが実行される
  - 起動されると，タスクの先頭から実行される
- 何らかの事象発生を待っている状態　（広義の）待ち状態
  - ！事象の発生をループで待ってはいけない
  - 低優先度タスクが実行される
  - 事象が発生した場合に，前の続きから実行される

待ち状態の有用性

- プログラム中のどこを実行しているかで，状態を表現するプログラムで有用
- 特に，呼び出された関数の内部で待ち状態としたい場合

例）サーバからのデータ取出し処理（）
{
サーバにデータAを問い合わせるコマンドを送る;
サーバからの応答を待ち，変数Aに格納する;
サーバにデータBを問い合わせるコマンドを送る;
サーバからの応答を待ち，変数Bに格納する;
}

- ローカル変数で状態を保持できるのもメリットの1つ

タスク間の通信と同期

- タスク間通信・同期の必要性
  - タスクが協調して動作するためには，タスク間でデータをやりとりすること（＝タスク間通信）が必要
  - タスク間通信の際には，タスク同士で動作タイミングをあわせること（＝タスク間同期）が必要
  - 複数のタスクが同一の資源を取りあう場合にも，タスク間同期が必要（排他制御）
- タスク間通信・同期のタイプ
  - ！タスクを仮想化されたプロセッサと捉えるなら，タスク間の通信・同期は，プロセッサ間の通信・同期を仮想化したもの
  - 共有メモリによる通信
  - メッセージによる通信

共有メモリによる通信

- 複数のタスクからアクセスできるメモリ領域（共有メモリ）上に受け渡しするデータを置く
- 共有データを読み書きする際には，排他制御することが必要．RTOSは排他制御のための機能を持つ
  （共有データが1度に読み/書きできる場合は例外）
  - 割込み/ディスパッチの禁止，タスク優先度の変更
  ！*マルチプロセッサへの拡張性に注意*
  - セマフォ，ミューテックス
  ！*デッドロックと優先度逆転に注意*
- 共有データを速やかに処理させたい場合には，事象の発生を知らせる機能を用いる
  - タスクの起動/起床
  - イベントフラグ，条件変数（Condition Variable）

メッセージによる通信

- RTOSが持つメッセージ通信のための機能を用いて，タスク間でメッセージを受け渡しする

- メッセージ通信機能のタイプ
  - コネクションあり *or* なし，単方向 *or* 双方向，1対1（送受信できるタスクが固定）*or* n対1 *or* n対n
  - 通信相手の指定方法
    - 通信オブジェクトを指定 *or* 相手タスクを指定 *or* …
  - 同期メッセージ通信 *or* 非同期メッセージ通信
  - （非同期通信で）バッファにメッセージが無い時の振舞
    - 待つ（ブロッキング）*or* 待たない（ノンブロッキング）
  - （非同期通信で）バッファがフルの時の振舞い
    - 待つ（ブロッキング）*or* 待たない（ノンブロッキング）*or* 上書きする

メッセージによる通信（続き）

- メッセージ通信機能のタイプ（続き）
  - メッセージが固定長 *or* 可変長（任意長）
  - （可変長の時に）パケットの単位がある *or* ない
  - ポインタを渡す *or* コピーする
    （メッセージが1度に読み/書きできる場合は意味が無い）
  - （非同期通信で）メッセージのキューイング順序
    - FIFO順 *or* 優先度順
  - 受信/送信待ちタスクのキューイング順序
    - FIFO順 *or* 優先度順
  - その他の特殊な機能
    - マルチキャスト/ブロードキャスト
    - 状態メッセージ（OSEK/VDX仕様のunqueued message）
  - ! RTOSの持つメッセージ通信機能（複数の機能を持つ場合も多い）の性質を良く理解することが必要

静的OS（Static Operating System）

! 専用システムであるという特性を活かしたOS技術

- 静的OSとは？
  - –使用するOS資源（タスク，セマフォなど）を静的に（設計時に）定義するOS
    例）多くの μITRON仕様OS
    OSEK/VDX仕様OS，AUTOSAR OS仕様（OS資源を動的に生成するAPIが仕様に規定されていない）
- 静的OSの利点
  - –メモリ容量（特にRAM容量）を小さくできる
  - –システム起動時間を短縮できる
  - –システムの動作中にメモリ不足になることがない
  - –静的な情報を使った最適化が可能

静的OS（Static Operating System）（続き）

- コンフィギュレーション記述の方法
  （1）コンフィギュレーション記述の言語を定め，そこからツールにより構成・初期化ファイルを生成する方法
    - ITRON仕様の静的API
    - OSEK/VDX仕様のOIL（OSEK Implementation Language）
    - AUTOSAR仕様ではXMLのフォーマットを規定
  （2）GUIベースのコンフィギュレーションツールを用意する方法
  （3）構成・初期化ファイルを直接記述する方法（C言語の初 期化文の形で記述するのが一般的）
    - 静的な情報を使った最適化は難しい

RTOSを使用するメリット

- ソフトウェアの構造化による生産性，保守性，信頼性の向上
  ソフトウェアの構造化 ≒ *モジュール化*
- 時間制約を持った多重処理システムの構築を容易に
  –論理的な処理順序と時間的な処理順序の分離により，時間制約を持ったソフトウェアの保守性・再利用性が向上
  ➡ソフトウェア部品を用いたソフトウェア開発（コンポーネントベース開発）の基盤
- ハードウェア（特にプロセッサ）の違いを隠蔽
  –ハードウェアの細部を知らなくても，アプリケー ションが構築できる
  –「リアルタイムカーネルは，プロセッサのデバイスドライバである」

RTOSを使用するデメリット

RTOSを使用するメリットとの引き換え

- RTOS自身のオーバヘッド
  - –オーバヘッドによる実行性能の低下
  - –RTOS自身のワークエリアによるメモリの消費
  - → リアルタイムカーネルでは，半導体技術やRTOS実装技術の進歩により，問題になる場面は減ってきた
- プリエンプティブスケジュ–リングのデメリット
  - –タスクのスタックエリアによるメモリの消費
  - –非決定性が増すことによるデバッグ・テストの困難化
  - → 行儀のよいタスク間同期・通信の設計が必要
- RTOSのブラックボックス化による解析の困難性
  - → RTOSに対応したツールの利用でかなり改善できる

! RTOSの原理・内部構造を知るべき

TOPPERS

ITRONの機能概要

1. ITRON仕様とは
2. リアルタイムOSの基礎
3. ITRON仕様の機能概要
4. TOPPERS/JSPカーネル

TOPPERS

ITRON仕様の機能解説

- μITRON4.0仕様を例として、リアルタイムカーネルの機能について解説。μITRON4.0仕様のスタンダードプロファイルのシステムコール（μITRON仕様ではサービスコールと呼んでいる）はすべてカバーする
- μITRON4.0仕様書は以下のURLからダウンロード可能

  http://www.ertl.jp/ITRON/

ITRON4．0仕様書の読み方

- μITRON4.0仕様書の構成　赤字の章が仕様の本体

  第1章　μITRON4.0仕様作成の背景

  第2章　ITRON仕様共通規定

  μITRON4.0仕様とそれと整合するように標準化されるソフトウェア部品仕様に共通の規定

  第3章　μITRON仕様の概念と共通定義

  複数の機能単位にまたがる概念や共通の定義

  第4章　μITRON4.0仕様の機能

  サービスコールと静的APIの機能（機能単位毎）

  第5章　付属規定

  仕様準拠の条件、自動車制御用プロファイルなど

  第6章　付録

  第7章　リファレンス

ITRON仕様のカーネルの機能(μITRON4.0仕様)

- カーネルの機能
  - タスク管理機能
  - タスク付属同期機能
  - タスク例外処理機能
  - 同期・通信機能
  - 拡張同期・通信機能
  - メモリプール機能
  - 時間管理機能
  - システム状態管理機能
  - 割り込み管理機能
  - サービスコール管理機能
- サービスコール数
  - フルセット
    - サービスコール : 166
    - 静的API : 21
  - スタンダードプロファイル
    - サービルコール : 70
    - 静的API : 11
  - 自動車制御用プロファイル
    - サービスコール : 43
    - 静的API : 8
  - 最小セット

！I/O操作のための機能は規定されていない

ITRON仕様のサービスコール(API)

- サービスコール名称のガイドライン

  例) act_tsk

  操作対象を表す. この場合はタスク(task)

  操作方法を表す. この場合は起動(activate)
  - サービスコールが成功するとE_OKが戻り値として返される
- 割込みハンドラ(厳密には)から呼び出せるサービスコールは"i"で始まる名称として区別

  例) iact_tsk
- 割込みハンドラから呼び出せないAPIがある
  - 待ち状態に入るAPI

μITRON4.0仕様におけるコンフィギュレーション

- カーネルオブジェクトは静的に生成
  - オブジェクト生成情報の、コンフィギュレーションファイルの中での記述方法を標準化（静的API）
- 静的APIの例

```
CRE_TSK(tskid, {tskatr, exinf, task, itskpri, stksz,stk});
DEF_INH(inhno, {inhatr, inthdr});
```

- コンフィギュレータは静的APIを解析して、カーネルの内部情報を生成する

タスクの生成

CRE_TSK	タスク生成（静的API）	【スタンダード】
cre_tsk	タスク生成	【スタンダード外】

【静的API】

CRE_TSK（ID tskid,{ATR tskatr, VP_INT exinf, FP task,
PRI itskpri, SIZE, stksz, VP stk}）

【C言語API】

ER ercd = cre_tsk(ID tskid, T_CTSK *pk_ctsk)

【パラメータ】

ID	tskid	生成対象のタスクのID番号
T_CTSK	*pk_ctsk	タスク生成情報を入れたパケット
ATR	tskatr	タスク属性（記述言語、初期状態）
VP_INT	exinf	タスクの拡張情報（タスクへのパラメータ）
FP	task	タスクの起動番地
PRI	itskpri	タスクの起動優先度
SIZE	stksz	タスクのスタック領域のサイズ（バイト数）
VP	stk	タスクのスタック領域の先頭番地

タスクの起動

act_tsk　　タスク起動　　【スタンダード】
iact_tsk　　【スタンダード】

【C言語API】
ER ercd = act_tsk(ID tskid);
ER ercd = iact_tsk(ID tskid);

【パラメータ】
ID　tskid　　起動対象のタスクのID番号

【リターンパラメータ】
ER　ercd　　エラーコード

【エラーコード】
E_CTX　　呼び出しコンテキストが不正
E_ID　　対象タスクID番号（tskid）が不正
E_NOEXS　　対象タスクが未登録状態
E_QOVR　　起動要求のキューイングオーバーフロー
その他のエラー　　E_SYS,E_NOSPT,E_OACV

