

中华人民共和国航空行业标准

FL 1119

HB/Z 360—2008

航空电子应用软件接口应用指南

Application guidance for avionics application software interface

2008—03—17 发布

2008—10—01 实施

国防科学技术工业委员会 发布

目 次

前言.....III

1 范围.....1

2 规范性引用文件.....1

3 术语和缩略语.....1

3.1 术语和定义.....1

3.2 缩略语.....2

4 概述.....2

4.1 通则.....2

4.2 软件结构.....3

4.3 一般要求.....3

5 功能描述.....4

5.1 分区管理.....4

5.2 进程管理.....7

5.3 时间管理.....8

5.4 分区间通信.....11

5.5 健康监控.....12

5.6 文件管理.....17

5.7 数据传输管理.....18

5.8 蓝图配置.....19

6 扩展接口要求.....20

6.1 分区管理.....20

6.2 进程管理.....25

6.3 时间管理.....30

6.4 分区间通信.....34

6.5 健康监控.....40

6.6 文件管理.....45

6.7 数据传输管理.....64

附录 A （规范性附录） 蓝图配置要求.....73

A.1 蓝图配置工具.....73

A.2 蓝图配置的生成.....73

A.3 基本图素.....73

A.4 蓝图配置示例.....73

A.5 XML-Schema 代码.....73

附录 B （资料性附录） 蓝图配置示例.....74

B.1 系统配置结构.....74

B.2 XML 配置文件结构.....76

附录 C （资料性附录） XML-Schema 代码.....80

C.1 代码说明.....80

C.2 代码示例.....80

图 1 核心模块组件间的关系图.....3

图 2 分区模式转换图.....5

图 3 进程状态图.....7

图 4 进程状态转换与分区模式转换之间的关系图.....7

图 5 时间补偿图.....9

图 6 动态跟踪时间修正原理.....11

表 1 软件层间的依赖性.....3

表 2 主时间框架表.....6

表 3 时间类型.....9

表 4 操作系统健康监控表.....13

表 5 模块健康监控表.....13

表 6 分区健康监控表.....13

表 7 故障代码表.....13

表 8 文件位置调整关系表.....53

前 言

本指导性技术文件是 GJB 5357—2005《航空电子应用软件接口要求》的配套标准，规定了航空电子系统中应用软件与嵌入式实时操作系统之间接口的应用指南。

本指导性技术文件的附录 A 为规范性附录，附录 B 和附录 C 均为资料性附录。

本指导性技术文件由中国航空工业第一集团公司提出。

本指导性技术文件由中国航空综合技术研究所归口。

本指导性技术文件起草单位：中国航空工业第六三一研究所、中国航空综合技术研究所。

本指导性技术文件主要起草人：叶 宏、黄永葵、任晓瑞、甯 清、徐晓光、齐 昱、胡 辛。

航空电子应用软件接口应用指南

1 范围

本指导性技术文件规定了航空电子系统中应用软件与嵌入式实时操作系统之间接口的应用指南,主要包括功能指南和接口指南。

本指导性技术文件适用于航空电子系统软件产品的开发,包括机载设备和地面支持设备中的软件、独立的软件产品和用于产品验证和确认的软件。

2 规范性引用文件

下列文件中的条款通过本指导性技术文件的引用而成为本指导性技术文件的条款。凡是注日期的引用文件,其随后所有的修改单(不包含勘误的内容)或修订版均不适用于本指导性技术文件,然而,鼓励根据本指导性技术文件达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件,其最新版本适用于本指导性技术文件。

GJB 5357—2005 航空电子应用软件接口要求

3 术语和缩略语

3.1 术语和定义

GJB 5357—2005 确立的以及下列术语和定义适用于本指导性技术文件。

3.1.1

首次释放点 first release point

对于周期进程,进程首次释放点是指该进程所属分区(在正常模式下)下一个周期中首个分区窗口的开始时间。

对于非周期进程,进程首次释放点为当前时间点。

3.1.2

下次释放点 next release point

周期进程的上次释放点时间加上进程的周期值。

3.1.3

应用分区 application partition

装载应用软件代码和数据的分区,应用分区中应用软件通过 GJB 5357—2005 定义的接口访问操作系统核心层。

3.1.4

系统分区 system partition

不同于一般应用分区的一种特殊分区,系统分区中的软件通过 GJB 5357—2005 定义的接口和本指导性技术文件定义的扩展接口访问操作系统核心层。

3.1.5

关键级别 criticality level

表示该分区的失效对整个飞机系统性能的影响程度。

3.1.6

绝对全局时间 absolute global time

一种日历时间,通常等同于当地的区域时间。

3.1.7

绝对本地时间 **absolute local time**

本载机航空电子系统系统时间，一般用于应用间同步。

3.1.8

相对本地时间 **relative local time**

核心模块中用于分区的管理，实现进程调度的时间。

3.1.9

传输连接 **transfer connection**

一种数据通信的方式，它由通道与硬件设备相连接组成。

3.1.10

蓝图配置 **blueprint**

蓝印配置 **t**

描述航空电子系统资源分配及其操作动作的定义。

3.1.11

锁定级别 **lock level**

描述进程抢占的级别。

3.1.12

模块操作系统接口 **module operating system interface**

操作系统核心层与硬件模块支持层之间的接口。

3.2 缩略语

下列缩略语适用于本指导性技术文件。

AGT——absolute global time，绝对全局时间；

ALT——absolute local time，绝对本地时间；

APEX——APplication/Executive，应用执行；

FIFO——first-in first-out，先入先出；

HM——health monitoring，健康监控；

LIFO——last-in first-out，后入先出；

MOS——module operating system interface，模块操作系统接口；

MSL——modual support level，硬件模块支持层；

RLT——relative local time，相对本地时间；

TC——transfer connection，传输连接；

XML——extensive makeup language，可扩展标记语言。

4 概述

4.1 通则

GJB 5357-2005 定义了航空电子系统软件的三层结构，即应用软件层、操作系统核心层和硬件模块支持层。应用软件层主要包括航空电子各功能模块的软件(如雷达应用、任务管理、存储管理和飞行管理系统等)；操作系统核心层负责航空电子任务的管理和资源的分配等工作；硬件模块支持层为操作系统和航空电子软件提供了硬件平台的支持服务。三层软件的相关依赖性见表 1。

本指导性技术文件针对 GJB 5357-2005 的各条款，详细描述了航空电子软件的层次和功能划分，并对 GJB 5357-2005 未能详细说明的部分内容进行了详细说明，这些内容有分区模式转换、分区调度、HM、故障类型定义和 XML 的配置定义。

本指导性技术文件所涉及的内容，如果 GJB 5357-2005 已定义明确，一般不再重新定义，作为指

南，仅对其中的概念作进一步说明。

表 1 软件层间的依赖性

软件层	飞机依赖	硬件依赖
应用软件层	相关	无关
操作系统核心层	无关	无关
硬件模块支持层	无关	相关

4.2 软件结构

GJB 5357-2005 给出了软件三层结构中各层次的功能划分。包括应用软件层、操作系统核心层和

对于驻留在

APEX 接口。

软件运行的

服务。同时

信和故障管

管理、数据传

调度、通信以

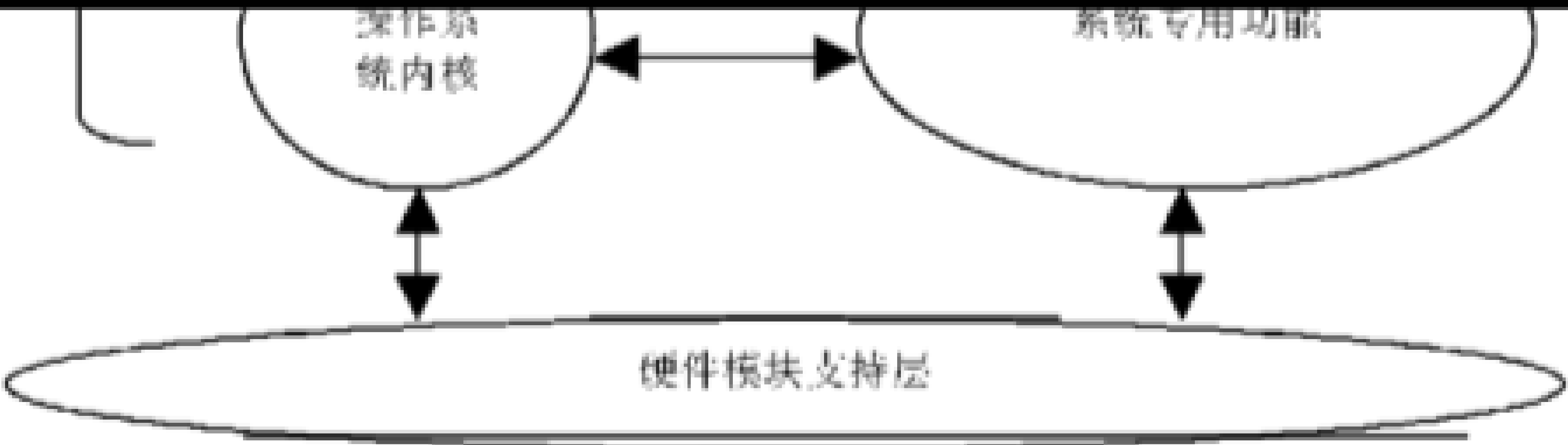
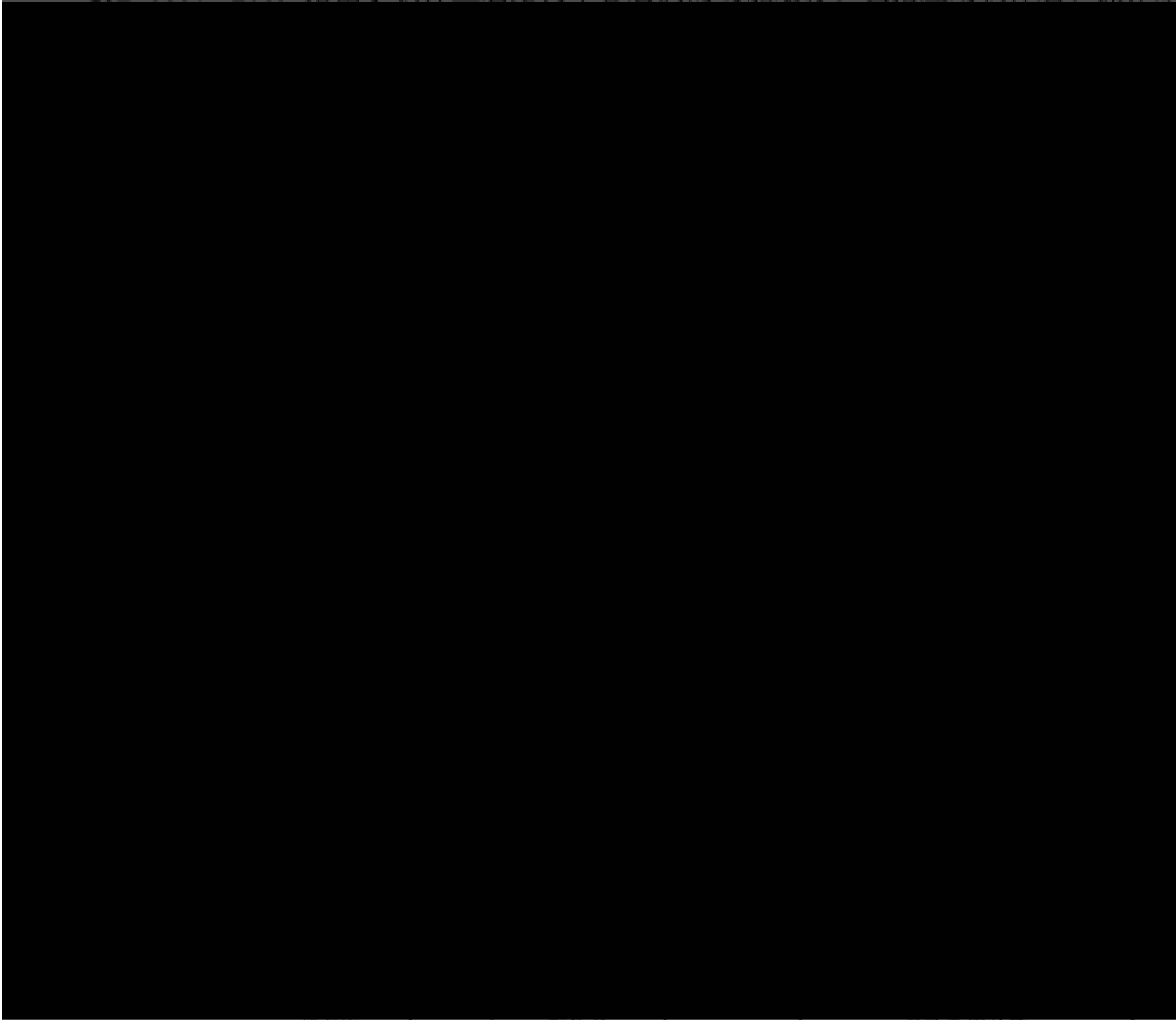


