國立屏東大學 即時與嵌入式系統實驗室

Xenomai

Prerequisite

在學習Xenomai前應具備基礎以上的作業系統知識、平行運算(Parallel Computing)及緒程式設計(Threading Programming)可先參考以下連結：

Parallel Computing

- Introduction to Parallel Computing Tutorial(HPC@LLNL)

PThreads Programming

- POSIX Threads Programming(HPC@LLNL)
- PThreads Programming

關於PThreads的程式範例

General Overview

Xenomai專案是讓Linux作業系統提供即時性的多種嘗試之一，相較於原本的Kernel，Xenomai在處理搶先(Preemption)時可以滿足更為嚴格的回應時間(Response Time)需求。具體來說，Xenomai為Linux系統新增了一個新的內核來專責處理具有時間要求的即時工作，並使用既有的內核來處理其它的工作，我們將這種運作方式稱為雙內核架構(Dual Kernel Architecture)，在此架構下，既有的內核與新增的內核各自執行它們所負責的工作，但新增的內核將以高於既有內核的優先權運行。

Xenomai 3有兩種不同的組態：

1. 雙內核(Dual Core，Codename Cobalt)，Xenomai實作了一個稱為Cobalt的內核，並與Linux原有的內核形成一個具有雙內核(Dual Kernel)的系統。在運行時Cobalt，負責接收處理所有來自硬體的搶先）將原有的內核視為一個閒置工作(Idle Process)，等到Cobalt沒有事情需要處理時，才會改由Linux原有的內核接手處理。如此可以讓「搶先(Preemption)」更有快速地被處理。
2. 單內核(Single Core，Codename Mercury):Xenomai的另一種實作稱為Mercury，其是以原有的Linux內核為基礎，使用PREEMPT-RT[1)2)]進行修補(Patch)以增進搶先處裡的即時性。

How to pronounce Xenomai? Xenomai有三個音節，分別是Xe no mai，可唸做「ㄌㄟ」.「ㄋㄨ」.「ㄇㄞˇ」）

關於以上的兩種實作，Huang等人[3)] 已經實驗驗證了雙內核的Cobalt效能優於單內核的Mercury.

雙內核/Cobalt

Cobalt是和Linux原有的內核共同運行的一個內核。當系統以這種**雙內核(Dual Kernel)**方式運行時，Cobalt負責所有時間攸關的工作[4]，例如中斷處理、即時執行緒(real-time threads)的排程等；其餘的工作則由原本的Linux內核加以執行。當然，從系統整體的角度來看，由Cobalt所負責的工作將會以較高的優先權執行，而由Linux原內核所執行的工作之優先權則相對較低，Xenomai還提供了實作在libcobalt函式庫的RTOS API，呼叫它們就可以透過Cobalt內核來進行具有即時性[5]的操作，其中包括了Xenomai 3所實作的部份POSIX 1003.1c服務。