μITRON4．0仕様の機能概要

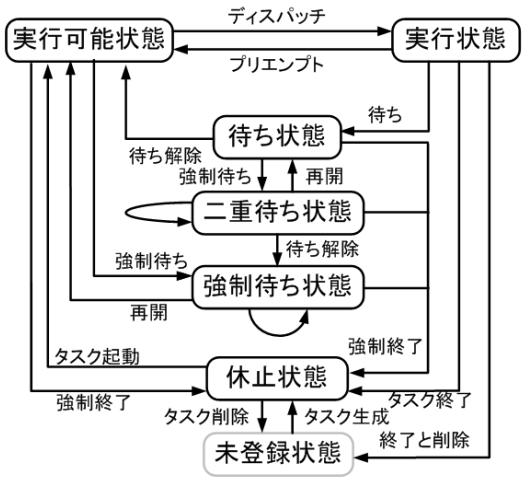
- タスク管理機能
- タスク付属同期機能
- タスク例外処理機能
- 同期・通信機能　セマフォ、イベントフラグ、データキュー、イベントフラグ
- 拡張同期・通信機能　ミューテックス、メッセージバッファ、ランデブ
- メモリプール機能　固定長/可変長メモリプール
- 時間管理機能　システム時間管理、周期/アラーム/オーバーランハンドラ
- システム状態管理機能　システム状態参照機能
- 割込み管理機能　割込みサービスルーチン
- サービスコール管理機能
- システム構成管理機能

＊緑字はスタンダード外

マルチタスク機構

- スケジューリング方式
  - μITRON4.0では、プリエンプティブな優先度ベーススケジューリングによるマルチタスク機構をサポート
  - 同優先度のタスクはFCFS（First Come First Served）方式でスケジューリング．同優先度のタスク内での実行順序を変更する機能を用意．これを用いて、同優先度内でのラウンドロビンスケジューリングも実現可能
  - 優先度付けは静的が基本だが、優先度を動的に変更する機能も用意
- タスク状態
  - 実行、実行可能、休止、待ち、強制待ち、二重待ち、未登録の7つの状態をサポート

タスクの状態遷移



- **実行状態（RUNNING）**
- **実行可能状態（READY）**
- **待ち状態（WAITING）**
  - タスクが自ら待ち状態に入る
- **強制待ち状態（SUSPENDED）**
  - 他のタスクにより待ち状態にされた
- **二重待ち状態（WAITTING-SUSPENDED）**
  - 上の2つが重なった状態
- **休止状態（DORMANT）**
- **未登録状態（NON-EXISTENT）**

タスク管理機能

タスクの状態を直接的に操作するための機能

- タスク管理機能のサービスコール

cre_tsk	タスクの生成
del_tsk	タスクの削除
act_tsk, iact_tsk	タスクの起動
can_act	タスクの起動要求の無効化
ext_tsk	自タスクの終了
ter_tsk	タスクの強制終了
chg_pri	タスクの優先度の変更
get_pri	タスクの優先度の参照
ref_tsk	タスクの状態参照

- タスクの起動要求（act_tsk）のキューイングとその無効化（can_act）

タスク付属同期機能

タスクを直接操作して同期を実現するための機能

- タスク付属同期機能のサービスコール

slp_tsk, tslp_tsk	自タスクを起床待ち状態に
wup_tsk, iwup_tsk	他タスクの起床
can_wup	起床要求の無効化
rel_wai, irel_wai	タスクの待ち状態の強制解除
sus_tsk	タスクを強制待ちに
rsm_tsk, frsm_tsk	タスクを強制待ちからの再開
dly_tsk	自タスクを遅延

- タスクの起床要求（wup_tsk）のキューイングとその無効化（can_wup）
- タイムアウト付きの起床待ち（tslp_tsk）と自タスクの遅延（dly_tsk）

タスク例外処理

- タスクに対する割込み/例外に対応（仮想化された割込み）
- 明示的なサービスコール呼び出しにより、タスクに例外を発生させる
- 次にそのタスクが実行されるタイミングで、タスク例外処理ルーチンが呼び出される
- UNIXのシグナル処理/シグナルハンドラに相当

- タスク例外処理サービスコール

def_tex	タスク例外処理ルーチンの定義
ras_tex, iras_tex	タスク例外処理の要求
dis_tex	タスク例外処理の禁止
ena_tex	タスク例外処理の許可
sns_tex	タスク例外処理禁止状態か?

タスク間同期・通信機能

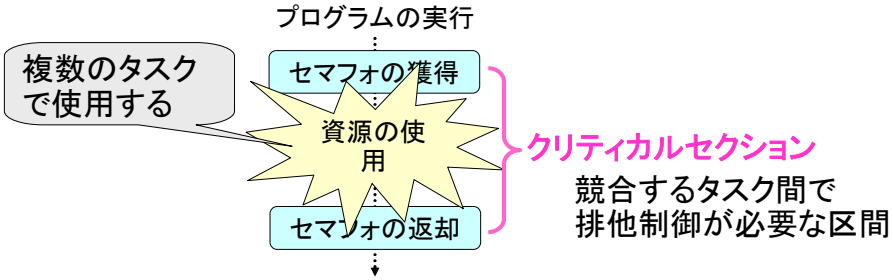
μITRON4.0では、タスク間同期・通信のために以下の機能を用意

- （主に）共有メモリによる通信のための機能
  - セマフォ*（排他制御など）
  - イベントフラグ*（事象通知）
  - ミューテックス（排他制御）
- メッセージ通信のための機能
  - データキュー*（同期・非同期、1ワードのメッセージ）
  - メールボックス*（非同期、ポインタを送受信）
  - メッセージバッファ（同期・非同期、可変長メッセージ）
  - ランデブ（同期、双方向に通信、可変長メッセージ）

*スタンダードプロファイルに含まれる機能

セマフォ（Semaphore）

- 使用されていない資源の有無や数量をセマフォの資源数として管理することにより排他制御を実現
- 資源を使用する前にセマフォ資源を獲得し，資源を使用した後にセマフォ資源を返却する．
- 資源が全て使用されていれば，セマフォ資源獲得の時点で待ち状態になる．

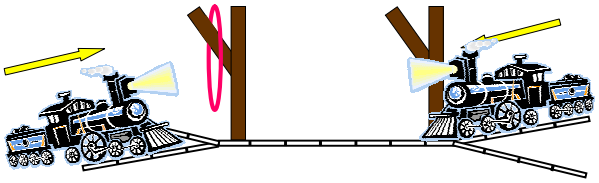


セマフォ（Semaphore）：サービスコール&語源

- サービスコール

cre_sem	セマフォの生成
del_sem	セマフォの削除
sig_sem, isig_sem	セマフォ資源の返却
wai_sem, pol_sem, twai_sem	セマフォ資源獲得

- 語源
  鉄道の単線区間で進入制御に用いていた「輪」．
  単線区間に入る場合は，この輪（セマフォ）を取る

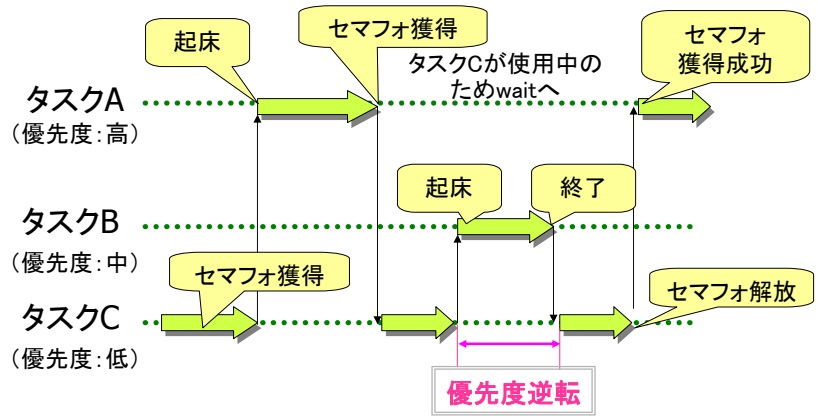


輪 ：セマフォ
列車 ：タスク
単線区間 ：資源

セマフォ（Semaphore）：優先度逆転

- 高優先度のタスクが低優先度のタスクの実行により待たされること

例）高優先度のタスクAがそれより低優先度のタスクBの実行により待たされる．（注 タスクCに待たされるのは本質的）



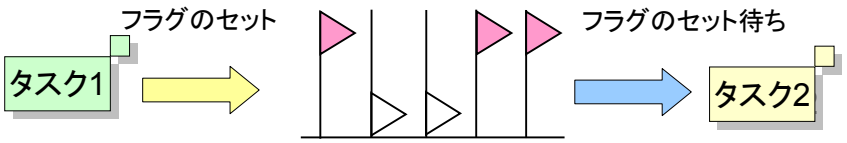
TOPPERS

イベントフラグ（EventFlag）

- タスク間でイベントの発生を伝えることで同期するための機構．1つのイベントフラグは、複数のフラグで構成

- サービスコール

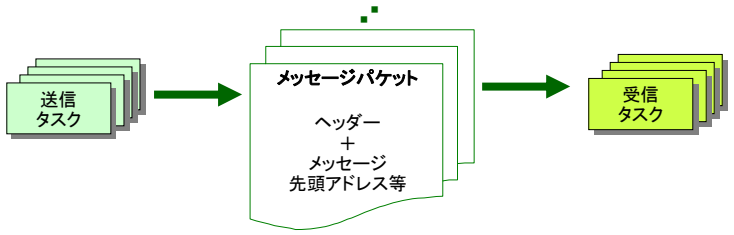
cre_flg	イベントフラグの生成
del_flg	イベントフラグの削除
set_flg, iset_flg	イベントフラグのセット
clr_flg	イベントフラグのクリア
wai_flg, pol_flg, twai_flg	イベントフラグ待ち



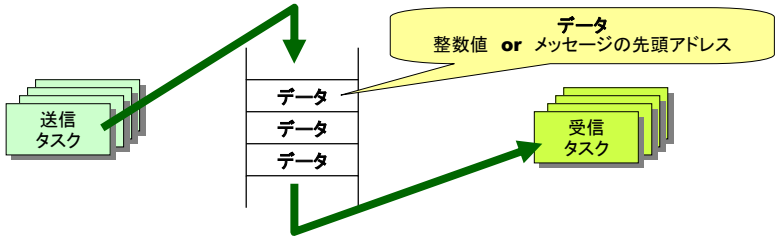
TOPPERS

メイルボックス と データキュー：概要

- メイルボックス：共有メモリ上に置かれたデータをやり取りする



- データキュー ：1ワードのメッセージをやり取りする



TOPPERS

データキュー（Data Queue）

- 1ワードメッセージの非同期メッセージ通信機構
- メッセージは，内部バッファにコピーされる
- メッセージが無い/フルの時は，待ち合わせる
- データキュー領域のサイズを0に設定すると、同期メッセージ通信も実現可能
- サービスコール