图 1 核心模块组件间的关系图

4.3 一般要求

4.3.1 通则

本指导性技术文件主要给出了 GJB 5357-2005 中定义的各项功能模块的详细说明及扩展，包括对分区管理、进程管理、时间管理及健康监控功能的详细说明。同时对 GJB 5357-2005 中未定义的或必

要的功能分别进行增补或扩展，包括文件管理、数据传输定义和蓝图配置描述语言。

4.3.2 分区类型和调度要求

分区类型主要有系统分区和应用分区两种。系统分区可调用除应用分区所能调用的应用接口外，还可调用操作系统专用接口服务，访问操作系统内核和硬件。

分区调度以基于主时间框架的调度方式为主。

4.3.3 进程管理要求

在 GJB 5357-2005 对进程管理描述的基础上，本指导性技术文件详细说明了进程的状态转换与分区状态之间的关系，同时增加对进程时间补偿的描述。

4.3.4 时间管理要求

GJB 5357-2005 中已定义了模块内分区的时间管理，时间用于实现分区调度、进程调度和数据传递等，本指导性技术文件详细说明了处理机间的同步和异步时间以及所有处理机需要使用时间，本指导性技术文件主要对系统中三个基本时间 (AGT、ALT 和 RLT) 做了进一步划分与描述。

4.3.5 分区间通信

分区间通信主要通过端口和通道来实现。本指导性技术文件对 GJB 5357-2005 中已描述的端口控制流程将做进一步的说明，同时增加了通道定义和通道控制的说明。

4.3.6 健康监控和故障类型要求

GJB 5357-2005 定义了健康监控功能、作用和故障响应与恢复要求。本指导性技术文件将详细说明健康监控的分类和故障分类定义。

健康监控分为系统级健康监控、模块级健康监控、分区级健康监控和进程级健康监控；限于适用范围，本指导性技术文件只说明了有关模块级、分区级和进程级的故障管理。

故障类型是由发生故障时所处的系统状态来决定的。系统状态分为模块级、分区级和进程级，故障的类型也同样分为模块级、分区级和进程级故障。

4.3.7 文件管理要求

文件管理是综合化航空电子系统对数据管理的关键部分，主要实现航空电子各子系统的日志记录、故障记录和运行记录等，为系统的容错和重构提供必要的恢复数据。本指导性技术文件增加了对文件管理的接口要求。

4.3.8 数据传输管理要求

数据传输采用分层管理方式，共分为四层，即端口层、通道层、传输连接层和接口层。本指导性技术文件增加了对数据传输管理的描述。

4.3.9 蓝图配置要求

蓝图配置是操作系统支持航空电子应用的基础，蓝图定义了操作系统对资源需求的描述、任务调度规则的描述和故障处理规则的描述等信息。蓝图配置的图素要求见附录 A。

5 功能描述

5.1 分区管理

5.1.1 概述

操作系统应支持分区管理，应允许分区根据不同的关键级别运行在同一个核心模块上，且不影响其它分区的时间特性和空间特性。

5.1.2 分区模式转换

分区被启动后，进入一种模式，分区所用的资源 (通道、进程、队列、信号量和事件等) 都在系统建立时配置好，相关资源在初始化阶段中创建成功。分区进入正常模式后，分区调用应用接口可将自己的模式从一种模式转换到另一种模式。核心模块上的一个分区的模式是与其它分区的模式相互独立的，这样，可能有一些分区处于冷启动模式，同时其它分区处于正常模式或热启动模式。为了处理一个严重故

障，健康监控可设置故障分区，也可将整个核心模块设置为重新启动模式或空闲模式。

在执行中，各分区(包括应用分区和系统分区)始终是处于以下几种模式之一：空闲模式、正常模式、冷启动模式和热启动模式。对这四种模式说明如下：

- a) 空闲模式：在此模式下，分区在分配的分区窗口内不执行任何进程，即分区不进行初始化(例如，分区相关的端口都不初始化)、没有进程执行、没有资源消耗；但分配给分区的分区时间窗口不变。
- b) 正常模式：在此模式下，分区内的进程调度处于被激活；分区内的所有进程都已创建好，且处于就绪状态的进程运行。系统是处于一个可操作的状态。
- c) 冷启动模式：在此模式下，初始化阶段正在进行中；如果锁定级别为零，那么禁止进程抢占(进程调度被禁止)，且分区正在执行它自己的初始化代码。
- d) 热启动模式：在此模式下，初始化阶段正在进行中；如果锁定级别为零，禁止进程抢占(进程调度被禁止)，且分区正在执行它自己的初始化代码。这时的模式类似于冷启动模式，但初始环境(分区开始的硬件上下文关系)不同，也不需要从非易失存储区复制代码到随机存储器中。

冷启动模式和热启动模式的差异主要依赖于硬件的能力和系统的规定。一些事件(下电中断、硬件复位、健康监控动作或一个设置分区模式服务)可能会导致冷启动模式或热启动模式的转换，不管是冷启动还是热启动，分区的入口点都是一个。

分区模式转换图见图 2，模式的转换要求如下：

- 冷启动模式到冷启动模式，见图 2 中 1a；热启动模式到热启动模式，见图 2 中 1b。分区是在一个初始化过程中，需要通过重启回到相同的初始化阶段。
- 热启动模式到冷启动模式，见图 2 中 2。分区是在一个初始化过程中，需要通过重启回到另外一种初始化阶段。
- 冷启动模式到空闲模式，见图 2 中 3a；热启动模式到空闲模式，见图 2 中 3b；正常模式到空闲模式，见图 2 中 3c。调用设置分区模式服务，并将分区模式设为空闲模式；健康监控产生一个恢复活动(例如，按分区的健康监控表产生一个恢复活动)，就是将当前正在工作的分区停止。
- 冷启动模式到正常模式，见图 2 中 4a；热启动模式到正常模式，见图 2 中 4b。当分区已完成初始化，且调用设置分区模式服务，设定的模式是正常模式时，分区就进入正常模式工作；
- 正常模式到冷启动模式，见图 2 中 5a；正常模式到热启动模式，见图 2 中 5b。分区处于正常工作模式下，但应重新启动。
- 空闲模式到冷启动模式，见图 2 中 6a；空闲模式到热启动模式，见图 2 中 6b。由于外部原因(例如：电源中断、核心模块复位、应用程序的复位等)，分区从空闲模式切换到冷/热启动模式。

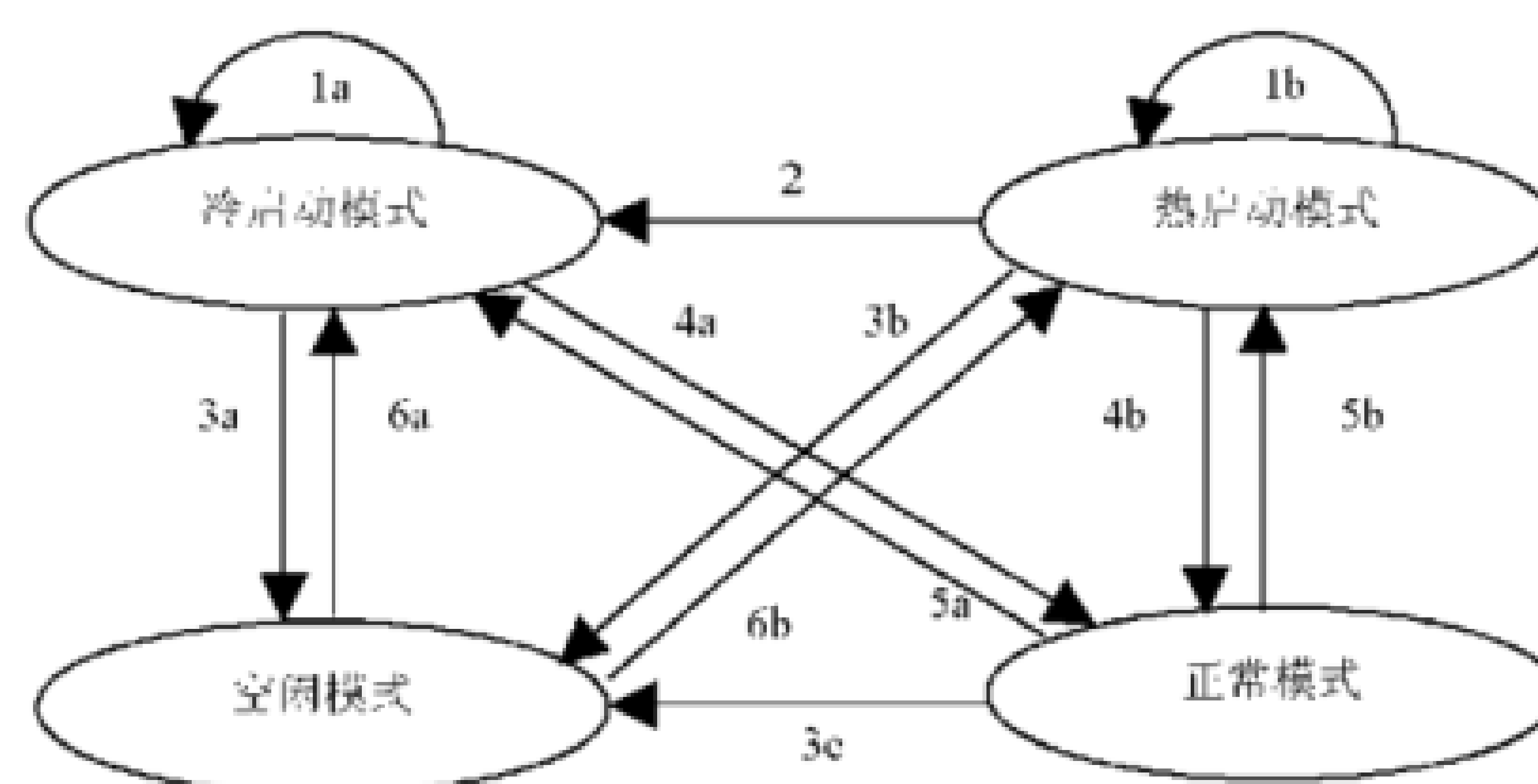


图 2 分区模式转换图

5.1.3 分区调度

本指导性技术文件在 GJB 5357-2005 中 4.2.4 原则的基础上进行了详细说明。分区调度主要是按固定的、基于周期的时间序列进行资源分配；每个分区按主时间框架分配给它的分区窗口（一个或多个）被调度程序所激活；分区可调度的单元是一个分区，分区没有优先级。操作系统专门控制分区的资源分配。

为了周期性地激活各分区，操作系统需要维护一个固定时间间隔（周期地重复在模块中的执行操作）的主时间框架表。各分区是按主时间框架中的一个或几个分区窗口来激活的，各分区窗口是由距离主时间框架开始点的偏移量和各分区的执行时间来定义的。各分区的激活次序是由配置表所定义的配置时间决定的。

在各调度循环中，至少应给每个分区分配一个时间窗口，也可是多个，并且分区的时间窗口不要求是相邻的，在一个时间框架中允许有空闲的时间窗口。在主时间框架中，每个分区的时间窗口至少应激活一次。

一个核心模块可能包含几个正在以不同周期运行的分区。主时间框架的时间定义为模块中所有分区周期的最小公倍数。在核心模块中，各分区的周期要求必须在主时间框架中得到满足，包括恰当地设置各分区的时间窗口大小、出现次数等。

分区调度支持多重调度方式（时间表调度，中断处理方式），时间表的重加载就是主时间框架表的重加载。系统分区可通过重新加载分区时间表，将分区的调度从下一个主时间框架中切换到一个新的主时间框架调度表。

主时间框架是由若干个分区窗口组成，每个分区窗口对应于一个分区（可以是用户分区，也可以是系统分区）。

每个分区窗口是由以下几个部分组成的，即分区名称、分区执行时间起点、分区窗口持续时间和分区释放点。

操作系统在对分区调度时，应用分区标识来调度的，且通过分区标识，查询分区控制块表，可得到分区的相关信息。

分区的调度表格式见表 2。

表 2 主时间框架表

分区窗口	分区名称	分区执行时间起点 ns	分区窗口的持续时间 ns	分区释放点
窗口 1				
窗口 2				
窗口 3				

对表 2 中的表头说明如下：

- a) 分区名称：表示执行分区的名字，是一个字符串。
- b) 分区执行时间起点：表示分区开始执行点与主时间框架的起始点之间相对时间，是一个以纳秒为单位的数值。
- c) 分区窗口的持续时间：表示要执行的分区在该分区窗口中所执行的时间值，是一个纳秒值。
- d) 分区释放点：一个分区的释放点就是该分区周期的起始点。对于周期进程，该进程第一个释放点是取决于该分区下一个周期的第一个分区窗口。对于非周期进程，该进程的释放点取决于当前时间。分区释放点为一布尔量，该值为假则表示不是分区释放点，为真表示是分区释放点。