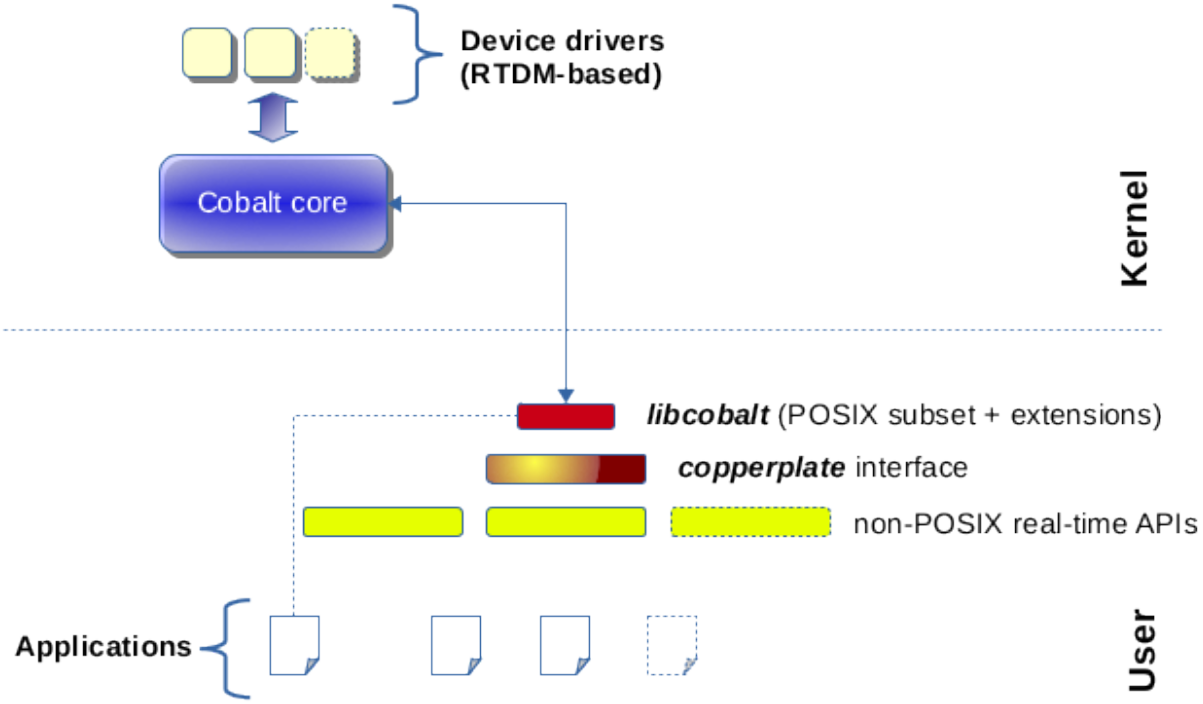

Fig. 1: Cobalt架構圖

在figure 1左上方的device drivers(裝置驅動程式)是指依據Xenomai所實作的Real-Time Driver Model(RTDM，即時驅動程式模型)[6][7]，使用者模式(User Mode)的應用程式(User Applications)可以透過對libcobalt裡的API呼叫，來經由Colblt內核存取(操作)這些裝置驅動程式。值得一提的是，在libcobalt裡的API只是POSIX 1003.1c一部份的實作，若是在應用程式裡要使用非POSIX 1003所定義的即時操作(換句話說，進行非POSIX的即時呼叫)，則是透過cooperplate介面來進行對應。

Xenomai 3使用Optimistic interrupt protection 機制減少 中斷遮罩(Interrupt Mask)的使用（changing the interrupt mask) – 一般的做法在每次進入critical section時都要中斷遮罩(interrupt mask)，而Optimistic interrupt protection可以不用；因此Xenomai可以有效減低latency(因為Latency很大因素就是來自於interrupt handling）

單內核/Mercury

Mercury使用PREEMPT-RT Patch讓Linux系統原有的內核能夠運行即時工作。相較於Cobalt，Mercury是一個單內核的解決方案。通常應用程式若要進行即時的操作，必須先確保Linux內核有啟動(Enable)PPEEMPT-RT，不過視應用程式對即時性要求的不同，若是有比較寬鬆的回應時間(Responsiveness)或是較大的完成時間(Jitter)要求等(甚至是可以允許一並比例的工作錯失截限時間),

有時也可以沒有PREEMPT-RT的情況下執行。

在Mercury這種單內核的組態下，所有Xenomai 3所提供的非POSIX的API都有在原本的執行緒函式庫(包含NPTL以及傳統的linuxthreads)進行精準的模擬。

Xenomai在Mercury這種單內核的組態下，所提供的所有非POSIX的API都在可以在原生的執行緒函式庫(包含NPTL以及傳統的linuxthreads)得到精確的??

In this single kernel configuration, all the non-POSIX RTOS APIs Xenomai 3 provides are accurately emulated over the native threading library (preferably NPTL, but also supports linuxthreads for legacy setups).