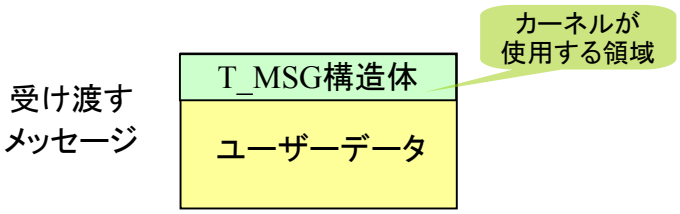
cre_dtq	データキューの生成
del_dtq	データキューの削除
snd_dtq, psnd_dtq, tsnd_dtq, isnd_dtq	データキューへの送信
rcv_dtq, prcv_dtq, trcv_dtq	データキューからの受信
fsnd_dtq, ifsnd_dtq	データキューへの上書き送信

TOPPERS

メイルボックス（Mail Box）

- 可変長メッセージの非同期メッセージ通信機構
  - カーネルがリンクに用いるための領域を、メッセージの先頭にアプリケーション側で用意
- サービスコール

cre_mbx	メイルボックスの生成
del_mbx	メイルボックスの削除
snd_mbx	メイルボックスへの送信
rcv_mbx, prcv_mbx, trcv_mbx	メイルボックスからの受信



ミューテックス

cre_mtx	ミューテックスの生成
del_mtx	ミューテックスの削除
loc_mtx,ploc_mtx tloc_mtx	ミューテックスのロック
unl_mtx	ミューテックスのロック解除

- 優先度継承/優先度上限プロトコルをサポートする排他制御機構
  ⇒POSIXリアルタイム拡張のmutexに相当
- 上限のない優先度逆転を防止
- 最大資源数が1のセマフォとの違い
  - ロックしたタスク以外はロック解除できない
  - タスク終了時に自動的にロック解除される

メモリ管理機能

- システム共通のメモリ領域をタスクに割り当てる機能
  - C言語のmalloc/freeと類似の機能をRTOSが持つ
  - 多くのRTOSがサポート
- 多重メモリ空間を提供する機能（MMUを使う）
  - 現状ではMMUを使うことによるオーバーヘッドが大きく、比較的大規模なRTOSのみでサポート

メモリプール

- システム共通のメモリ領域を複数のメモリプールに分割
  - 目的によってメモリプールを分けると、重要な処理にメモリを残しておくといった使い方が可能
- μITRON4.0では、メモリが足りないときに待ちに入る機能
- μITRON4.0仕様では、固定長と可変長の両方をサポート

μITRONのメモリプール

- 固定長メモリプール機能のサービスコール

cre_mpf	固定長メモリプールの生成
del_mpf	固定長メモリプールの削除
get_mpf, pget_mpf tget_mpf	固定長メモリブロックの獲得
rel_mpf	固定長メモリブロックの解放

- 可変長メモリプール機能のサービスコール

cre_mpl	可変長メモリプールの生成
del_mpl	可変長メモリプールの削除
get_mpl, pget_mpl tget_mpl	可変長メモリブロックの獲得
rel_mpl	可変長メモリブロックの解放

時間管理機能：概要

- システム時刻管理
  - システム時刻（カーネルが管理する現在時刻）を管理するための機能
- タイムイベントハンドラ
  - 時間の経過をきっかけに呼び出されるルーチン
  - μITRON4.0仕様では以下のタイマハンドラを用意
    - 周期ハンドラ（周期的に呼ばれる）
    - アラームハンドラ（指定した時刻に呼ばれる）
    - オーバランハンドラ（オーバラン時に呼ばれる）
- 時間同期のためのその他の機能
  - タスクの遅延（タスクを一定時間待たせる）
  - 待ちに入るサービスコールのタイムアウト機能

時間管理機能：サービスコール1

- システム時刻管理のためのサービスコール

set_tim	システムクロックの設定
get_tim	システムクロックの読み出し
isig_tim	タイムティック供給

- 周期ハンドラ操作のためのサービスコール

cre_cyc	周期ハンドラ生成
del_cyc	周期ハンドラの削除
sta_cyc	周期ハンドラの起動
stp_cyc	周期ハンドラ停止

時間管理機能 ：サービスコール2

- アラームハンドラ操作のためのサービスコール

cre_alm	アラームハンドラの生成
del_alm	アラームハンドラの削除
sta_alm	アラームハンドラの起動
stp_alm	アラームハンドラの停止

- オーバーランハンドラ操作のためのサービスコール

def_ovr	周期ハンドラ生成
sta_ovr	周期ハンドラの削除
stp_ovr	周期ハンドラの起動

  - タスクが設定されたプロセッサ時間を使い切った時に例外を発生させる機能
  - オーバーランハンドラはシステムに一つのみ登録可能
  - プロセッサ時間の測定精度は実装依存

システム状態管理機能

システム状態を参照/変更するための機能

- システム状態管理機能のサービスコール

rot_rdq, irot_rdq	タスクの実行順序の回転
get_tid,iget_tid	実行状態のタスクのIDの取り出し
loc_cpu, iloc_cpu	CPUロック状態に
unl_cpu_iunl_cpu	CPUロック状態の解除
dis_dsp	ディスパッチ禁止
ena_dsp	ディスパッチ許可
sns_ctx	非タスクコンテキスト実行中か?
sns_loc	CPUロック状態か?
sns_dsp	ディスパッチ禁止状態か?
sns_dpn	ディスパッチ保留状態か?

割込み処理と割込み管理機能

- 割込み処理（外部割込み）
  - μITRON仕様では、割込みハンドラをアプリケーション側で記述できる
  - 割込みが発生すると、カーネル内の出入り口処理を経由して、割込みハンドラが呼び出される
  - 割込みハンドラの中でサービスコールを呼び出して、タスクの起動/起床などが可能
  - 割込みハンドラ処理はタスクよりも優先
    - ディスパッチが遅延する
- 割込み管理機能のサービスコール

def_inh	割込みハンドラの定義

  - この他に、割込みを個別に禁止/許可する機能など

CPU例外処理とCPU例外管理機能

- プロセッサレベルの例外処理（CPU例外）
  - 割込み処理と類似
  - CPU例外が発生すると、カーネル内の出入口処理を経由して、CPU例外ハンドラが呼び出される
  - CPU例外ハンドラの中でシステムコールを呼び出して、タスクに対して例外を発生させることが可能
  - CPU例外ハンドラの処理はタスクよりも優先

    ！タスク例外との違いに注意
- CPU例外管理のシステムコール
  - def_exc ･･･ CPU例外ハンドラの定義

まとめ

- ITRONの同期・通信機能
  - セマフォ
  - イベントフラグ
  - メイルボックス
  - データキュー
- それぞれの機能の性質を理解して，使用目的に最適な機能を選択することが重要．
- 細かいパラメータの設定で，振る舞いが変化するので，プログラミングの際には注意する．

ITRONの基礎知識

1. ITRON仕様とは
2. リアルタイムOSの基礎
3. ITRON仕様の機能概要
4. TOPPERS/JSPカーネル

TOPPERS/JSPカーネルについて

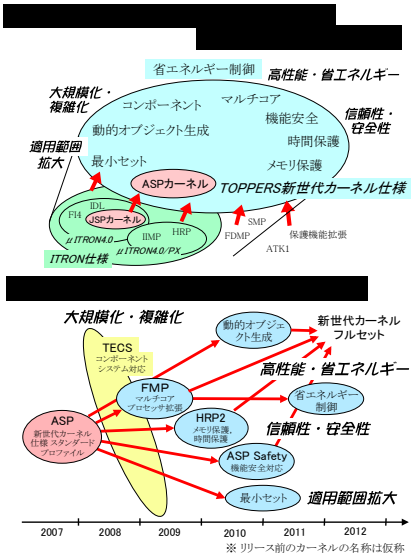
- JSPとは
  - JSP = Just Standard Profile
  - TOPPERSプロジェクトで開発されているμITRON4.0仕様のスタンダードプロファイルに準拠したオープンソースのリアルタイムOS．最新版は Release 1.4.3
- 開発の目的
  - μITRON4.0仕様の評価、リファレンス実装
  - 研究・教育機関における研究・教育のプラットホーム
  - ソフトウェア部品（IP）開発のプラットホーム
  - 評価目的・プロトタイプ開発への利用
  - 実製品への適用
- ターゲット環境
  - SH1/2,SH3, H8, ARMv4, MIPS, PowerPC

TOPPERS/JSPカーネルの特徴

- 読みやすく改造しやすいソースコード
  - 定量的な評価は難しいが、読みやすさには自身あり
  - 可能な限り #ifdef を追放
  - 様々な改造・拡張の実績あり
- 他のターゲットへのポーティングが容易な構造
  - 実行性能を落とさないプロセッサの抽象化
  - 新しいプロセッサへのポーティングを2日間で行った例
- 高い実行性能と小さいRAM使用量
  - ！大部分をC言語で記述したカーネルとしては
- LinuxおよびWindows上でのシミュレーション環境
  - プロトタイプ開発、教育用に最適
- オープンソースソフトウェアのみで開発環境まで

TOPPERS新世代カーネルについて

- 組込みシステムにおける新しい要求に応えるために、新しい世代のリアルタイムOSの開発を進めている
- 開発方針
  - μITRON4.0仕様をベースに拡張・改良を加える
  - ソフトウェアの再利用性を重視する
  - 高信頼性・安全なシステム構築を支援する
  - アプリケーションシステム構築に必要な機能は積極的に取り組む



TOPPERS/JSPカーネルの導入

1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

ITRON導入実習 : 実習内容

- Mindstorms nxt用のTOPPERS/JSPカーネルの導入を行う

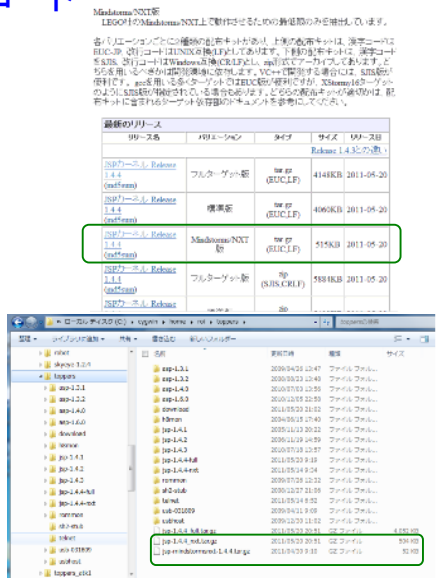
- TOPPERS/JSPカーネルについての説明
- カーネルのダウンロードと環境つくり
  - ダウンロードとツールを作成します
- 開発・実行環境の使用方法についての説明
  - 開発環境と2種類の実行環境の使用方法について解説
- カーネルのビルドと実行
  - Mindstorms nxt用カーネルをビルドし、サンプルプログラムを実行します
- サンプルプログラムの説明
  - サンプルプログラムの説明と実行確認