5.1.4 分区管理扩展接口的功能要求

相对于 GJB 5357-2005，本指导性技术文件主要增加了以下分区管理接口要求：

- a) 创建分区服务：

- b) 撤消分区服务;
- c) 加载分区调度表服务。

5.2 进程管理

5.2.1 概述

在 GJB 5357-2005 中定义的进程管理的基础上, 本指导性技术文件主要就进程的状态转换与分区状态两者之间的关系进行详细描述。

5.2.2 进程状态转换

5.2.2.1 进程状态

在 GJB 5357-2005 中规定, 分区内的进程处于休眠态、就绪态、运行态和等待态四种状态之一。通过进程调度, 它们能从一种状态转换到另一种状态。

进程的状态如图 3 所示, 与 GJB 5357-2005 相比, 本指导性技术文件增加了以下两个条件下的状态转换:

- a) 允许在等待态下转入等待态;
- b) 允许进程从休眠态到等待态的转换。



图 4 进程状态转换与分区模式转换之间的关系图

5.2.2.3 状态转换

状态转换规则如下：

- a) 休眠态到就绪态(见图 4 中(1))：进程由分区内的其它进程启动(当分区处于正常模式)。
- b) 就绪态到休眠态(见图 4 中(2))：进程被分区内的其它进程停止(当分区处于正常模式)。
- c) 等待态到就绪态(见图 4 中(3))：
 - 1) 当分区从初始化阶段转为正常模式时，在初始化阶段启动的非周期进程(在初始化阶段未被挂起)由等待态转为就绪态(见图 4 中(3a))；
 - 2) 进程解挂，或进程等待的资源变为可用(包括延迟等待的时间到期)；当周期进程等待周期时，等待的周期释放点到达时该周期进程自动变为就绪态(见图 4 中(3b))。
- d) 就绪态到等待态(见图 4 中(4))：进程被分区内的其它进程挂起。
- e) 运行态到就绪态(见图 4 中(5))：进程等待一个时间值为零的时间量，或被分区内的其它进程抢占。
- f) 就绪态到运行态(见图 4 中(6))：进程被选择执行。
- g) 运行态到等待态(见图 4 中(7))：进程挂起自己，或进程试图访问当前得不到的资源(延时、信号量、周期、事件和消息)并进入等待；周期进程调用周期等待服务等待其周期时间到，即等待下一个分区释放点的到达。
- h) 运行态到休眠态(见图 4 中(8))：进程停止自己。
- i) 等待态到等待态(见图 4 中(9))：
 - 1) 正在等待访问资源的进程被挂起，或当正在等待访问资源且被挂起的进程被解挂，或资源变为可用，或延迟到期(见图 4 中(9a))；
 - 2) 当分区从初始化阶段转为正常模式时，在初始化阶段启动的周期进程，或是在初始化阶段挂起/延时启动的非周期进程(见图 4(9b))；
 - 3) 在冷/热启动模式下挂起进程(见图 4(9c))。
- j) 等待态到休眠态(见图 4 中(10))：进程被分区内的其它进程停止(适用于初始化阶段和正常模式)。
- k) 休眠态到等待态(见图 4 中(11))：
 - 1) 在冷/热启动模式下启动一个进程(见图 4 中(11a))；
 - 2) 在正常模式下，启动周期进程(等待释放点)(见图 4(11b))。
- l) 休眠态到休眠态(见图 4 中(12))：分区从初始化阶段到正常模式时未被启动的进程。

5.2.3 进程的时间补偿

当进程被启动时(无论是通过服务调用，还是在分区初始化阶段结束时)，进程的截止期被设置为当前相对本地时间加上进程的时间容量。时间容量对于进程来说，是一个绝对时间值，而不是指进程的运行时间。这就意味着一个进程还未运行，截止期就已超时了。操作系统应提供时间补偿服务来延期进程的截止期。时间补偿功能见图 5 所示。

5.2.4 进程扩展接口功能要求

相对 GJB 5357-2005 中进程管理定义的接口，本指导性技术文件增加以下进程管理接口要求。包括：

- a) 进程延迟启动服务；
- b) 进程获得自身标识服务；
- c) 时间补偿服务。

5.3 时间管理

5.3.1 概述

航空电子系统是由多个核心模块构成的分布式系统，为了协调系统资源，每个核心模块应维护一个

时间。GJB 5357-2005 中已经定义了核心模块内分区的时间管理，用于实现分区、进程的调度和数据传递等，而在所有模块间需要使用精确的 ALT，核心模块必须访问整个飞机的时间资源，达到核心模块间以及模块与外部环境间的协调工作。

在航空电子系统中，时间管理主要提供以下飞机需要：

- a) 系统事件的时间记录和输出，如处理传感器数据的时间戳；
- b) 任务执行中子系统元素的调度与同步；
- c) 飞机中不同子系统间的合作与同步；
- d) 飞机间联合操作的合作与同步。



图 5 时间补偿图

在航空电子系统中，必须有三个基本时间，即 AGT、ALT 和 RLT。模块内用于管理分区、进程调度的时间称为 RLT，它应比其他时间的精度高，主要被用于模块中紧耦合进程的同步与调度；ALT 主要用于载机航空电子系统应用的同步与调度；AGT 是所有飞机共同拥有的日历时间，其计时单位是年、月、日、时、分、秒。这三个时间的精度要求可参考表 3。GJB 5357-2005 中定义的时间管理接口为 RLT。

表 3 时间类型

类型	精度 ms	范围
AGT	1~10	>60d
ALT	0.001~0.1	>24h
RLT	0.001	>60s

5.3.2 AGT

AGT 实质上是日历时间，提供了以年、月、日、时、分、秒为单位时间度量。通常用为当地的区域时间(如格林威治时间)。这个时间一般是通过时间源传输到飞机中，用于同步外部通信。

AGT 功能定义如下：

- a) 是航空电子系统的一个或一些时间的时间源;
- b) 存在于航空电子系统中的所有核心模块;
- c) 提供应用软件和操作系统的接口。

5.3.3 ALT

ALT 应比 AGT 的精确度高,它被用于应用同步(例如 ALT 用于多余度通道的同步处理)。该时间的初始化取决于飞机上的某个事件(如应用加电),这个时间的计时由后备电池维持,并当 AGT 有效时,重新进行时间同步。

ALT 功能定义如下:

- a) 作为航空电子处理系统的系统时间;
- b) 由一个逻辑的时间维护中心维护;
- c) 分布并同步贯穿在航空电子系统的所有核心模块中;
- d) 独立于 AGT。

5.3.4 RLT

RLT 的精度高于其他两个时间,主要被用于紧耦合进程的同步与调度,同时也可用于高速功能模块间的位一位同步(bit by bit synchronisation)、传出延迟等,因此,这个时间需要一个非常精确的硬件时钟管理。

RLT 的功能定义如下:

- a) 作为模块中进程的调度时间;
- b) 独立于 ALT。

5.3.5 时间管理方法

5.3.5.1 分布式机制

分布式机制适应于从一个参考源发送一个时间信号到时钟。该方法可使用自顶向下的发送协议,但是,对系统中同一个事件发生的时间,从两个不同的模块获取到的最大时间差应在一个已知的范围内。为支持 ALT 的管理并遵守时间的精确要求,系统可使用一个高确定的硬件资源保障网络上的同步,这种方式可作为 ALT 时间源的一部分。

5.3.5.2 同步方法

无论从不同地方传过来的物理时间存在多大的差异,时钟同步允许系统维护一个大约相等的公共系统(全局)时间。同步方法可用硬件方法实现或用软件方法实现,也可用混合方法(或前两种方法混合)实现。对这三种同步方法说明如下:

- a) 硬件同步方法:为了时间同步,硬件方法采用了直接修正物理时钟参数。即使用专用电路的物理时钟硬件直接提供系统时间。这种方法可提供高精确度的时间信号,但是成本过高,因此仅可用于要求高时间精确度。
- b) 软件同步方法:软件同步一般使用一个物理时钟或定时器修正一个或多个同步时钟。同步时钟是一个使用了同步算法的物理时钟,提供给系统以修正同步时钟的漂移速率。软件同步通过通信网络在系统的不同区域间交换信息(请求-应答机制)来实现。这种方法与硬件方法相比,具有一定的灵活性,且成本较低。但是总线的宽度和时间的精确性是实施中必须考虑的问题。
- c) 混合同步方法:混合同步方法的目的是增加软件精确性和较少硬件成本。

5.3.5.3 时间修正

处于激活状态的同步机制一般从当前的校准本地时间中获取时间斜率(不考虑时间是否到达),算出时间漂移的衰减结果。当时间修正信号到达时,处于激活状态的同步机制将比较接收的时间与维护时间的差异,并计算出衰减的调整值。这个处理过程称之为动态同步。图 6 给出了动态跟踪时间修正原理。

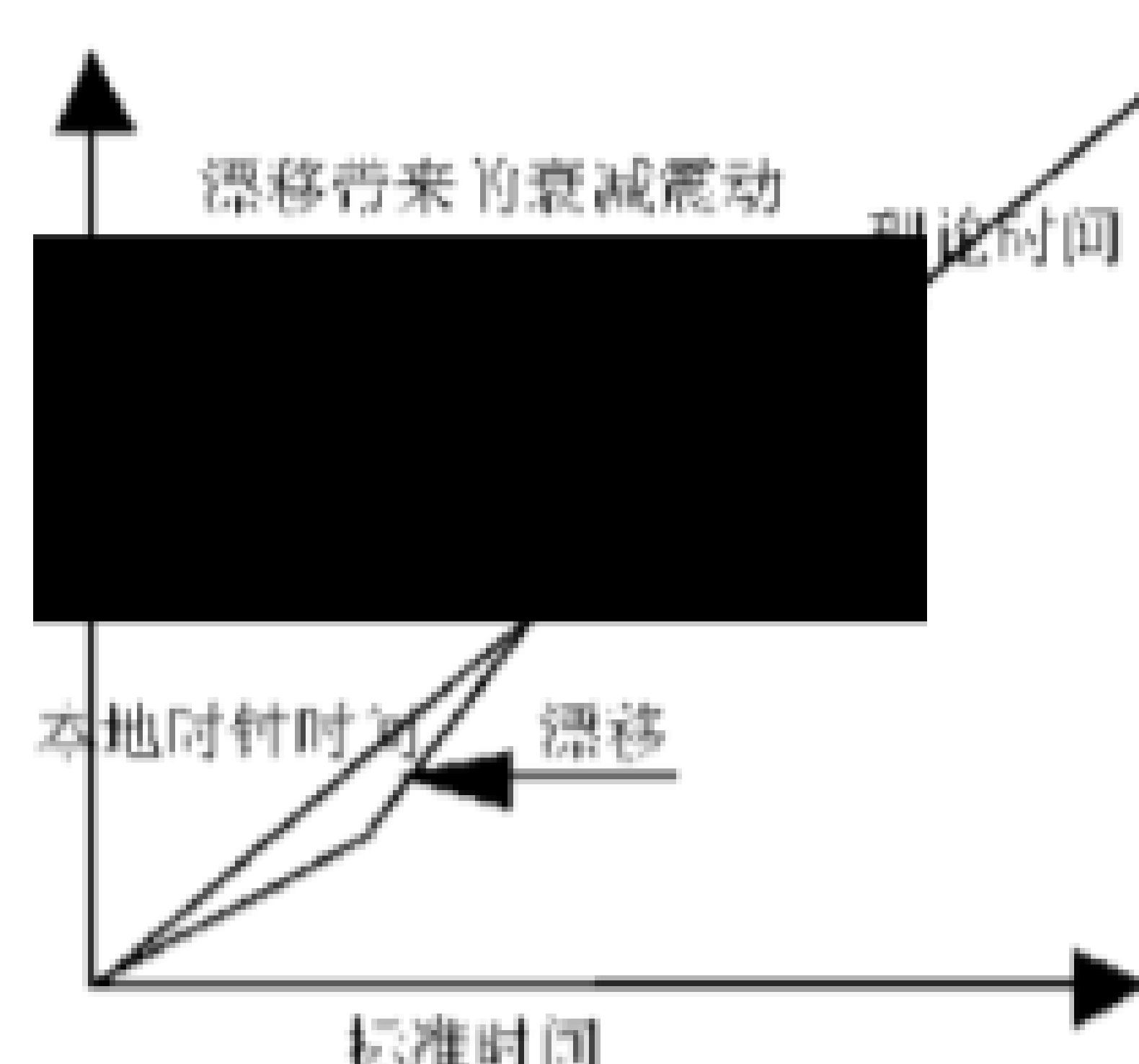


图6 动态跟踪时间修正原理

5.3.6 时间扩展接口功能要求

相对 GJB 5357-2005 中时间管理定义的 RLT 应用软件接口, 本指导性技术文件增加了以下时间管理接口要求:

