ICS 25.040.30
CCS J 28

NATIONAL STANDARD OF THE PEOPLE'S REPUBLIC OF CHINA

GB/T 33264-2016

Framework of a real-time robot operating system on multi-core processors

面向多核处理器的机器人实时操作系统应用框架

Issued on: December 13, 2016 **Implemented on: July 1, 2017**

Issued by: General Administration of Quality Supervision, Inspection and Quarantine;
Standardization Administration of the PRC

Table of Contents

Foreword

This standard was drafted in accordance with the rules given in GB/T 1.1-2009.

It was proposed by the China Machinery Industry Federation and is under the jurisdiction of the National Technical Committee on Automation Systems and Integration Standardization (SAC/TC 159).

The main drafting organisations are Beihang University, Capital Normal University, Beijing University of Chemical Technology, the Beijing Institute of Automation for Machinery Industry and the China Machinery Productivity Promotion Centre.

The principal drafters include Shao Zhenzhou, Wei Hongxing, Tan Jindong, Guan Yong, Zhang Jie, Chen Youdong and Huang Zhen.

1 Scope

This standard specifies the application framework for robot real-time operating systems on multi-core processors, together with the design principles for such systems.

It applies to developers of robot real-time operating systems, developers of robot application software, and users of robot operating systems.

Two kinds of computing that do not mix

A robot has two kinds of computing to do, and they have incompatible requirements.

Servo control runs on a hard deadline. The loop closing around a joint executes at a fixed rate - typically one to eight kilohertz - and a result that arrives late is not a late result but a wrong one, because the actuator has already moved.

Perception, planning, mapping and coordination have no such deadline. They will consume as much computing as they are offered and produce a better answer for it, and a delay of some milliseconds is usually of no consequence.

Running both on one processor forces a compromise: either the real-time work is starved by the rest, or the rest is throttled to protect it. Neither is satisfactory, and both have been the normal state of affairs.

4 What multi-core changes

Multi-core hardware offers the obvious way out - give the real-time work its own cores and let the rest have the others - and this framework is about organising that properly rather than ad hoc.

The design principles listed are worth reading as a set. Cross-platform operation means a robot's software survives a change of processor, which over a product's life is not a small consideration.

Structural separation keeps the real-time and non-real-time halves genuinely apart, so that a fault or a load spike on one side cannot reach the other.

Functional modularity, distributed management across cores and extensibility complete the list, and together they describe a system in which cores are treated as nodes in a distributed system rather than as a single machine with several processors.

5 Interrupts and distribution

Interrupt management gets its own subclause because on a multi-core system interrupt routing is what determines whether the separation actually holds.

An interrupt delivered to a core running real-time work steals time from it, and a system that lets every device interrupt every core has no real-time guarantee however carefully the tasks were assigned.

Distributed operation and the communication interfaces then cover how the nodes talk to one another - which for cores on one chip is a shared memory question, and for a robot with several processors is a bus question.

The custom message provision for node communication is the extensibility hook: a

framework that only carries the message types its authors imagined stops being useful the moment somebody adds a sensor.

5 Two APIs, and why

The framework specifies a real-time operating system function interface and, separately, a non-real-time node application programming interface, with a modular node API alongside.

Having two APIs rather than one is deliberate and is the practical consequence of the separation principle.

Code on the real-time side may only call functions with bounded execution time - no dynamic allocation, no blocking, no unbounded loops - and offering it the full API would be offering it a way to break its own guarantees.

Annex A, listing the key functions and variables that must be migrated when a real-time system is ported, is the annex a developer will actually use, and Annex B does the same for the non-real-time side.