TOPPERS/JSPカーネルの導入

1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

TOPPERS/JSPのダウンロード

- TOPPERSプロジェクトのWebから最新版のjspカーネル Release 1.4.4_nxtをダウンロードします
- 本教材のjsp-mindstormsnxt-1.4.4.tar.gzと供にログインディレクトリに置いてください
- jsp-1.4.4-nxtにMindstorms nxt拡張ソースを加えてビルド環境を構築します



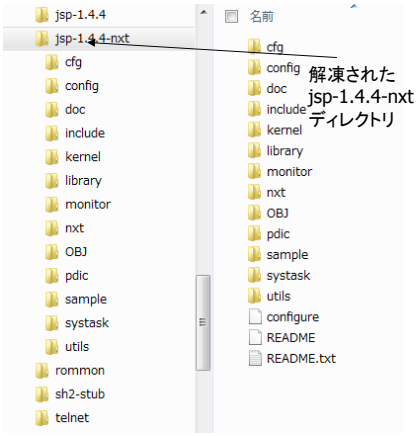
ソースコードについての注意

- JSPのカーネルソース、教材のプログラムの漢字コードはEUC-JP、改行コードはUNIXと互換のLFとなっています
- Cygwinの環境で参照、修正を行う場合は、UNIX互換漢字コード、改行コード対応のエデッタを使用してください

漢字コード、改行コードを自動変換する
エデッタ：TeraPad

プログラムの解凍

- Cygwinからログインディレクトリに移動して、jspとjsp-mindstormsnxtをtarコマンドで解凍します
- jsp-mindstormnxtは先に解凍したjsp-1.4.4-nxtに上書きされます

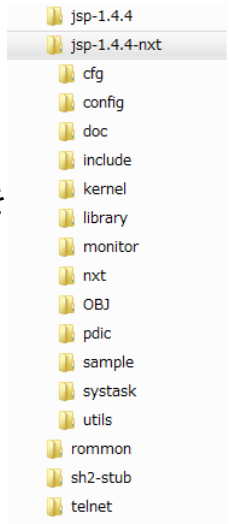



```
$ tar zxvf jsp-1.4.4_nxt.tar.gz
$ tar zxvf jsp-mindstormsnxt-1.4.4.tar.gz
```


TOPPERS

コンフィギュレータの構築

- JSPは静的APIサポートしているため、静的APIをプログラミング言語に変換するコンフィギュレータが用意されている
- Cygwinでは、cfgの下のMakefileを用いてコンフィギュレータのビルドを行う必要があります
- jsp-1.4.4-nxt/cfgに移動して、makeコマンドでコンフィギュレータをビルドします

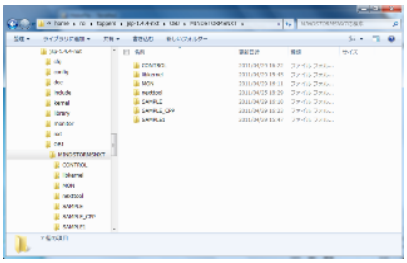



```
$ cd jsp-1.4.4-nxt/cfg
$ make depend
$ make
```


TOPPERS

TOPPERS/JSP用ワークスペースの作成

- 開発用のワークスペースを作成します
- この教材では、jsp-1.4.4-nxt/OBJ/MINDSTORMSNXTに設定されています
- 4つのプログラムが用意されています



ディレクトリ	内容
SAMPLE	走行体サンプルC言語版
SAMPLE_CPP	走行体サンプルC++言語版
MON	sample1+ROMモニタ
CONTROL	走行体Bluetoothラジコン制御

TOPPERS/JSPカーネルの導入

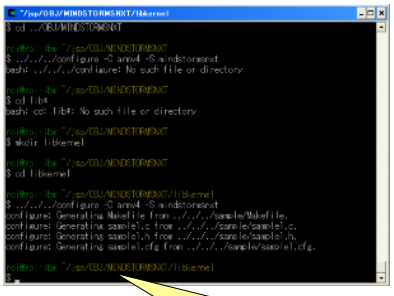
1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

MINDSTORMS NXT用ビルドを行うための注意点

- 通常のエバレーションボードでは、操作のためのシリアルデバイスや電源スイッチが実装されています
- MINDSTORMS NXTではシリアルコネクタが実装されておらず、電源オンオフのためにキー操作を必要とします
- MINDSTORMS NXTの実装では、シリアルデバイスの代わりにBluetooth通信をサポートしたり、LCDやキー操作をサポートするためにECROBOTインターフェイスやデバイスドライバの取り込みを行わなければなりません
- 以下のライブラリィを前もって作成する必要があります
  - ecrobotインターフェイスライブラリィ
  - デバイスドライバライブラリィ

SAMPLE1のビルドと実行

- まず、JSPカーネルライブラリィ化を行います
- これにより、プログラムの開発時、毎回カーネルプログラムをコンパイルしなくてよくなります
- 以下のコマンドでlibkernelディレクトリに標準のsample1プログラムを作ります
- カーネルライブラリィの指定は各MakefileのKERNEL_LIB変数にライブラリィディレクトリを登録することで有効となります



configureコマンドで標準的なsample1のプログラムが設定される

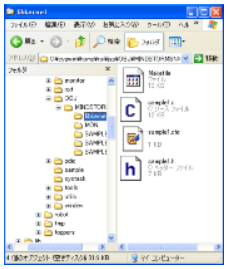
```
$ cd ../OBJ/MINDSTORMSNXT
$ mkdir libkernel
$ cd libkernel
$ ../../../configure –C armv4 –S mindstormsnxt
```

MINDSTORMS NXTのカーネル実装1

- 標準的なTOPPERS/JSPの実装ではlibkernelディレクトリで生成した3つのプログラムでsample1を実行できます
- Mindstorms nxtではシリアルデバイス用のコネクタがなかったり、電源操作をパネルで行うために、標準的なプログラムにいくつかに拡張を行っています
- diffコマンドプログラムの比較を行います

```
$ diff ../libkernel ../SAMPLE1
```

追加した機能	詳細説明
Bluetoothのスレーブモードのコネクション処理	カーネル内にBluetoothのデバイスドライバを実装しています。パソコンとストリームモードで通信するために、コネクション処理を行う必要があります
プラットフォームモジュールの取り込み	LCDやキーを操作するためにプラットフォーム機能を取り込む必要があります



TOPPERS

MINDSTORMS NXTのカーネル実装2

- カーネルライブラリィ以外に、プラットフォームライブラリィを準備する必要があります

ファイル	変更内容
Makefile	カーネルライブラリの指定、プラットフォームとデバイスドライバのinclude pathとライブラリィの指定、Bluetooth用のコネクション関数ファイル(bluetooth.o)の取り込み、rxeファイルの設定
sample1.c	Bluetooth接続待ち画面の表示機能を追加(show_main_screen,display_status_bar関数コール)
sample1.cfg	プラットフォーム機能の取り込み(#include ../../../nxt/ecrobot/ecrobot.cfg)
sample1.h	デフォルトスタックサイズを指定

- 以下のコマンドでカーネルライブラリィ(libkernel.a)を作成します

```
$ make depend
$ make libkernel.a
```


TOPPERS

NXT用ライブラリィの作成

- jsp/nxtディレクトリに移動して、ecrobotインターフェイスライブラリィ(libercobot.a)とデバイスドライバライブラリィ(libdevice.a)を作成します
- このライブラリィはMakefileの指定でサンプルプログラムに取り込まれます

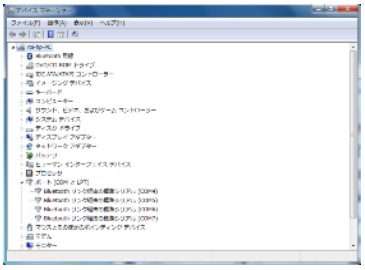

```
$ cd ../../../nxt
$ make
```

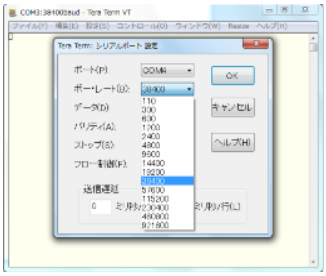

Bluetoothをパソコンに登録

- MINDSTORMS NXTを起動しBluetooth ⇒ Searchにてパソコンが表示されるまで待ちます
- パソコン名が表示されたら選択します
- PINCODE(ペアリングコード)は指定されたコードを設定してください
- 注意:走行会に参加した場合、多くの参加者がおり、他の参加が同一のPINCODEで設定している場合、正しくコネクションができません

Bluetoothリンクのシリアルポートの確認

- デバイスマネージャを開いてBluetoothに対応したシリアルポートを確認します
- Bluetoothを使ってMINDSTORMS NXTと通信するために、Tera Termをインストールします
- Tera TermでBluetoothのリンクシリアル(左の環境ではCOM4)を通してNXTとのstream通信 します





TOPPERS

PINCODEを設定する

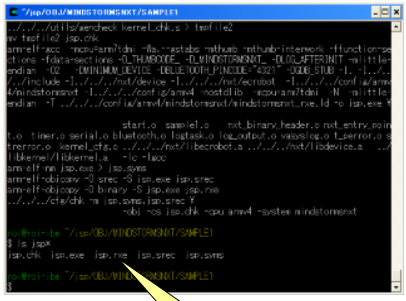
- sample1用のプログラムをパソコンに登録したPINCODEに変更します
- PINCODEを”4321”に設定したとして、プログラムのPINCODEを変更します
- エデッタでconfig/armv4/mindstormsnxt/hw_serial.hを開き、BLUETOOTH_PINCODEの定義を追加します

```
/*
 * ピンコードの指定がない場合のデフォルトを"1234"とする
 */
#ifndef BLUETOOTH_PINCODE
#define BLUETOOTH_PINCODE      "4321"
#endif
```


TOPPERS

sample1のビルド

- SAMPLE1ディレクトリに移動して実行プログラムをビルドします
- ビルドが終了するとMINDSTROMS NXTでの実行ファイルjsp.rxeが作られます

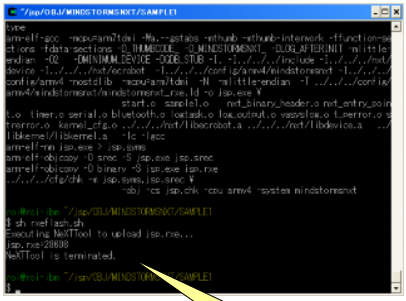


ビルドが終了するとjsp.rxeが作られる

```
$ cd ../SAMPLE1
$ make depend
$ make
```


実行ファイルをNXTにアップロードする

- MINDSTORMS NXTとパソコンをUSBケーブルでつなぎ、電源を入れる
- Cygwinからrxeflash.shシェルスクリプトを起動する
- 正しくアップロードされるとファイルサイズを表示します