- a) 对时钟的绑定操作;
- b) 对时钟的配置操作;
- c) 获得 AGT;
- d) 获得 ALT。

5.4 分区间通信

5.4.1 概述

通过消息连接分区的基本机制是通道, 通道定义了一个源分区与一个或多个目的分区之间的逻辑连接。通道由一个或多个端口以及相关的资源组成, 端口提供允许分区在特定通道上发送或接收消息所必需的资源。一个分区可使用多个通道进行数据交换, 每一个通道都有自己的源端口和目的端口。

分区间通信支持同一模块上不同分区之间的通信、不同模块上分区之间的通信和分区与设备之间的通信。

对于同一模块上分区之间的通信, 消息直接从源分区的发送端口到达目的分区的接收端口。对于这种方式, 通道连接了一个发送端口和一个(或多个)接收端口。

对于不同模块上分区之间的通信, 发送端通过网络接口(或设备)发送出消息, 接收端通过网络接口(或设备)接收消息。对于这种方式, 通道绑定在 TC 上, 发送端口的消息被传递给通道绑定的 TC, TC 将消息传递给接口设备, 当消息到达接收端时, 接口设备将数据传递给 TC, TC 将接收到的消息传递给绑定在该 TC 通道上的接收端口。

对于分区与设备之间的通信, 通道绑定在 TC 上, 对于发送方向, 发送端口的消息传递给 TC, TC 调用设备驱动发送消息。对于接收方向, TC 调用设备驱动接收消息, 将接收到的消息传递给该 TC 绑定的通道的接收端口。

5.4.2 端口属性

端口属性除符合 GJB 5357-2005 中的 4.7.6 规定之外, 还应符合以下规定:

- a) 排队规则: 该属性仅对队列端口有效。当进程向队列发送端口发送消息时, 如果该端口此时没有空闲缓冲区接收消息, 且进程允许等待, 那么进程将被挂起; 当进程从队列接收端口接收消息时, 如果此时队列端口没有接收到消息, 且进程允许等待, 那么调用进程将被挂起。被挂起的进程应被加入到端口的等待队列中。该属性指明端口加入到等待队列的规则是 FIFO 或是优先级的方式。
- b) 分区标识: 用于指示可访问该端口的分区, 在配置时该属性是该端口所在分区的分区名称。
- c) 消息的存储区域大小: 对于队列端口, 该属性定义了一个消息的最大长度和端口允许缓冲消息的最大个数; 对于采样端口, 该属性定义了端口缓冲区的大小。

5.4.3 端口控制

端口控制除符合 GJB 5357-2005 中 4.7.7 规定外, 还应符合以下规定:

- a) 对于采样端口, 允许发送可变长度的消息, 消息的长度由发送函数中的消息长度来指定;
- b) 对于采样端口有效性标识的说明。当应用从端口读消息时, 此时系统时间与消息到达该接收端口时的系统时间的差值如果大于采样端口的刷新率, 那么端口的有效性标识为假, 否则端口的有效性标识为真。

5.4.4 通道属性

通道属性应包括:

- a) 通道标识: 在载机航空电子系统中通道的标识是唯一的。
- b) 通道类型: 说明通道是 FIFO 通道或是 LIFO 通道。对于 FIFO 的通道, 发送端口的数据按 FIFO 的规则到达接收端口; 对于 LIFO 的通道, 发送端口的数据按 LIFO 的顺序到达接收端口。
- c) TC 列表: 绑定到该通道所有 TC 的标识。
- d) TC 数目: TC 列表中所有的 TC 数目。
- e) 接收端口数目: 通道允许连接的接收端口的最大数目。

5.4.5 通道控制

在使用端口进行通信之前, 必须对通道进行初始化。通道的初始化必须在 TC 创建之后进行, 不允许在 TC 建立之前对通道初始化。通道初始化根据通道的配置进行。通道初始化后, 如果通道需要通过 TC 来收发消息, 必须将通道绑定在 TC 上。

根据通道连接方式的不同, 通道的操作如下:

- a) 如果通道连接的端口在一个模块内, 那么发送端口的消息将沿通道直接到达接收端口。
- b) 如果通道连接的端口不在一个模块内, 那么通道将通过 TC 来发送数据, 数据到达接收端时, 通道通过 TC 来接收数据。
- c) 如果通道连接的是发送端口和设备, 通道通过 TC 发送端口的数据, TC 调用驱动程序将数据传送到设备。如果通道连接的是接收端口和设备, 那么通道通过 TC 来接收设备收到的数据, 将收到的数据传递给通道上的接收端口。
- d) 如果通道连接多个端口, 那么在一个模块内按照 a) 操作, 其它将按照 b) 操作。

5.4.6 端口扩展接口要求

为了降低分区间通信的开销, 使应用程序方便地使用端口进行通信, 本指导性技术文件增加了 4 个服务接口。这些增加的接口允许应用直接访问端口的缓冲区, 可有效的减少通信时数据拷贝的次数。

由于允许应用直接访问端口的缓冲区, 而操作系统也应对端口缓冲区进行操作, 所以增加接口必须保证应用和操作系统对端口缓冲区的互斥访问。

相对 GJB 5357-2005 中分区间通信定义的端口应用软件接口, 本指导性技术文件增加以下分区间通信的接口要求:

- a) 锁定端口缓冲区;
- b) 接收端口缓冲区;
- c) 发送端口缓冲区;
- d) 释放端口缓冲区。

5.5 健康监控

5.5.1 概述

健康监控用于监视模块硬件、应用软件和操作系统的状态。当发现故障时, 健康监控将记录故障并进行故障隔离, 防止故障蔓延, 同时按故障级别(模块级、分区级和进程级)进行必要的恢复操作。

在以下各部分软件中均包含有健康监控的部分:

- a) 操作系统: 操作系统控制着整个健康监控的执行。操作系统使用健康监控配置表处理可预见的故障;

- b) 应用分区：应用确定的故障或由逻辑或计算所引起的故障，应用分区可采用健康监控服务将故障数据传送给操作系统或适当的系统分区；
- c) 系统分区：系统分区能作为系统的故障处理平台，操作系统可把故障信息通过定义在健康监控表(模块健康监控表和分区健康监控表)中的回调机制传送给系统分区；应用分区可通过端口报告故障到系统分区。

5.5.2 健康监控的分类和各种健康监控的功能要求

健康监控分为进程级健康监控、分区级健康监控和模块级健康监控，这样的划分是基于其作用的范围和管理各级别故障的需要所确定的。健康监控诊断出的故障应进行分类，并以相应的故障代码进行标识。通过故障代码和系统状态来决定故障的分类；在分区健康监控表和模块健康监控表中，通过故障代码、故障发生时的系统状态与具体的故障处理程序发生联系。对进程级的故障响应是通过健康监控进程处理(该应用分区中优先级最高的进程)的，应用程序可连接自己的故障处理程序。

健康监控应包含三个健康监控表，即操作系统健康监控表、模块健康监控表和分区健康监控表。操作系统健康监控表是操作系统故障的派遣者，它由故障代码、系统状态级别和派遣级别组成。根据故障代码和故障事件发生时的系统状态，获得故障事件的派遣级别，据此级别派遣到不同的健康监控任务。每个核心模块应有一个操作系统健康监控表，它格式见表 4。

表 4 操作系统健康监控表

故障代码	系统状态级别	派遣级别

模块健康监控表主要是针对派遣到模块级健康监控任务处理的故障，进行故障恢复策略的制定，每个核心模块应有一个模块健康监控表，它的格式见表 5。

表 5 模块健康监控表

模块故障代码	系统状态级别	故障处理程序

分区健康监控表主要是针对派遣到分区级健康监控任务处理的故障，进行故障恢复策略的制定；每个分区应有一个独立的分区健康监控表，它的格式见表 6。

表 6 分区健康监控表

分区故障代码	系统状态级别	故障处理程序

进程级健康监控没有静态配置表，主要靠健康监控进程(一种特殊的进程)处理分区内的各种故障对于进程级健康监控如果用户没有创建自己的处理进程，则故障应进行上报，转分区级健康监控处理。

故障类型代码定义见表 7。

表 7 故障代码表

故障代码	含义	故障代码	含义
DEADLINE_MISSED	截止时间失效	APPLICATION_ERROR	应用错
NUMERIC_ERROR	数字错	ILLEGAL_REQUEST	非法请求
STACK_OVERFLOW	栈溢出	MEMORY_VIOLATION	存储器访问违例
HARDWARE_FAULT	硬件故障	POWER_FAIL	电源失效
KERNEL	内核错	CONFIG_ERROR	配置错

表 7(续)

故障代码	含义	故障代码	含义
INIT_ERROR	初始化错	PARTITION_OVERFLOW	分区溢出
PARTITION MODE SET	分区模式设置错	APEX INTERNAL ERROR	应用接口内部错
HARD DEADLINE MISSED	硬截止期失效	HM INTERNAL ERROR	健康监控内部错
PORT_INTERNAL_ERROR	端口内部错	LOST_TICKS	系统时钟丢失
OTHER	其它错	HM_ERROR	健康监控错
HMQ_OVERFLOW	健康监控消息队列溢出	DATA_LOSS	数据丢失
HM DEADLINE MISSED	健康监控不能执行	DEFAULT	缺省

故障的响应还应依赖于诊断故障发生时的系统状态。系统状态应由操作系统管理。与故障分级一样，系统状态也划分为三级，即模块级、分区级和进程级。系统状态定义如下：

- a) 模块级：
 - HM_UNKNOWN_STATUS 不可确定的系统状态；
 - HM_MODULE_INIT_STATUS 模块初始化状态；
 - HM_MODULE_HM_STATUS 模块健康监控任务执行状态；
 - HM_SYS_FUNC_STATUS 非健康监控任务或分区相关任务(即中断运行状态)；
 - HM_PARTITION_SWITCH_STATUS 分区切换状态。
- b) 分区级：
 - HM_PARTITION_HM_STATUS 分区健康监控任务执行状态；
 - HM_PARTITION_INIT_STATUS 分区初始化状态；
 - HM_SYSCALL_STATUS 系统调用状态。
- c) 进程级：
 - HM_PROCESS_EXEC_STATUS 进程执行状态；
 - HM_PROCESS_MGMT_STATUS 进程管理状态(系统内核或中断状态)。

5.5.3 故障的分类和恢复动作功能要求

5.5.3.1 概述

故障可发生在系统级、模块级、分区级或进程级。故障的级别主要是由发生的故障和发生故障时系统状态相符的健康监控表所决定的。故障可在它所属范围内处理，如果处理不了，则上报到更高一级处理。

5.5.3.2 故障级别

5.5.3.2.1 概述

故障分为系统级、模块级、分区级和进程级。系统级故障与载机航空电子系统相关，本指导性技术文件不做定义。

5.5.3.2.2 进程级故障

进程级故障只影响分区内的一个进程或多个进程，或影响整个分区。进程级故障有以下几种：

- a) 应用进程产生的应用故障；
 - b) 非法系统请求；
 - c) 进程执行故障(溢出、存储区冲突等)。
- 当处理进程级故障时，不会影响分区的执行。

5.5.3.2.3 分区级故障

分区级故障只影响一个分区。分区级故障有以下几种：

- a) 分区初始化时发生的分区配置故障；
- b) 分区初始化故障；
- c) 进程管理中发生的故障；
- d) 故障处理进程中发生的故障(或健康监控进程发生的故障)。

当处理分区级故障时，不会影响其他分区的执行。

5.5.3.2.4 模块级故障

模块级故障影响模块内的所有分区。模块级故障有以下几种：

- a) 模块初始化时发生的模块配置故障；
- b) 模块初始化时其它故障；
- c) 系统特权指令执行时发生的故障；
- d) 分区切换时发生的故障；
- e) 电源故障。

5.5.3.3 故障诊断和响应机制

5.5.3.3.1 概述

应诊断出来的故障如下：

- a) 硬件故障；
- b) 操作系统软件故障；
- c) 应用软件和分区软件故障。

故障响应取决于所发生的故障和该故障发生时所处的系统状态，不同故障和不同系统状态会产生不同的故障处理机制。各种故障的响应执行应是相互独立的。

对模块级和分区级的故障响应是采用一个模块健康监控表和每个分区独立的分区健康监控表以表驱动方式处理。

进程级的故障响应是应用程序可创建自己的故障处理程序进行进程级故障处理。

进程级故障可派遣到进程级故障处理。如果进程级故障处理程序不能处理，则派遣到分区级故障处理；如果分区级故障处理程序不能处理，则派遣到模块级故障处理。

分区级故障可派遣到分区级故障处理。如果分区级故障处理程序不能处理，则派遣到模块级故障处理。

模块级故障可派遣到模块级故障处理。如果模块级故障处理不能处理，则派遣到更高级故障处理(如系统级故障处理)。

健康监控的回调可定义在模块级或分区级上。只要发生一个分区级故障或模块级故障，操作系统能通过回调服务将故障传递到系统健康监控。系统健康监控回调过程的启动依赖于故障是发生在分区级的故障、还是模块级的故障。每个分区对应唯一一个健康监控的回调服务。传给健康监控回调服务的参数是操作系统诊断故障的标识和引起分区或模块健康监控表启动的系统状态。健康监控回调的入口点是在模块或分区健康监控表中配置好的。任何分区健康监控回调的设置不会影响到分区执行。