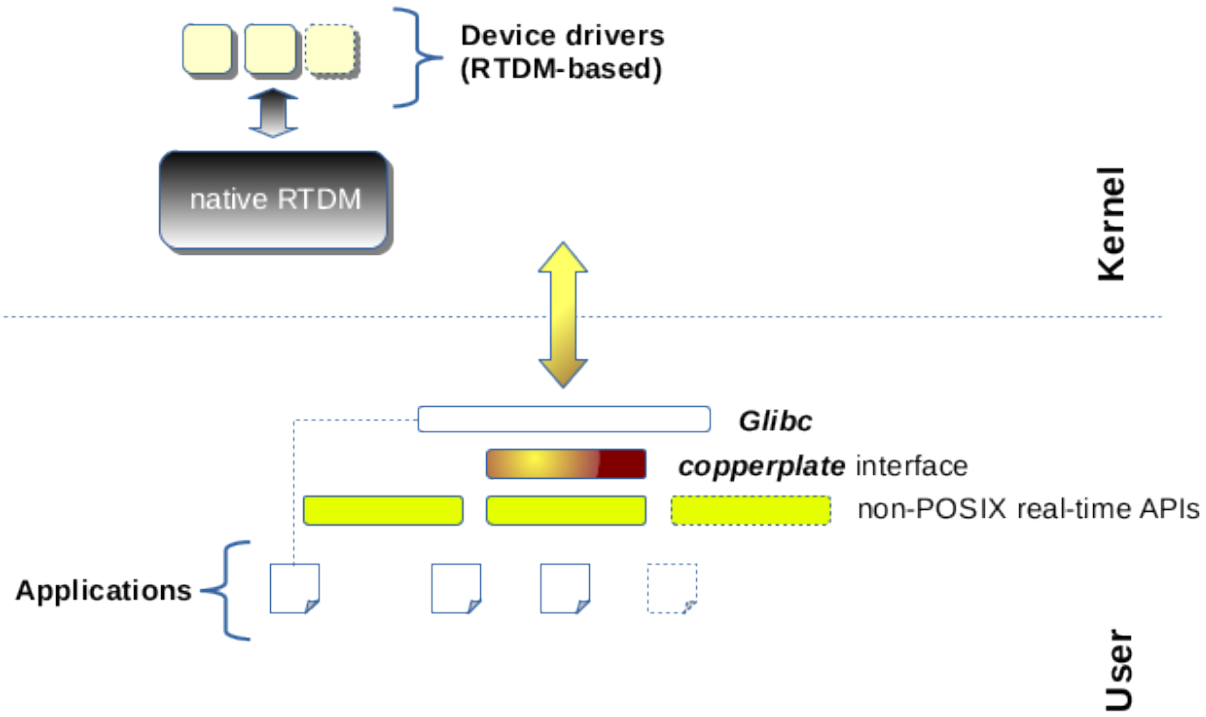

Fig. 2: Mercury架構圖

如何選擇適合的Xenomai組態

若是應用程式對於即時性的要求並沒有太嚴格，且能夠接受在不具即時性的內核上執行RTOS API以及高延遲的搶先處理(也就是Linux原本的CONFIG_PREEMPT)那麼就適合使用Xenomai 3的Mercury單內核組態 — 因為比起原始的Linux的系統Mercury讓應用程式在執行上更能滿足時間要求(也就是能夠比起原本的Linux系統更好，或者反過來說，至少不會更糟)。

但是當應用程式對於即時性的要求較高時Mercury可能就不適合。

一個名為Dovetail的機制(Kernel軟體層)，用以將所有關鍵的事件送到延伸的第二內核(dual kernel extension core)

同步方法的比較

雙內核組態採用了簡單的鎖定方法，在即時活動(意即工作)同時在不超過4個CPU核心上運行時，此組態通常會有較好的表現。反之，當即時工作在超過4個以上的CPU核心上執行時，雙內核組態的效能較差。至於當即時工作在超過4個CPU核心上執行時，單內核組態能夠得到更好的效能表現。

更具體來說，Xenomai在幾個CPU核心上執行並不是效能的關鍵，即時工作同時在多少個CPU核心上執行，且接收來自即時工作的中斷才是影響效能的關鍵。

至於Xenomai 4則使用了一個（不同於Xenomai 3的）同步方法，採用16-way server pinning，在使用超過4個CPU核心時，仍能提供良好的效能表現。