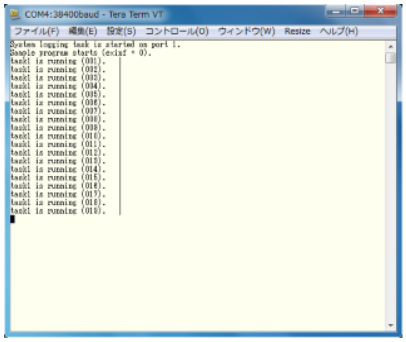


正しくアップロードされるとファイルサイズが表示される

```
$ sh rxeflash.sh
```


jspを実行する

- MINDSTROMS NXTのMy Files ⇒ Software Files ⇒ jspを選んで実行する
- 「WAIT BLUETOOTH CONNECTION」の画面が表示されたら、パソコンでTera Termを起動します
- Bluetoothに対応したシリアルポートを開くと、sample1の表示が始まります



TOPPERS

nxtBIOSの対応

- ボード依存部のMakefile.configのDBGENVの設定変更により、nxtBIOS用の実行ファイル(.bin)をビルドできます
  - Config/armv4/mindstormsnxt/Makefile.config
  - DEGENVをRXEからROMに変更
- 再度ビルドを行うと、jsp.binが作られます

```
$ make realclean
$ make depend
$ make
```

- 以下のコマンドでnxtBIOSをインストールしたMINDSTORMS-NXTにアップロードします

```
$ sh appflash.sh
```


TOPPERS

TOPPERS/JSPカーネルの導入

1. 実習内容の説明
2. 開発環境の構築
3. ビルドと実行
4. サンプルプログラムの確認

sample1プログラムの内容

- Sample1プログラムはひとつのメインタスクと3つの並列実行タスクで構成される
- メインタスクは種々のRTOS評価用メッセージを並列実行タスクに送り、並列実行タスクが表示するメッセージにてTOPPERS/JSPのサービスコールと動作の評価を行う
- 並列実行タスクは同一優先度で、待ち状態を作らずに実行する

タスクID	優先度	タスク関数	引数	機能
TASK1	10	task	1	並列実行されるタスクは，task_loop 回空ループを実行する度に，タスク＊ が実行中であることをあらわすメッセージを表示する．
TASK2	10	task	2	
TASK3	10	task	3	
MAIN_TASK	5	main_task	0	コマンドの読み込みと並列タスクへの通知

sample1用コマンド1

- sample1のコマンドは、並列実行タスクに対するコマンドと、全体の制御を行うコマンドがあります
- 以下に全体の制御に関するコマンドを記載します

コマンド	機能
1	以後のコマンドの対称を並列実行タスク1とする
2	以後のコマンドの対称を並列実行タスク2とする
3	以後のコマンドの対称を並列実行タスク3とする
v	発行したシステムコールを表示する（デフォルト）
q	発行したシステムコールを表示しない
r	三つの優先度（9、10，11）のレディキューを回転させる
Q	プログラムを終了する
Ctrl-C	プログラムを終了する

sample1コマンド2

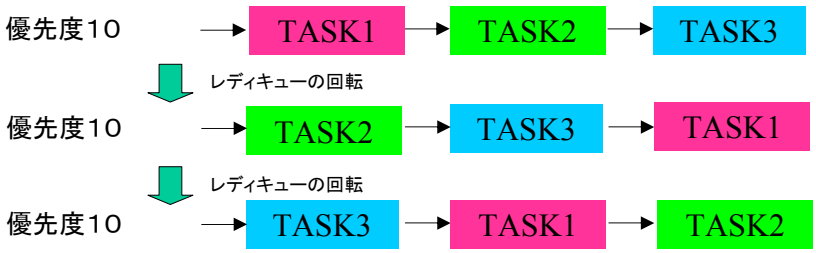
- 並列実行タスクに対する簡単なコマンドは以下の通り

コマンド	機能
a	タスクをact_tskにより起動する
A	タスクに対する起動要求をcan_actによりキャンセルする
e	タスクにext_tskを呼び出させ、終了させる
t	タスクをter_tskにより強制終了させる
s	タスクにslp_tskを呼び出させ、起床待ちにさせる
S	タスクにtslp_tsk（10秒）を呼び出させ、起床待ちにさせる
w	タスクをwup_tskにより起床させる
W	タスクに対する起床要求をcan_wupによりキャンセルする
l	タスクをrel_waiにより強制的に待ち状態にする
＞	タスクの優先度を9（HIGH_PRIORITY）にする
＝	タスクの優先度を10（MID_PRIORITY）にする
＜	タスクの優先度を11（LOW_PRIORITY）にする
d	タスクにdly_tsk（10秒）を呼び出させ、時間経過待ちにさせる

演習：レディキューの回転

- 起床時、3つの並列実行タスクは優先度10でレディ状態ですが、並列実行タスクには待ち状態がないため、レディキューの先頭のTASK1が常に実行されます
- rコマンドでレディキューを回転させ、他のタスクの実行を確認しましょう

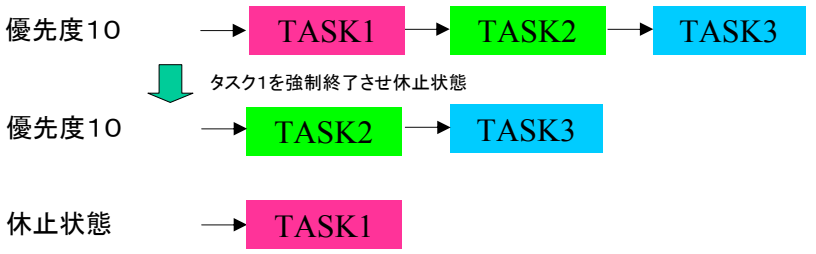
■ 起動時の優先度10（MID_PRIORITY）のレディキューの状態



TOPPERS

演習：タスクの起床と終了

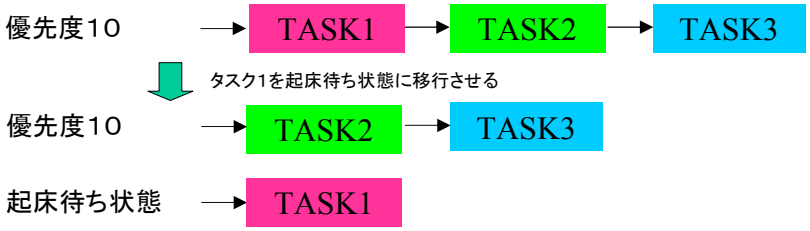
- tコマンドを用いて、TASK1を強制終了させ、レディキューの回転を行っても、TASK1が実行しないことを確認してください
- aコマンドでTASK1を起動させ、レディキューの回転で、TASK1が実行されることを確認してください



TOPPERS

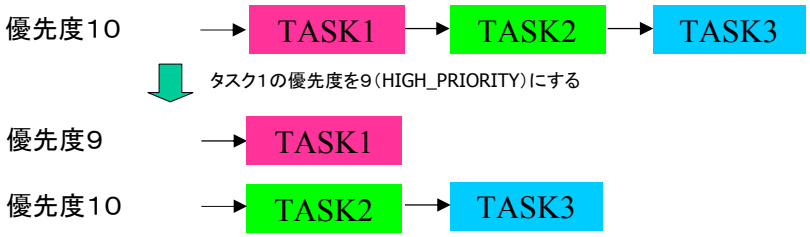
演習：タスクの起床待ち

- sコマンドでTASK1を起床待ち状態に、レディキューの回転でTASK1が実行されないことを確認してください
- wコマンドでTASK1を起床させ、レディキューの回転でTASK1が実行されることを確認してください
- dコマンドでTASK1を時間待ち状態にさせ、10秒間はレディキューの回転を行っても実行しないことを確認してください



演習：優先順位の変更

- ＞コマンドでTASK1の優先度を高くして、レディキューの回転を行っても、TASK1しか実行しないことを確認してください
- ＜コマンドでTASK1の優先度を低くして、レディキューの回転を行っても、TASK1が実行されないことを確認してください



走行体サンプルプログラム

1. プログラム解説
2. プラットフォームの構造
3. デバイスドライバ
4. ログ設定
5. 超音波センサを使う

走行体サンプルプログラムの構成

- OBJ/MINDSTORMSNXT/SAMPLEディレクトリに走行体用のサンプルプログラムが格納されています
- サンプルプログラムは5のファイルで構成されています

ファイル名	機能
sample.cfg	走行体のコンフィギュレーションファイル
sample.c	走行体のC言語サンプルプログラム
sample.h	走行体のインクルードファイル
Makefile	走行体プログラムのメイクファイル
rxeflash.sh	走行体アップロード用のシェルスクリプト

sample.cの解説

- 走行体サンプルはTaskDispタスクとTaskMainタスクで構成されます
- 2つのタスクはITRON仕様のサービスコール以外に、ecrobotプラットフォームは提供する各種のサービスを関数呼び出しにより実行要求する
- LCDのキー操作はプラットフォームの機能となる

タスク	機能
TaskDisp	最初に起動するタスクで、初期化とLCDの表示を行います LCDのバナー表示、bluetoothのコネクション待ち表示、走行中の表示を行います
TaskMain	TaskDispによりタスク起動し、タッチセンサ押下後4MS単位にbalancerモジュールを呼び出し左右のモータに回転指定を行います

TOPPERS

TaskDispの解説1

- TaskDispは、初期化、LCD表示、通信を行います

```
void TaskDisp(VP_INT exinf)
{
        T_SERIAL_RPOR rpor;
        UB buf;
        int wtime = 0;
        char key;
        ER ercd;

        vmsk_log(LOG_UPTO(LOG_INFO), LOG_UPTO(LOG_EMERG));
        syscall(serial_ctl_por(CONSOLE_PORTID, (IOCTL_CRLF|IOCTL_FCSND|IOCTL_FCRCV)));
        show_splash_screen();
        ecrobot_init_nxtstate();
        ecrobot_init_sensors();
        ecrobot_init_sonar_sensor(NXT_PORT_S2);
        show_main_screen();
        display_status_bar();
        ecrobot_sound_tone(1000, 200, 50);
        mstate = MINIT;
        keystate = ecrobot_get_touch_sensor(NXT_PORT_S4);
        nxt_motor_set_count(NXT_PORT_A, 0);
        act_tsk(TASK0);
        while(1){
            ecrobot_poll_nxtstate();
            ercd = serial_ref_por(CONSOLE_PORTID, &rpor);
            if(ercd == E_OK && rpor.reacnt){
```

LCDのバナー表示
状態表示の初期化
センサデバイスの初期化
超音波センサの初期化
Bluetooth接続待ち表示
ブザーを鳴らす
走行体の状態：初期状態
キーセンサ情報の取り込み
倒立モータを倒立位置に
TaskMainを起動
センサ情報取り込み
Bluetoorh受信