5.5.3.3.2 进程级故障响应机制

进程级故障的响应是由分区内的一个特殊进程(最高优先级的)来处理的，这个特殊进程就是故障处理进程。故障处理进程只有在正常工作模式下才被激活，并通过一个健康监控服务识别发生的故障和发生故障的进程，然后采用进程级的恢复动作(如停止和开始进程)或分区级恢复动作(如设置分区的模式为空闲态、冷启动态和热启动态)。在故障处理进程中发生的故障应归结为分区级故障。

一个故障代码可派遣为几个进程级故障(如数据错可对应于溢出、零作除数和浮点下溢等)。健康监控服务可将故障代码返回给发生故障的应用。为了保证应用的灵活性，故障代码的处理程序应是相互独立的。这些故障代码和各代码所对应的实例说明如下：

- a) 截止期超时错：进程截止期超时；

- b) 应用故障：由一个应用进程引起的故障；
- c) 数据错：在进程执行期间发生的溢出故障、零作除数和浮点故障等；
- d) 非法请求错：一个进程对操作系统的非法请求；
- e) 栈溢出错：进程的栈溢出故障；
- f) 存储区冲突错：在进程执行期间，发生的存储保护错和特权指令冲突；
- g) 硬件故障：在进程执行期间，发生的存储区奇偶错和输入/输出错；
- h) 电源故障：发生电源中断时的通知(如保留应用特殊状态的数据)。

如果在分区内，当一个进程级故障处理不存在时，操作系统将采取启动分区级故障处理机制。

5.5.3.3.3 分区级故障响应机制

分区级故障响应按以下几种途径处理：

- a) 通过检查健康监控配置表的健康监控回调服务的相关参数，操作系统将调用一个为分区所识别的过程进行处理；
- b) 在完成了健康监控回调服务的基础上，操作系统检查故障代码在分区健康监控配置表中的响应活动；
- c) 操作系统执行配置表中所响应的标识。

5.5.3.3.4 模块级故障响应机制

对于模块级故障的响应是按以下几种途径处理的：

- a) 通过检查健康监控配置表中健康监控回调服务的相关参数，操作系统将调用一个为模块所识的过程进行处理；
- b) 在完成了健康监控回调服务的基础上，操作系统检查故障代码在模块健康监控配置表中的响应活动；
- c) 操作系统执行配置表中所响应的标识。

5.5.3.4 恢复活动

5.5.3.4.1 概述

恢复活动包括进程级恢复活动、分区级恢复活动和模块级恢复活动。

5.5.3.4.2 进程级故障的恢复活动

进程级故障的恢复活动应符合 GJB 5357-2005 中 4.8.5 的规定。

5.5.3.4.3 分区级故障的恢复活动

分区级故障的恢复活动是在健康监控配置表中配置好的。用户必须为每个分区定义相关的分区健康监控表。对于各分区，故障代码和系统状态所决定的故障响应应考虑分区的执行能力(如是否可复位、进行降级处理等)。另外，恢复活动可在健康监控的回调服务中执行。恢复活动通常包括以下内容：

- a) 不动作，但是故障可通过健康监控回调服务处理；
- b) 停止分区执行；
- c) 停止并重新启动分区执行。

5.5.3.4.4 模块级故障的恢复活动

模块级故障的恢复活动是在健康监控配置表中配置好的。用户必须为每个模块定义相关的模块健康监控表。另外，恢复活动可执行在健康监控的回调服务中。恢复活动通常包括以下内容：

- a) 不动作，但是故障可通过健康监控回调服务处理；
- b) 停止模块执行；
- c) 停止并重新启动模块执行。

5.5.4 健康监控扩展接口的功能要求

健康监控扩展接口是为了使用户更好地使用操作系统的功能而给用户提供的扩展接口。

相对 GJB 5357-2005 中健康监控定义的应用软件接口，本指导性技术文件增加了以下健康监控的

接口要求：

- a) 重新加载系统健康监控表服务；
- b) 重新加载模块健康监控表服务。

5.6 文件管理

5.6.1 总要求

文件是一种数据存储的机制，文件管理包含进程存取文件所必需的信息，其内容包括文件所有者、访问权限、文件长度和当前访问的文件位置。进程通过打开、读、写、关闭、锁定和解锁等操作系统服务处理文件。进程通过一个字符串来创建/打开文件，该字符串唯一地对应一个文件，操作系统将该字符串转换为文件标识符，对文件的进一步操作都是通过文件标识符进行的。

5.6.2 文件属性定义

5.6.2.1 文件访问权限

文件的访问权限定义包括：

- a) READ：读操作；
- b) WRITE：写操作；
- c) DELETE：删除文件；
- d) DEFAULT：缺省。

5.6.2.2 文件访问方式

文件访问方式指文件允许进程并行操作的方式，包括：

- a) SHARE：共享方式；
- b) EXCLUSIVE：独占方式，文件暂时正在被一进程访问，不允许其它进程访问。

5.6.2.3 文件定位方式

文件定位方式主要用于对文件当前位置的调整，定位方式的定义包括：

- a) START_OF_FILE：从文件头开始定位；
- b) CURRENT_POSITION：从当前位置开始定位；
- c) END_OF_FILE：从文件尾开始定位。

5.6.2.4 日志文件记录格式

健康监控日志文件负责记录故障事件发生时的现场信息。

若使能自动记录日志文件，故障事件发生时的现场信息自动记录在 LOG 文件中；自动记录未使能时，可通过显式调用写日志服务将故障事件记录到日志文件。

日志文件应包含以下信息：

- a) 故障代码：故障状态的编码代号；
- b) 时间戳：发生故障时的绝对本地时间；
- c) 地址：发生故障时的程序运行地址；
- d) 故障级别：模块级、分区级、进程级；
- e) 系统状态：发生故障时的系统状态；
- f) 分区号：发生故障的分区标识符；
- g) 进程名：发生故障的进程名；
- h) 进程标识：发生故障的进程的标识符；
- i) 信息：用户对故障事件的描述信息；
- j) 信息长度：描述信息的长度。

5.6.3 文件管理应用软件接口要求

文件管理应提供以下应用软件接口要求：

- a) 创建文件服务；

- b) 删除文件服务;
- c) 打开文件服务;
- d) 关闭文件服务;
- e) 锁定文件服务;
- f) 解锁文件服务;
- g) 获得文件属性服务;
- h) 调整文件指针服务;
- i) 读文件服务;
- j) 写文件服务;
- k) 获得文件缓冲区服务;
- l) 释放文件缓冲区服务;
- m) 创建目录服务;
- n) 删除目录服务;
- o) 清空日志文件服务;
- p) 读日志记录服务;
- q) 写日志记录服务。

5.7 数据传输管理

5.7.1 通则

数据传输采用四层的管理方式,即端口层、通道层、传输连接层和接口层。每一层都有自己的协议,协议规定了同层之间数据交换格式。不允许在协议上下层之间直接拷贝数据。每一层的实现应与其他层无关。层与层之间的接口应清晰,以保证其中一层发生改变时,不影响其他层。

5.7.2 端口层管理

端口层主要功能应包括:

- a) 为应用软件提供标准的通信接口;
- b) 接收应用软件发送的数据,并传递给通道层处理;
- c) 接收通道层传递的数据,并传递给应用软件。

端口层提供采样端口和队列端口两类端口。采样端口允许消息覆盖,应用向采样端口发送数据时,如果前一个数据存在,那么新数据将覆盖旧数据;当通道上有数据到达时,如果端口中的前一个数据没有被取走,新数据将覆盖旧数据。队列端口不允许消息覆盖,当向一个队列端口发送数据时,如果端口没有缓冲区,那么调用进程将被挂起或取消本次发送;当从队列端口接收数据时,如果端口内没有数据,那么调用进程将被挂起或取消本次接收。

采样端口不允许消息分段,队列端口允许消息分段,分段大小由端口属性决定。

5.7.3 通道层管理

通道层主要功能应包括:

- a) 将端口层的数据分割成合适的消息段并发送;
- b) 将接收到的消息段排序重组形成完整消息传递给端口层。

通道是发送端和接收端之间的逻辑连接,通道定义并不局限于一个模块内。通道上发送端的数据将沿通道到达接收端。一个通道可连接多个接收端口,但最多仅能连接一个发送端口,一个端口只能连接到一个通道上。

通道连接的发送端口和接收端口可在一个核心模块内,也可在不同的核心模块内,如果在同一核心模块内,发送端口的数据将沿通道直接到达接收端口;如果不在一个核心模块内,发送端口的数据将通过 TC 到接收端口。

通道允许进行一对多通信。通道连接的接收端口可有多,发送端口的数据将沿通道到达所有的接

收端口，不管接收端口在本核心模块内还是在核心模块外。

通道允许消息分段。在发送时通道层必须将端口层的消息分割成合适的消息段后再发送；同样，通道层也必须对接收到的消息段进行排序重组，形成完整的消息后才能传递给端口层。

5.7.4 传输连接层管理

TC 层主要功能应包括：

- a) 接收通道层传递来的数据，并为数据选择路由，调用接口层服务发送数据；
- b) 从接口层接收数据，将收到的数据传递给通道层。

当通道层调用 TC 层服务发送一个数据时，TC 根据传输连接层的配置为该数据选择路由，并调用接口层服务发送数据，如果不能发送数据，必须向通道层返回相应的故障信息。当 TC 收到一个数据后，必须通知通道层，通道层调用传输连接层提供的服务接收数据。

5.7.5 接口层管理

接口层主要是接口相关的设备驱动程序，接口层必须向 TC 层提供标准的设备驱动服务。

5.7.6 数据传输管理扩展接口要求

数据传输管理应提供以下应用软件接口要求：

- a) 绑定 TC 和通道；
- b) 配置接口；
- c) 配置网络；
- d) 创建 TC；
- e) 配置所有通道；
- f) 配置并绑定所有通道；
- g) 删除 TC；
- h) 解除通道和 TC 的绑定。

5.8 蓝图配置

5.8.1 蓝图功能要求

蓝图用于描述航空电子系统的配置。它可被操作系统访问，但应用软件不能直接访问。

5.8.2 蓝图配置组成

5.8.2.1 概述

蓝图配置应由分区配置、分区调度配置、系统健康监控表配置、模块健康监控表配置和连接表配置组成。

5.8.2.2 分区配置

分区配置应由分区健康表监控配置、分区内进程配置、分区内端口配置、分区存储配置和分区属性配置组成。对分区的配置说明如下：

- a) 分区健康监控表配置：用于描述分区级健康监控的动作表、回调对象、回调服务地址、分区健康监控任务栈的大小等配置。其中动作表中每一元素的配置应包括故障代码、系统状态和对应处理程序。
- b) 分区内进程配置：用于描述一个分区内的每一个用户进程的属性，应包括进程名称、进程的入口函数、进程栈大小、基本优先级、进程的周期、进程的执行时间、进程的截止期、进程是否需要浮点支持等配置。
- c) 分区内端口配置：用于描述分区内的每一个端口的属性，应包括端口名称、端口所在分区名称、端口方向、端口模式、最大消息长度、最大消息数目、消息段长度、等待进程的排队规则、端口刷新率和端口连接的通道。
- d) 分区存储配置：用于描述分区的内存分配，应包括分区大小、分区代码的物理起始地址、分区的栈大小、分区的堆大小、存储器类型和访问权限等。

- e) 分区属性配置：包括分区名称、分区入口、分区配置文件路径、分区周期、分区执行时间、分区的关键级、该分区是否是系统分区、分区内的消息队列数目、分区内的事件数目、分区内的信号量数目、分区内的黑板数目和分区内的进程数目。

5.8.2.3 分区调度配置

描述一个主时间框架。该框架内应包含多个分区窗口，每一个分区窗口对应一个分区，每一个分区窗口配置应包括分区名称、分区窗口起始时间、分区窗口的持续时间和分区释放点标志。

5.8.2.4 系统健康监控表配置

系统健康监控表配置用于确定故障的派遣级别，包括故障代码、系统状态、派遣级别。

5.8.2.5 模块健康监控表配置

模块健康监控表配置主要配置模块级故障的处理，包括故障处理表、回调对象、回调服务地址、模块健康监控任务栈大小。其中，故障处理表中每一元素的配置应包括故障代码、系统状态、对应的处理程序。同样的故障在不同系统状态下的故障处理程序可能不相同。