Xenomai API

Xenomai的API包含了一部份(基礎)的POSIX API，以及一些非POSIX的API — 不過Xenomai只有實作部份的POSIX API(實作在libcobalt函式庫裡)。至於非POSIX的API則是實作於libcobalt之上，我們可以透過copperplate介面加以呼叫。

以下為POSIX所定義的API及Xenomai所實作的部份API：

- 定義在Xenomai 3.0.5裡的POSIX介面的API
- POSIX 1003 2007版的定義

除了POSIX介面的API以外，Xenomai還有以下的API:

- Alchemy API
- Smokey API

Cobalt的更多細節

ADEOS

Cobalt之所以能和linux原本的內核形成雙內核的運作方式，是得利於Adeos(Adaptive Domain Environment for Operating System)

Adeos是2001年，由Karim Yaghmour所設想的 [8)]

其後於2002年，更進一步探討了在SMP環境下的實作議題 [9)]

這些想法後來由Philippe Gerum進行實作，但在實際的做法上，與原始的設想有些差異。 2005年6月，Philippe Gerum發表了Adeos裡用於在兩個內核間共享中斷的I-pipe核心機制. [10)]

2000年，Karim发表了一篇名为《操作系统的自适应域环境》的论文（即Adeos,Adaptive Domain Environment of Operating System，该论文描述了一种简单而智能的方案，用于在同一系统上运行的多个内核之间共享公共硬件资源。他通过“pipeline”抽象来说明在x86硬件上共享中断的基本机制，根据整个系统给定的优先级，依次向每个内核传入中断。他倡导一种对硬件中断进行优先级排序的新方法，以便可以开发基于Linux内核的实时扩展，而无需使用当时已被某些专有RTOS供应商申请授予专利方法（这里

的RTOS供应商和专利指的就是WindRiver和RTlinux使用的RTHAL技术）。

ADEOS (Adaptive Domain Environment for Operating System)提供了一个灵活的环境，可以在多个操作系统之间或单个OS的多个实例之间共享硬件资源，从而使多个优先级域可以同时存在于同一硬件上。早期在xenomai 2上使用。

2005年6月17日Philippe Gerum发布用于Linux内核的I-pipeI-pipe基于ADEOS但是I-pipe更精简，并且只处理中断xenomai3使用I-pipe和后来改进的dovetail

Xenomai 4採用了I-pipe的後繼者Dovetail

Xenomai 編譯與開發工具

編譯Xenomai 3的官方文件

在Rasberry Pi 4上編譯Xenomai 4

目前Xenomai 4以EVL core取代Cobalt以Dovetail取代I-pipe但EVL Core仍不穩定 有人是使用Xenomi 3.2的colbalt加上dovetail參考此處

Xenomai Programming

Tunables

Tunables(可調參數)是一種用以設定系統的組態值(例如可用以調整記憶體heap)或是控制系統執行階段的行為(例如設定Verbosity Level的數值改變log資料的詳細度)的變數。

有些Tunables又被稱為Configuration Tunables(組態可調參數)， 它們可以在應用程式比較前面的執行的程式碼進行更新，直到系統開始初化核心資料結構與啟動服務後，就變成唯讀。

另一些Tunables則被稱為執行階段可調參數，可以在應用程式執行時的任意時間進行更改。

要存取configuration 與runtime tunable當前的數值可以透過以下的API(定義在xenomai/tunables.h裡的巨集)進行：

```
{get/set}_config_tunable(name)
{get/set}_runtime_tunable(name)
```

下面是設定並印出verbosity_level這個runtime tunable數值的例子：

```
#include <stdio.h>
#include <xenomai/tunables.h>
```

```
set_runtime_tunable(verbosity_level, 1);
printf("verbosity_level=%d\n", get_runtime_tunable(verbosity_level));
```