TOPPERS

TaskDispの解説2

```
                seril_rea_dat(CONSOLE_PORTID, buf, 1);
                if(buf[0] == 'g' && mstate == MWAIT)
                        mstate = MRUNNING;
                else if(buf[0] == 's' && mstate == MRUNNING){
                        mstate = MSTOP1;
                        wtime = 0;
                }
        }
        key = ecrobot_get_touch_sensor(NXT_PORT_S4);
        if(key != keystate){
                if(mtate == MWAIT)
                        mstate = MRUNNING;
                else if(mstate == MRUNNING){
                        mstate = MSTOP1;
                        wtime = 0;
                }
                keystate = key;
        }
        if(wtime > 0){
                if(--wtime > 0 && mstate >= MSTOP1 && mstate mstate < MPREWAIT){
                        if(++mstate != MPREWAIT)
                                wtime = 1;
                }
        }
        ecrobot_status_monitor("sample JSP"); /* LCD display */
        dly_tsk(40U); /* 40msec wait */
    }
}
```

走行体の状態：RUN

走行体の状態：停止1状態

キーセンサの押下あり

停止状態状態の状態遷移

TaskMainの解説1

- CYC0は周期ハンドラで4MS周期にTaskMainをWakeUpします

```
static void tail_control(signed int angle)
{
        float pwm = (float)(angle – nxt_motor_get_count(NXT_PORTA))*PGAIN;
        if(pwm > PWM_ABS_MAX)
            pwm = PWM_ABS_MAX;
        else if(pwm < -PWM_ABS_MAX)
            pwm = -PWM_ABS_MAX;
        nxt_motor_set_speed(NXT_PORT_A, (signed char)pwm, 1);
}

void cyc0(VP_INT exinf)
{
        isig_sem(EVT_SEM);
}

void TaskMain(VP_INT exinf)
{
        signed char forward;      /* 前後進命令: -100(後進)～100(前進)   */
        signed char turn;         /* 旋回命令: -100(左旋回)～100(右旋回) */
        signed char pwm_L, pwm_R; /* 左右モータPWM出力 */
```

直立モータの角度指定

セマフォを使ってTaskMainをWakeUp

TaskMainの解説2

```
        mstate = MWAIT;
        ecrobot_set_light_sensor_active(NXT_PORT_S3);      /* 光センサ赤色LEDをON */
        while (mstate != MRUNNING){                        /* タッチセンサ押下待機 */
                tail_control(TAIL_ANGLE_STAND_UP);  /* 完全停止角度に制御 */
                dly_tsk(10);                                                    走行体の状態：走行待ち状態
        }
        balance_init();                                    /* 倒立振子制御初期化 */
        nxt_motor_set_count(NXT_PORT_C, 0);        /* 左モータエンコーダリセット */
        nxt_motor_set_count(NXT_PORT_B, 0);        /* 右モータエンコーダリセット */
        forward = 50;                                      /* 前進命令 */
        sta_cyc(CYC0);
        while(mstate < MPREWAIT){                                               走行体の状態：走行状態
                if(mstate == MRUNNING){
                        tail_control(TAIL_ANGLE_DRIVE);  /* バランス走行用角度に制御 */
                        if(ecrobot_get_light_sensor(NXT_PORT_S3) >= (WHILE+BLACK)/2)
                                turn = 50;
                        else
                                trun = -50;
                }                                                               走行体の状態：停止待ち状態
                else if(mstate == MSTOP1){
                        nxt_motor_set_count(NXT_PORTA, 0); /* 完全停止用モータエンコーダリセット */
                        mstate = MSTOP2;
                }                                                               走行体の状態：停止状態
                else{
                        tail_control(TAIL_ANGLE_STAND_UP); /* 完全停止角度に制御 */
                        forward = 0;                               /* 停止命令 */
                        turn = 0;
                }
```


TaskMainの解説3

- while文内は4ms周期で動作する

```
                /* 倒立振子制御(forward = 0, turn = 0で静止バランス) */
                balance_control(
                        (float)forward,                         /* 前後進命令              */
                        (float)turn,                            /* 旋回命令                */
                        (float)ecrobot_get_gyro_sensor(NXT_PORT_S4), /* ジャイロセンサ値          */
                        (float)GYRO_OFFSET,                     /* ジャイロセンサオフセット値 */
                        (float)nxt_motor_get_count(NXT_PORT_C),     /* 左モータ回転角度[deg]     */
                        (float)nxt_motor_get_count(NXT_PORT_B),     /* 右モータ回転角度[deg]     */
                        (float)ecrobot_get_battery_voltage(),       /* バッテリ電圧[mV]          */
                        &pwm_L,  /* 左モータPWM出力値 */
                        &pwm_R); /* 右モータPWM出力値 */
                nxt_motor_set_speed(NXT_PORT_C, pwm_L, 1); /* 左モータPWM出力セット */
                nxt_motor_set_speed(NXT_PORT_B, pwm_R, 1); /* 右モータPWM出力セット */

                wai_sem(EVT_SEM);       /* 4MS待ち */
        }
        ecrobot_set_motor_speed(NXT_PORT_C, 0);   /* 左モータPWM出力セット */
        ecrobot_set_motor_speed(NXT_PORT_B, 0);   /* 右モータPWM出力セット */
        ecrobot_sound_tone(1000, 20, 50);
        ack_tsk(TASK0);                                                         TaskMainを再起動
}
```


コンフィグレーションファイル

- 走行体サンプルで使用するRTOSオブジェクト設定を行う
- プラットフォームのオブジェクトはecrobot.cfgに記載

```
#define _MACRO_ONLY
#include "sample.h"
INCLUDE("¥"sample.h¥"");

CRE_TSK(TASK0, { TA_HLNG , 0, TaskMain, 2, 1024, NULL });
CRE_TSK(TASK1, { TA_HLNG | TA_ACT, 0, TaskDisp, 10, 1024, NULL });

CRE_SEM(EVT_SEM, { TA_TPRI, 0, 1 });

CRE_CYC(CYC0, { TA_HLNG, 0, cyc0, 4, 0 });

#include "../../../nxt/ecrobot/ecrobot_jsp.cfg"
#include "../../../systask/timer.cfg"
#include "../../../systask/serial.cfg"
#include "../../../systask/logtask.cfg"
```

タスクの設定

セマフォの設定

周期ハンドラの設定

ecrobotプラットフォームのコンフィギュ設定

Makefile

- Makefileの拡張部分

```
#
#  共通コンパイルオプションの定義
#
COPTS := $(COPTS)
CDEFS := $(CDEFS)
INCLUDES := -I. -I$(SRCDIR)/include -I$(SRCDIR)/nxt/device -I$(SRCDIR)/nxt/ecrobot

 -I$(SRCDIR)/nxt/balancer  $(INCLUDES)
LDFLAGS := -nostdlib $(LDFLAGS)
LIBS := $(LIBS) $(CXXLIBS) -lc -lgcc
CFLAGS = $(COPTS) $(CDEFS) $(INCLUDES)

#
#  アプリケーションプログラムに関する定義
#
UNAME = sample
UTASK_CFG = $(UNAME).cfg

UTASK_DIR = $(SRCDIR)/library
UTASK_ASMOBJS =
ifdef USE_CXX
    UTASK_CXXOBJS = $(UNAME).o
    UTASK_COBJS =
else
    UTASK_COBJS = $(UNAME).o
endif
UTASK_CFLAGS =
UTASK_LIBS = $(SRCDIR)/nxt/libecrobot.a $(SRCDIR)/nxt/libdevice.a $(SRCDIR)/nxt/libbalancer.a
```

プラットフォームのインクルードパス指定

ソースファイルの取り込み

プラットフォームのプログラムをライブラリィで取得

光センサーとジャイロセンサーデフォルト値設定

- サンプルソースの垂直位置のジャイロセンサー値と白黒の光センサー値は機種固有で異なります
- 自分の走行体にあった値を設定しなければなりません
- この値はプラットフォームの標準的なLCD表示（次の章で説明）から読み取れます

```
#include <s_services.h>
#include "sample.h"
#include "kernel_id.h"
#include "ecrobot_base.h"
#include "ecrobot_interface.h"
#include "balancer.h" /* 倒立振子制御用ヘッダファイル */

/* 下記のパラメータはセンサ個体/環境に合わせてチューニングする必要があります */
#define GYRO_OFFSET  589 /* ジャイロセンサオフセット値(角速度0[deg/sec]時)  */
#define WHITE        500 /* 白色の光センサ値 */
#define BLACK        700 /* 黒色の光センサ値 */
```


TOPPERS

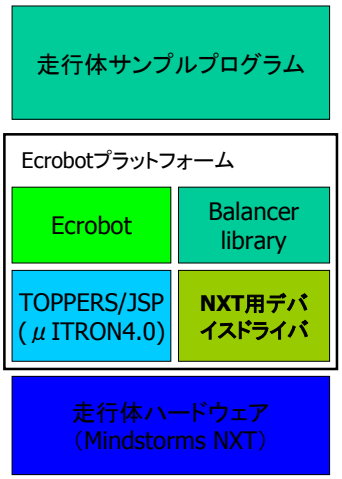
走行体サンプルプログラム

1. プログラム解説
2. プラットフォームの構造
3. デバイスドライバ
4. ログ設定
5. 超音波センサを使う

TOPPERS

Ecrobotプラットフォームがアプリサービスを行う

- Ecrobotプラットフォームの機能
  - ユーザデバイスドライバ
  - LCDの各種表示
  - キー処理
  - エクセプション発生時処理
- 走行体サンプルプログラムから利用できるI/F
  - μITRON4．0仕様スタンダードプロファイル
  - EcrobotデバイスドライバI/F
- プラットフォームのプログラムはライブラリィで提供



アプリケーションサービス

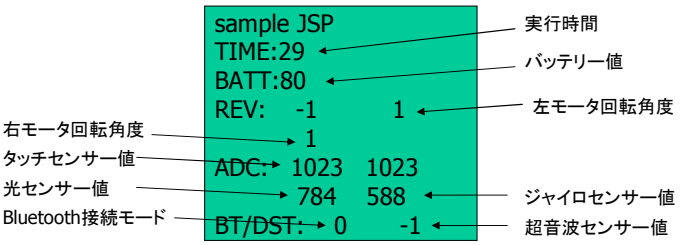
- アプリケーションに対して種々のサービスを行います
- サービスは関数コールの形となります
- 詳細はecrobot_intercae.hを参照してください
- プロトタイプ宣言は以下のインクルードファイルで定義
  - ecronot_base.h ecrobot_interface.h

```
例）ジャイロセンサー値の取り込み
U16 ecrobot_get_gyro_sensor(U8 port_id);
port_idは接続するポートのID、NXT_PORT_S4 = 3
取得値は0から1023までの値

例）超音波センサー値の取り込み
S32 ecrobot_get_sonar_sensor(U8 port_id);
port_idは接続するポートのID、NXT_PORT_S2=1
取得値は0から255までの値
```