5.8.2.6 连接表配置

连接表配置主要配置数据传输管理中的四层间的连接关系，定义了发送端和接收端之间的连接。连接表应包括通道配置表、TC 配置表和接口配置表。

6 扩展接口要求

6.1 分区管理

6.1.1 分区管理的数据类型

6.1.1.1 概述

分区管理的数据类型应包括分区配置类型、分区调度配置表类型和分区存储分配记录表类型。

6.1.1.2 分区配置类型

```
#define NAME_LENGTH      30
typedef char              NAME_TYPE[NAME_LENGTH];
typedef struct
{
    NAME_TYPE              Partition_name;          /* 分区名 */
    NAME_TYPE              Partition_entry;          /* 分区入口点(热启动/冷启动) */
    HM_TABLE_CFG_RECORD *Partition_HM;              /* 指向分区健康监控表的指针 */
    APEX_CHAR              CFGPath[64];              /* 分区配置文件的路径 */
    SYSTEM_TIME_TYPE       Period;                    /* 周期 */
    SYSTEM_TIME_TYPE       Duration;                  /* 持续时间 */
    APEX_INT32              Criticality;               /* 关键级别 */
    PARTITION_MEM          MemoryCFG;                 /* 分区存储分配信息 */
    APEX_CHAR              System_partition;          /* 系统分区标识，缺省是应用分区 */
    APEX_UINT32             WorkerTasksnum;           /* Worker 任务的数量(缺省值为 1) */
    APEX_UINT32             Messagequeue_number;      /* 消息队列个数 */
    APEX_UINT32             Event_number;              /* 事件个数 */
    APEX_UINT32             Scmaphore_number;         /* 信号量个数 */
    APEX_UINT32             Blackboard_number;        /* 黑板个数 */
    APEX_UINT32             Processes_number;         /* 分区内的进程数量 */
    PROCESS_ATTRIBUTE_TYPE *ProcessesPTR;            /* 分区内所有进程的链表 */
    PORT_CFG_TABLE_TYPE     Ports_table;              /* 分区端口的配置属性 */
}
```

```
} PARTITION_CFG_RECORD;
```

6.1.1.3 分区调度配置表类型

```
typedef struct
{
    NAME_TYPE      Partition_name; /*分区的名字 */
    SYSTEM_TIME_TYPE Widows_start; /* 分区执行时间起点, 最小单位: 纳秒 */
    SYSTEM_TIME_TYPE Windows_duration; /*分区窗口的持续时间, 最小单位: 纳秒 */
    APEX_BOOLEAN    Period_start_flag; /*分区释放点, =true 是发信号, =false 不发信号*/
} USER_CONFIGURE_TIME_FRAME;
```

6.1.1.4 分区存储分配记录表类型

```
typedef struct
{
    APEX_INT32      SizeBytes; /* 大小 */
    APEX_INT32      PhysicalAddress; /* 物理地址 */
    APEX_INT32      StackSize; /* 本分区的栈大小 */
    APEX_INT32      HeapSize; /* 本分区的堆大小 */
    APEX_CHAR       MemoryType; /* 存储类型 */
    APEX_CHAR       Access; /* 访问权限 */
} PARTITION_MEM;
```

6.1.2 分区管理服务

6.1.2.1 总要求

分区管理服务总要求如下:

- a) 相对于 GJB 5357-2005 分区管理, 本指导性技术文件增加了以下服务例程:
 - 1) 创建分区(CREATE_PARTITION);
 - 2) 撤消分区(DESTROY_PARTITION);
 - 3) 加载分区调度表(LOADING_PARTITION_SCHEDULE_TABLE)。
- b) 相对于 GJB 5357-2005 分区管理, 本指导性技术文件更改的服务例程是:
 设置分区模式(SET_PARTITION_MODE)。

6.1.2.2 创建分区

创建分区要求如下:

功能描述: 根据给定的分区参数, 创建指定分区, 建立有关分区的控制结构, 并返回创建好的分区的标识值, 否则返回创建不成功的返回值。本服务仅限系统分区使用。

调用格式:

```
RETURN_CODE_TYPE CREATE_PARTITION(
    PARTITION_CFG_RECORD *RECORD,
    APEX_UINT32          LENGTH,
    PARTITION_ID_TYPE    *IDENT
)
```

操作:

```
{
    // Error:
    if(检查输出值的地址是否为 NULL, 如果为 NULL, 则返回 INVALID_PARAM)
        RETURN_CODE=INVALID_PARAM;
```

```
if(应检查当前分区是否为系统分区，如果不是系统分区，则返回 NO_ACTION)
    RETURN_CODE=NO_ACTION;
if(检查分区配置(分区和端口)，如果不合法，则返回 INVALID_PARAM)
    RETURN_CODE=INVALID_PARAM;
// NORMAL:
if(要创建的分区的映像不存在 )
{
    加载分区的执行映像;
}
填写分区的起始信息;
创建指定的分区;
建立相应的数据结构，记录相应的数据到指定的全局数据结构中;
创建分区级健康监控任务;
设置本分区为冷启动态;
端口控制块表的初始化;
启动该分区执行;
IDENT=创建好的分区的标识;
RETURN_CODE=NO_ERROR;
return(RETURN_CODE);
}
```

输入值:

RECORD 分区配置信息的起始地址
LENGTH 分区配置信息的长度

输出值:

IDENT 分区标识

返回值说明:

返回值	注释
NO_ERROR	成功完成
NO_ACTION	当前分区不是系统分区
INVALID_PARAM	分区配置(分区和端口)不合法，或输出值的地址为 NULL

6.1.2.3 撤消分区(DESTROY_PARTITION)

撤消分区要求如下:

功能描述: 根据相应的分区标识符，将指定的分区撤消(包括: 删除分区的控制块、释放分区的存储空间、将主时间框架中分配给该分区的时间窗口的撤消标识置为已撤销)。正确执行此服务的条件是被撤消的分区处于空闲态。本服务仅限系统分区使用。

调用格式:

```
RETURN_CODE_TYPE DESTROY_PARTITION (
    PARTITION_ID_TYPE IDENTIFIER
)
```

操作:

```
{
    // Error:
    if(检查当前分区是否为系统分区，如果不是系统分区，则返回 NO_ACTION)
```

```
    RETURN_CODE=NO_ACTION;
if(检查所给分区的模式是否为 IDLE，如果不是，则返回 NO_ACTION)
    RETURN_CODE=NO_ACTION;
//NORMAL:
检查分区调度运行表，将对应的分区的时间窗口的是否删除的标识置为：TRUE；
清除该分区在系统中与通信相关的数据结构；
删除指定分区的健康监控；
删除指定的分区；
将分区对应的控制块表项删除并释放相关的空间；
RETURN_CODE=NO_ERROR;
return (RETURN_CODE);
}
输入值：
    IDENTIFIER          分区标识
输出值：
    无
返回值说明：
    返回值              注释
    NO_ERROR            成功完成
    NO_ACTION           当前分区不是系统分区，或当前分区的模式不是空闲态
```

6.1.2.4 加载分区调度表(LOADING_PARTITION_SCHEDULE_TABLE)

加载分区调度表要求如下：

功能描述：为操作系统的分区调度加载分区调度表，当本服务执行完成后，操作系统将在下一个主时间框架起始点开始，使用新的调度表。系统分区可通过本函数为系统重新加载调度表，以改变分区的执行序列。本服务仅限系统分区使用。

调用格式：

```
RETURN_CODE_TYPE  LOADING_PARTITION_SCHEDULE_TABLE(
    USER_CONFIGURE_TIME_FRAME  *SCHEDULE_TABLE,
    APEX_UINT32                LENGTH
)
```

操作：

```
{
    // Error:
    if(检查当前分区是否为系统分区，如果不是系统分区，则返回 NO_ACTION。)
        RETURN_CODE=NO_ACTION;
    if(检查输入指针是否为空，如果为空，则返回非法参数)
        RETURN_CODE=INVALID_PARAM;
    if(如果所给长度(LENGTH)除以结构(USER_CONFIGURE_TIME_FRAME)的大小，所获得的余数不为零，则返回非法参数)
        RETURN_CODE=INVALID_PARAM;
    if(如果分区窗口的个数超过配置个数时，则返回非法参数)
        RETURN_CODE=INVALID_PARAM;
    if(对分区释放点的判断，如果不为 true 或 false，则返回非法参数)
```

```
    RETURN_CODE=INVALID_PARAM;
    if(对分区调度表的合法性检查, 如果不合法, 则返回非法参数)
        RETURN_CODE=INVALID_PARAM;
    //NORMAL:
    分析分区窗口, 完成合法性检查;
    转换时间;
    创建分区调度运行表;
    重新加载分区调度表;
    RETURN_CODE=NO_ERROR;
    return (RETURN_CODE);
}
```

输入值:

SCHEDULE_TABLE	分区调度表
LENGTH	调度表的长度

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成
NO_ACTION	当前分区不是系统分区
INVALID_PARAM	分区时间调度表不合法

6.1.2.5 设置分区模式 (SET_PARTITION_MODE)

设置分区模式要求如下:

功能描述: 该服务用于应用分区初始化完成之后, 设置当前分区的操作模式为正常模式; 当诊断和处理故障时, 这个服务用于设置分区返回到空闲模式(分区停止)、冷启动或热启动(分区重新启动)。

调用格式:

```
RETURN_CODE_TYPE  SET_PARTITION_MODE(
    OPERATING_MODE_TYPE  OPERATING_MODE
)
```

操作:

与 GJB 5357-2005 的不同之处在于:

在对出错的判断中增加了:

```
if(OPERATING_MODE 是热启动方式且分区当前的操作模式是冷启动模式)
    RETURN_CODE=INVALID_MODE;
```

在对正常处理中增加了:

```
if(OPERATION_MODE 是 NORMAL)
{
    把所有先前启动的周期进程的下一个释放点设置为它们分区的下一个周期的起始点;
    把所有先前延迟启动的周期进程的下一个释放点设置为它们分区的下一个周期的起始点
    +包括它们的延迟时间;
    计算分区内所有非休眠态进程的截止时间;
    激活进程调度;
}
```


输入值:	
OPERATING_MODE	操作模式
输出值:	
无:	
返回值说明:	
返回值	注释
NO_ERROR	成功完成
NOT_AVAILABLE	查找失败分区控制块不成功
INVALID_MODE	模式不匹配
NO_ACTION	输入的模式等于分区的当前模式(除冷启动模式和热启动模式外)

6.2 进程管理

6.2.1 进程管理的数据类型

与 GJB 5357-2005 中的数据类型一致。

6.2.2 进程管理服务

6.2.2.1 总要求

进程管理服务的总要求如下:

- a) 相对于 GJB 5357-2005 进程管理, 本指导性技术文件增加了以下服务例程:
 - 1) 延迟启动(Delayed_Start);
 - 2) 获得自身标识(Get_My_Id);
 - 3) 时间补偿(Replenish)。
- b) 相对于 GJB 5357-2005 进程管理, 本指导性技术文件更改了服务以下例程:
 - 1) 锁定优先级抢占(Lock_Preemption);
 - 2) 挂起其他进程(Suspend);
 - 3) 开始进程的执行(Start);
 - 4) 不锁定优先级抢占(Unlock_Preemption)。

6.2.2.2 延迟启动(Delayed_Start)

延迟启动要求如下:

功能描述: 初始化一个进程的所有属性为默认值, 重置该进程的栈, 将进程置为等待状态(如设置进程从休眠态到等待态)。如果分区操作模式为正常模式, 则进程截止期会被重新计算, 根据给定的延时时间计算进程的释放点。该服务允许当前进程启动其他进程。当延时时间值为零时, 该服务等同于 Start 服务。

调用格式:

```
RETURN_CODE_TYPE Delayed_Start (
    PROCESS_ID_TYPE Process_Id,
    SYSTEM_TIME_TYPE Delay_Time
)
```

操作:

```
{
    // Error:
    if(Process_Id 标识的进程不存在)
        RETURN_CODE=INVALID_PARAM;
    if(Process_Id 标识的进程状态不为休眠态 DORAMNT)
        RETURN_CODE=NO_ACTION;
```

```

if(PROCESS_ID 标识的进程是周期进程，且延迟时间的值大于等于指定进程的周期)
    RETURN_CODE=INVALID_PARAM;

```

```

// Normal:

```

```

if(进程是非周期进程)