Configuration tunable也可以透過命令列選項進行選項設定，同時要注意的是命令列選項的優先層級高於set_config_tunable()。

Configuration tunables

NAME	DESCRIPTION	DEFAULT
cpu_affinity	same as –cpu-affinity option	any online CPU
no_mlock	same as –no-mlock option	off
no_sanity	same as –no-sanity option	!CONFIG_XENO_SANITY
no_registry	same as –no-registry option	off (i.e. enabled)
mem_pool_size	same as –mem-pool-size option	1Mb
session_label	same as –session option	none (i.e. anonymous)
registry_root	same as –registry-root option	CONFIG_XENO_REGISTRY_ROOT
shared_registry	same as –shared-registry option	off (i.e. private)

Runtime tunables

NAME	DESCRIPTION	DEFAULT
verbosity_level	same as –verbose option	1
trace_level	same as –trace option	0

開發者也可以自訂tunable可調參數(使用define_{config/runtime}_tunable(name) 與read_{config/runtime}_tunable(name))。或是覆載(Overriding)既有的tunable的值，詳細內容可參考官方的說明文件

Entry Point

程式還是和一般的C/C++程式相同，都是從main()開始執行，但視程式所要使用的API。必須透過自動(Automatic)或手動(Manual)設定好對應的啟動程序(Bootstrap)。自動的啟動程序是透過編譯器參數xeno-config –ldflags來完成的，例如使用xeno-config –posix –alchemy –ldflags來設定程式要使用posix及alchemy API。有了這些設定之後，Xenomai 3的函式庫將會負責完成適切的啟動程序。開發者除了可以透過編譯器參數來設定自動啟動程序外，也可以透過程式碼來自行完成啟動，這種做法則稱為手動啟動。選擇使用手動啟動程序的話，必須使用xeno-config–ldflags–no-auto-init編譯器參數來關閉自動啟動程序，並且在程式中必須透過對xenomai_init(&argc, &argv)的呼叫來完成啟動程序的設定，詳細內容亦可參考官方文件。

啟動程式時的命令列引數 (Command Line Arguments)

啟動程式時的命令列引數在自動啟動程序(Automatic Bootstrap)模式下，Xenomai將會在啟動完成後將命令列引數交付給main()函式使用。但在手動啟動程序下，

If the automatic bootstrap mode is used, the application directly receives the set of command line switches passed on invocation in the main() routine, expunged from the Xenomai 3 standard options.

[1] PREEMPT_RT patch versions, The Linux Foundation Wiki, Available at https://wiki.linuxfoundation.org/realtime/preempt_rt_versions
[2] Daniel Bristot de Oliveira and Romulo Silva de Oliveira1, Timing analysis of the PREEMPT RT Linux kernel, SOFTWARE: PRACTICE AND EXPERIENCE, Vol. 46, 2016, pp. 789–819.
[3] Ching-Chun Huang, Chan-Hsiang Lin, and Che-Kang Wu, Performance Evaluation of Xenomai 3, Proceedings of the 17th Real-Time Linux Workshop (RTLWS), Graz, Austria, October 21 to 22, 2015.
[4] Time-critical工作意指工作的執行除正確以外還與其完成時間高度相關，例如車駕系統必須在特定時間內完成的煞車工作就是一例。值得注意的是time-critical一般中譯為時間關鍵，但本文選譯意涵更為接近原意的時間攸關一詞。
[5] 具有即時性的意思係指具有截限時間限制等要求。
[6] 提供使用者及開發人員關於即時裝置驅動程式的一致性介面。
[7] Rollinger, B., Bleiner, D., Chokani, N., & Abhari, R. S., The Real-Time Driver Model and First Applications. 7th Real-Time Linux Workshop, Lille, France, 2008
[8] Karim Yaghmour, Adaptive Domain Environment for Operating Systems, Technical Report, Opersys, Sherbrooke, Canada, February 2001, https://www.opersys.com/ftp/pub/Adeos/adeos.pdf.
[9] Karim Yaghmour, A Practical Approach to Linux Clusters on SMP Hardware, July 2002, https://www.opersys.com/ftp/pub/Adeos/practical-smp-clusters.pdf.
[10] Philippe Gerum, Xenomai - Implementing a RTOS Emulation Framework on GNU/Linux, White Paper, Xenomai, 2004.