標準的なLCD表示

- 円滑に実行状態を認識できるように、標準的なLCD表示関数を用意してあります
  - show_splash_screen　バナー表示
  - show_main_screen　キーの部品表示
  - display_status_bar　Bluetoothの状態表示
  - ecrobot_status_monitor 走行体のデバイス状態表示



TOPPERS

キー処理

- キーの取り込みように2つの取得関数があります
  - ecrobot_init_nxtstate　取得情報の初期化
  - ecrobot_poll_nxtstate　情報の取得
- 電源OFF（EXIT_PRESSED）、リスタート（STOP_PRESSED）、スキップ（RUN_PRESSED）のセンスと実行はcheck_NXT_buttons関数で処理します
- この関数はコンフィギュレーションファイル中の周期ハンドラを定義するCRE_CYC静的APIで10ms周期で自動実行するように指定してあります

```
CRE_CYC(CYC2, { TA_HLNG|TA_STA, 0, check_NXT_bottons, 10, 0 });
```


TOPPERS

エクセプションの表示

- Bluetoothによるログ表示ではタスクを使ったデータ通信のため、割込みやエクセプション中の表示には使用できません
- データエラーやインストラクションエラーなどの実行継続できないエクセプションが発生した場合、LCDに表示し、処理停止を行います
- コンフィギュレーションファイル中のDEF_EXC静的APIで1番と4番のエクセプションを設定しなおしています

```
#ifndef CPUEXC1
DEF_EXC(1, { TA_HLNG, instruction_abort} );
DEF_EXC(4, { TA_HLNG, data_abort} );
#endif /* CPUEXC1 */
```


走行体サンプルプログラム

1. プログラム解説
2. プラットフォームの構造
3. デバイスドライバ
4. ログ設定
5. 超音波センサを使う

MINDSTORMS NXTのハードウェア構成

- NXTではユーザから見たデバイスI/FとAT91SAMS上のデバイスがオーバーラップしており、デバイスからI/Fデバイスを呼び出すものも多い（ユーザに提示しないI/Fもある）

デバイス	I/F	使用モジュール	機能	備考
NXT_AVR	TWI	TWI	電源、電圧、キー情報取得	
MOTOR	TWI	TWI+PIO	モータ制御	
NXT_LCD	SPI	SPI	LCD表示	
I2C	I2C	PIO+TC0	超音波センサデバイスとの通信	ソフト制御
USB-DEVICE		UDP	USBデバイス	
PIT		PIT	システムタイマー	RTOS使用
SOUND		SSC+PIO	サウンド機能	
BLUETOOTH	UART	US1+TC2	BLUETOOTHデバイスとの通信	RTOS使用
SENSOR	TWI	TWI+PIO	センサデバイスとの通信	
UART	UART	US0(,DEBU)	シリアル通信	コネクタなし

NXT用デバイスインターフェイス1

- I2cは直接ユーザからは呼び出さない

デバイス	主要な関数	基本機能	内容

nxt_avr	void nxt_avr_init(long exinf);
	void nxt_avr_power_down(void)
	U32 buttons_get(void);
	void twi_isr_C(void);
		電源制御、KEYのセンス	twiにてデバイスと通信、twi割込みと1msの周期ハンドラで制御

i2c	void i2c_init(long exinf);
	void i2c_disable(int port);
	void i2c_enable(int port);
	int i2c_busy(int port);
	int i2c_start_transaction();
	void i2c_timer_isr_C(void);
		超音波センサデバイスとの通信	Timer#0の割り込みを処理用に使用している。直接ソフト制御でI2C-BUSを制御している

motor	void nxt_motor_init(long exinf);
	void nxt_motor_set_speed();
	void nxt_motor_set_count();
	void nxt_motor_get_count();
	void nxt_motor_isr_C(void);
		3つまでのモータの制御	PIOA,B,Cにて割込みを用いて制御、1msの周期ハンドラでセンシングしている

NXT用デバイスインターフェイス2

- DisplayはLCDドライバを呼び出す

デバイス	主要な関数	基本機能	内容

sound	void sound_init(long exinf);
	void sound_freq_vol();
	int sound_get_time(void);
	void sound_play_sample();
	void sound_isr_C(void);
		音の制御	サウンド用のLSI-SSC音の制御、SSCの割込みを使用する

sensor(pio)	void init_sensors(void); void read_buttons(int, short *);	センサ値の取得	PIOのポーリング、twi経由のデータ取得

display	void disply_init(long exinf);
	void display_update(void);
	void display_force_update();
	void display_clear(U32 u);
	void display_goto_xy();
	void display_string();
	void display_hex();
	void spi_isr_C(void);
		LCDの表示	SPI-I/FによるLCDドライバの制御、SPIの割込みを使用する

NXT用デバイスインターフェイス3

- TWI、SPIはデバイスとの通信規格です
- これらのデバイスはサブプロセッサやLCDモジュールとの通信を行うためのもので、ユーザから隠蔽されています

デバイス	主要な関数	基本機能	内容

twi	int twi_init(void);
	void twi_reset(void);
	void twi_start_read();
	void twi_start_write();
	void twi_isr_C(void);
		TWI通信機能	TWIにてサブプロセッサとの通信を行っている。サブプロセッサは多機能である

spi	int nxt_spi_init(void);
	void nxt_spi_init(void);
	void nxt_spi_write();
	int nxt_spi_display();
	void nxt_spi_refresh();
	void spi_isr_C(void);
		SPI通信機能	SPIにてLCDモジュールと通信を行っている

NXT用デバイスインターフェイス4

- BLUETOOTHとPITはRTOSで指定したI/Fで使用する。

デバイス	主要な関数	基本機能	内容

udp(usb)	int udp_init(void);
	void udp_reest(void);
	int udp_write();
	int udp_read();
	void udp_isr_C(void);
		USBの制御	USB DEVICEの制御、UDP割込みを用いたコントロール通信,BULK_IN,BULK_OUT（現状未デバッグ）

uart	void uart_init(U32,…);
	void uart_close(U32);
	void uart_get_byte(U32, U8*);
	void uart_put_byte(U32, U8);
	void uart_put_str(U32, U8*);
	void uart_isr_C_0(void);
	void uart_isr_C_1(void);
		シリアル通信	UARTを用いたシリアル割込み通信（UART1はbluetoothに対応で使用不可）

走行体サンプルプログラム

1. プログラム解説
2. プラットフォームの構造
3. デバイスドライバ
4. ログ設定
5. 超音波センサを使う

演習:ジャイロオフセット値を自動設定する

- ジャイロオフセット値は走行体は直立状態のジャイロセンサ値です
- ジャイロセンサの個体差により、サンプルプログラムでは設定値(GYRO_OFFSET)を変更する必要があります
- タッチセンサ押下時、走行体が直立状態として、その時点でのジャイロセンサの値をGYRO_OFFSETとするようにプログラムを修正してください
- また、設定したGYRO_OFFSET値がbluetoothを通して確認できるようにログ表示してください

演習:ログ表示の方法

- sample1のプログラムではsyslog関数を用いてログの表示を行っていた。
- 表示はシリアルデバイス(Bluetooth)を介して行われる。通信モードはserial_ctl_por関数で設定される

```
/*
 *  ログ情報(コメント)を出力するためのライブラリィ関数(vasyslog.c)
 */
extern ER syslog(UINT prio, const char *format, ...) throw();
```

```
extern ER serial_ctl_por(ID portid, UINT ioctl) throw();
```

定義	値	機能
IOCTL_NULL	0x0000u	指定なし
IOCTL_ECHO	0x0001u	受信した文字をエコーバック
IOCTL_CRLF	0x0010u	LFを送信する前に、CRを付加する
IOCTL_FCSND	0x0100u	送信に対してフロー制御を行う
IOCTL_FCANY	0x0200u	どのような文字でも送信再開
IOCTL_FCRCV	0x0400u	受信に対してフロー制御を行う

通信モードの設定

ログ表示の重要度レベルの設定

- 重要度指定とvmsk_log関数による重要度設定によって出力されるログ情報を指定できます
- Logmaskはログ記録の有無の指定、lowmaskを低レベル出力するかどうかの判定、本システムでは低レベル出力はサポートしていません

```
/*
 *  出力すべきログ情報の重要度の設定
 */
extern ER vmsk_log(UINT logmask, UINT lowmask) throw();
```

重要度	値	定義
LOG_EMERG	0	シャットダウンに値するエラー
LOG_ALERT	1	
LOG_CRIT	2	
LOG_ERROR	3	システムエラー
LOG_WARING	4	警告メッセージ
LOG_NOTICE	5	
LOG_INFO	6	
LOG_DEBUG	7	デバッグ用メッセージ

プログラムの修正

- sample.cを修正して、gyro_offsetを変数指定に変更します
- タッチセンサ押下時に、ジャイロセンサの値を読みこみgyro_offsetに設定するように変更します

```
static char mstate;                                    /* 走行体の状態 */
static char keystate;                                  /* タッチセンサーの状態 */

static int gyro_offset;
//*****************************************************************************
// 関数名：tail_control
// 引数 ：angle (モータ目標角度[度])
// 返り値：無し
// 概要：走行体完全停止用モータの角度制御
//*****************************************************************************
static void tail_control(signed int angle)
{
        float pwm = (float)(angle - nxt_motor_get_count(NXT_PORT_A))*P_GAIN; /* 比例制御 */
        /* PWM出力飽和処理 */
        if(pwm > PWM_ABS_MAX)
```


TaskMain関数の修正１

- TaskMain関数を修正します
- タッチセンサ押下待機の後、gyro_offset変数に現在のジャイロセンサの値を取り込みログ表示します

```
void TaskMain(VP_INT exinf)
{
        signed char forward;     /* 前後進命令: -100(後進)～100(前進)   */
        signed char turn;        /* 旋回命令: -100(左旋回)～100(右旋回) */
        signed char pwm_L, pwm_R; /* 左右モータPWM出力 */

        mstate = MWAIT;
        ecrobot_set_light_sensor_active(NXT_PORT_S3);      /* 光センサ赤色LEDをON */
        while(mstate != MRUNNING){                         /* タッチセンサ押下待機 */
                tail_control(TAIL_ANGLE_STAND_UP);   /* 完全停止用角度に制御 */
                dly_tsk(10);
        }
        gyro_offset = ecrobot_get_gyro_sensor(NXT_PORT_S4);
        syslog(LOG_NOTICE, "gyro offset def=%d cur=%d", GYRO_OFFSET, gyro_offset);

        balance_init();                                    /* 倒立振子制御初期化 */
        nxt_motor_set_count(NXT_PORT_C, 0); /* 左モータエンコーダリセット */
        nxt_motor_set_count(NXT_PORT_B, 0); /* 右モータエンコーダリセット */
        forward = 50;                                                  /* 前進命令 */
        sta_cyc(CYC0);
```