```

```

{
    //根据延迟时间 DELAY_TIME 启动非周期进程
    设置 PROCESS_ID 标识的进程的初始优先级为当前优先级;
    重置上下文和栈;
    if(分区操作模式为 NORMAL)
    {
        if(DELAY_TIME==0) {
            设置指定进程的状态为就绪态;
            设置 PROCESS_ID 指定的进程 DEADLINE_TIME 值(当前系统时钟 +
            TIME_CAPACITY);
        }
        else {
            设置指定进程的状态为等待态;
            设置 PROCESS_ID 指定进程的 DEADLINE_TIME 值(当前系统时钟 +
            TIME_CAPACITY + DELAY_TIME);
            设置 PROCESS_ID 指定的进程等待时间为 DELAY_TIME;
            //进程 PROCESS_ID 等待时间 DELAY_TIME 将变为就绪态}
            if(抢占调度算法被激活) 进程重调度;
        }
    }
    else
    {
        //分区的模式为冷启动或热启动模式
        设置 PROCESS_ID 所指进程为等待状态; 等待分区进入 NORMAL 模式;
        //当分区完成初始化, 转为 NORMAL 模式时, 系统将对所有等待 NORMAL 模式的
        //进程进行相应的启动, 包括设置进程状态和计算进程截止期, 并进行重调度。
    }
}

```

```

} //END-OF 根据延迟时间 DELAY_TIME 启动非周期进程

```

```

else

```

```

{
    //根据延迟时间 DELAY_TIME 启动周期进程;
    设置 PROCESS_ID 标识的进程的初始优先级为当前优先级;
    重置上下文和栈;
    if(分区操作模式为 NORMAL) {
        设置指定进程的状态为等待态;
        设置 PROCESS_ID 指定的进程的首次释放点(包括延迟启动时间);
        设置 PROCESS_ID 指定的进程 DEADLINE_TIME 值(首次释放点 +
        TIME_CAPACITY);
        // PROCESS_ID 指定的进程等待其首次释放点; }
    }
    else {
        //分区的模式为冷启动或热启动模式
        设置 PROCESS_ID 所指进程为等待状态; 等待分区进入 NORMAL 模式;
    }
}

```

```
        //当分区完成初始化，转为 NORMAL 模式时，系统将对所有等待 NORMAL 模
        //式的周期进程，计算期首次释放点和首次截止期。}
    } //END-OF 根据延迟时间 DELAY_TIME 启动周期进程
    RETURN_CODE=NO_ERROR;
    return (RETURN_CODE);
}
```

输入值:

PROCESS_ID	进程标识
DELAY_TIME	延迟时间

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	PROCESS_ID 标识的进程不存在
INVALID_PARAM	延迟时间 DELAY_TIME 的值大于等于指定进程的周期
NO_ACTION	PROCESS_ID 标识的进程状态不为休眠态 DORAMNT

6.2.2.3 获得自身标识 (GET_MY_ID)

获得自身标识要求如下:

功能描述: 用于获取当前进程的标识符。

调用格式:

```
RETURN_CODE_TYPE  GET_MY_ID(
    PROCESS_ID_TYPE  *PROCESS_ID
)
```

操作:

```
{
// Error:
    if(当前进程没有标识符，如故障处理进程)
        RETURN_CODE=INVALID_MODE;
// Normal:
    PROCESS_ID=当前进程的标识符;
    RETURN_CODE=NO_ERROR;
    return (RETURN_CODE);
}
```

输入值:

无

输出值:

PROCESS_ID	进程标识
------------	------

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_CONFIG	当前进程没有标识符

6.2.2.4 时间补偿 (REPLENISH)

时间补偿要求如下：

功能描述：该服务根据 BUDGET_TIME 的值，更改请求进程的截止期时间，对于周期进程，更改后的截止期时间不能超过周期进程的下一个释放点。

调用格式：

```
RETURN_CODE_TYPE REPLENISH(  
    SYSTEM_TIME_TYPE BUDGET_TIME  
)
```

操作：

```
{  
// Error:  
    if(进程是故障处理进程或分区的模式不是正常 NORMAL 模式)  
        RETURN_CODE=NO_ACTION;  
    if(请求的进程是周期进程且新的截止期超过进程的下一个释放点)  
        RETURN_CODE=INVALID_MODE;  
    if(BUDGET_TIME 超出范围)  
        RETURN_CODE=INVALID_PARAM;  
// Normal:  
    DEADLINE_TIME=(当前系统时间 + BUDGET_TIME);  
    //如果 BUDGET_TIME 为无限，则 DEADLINE_TIME 也为无限;  
    RETURN_CODE=NO_ERROR;  
    return(RETURN_CODE);  
}
```

输入值：

BUDGET_TIME 补偿时间

输出值：

无

返回值说明：

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	BUDGET_TIME 超出范围
INVALID_MODE	新的截止期时间超出了下一个释放点时间
NO_ACTION	调用进程是故障处理进程或分区的模式不是正常 NORMAL 模式

6.2.2.5 锁定优先级抢占(LOCK_PREEMPTION)

本指导性技术文件增加的限制是：不允许故障处理进程调用本服务，同时本服务仅允许在 NORMAL 模式下调用。

6.2.2.6 挂起其他进程(SUSPEND)

本指导性技术文件增加了以下状态处理：

- a) 故障处理进程不能调用该服务挂起当前锁定调度的进程；
- b) 如果指定进程已经被挂起，本服务将返回 NO_ACTION。

6.2.2.7 开始进程的执行(START)

开始进程的执行要求如下：

功能描述：初始化一个进程的所有属性为默认值，重置该进程的栈，将进程置为就绪状态。如果分区的操作模式为正常模式，则进程截止期和下一个释放点会被重新计算。该服务允许当前进程启动其他

进程。

调用格式：

```
RETURN_CODE_TYPE  START(
    PROCESS_ID_TYPE  PROCESS_ID
)
```

操作：

```
{
```

// Error:

```
    if(PROCESS_ID 标识的进程不存在)
```

```
        RETURN_CODE=INVALID_PARAM;
```

```
    if(PROCESS_ID 标识的进程状态不为休眠态 DORAMNT)
```

```
        RETURN_CODE=NO_ACTION;
```

// Normal:

```
    if(进程是非周期进程)
```

```
    {    //启动非周期进程
```

```
        设置 PROCESS_ID 标识的进程的初始优先级为当前优先级;
```

```
        重置上下文和栈;
```

```
        if(分区操作模式为 NORMAL) {
```

```
            设置指定进程的状态为就绪态;
```

```
            设置 PROCESS_ID 指定的进程 DEADLINE_TIME 值 (当前系统时钟 +
            TIME_CAPACITY);
```

```
            if(抢占调度算法被激活)    进程重调度; }
```

```
        else{    //分区的模式为冷启动或热启动模式
```

```
            设置 PROCESS_ID 所指进程为等待状态; 等待分区进入 NORMAL 模式;
```

```
            //当分区完成初始化, 转为 NORMAL 模式时, 系统将对所有等待 NORMAL 模
```

```
            //式的进程设置进程状态为就绪态, 计算进程截止期, 并进行重调度。}
```

```
    }//END-OF 启动非周期进程
```

```
    else
```

```
    {    //启动周期进程;
```

```
        设置 PROCESS_ID 标识的进程的初始优先级为当前优先级;
```

```
        重置上下文和栈;
```

```
        if(分区操作模式为 NORMAL) {
```

```
            设置指定进程的状态为等待态;
```

```
            设置 PROCESS_ID 指定的进程的首次释放点;
```

```
            设置 PROCESS_ID 指定的进程 DEADLINE_TIME 值 (首次释放点 +
            TIME_CAPACITY);
```

```
            // PROCESS_ID 指定的进程等待其首次释放点; }
```

```
        else{ //分区的模式为冷启动或热启动模式
```

```
            设置 PROCESS_ID 所指进程为等待状态; 等待分区进入 NORMAL 模式;
```

```
            //当分区完成初始化, 转为 NORMAL 模式时, 系统对所有等待 NORMAL 模
```

```
            //式的周期进程, 计算其首次释放点和首次截止期。}
```

```
    }//END-OF 启动周期进程
```

```
    RETURN_CODE=NO_ERROR;
```

```

        return (RETURN_CODE);
    }

```

输入值：
 PROCESS_ID 进程标识

输出值：
 无

返回值说明：
 返回值 注释
 NO_ERROR 成功完成
 INVALID_PARAM PROCESS_ID 标识的进程不存在
 NO_ACTION PROCESS_ID 标识的进程状态不为休眠态 DORAMNT

6.2.2.8 不锁定优先级抢占 (UNLOCK_PREEMPTION)

本指导性技术文件增加的限制是：不允许故障处理进程调用本服务，同时本服务仅允许在 NORMAL 模式下调用。

6.3 时间管理

6.3.1 时间管理的数据类型

6.3.1.1 概述

时间管理的数据类型包括时钟信息类型、时钟方式类型和当地时钟信息类型。

6.3.1.2 时钟信息类型

```

Typedef struct {
    CLOCKMODE      clock_mode;
    APEX_UINT32    clock_id;           /* 时钟标识 ID */
    APEX_UINT32    tc_id_from_parent; /* 上层时间的接收通道标识 */
    APEX_UINT32    tc_id_to_parent;   /* 应答上层时间的通道标识 */
    SYSTEM_TIME_TYPE sync_wave_period; /* 同步周期 */
    APEX_UINT32    max_of_missed_alt; /* ALT 最大失效值 */
    SYSTEM_TIME_TYPE range_for_alt;   /* ALT 范围 */
    SYSTEM_TIME_TYPE alt_res_bound;   /* ALT 资源范围 */
    SYSTEM_TIME_TYPE max_alt_diff;    /* 最大 ALT 数 */
    SYSTEM_TIME_TYPE timeout;         /* 时限 */
} CLOCKINFO;

```

6.3.1.3 时钟方式类型

```

Typedef enum {
    MASTER_REFERENCE_CLOCK, /* 主参考时钟 */
    REFERENCE_CLOCK,       /* 参考时钟 */
    MODULE_CLOCK           /* 模块时钟 */
} CLOCKMODE

```

6.3.1.4 当地时钟信息类型

```

Typedef struct {
    CLOCKMODE      clock_mode;
    APEX_UINT32    clock_id;           /* 时钟标识 ID */
    APEX_UINT32    Tc_Id_To_Federated; /* 传输到当地时钟的通道标识 */
    APEX_UINT32    Tc_Id_From_Federated; /* 接收当地时钟的通道标识 */
}

```

| FEDERATEDINFO:

6.3.2 时间管理服务

6.3.2.1 总要求

相对于 GJB 5357-2005 时间管理, 本指导性技术文件增加了以下服务例程:

- a) 绑定时钟(ATTACH_CLOCK);
- b) 配置时钟(CONFIGURE_CLOCK);
- c) 获得绝对全局时间(GET_AGT_TIME);
- d) 获得绝对本地时间(GET_ALT_TIME)。

6.3.2.2 绑定时钟(ATTACH CLOCK)

绑定时钟要求如下：

功能描述：本服务用于捆绑一个当地时钟到模块时钟，其结构被传输到 MSL 层以支持时间管理的相关资源，它指派的信息用于处理核心模块时钟和远程当地时钟之间的关系。本服务应支持每一个当地时钟，当地时钟的配置是相互独立的，也就是说在开始时钟计数之前不用等待所有当地时钟配置完成。操作系统将在 MOS 接口中安排这个附加装置。当请求 MOS 接口安装这个时钟无效时，本服务将返回故障码，如果成功完成，则返回 NO ERROR。

调用格式:

```
RETURN_CODE_TYPE ATTACH_CLOCK(
    FEDERATEDINFO  ATTACHINFO
)
```

操作:

1

```
// Error:
```

if(调用本服务的分区不是系统分区或分区的模式是正常 NORMAL 模式)

RETURN CODE=NO ACTION;

if(当地时钟信息中定义的时钟方式无效)

RETURN CODE=INVALID MODE:

if(指定的当地时钟信息中标识不存在)

RETURN CODE=INVALID PARAM:

```
// Normal:
```

```
// 将当地时钟信息结构传递到MSL层,完成当地时钟与模块时钟的绑定;
```

if(调用成功)

RETURN CODE=NO ERROR;

else

RETURN CODE=NO ACTION;

```
return (RETURN CODE):
```

1

输入值:

ATTACHINFO

绑定信息

输出值:

无

返回值说明:

返回值

注释

NO ERROR

成功完成

INVALID_MODE	所选时钟方式无效
INVALID_PARAM	指定的当地时钟信息中标识不存在
NO_ACTION	当前操作的分区状态不合法或 MSL 层不可完成此操作

6.3.2.3 配置时钟(CONFIGURE_CLOCK)

配置时钟要求如下：

功能描述：本服务用于配置模块的时钟方式。其结构被传输到 MSL 层以支持时间管理的相关资源，它指派时钟方式和 AGT 的数据配置管理以及 AGL 的同步(取决于同步方法)。时钟模式定义了本时钟在航空电子系统的层次关系。操作系统应答应用程序的请求后，通过 MOS 接口配置模块的时钟方式。当请求的配置时钟方式无效时，本服务返回故障代码；如果成功完成，则返回 NO_ERROR。