TOPPERS

TaskMain関数の修正２

- Balance_control関数に渡すジャイロセンサオフセット値をGYRO_OFFSET（定数）からgyro_offset（変数）に変更します

```
        /* 倒立振子制御(forward = 0, turn = 0で静止バランス) */
        balance_control(
                (float)forward,                    /* 前後進命令              */
                (float)turn,                       /* 旋回命令                */
                (float)ecrobot_get_gyro_sensor(NXT_PORT_S4), /* ジャイロセンサ値          */
                (float)gyro_offset,                /* ジャイロセンサオフセット値 */
                (float)nxt_motor_get_count(NXT_PORT_C),    /* 左モータ回転角度[deg]     */
                (float)nxt_motor_get_count(NXT_PORT_B),    /* 右モータ回転角度[deg]     */
                (float)ecrobot_get_battery_voltage(),      /* バッテリ電圧[mV]          */
                &pwm_L,  /* 左モータPWM出力値 */
                &pwm_R); /* 右モータPWM出力値 */
        nxt_motor_set_speed(NXT_PORT_C, pwm_L, 1); /* 左モータPWM出力セット */
        nxt_motor_set_speed(NXT_PORT_B, pwm_R, 1); /* 右モータPWM出力セット */
```


TOPPERS

TaskDisp関数の確認

- TaskDisp関数スタート時に、vmsk_log関数を用いて表示レベルの設定とserial_ctl_por関数を用いて、シリアルデバイスの通信モードを設定します
- Bluetoothを使用したシリアルデバイスのポートIDはCONSOLE_PORTIDです

```
void TaskDisp(VP_INT exinf)
{
        T_SERIAL_RPOR rpor;
        UB buf[4];
        int wtime = 0;
        char key;
        ER ercd;

        vmsk_log(LOG_UPTO(LOG_INFO), LOG_UPTO(LOG_EMERG));
        serial_ctl_por(CONSOLE_PORTID,      (IOCTL_CRLF | IOCTL_FCSND | IOCTL_FCRCV));
        show_splash_screen();
        ecrobot_init_nxtstate();
        ecrobot_init_sesors();
```


演習：gyro_offsetの値を動的に変更します

- 走行体動作中に、gyro_offsetの値をbluetooth通信を用いて変更し、走行体の動作を確認しましょう
- TeraTermより'u'（up）、'd'（down）のキーを押し
  - upの場合、gyro_offsetを2プラスします
  - downの場合、gyro_offsetを2マイナスします
- 変更後のgyro_offsetの値をログ表示します
- 走行体の動作がどのように変化するか確認してください

シリアルデバイスの受信データ取り込み

- シリアルデバイスの受信レングスを取り出す関数（serial_ref_por）とシリアルデバイスから受信データを取り出す関数（serial_rea_dat）を用いてパソコンからのデータを取り出します

```
/*
 *  シリアルインターフェイスドライバの用いるパケット
 */
typedef struct {
        UINT reacnt;        /* 受信バッファ中の文字数 */
        UINT wricnt;        /* 送信バッファ中の文字数 */
    } T_SERIAL_RPOR;
/*
 *  シリアルポート状態の参照
 */
extern ER serial_ref_por(ID portid, T_SERIAL_RPOR *pk_rpor) throw();
/*
 *  シリアルポートからの受信
 */
extern ER_UINT serial_rea_dat(ID portid, char *buf, UINT len) throw();
```


TaskDisp関数の改造

- TaskDisp関数にキー入力確認とgyro_offset変更が行えるように修正します

```
while(1){
    ecrobot_poll_nxtstate();
    ecrobot_get_sonar_sensor(NXT_PORT_S2);
    ercd = serial_ref_por(CONSOLE_PORTID, &rpor);
    if(ercd == E_OK && rpor.reacnt > 0){
      serial_rea_dat(CONSOLE_PORTID, buf, 1);
        if(buf[0] == 'g' && mstate == MWAIT)
            mstate = MRUNNING;
        else if(buf[0] == 's' && mstate == MRUNNING){
            mstate = MSTOP1;
            wtime = STOPWAIT;
        }
        else if(buf[0] == 'u'){
            gyro_offset += 2;
            syslog(LOG_NOTICE, "[up]gyro offset def=%d cur=%d", GYRO_OFFSET,
                        gyro_offset);
        }
        else if(buf[0] == 'd'){
            gyro_offset -= 2;
            syslog(LOG_NOTICE, "[down]gyro offset def=%d cur=%d", GYRO_OFFSET,
                        gyro_offset);
        }
    }
```

キー入力有り無しの判定

キーデータの取り込み

走行体サンプルプログラム

1. プログラム解説
2. プラットフォームの構造
3. デバイスドライバ
4. ログ設定
5. 超音波センサを使う

演習：超音波センサを用いた自動停止機能

- 超音波センサを用いて障害物を検知すると自動停止するように改造しましょう
- 超音波センサはNXT_PORT_S2に接続しており、以下の関数で障害物までの距離を取り出せます。
  - ecrobot_get_sonar_sensor(NXT_PORT_S2);
  - 戻り値は0～255までの値でcmです
- 前進の設定はTaskMain中の変数forwardの値がゼロで停止、正の値で前進、負の値で後退です
- 超音波センサの値の設定はTaskDispで行うようにします

forwardを自動変数を外部関数から設定できるように

- forward自動変数を、他のタスク（関数）から設定できるように、関数外の変数に修正します
- ここでは、他のソースから変更できないようにstatic関数とします

削除して移動

```
static signed char forward;     /* 前後進命令: -100(後進)～100(前進)  */
static int  gyro_offset;
//*****************************************************************************
//  関数名：tail_control
// 引数  ：angle (モータ目標角度[度])
      :
//*****************************************************************************
//  タスク名：TaskMain
//  概要      ：メインタスク
//*****************************************************************************
void TaskMain(VP_INT exinf)
{
        signed char turn;        /* 旋回命令: -100(左旋回)～100(右旋回) */
        signed char pwm_L, pwm_R; /* 左右モータPWM出力 */
```


TOPPERS

停止時、回転をやめる

- 前進値がゼロの場合、回転値をゼロに設定

```
void TaskMain(VP_INT exinf)
{
    signed char turn;        /* 旋回命令: -100(左旋回)～100(右旋回) */
    signed char pwm_L, pwm_R; /* 左右モータPWM出力 */

    ecrobot_set_light_sensor_active(NXT_PORT_S3);      /* 光センサ赤色LEDをON */
    while (ecrobot_get_touch_sensor(NXT_PORT_S1) == 0){ /* タッチセンサ押下待機 */
      dly_tsk(10);
    }
      ..........
    forward = 50;                                              /* 前進命令 */
    sta_cyc(CYC0);
    while(1){
        if (forward == 0)
            turn = 0;
        else if (ecrobot_get_light_sensor(NXT_PORT_S3) <= (WHITE + BLACK)/2)
            turn = 50;  /* 右折 */
        else
            turn = -50; /* 左折 */
```


TOPPERS

超音波センサの取り込み

- 超音波センサの取得値を参照して、forwardの値を補正する機能をTaskDispに追加する

```
void TaskDisp(VP_INT exinf)
{
        ecrobot_init_sonar_sensor(NXT_PORT_S2); /* 超音波センサ(I2C通信)を初期化 */
        .........
        while(1){
                ecrobot_poll_nxtstate();
                 ecrobot_get_sonar_sensor(NXT_PORT_S2);
                 ercd = serial_ref_por(CONSOLE_PORTID, &rpor);
                    .........
                if(ecrobot_get_sonar_sensor(NXT_PORT_S2) <= 30){   /* 距離測定 */
                      if(forward != 0)
                           syslog(LOG_NOTICE, "[sonar sense]=%d",
                                        ecrobot_get_sonar_sensor(NXT_PORT_S2));
                       forward = 0;
                }
                else{
                       forward = 50;
                }
         }
```

ソナーセンサの読み取り値でforwardを変更するように修正

TOPPERS/JSP導入実習のまとめ

- コンパイルやリンク等の開発環境とターゲットボードがあれば、比較的簡単にTOPPERS/JSPの動作確認が行える
- ソースが供給されているので、RTOSのしくみを理解しやすい、問題発生時もソースを用いた問題解析が可能である
- RTOSを使用する中規模組込み開発では、多人数の開発者が分業して多くのプログラム開発やミドルウェアの導入を行うことが多い、タスクモニタをシステム用に機能拡張し、問題発生時に、問題の要因を簡単に判別できる開発環境を用意していくことが、開発効率や品質の向上につながる

まとめ

リアルタイムOSを用いたシステム設計

- リアルタイムOSは、プログラム規模が大きい中規模組込みシステムで使用すると効果的です
- リアルタイムOSは、CPU処理の代行を行う機能が主で、システム設計を行う場合、機器の機能に対応したベース機能を含んだプラットフォームを構築すると、開発工数の削減や品質の向上につながります
- プラットフォームは、機器の機能に応じたメンテナンス機能を用意した方が、効率的な開発が行えます
  - メンテナン機能の代表がタスクモニタです
- システム設計管理には、熟練したシステム管理者が必要です

μITRON仕様に関して

- μITRON仕様は、日本の組込みシステムでもっと多く使われているリアルタイムOSの仕様です
- リアルタイムOSは、組込み機器が多くのバリエーションを持つためCPUの機能の代行する機能に限定した仕様となっています
- タスクを用いたシステムでは、同期や通信などの機能を用いて各機能が円滑に動作するようにシステム設計する必要があります
- TOPPERS/JSPはμITRON4．0のスタンダードプロファイルに準じたリアルタイムOSで、TOPPERS/ASPはμITRON4.0仕様に新たな機能を付加し、オープンソースとして供給されています

著者リスト

- 開発環境とプラットフォーム
  - 竹内 良輔（(株)リコー）
- ITRON仕様について学ぶ
  - 高田 広章（名古屋大学）, 本田 晋也（名古屋大学）
- TOPPERS/JSPの導入
  - 竹内 良輔（(株)リコー）
- 走行体サンプルプログラム
  - 竹内 良輔（(株)リコー）

謝辞

本教材の開発において、NPO法人組込みソフトウェア管理者・技術者育成研究会およびNPO法人TOPPERSプロジェクトの作成した以下の教材を参考としました.

- OpenSESSAME Seminar組込みソフトウェア技術者・管理者向けセミナー初級者向けテキスト第5版
- TOPPERS初級実装セミナー教材

両団体および上記教材の作者の皆様へ感謝します.

本教材の開発において、NEXCESS初級コースおよび指導者養成コースの受講者の皆様のコメントを参考としました. 両コースの受講者の皆様へ感謝します.

謝辞

- プログラムの開発にあたり、以下のプログラムを参考としました.
  - 2009年度版ETロボコン用TOPPERS/JSP
  - LeJOS-OSEKnxt rev1.1.1.1