调用格式：

```
RETURN_CODE_TYPE CONFIGURE_CLOCK(  
    CLOCKINFO CONFIGINFO  
)
```

操作：

```
{  
// Error:  
    if(调用本服务的分区不是系统分区或分区的模式是正常 NORMAL 模式)  
        RETURN_CODE=NO_ACTION;  
    if(时钟信息中定义的时钟方式无效)  
        RETURN_CODE=INVALID_MODE;  
    if(要配置的时钟信息参数超界)  
        RETURN_CODE=INVALID_PARAM;  
// Normal:  
    // 将时钟信息结构传递到 MSL 层，完成时钟配置;  
    if(调用成功)  
        RETURN_CODE=NO_ERROR;  
    else  
        RETURN_CODE=NO_ACTION;  
    return(RETURN_CODE);  
}
```

输入值：

CONFIGINFO	配置信息
------------	------

输出值：

无

返回值说明：

返回值	注释
NO_ERROR	成功完成
INVALID_MODE	所选时钟方式无效
INVALID_PARAM	时钟信息内参数有错
NO_ACTION	当前操作的分区状态不合法或 MSL 层不可完成此操作

6.3.2.4 获得绝对全局时间(GET_AGT_TIME)

获得绝对全局时间要求如下：

功能描述：本服务将由 MSL 层维护的绝对全局时间传递给应用软件。为了获得准确的时间值，服

务的响应时间应等于或小于绝对全局时间的精度。如果从 MSL 取出的 AGT 值有效, 则服务返回 NO_ERROR, 如果 MSL 给出了故障条件, 则本服务返回故障代码。

调用格式:

```
RETURN_CODE_TYPE GET_ALT_TIME (
    SYSTEM_TIME_TYPE  *AGTIME
)
```

操作：

1

```
// Normal:
```

AGTIME=调用MSL层的获取AGT时间:

if(调用成功)

RETURN CODE=NO ERROR;

else

RETURN CODE=NO ACTION;

```
return (RETURN CODE);
```

1

输入值:

无

输出值:

AGTIME

绝对全局时间

返回值说明:

返回值

注释

NO ERROR

成功完成

NO ACTION

调用 MSL 服务不成功

6.3.2.5 获得绝对本地时间 (GET ALT TIME)

获得绝对本地时间要求如下:

功能描述：本服务将由 MSL 层维护的绝对本地时间传递给应用软件。绝对本地时间涉及到系统时间的同步，可被用于操作系统的同步或时间触发等功能。如果从 MSL 取出的 ALT 值有效，则服务返回 NO_ERROR；如果 MSL 给出了故障条件，则本服务返回故障代码。为了获得准确的时间值，服务的响应时间应等于或小于绝对本地时间的精度。

调用格式:

```
RETURN_CODE_TYPE GET_ALT_TIME (
    SYSTEM_TIME_TYPE  *ALTIME
)
```

操作:

1

```
// Normal:
```

ALTIME=调用 MSL 层的获取 ALT 时间;

if(调用成功)

RETURN CODE=NO ERROR;

Else

RETURN CODE=NO ACTION;

```
return (RETURN_CODE);
```

}	
输入值:	
无	
输出值:	
ALTIME	绝对本地时间
返回值说明:	
返回值	注释
NO_ERROR	成功完成
NO_ACTION	调用 MSL 服务不成功

6.4 分区间通信

6.4.1 分区间通信的数据类型

6.4.1.1 概述

分区间通信的数据类型包括端口配置类型和端口配置表类型。

6.4.1.2 端口配置类型

```
typedef struct
{
    NAME_TYPE          port_name;          /* 端口名称          */
    NAME_TYPE          partition_name;      /* 所属分区名称      */
    PORT_DIRECTION_TYPE direction;          /* 发送/接收          */
    PORT_MODE_TYPE      mode;               /* 采样/队列          */
    APEX_INT16          maxMsgSize;         /* 一个完整消息的最大长度 */
    APEX_INT16          maxMsgNumber;       /* 最大消息数          */
    APEX_INT16          msgSegmentLength;   /* 消息分段长度        */
    QUEUING_DISCIPLINE_TYPE discipline;     /* 排队规则            */
    SYSTEM_TIME_TYPE     refreshRate;        /* 刷新率              */
    APEX_INT32          channel_id;         /* 端口所使用的通道的标识 */
} PORT_CFG_RECORD;
```

6.4.1.3 端口配置表类型

```
typedef struct {
    PORT_CFG_RECORD *ports;                /* 端口配置          */
    APEX_INT16      sampling_port_num;      /* 采样端口数目      */
    APEX_INT16      queuing_port_num;       /* 队列端口数目      */
} PORT_CFG_TABLE_TYPE;
```

6.4.2 分区间通信服务

6.4.2.1 总要求

相对于 GJB 5357-2005 分区间通信，本指导性技术文件增加了以下服务例程：

- a) 锁定端口缓冲区 (LOCK_BUFFER);
- b) 接收端口缓冲区 (RECEIVE_BUFFER);
- c) 发送端口缓冲区 (SEND_BUFFER);
- d) 释放端口缓冲区 (UNLOCK_BUFFER)。

6.4.2.2 锁定端口缓冲区 (LOCK_BUFFER)

锁定端口缓冲区要求如下：

功能描述：该服务用于锁定队列端口的一个空闲缓冲区。如果端口有空闲缓冲区，该服务锁定一个

空闲缓冲区，并返回该空闲缓冲区的首地址；如果端口没有缓冲区，调用该服务的应用将被挂起或取消本次请求。一旦应用成功锁定一个端口的缓冲区，该缓冲区控制权属于并且只属于该应用，直到应用调用发送端口缓冲区服务释放这个缓冲区。应用不能通过该服务获得已经被锁定的缓冲区，操作系统也不能对应用已经锁定的缓冲区进行操作。该服务只适应于发送端口。

调用格式：

```
RETURN_CODE_TYPE LOCK_BUFFER (
    QUEUING_PORT_ID_TYPE    QUEUING_PORT_ID,
    MESSAGE_ADDR_TYPE        *MSG_BUFFER,
    MESSAGE_SIZE_TYPE        *LENGTH,
    SYSTEM_TIME_TYPE         TIME_OUT
)
```

操作：

```
{
    // Error:
    if(QUEUING_PORT_ID 无效)
        RETURN_CODE=INVALID_PARAM;
    if(QUEUING_PORT_ID 标识的不是队列端口)
        RETURN_CODE=INVALID_PARAM;
    if(MSG_BUFFER 非法或 LENGTH 非法)
        RETURN_CODE=INVALID_PARAM;
    if(指定端口不是源队列端口)
        RETURN_CODE=INVALID_MODE;
    //Normal:
    if(队列端口有缓冲区)
    {
        用缓冲区的地址设置输出参数 BUFFER_ADDRESS;
        用缓冲区长度设置 LENGTH;
        将缓冲区的控制权设置为该应用独占;
        RETURN_CODE=NO_ERROR;
    }
    else if(TIME_OUT 为零)
        RETURN_CODE=NOT_AVAILABLE;
    else if(如果调度被锁或调用该服务的进程是故障处理进程)
        RETURN_CODE=INVALID_MODE;
    else
    {
        if(TIME_OUT 的值不为无限大)
            按 TIME_OUT 初始化 一个超时时间计数器。
            设置调用进程的状态为等待端口有空闲缓冲区的状态;
            按端口的排队原则，把该进程加入到端口的等待队列中;
            进程重调度;
            //进程被阻塞到超时时间到或端口有空闲缓冲区
            if(超时时间到)
```

```

        RETURN_CODE=TIME_OUT;
    else    // 端口有空闲缓冲区
    {
        用缓冲区的地址设置输出参数 BUFFER_ADDRESS;
        用缓冲区长度设置 LENGTH;
        将缓冲区的控制权设置为该应用独占;
        if(TIME_OUT 的值不为无限大)
            停止超时时间计数器;
        RETURN_CODE=NO_ERROR;
    }
}
return (RETURN_CODE);
}

```

输入值:

QUEUING_PORT_ID	队列端口标识
TIME_OUT	超时时间

输出值:

MSG_BUFFER	消息存放的缓冲区
LENGTH	消息的长度

返回值说明:

返回值	注释
NO_ERROR	成功锁定一个缓冲区
INVALID_PARAM	QUEUING_PORT_ID 标识的端口不存在, 缓冲区地址指针、非法, 缓冲区长度指针非法
INVALID_MODE	——QUEUING_PORT_ID 标识的不是源端口 ——如果调度被锁、端口没有缓冲区且等待时间不为 0 ——如果调用进程是健康监控进程、端口没有缓冲区且等待时间不为 0
NOT_AVAILABLE	端口没有缓冲区, 且调用进程不等待
TIMED_OUT	端口没有缓冲区, 并且在超时时间内都没有

6.4.2.3 接收端口缓冲区(RECEIVE_BUFFER)

接收端口缓冲区要求如下:

功能描述: 该服务用于获得端口接收到数据的一个缓冲区。返回给调用该服务的应用缓冲区的地址, 应用可直接从该缓冲区取数据。通过该服务获得的缓冲区, 其控制权归该应用所有, 其他应用和操作系统都无权使用。该服务适应于接收端口。

调用格式:

```

RETURN_CODE_TYPE RECEIVE_BUFFER(
    QUEUING_PORT_ID_TYPE    QUEUING_PORT_ID,
    SYSTEM_TIME_TYPE        TIME_OUT,
    MESSAGE_ADDR_TYPE       *MSG_BUFFER,
    MESSAGE_SIZE_TYPE       *LENGTH
)

```

操作:

```

{

```

```
//Error:
    if(QUEUING_PORT_ID 无效)
        RETURN_CODE=INVALID_PARAM;
    if(MSG_BUFFER 非法或 LENGTH 非法)
        RETURN_CODE=INVALID_PARAM;
    if(QUEUING_PORT_ID 标识的端口不是接收队列端口)
        RETURN_CODE=INVALID_MODE;
//Normal:
    if(接收端口有消息)
    {
        用消息所在的缓冲区的地址设置 BUFFER_ADDRESS;
        用消息的长度设置 LENGTH;
        将消息的控制权设置为应用所独占;
        RETURN_CODE=NO_ERROR;
    }
    else if(TIME_OUT 为零)
        RETURN_CODE=INVALID_AVAILABLE;
    else if(调度被锁定或调用进程是故障处理进程)
        RETURN_CODE=INVALID_MODE;
    else
    {
        if(TIME_OUT 的值不为无限大)
            按 TIME_OUT 初始化 一个超时时间计数器;
            设置当前的进程的状态为等待消息缓冲的状态;
            按端口的排队原则, 把该进程加入到端口的等待进程队列中;
            进程重调度;
        if(超时时间到)
            RETURN_CODE=TIME_OUT;
        else
        {
            用消息所在的缓冲区的地址设置 BUFFER_ADDRESS;
            用消息的长度设置 LENGTH;
            将消息的控制权设置为应用所独占;
            RETURN_CODE=NO_ERROR;
        }
    }
    return (RETURN_CODE);
}
```

输入值:

QUEUING_PORT_ID	队列端口标识
TIME_OUT	超时时间

输出值:

MSG_BUFFER	消息存放的缓冲区
------------	----------

LENGTH	消息的长度
返回值说明:	
返回值	注释
NO_ERROR	成功从队列端口接收一个缓冲
INVALID_PARAM	QUEUING_PORT_ID 标识的端口不存在, 消息缓冲地址指针非法, 长度指针非法
INVALID_MODE	——QUEUING_PORT_ID 标识的不是口的端口 ——如果调度被锁、端口没有消息且等待时间不为 0 ——如果调用进程是健康监控进程、端口没有消息且等待时间不为 0
NOT_AVAILABLE	端口没有消息, 且调用进程不等待
TIMED_OUT	端口没有消息, 并且在等待时间内都没有

6.4.2.4 发送端口缓冲区(SEND_BUFFER)

发送端口缓冲区要求如下:

功能描述: 该服务用于发送队列端口被锁定的缓冲区。应用锁定发送端口的缓冲区, 并向该缓冲区复制数据后, 通过该服务将缓冲区的控制权转交给操作系统, 操作系统负责发送端口中的数据。操作系统发送完数据后, 该缓冲区为空闲缓冲区, 应用又可通过锁定缓冲区服务锁定该空闲缓冲区。在操作系统将缓冲区中的数据发送之前, 用户调用锁定缓冲区服务也不能获得对该缓冲区的控制。该服务只适应于发送端口。

调用格式:

```

RETURN_CODE_TYPE SEND_BUFFER(
    QUEUING_PORT_ID_TYPE QUEUING_PORT_ID,
    MESSAGE_ADDR_TYPE *MSG_BUFFER,
    MESSAGE_SIZE_TYPE LENGTH
)
```

操作:

```

{
// Error:
    if(QUEUING_PORT_ID 无效)
        RETURN_CODE=INVALID_PARAM;
    if(QUEUING_PORT_ID 标识的不是队列端口)
        RETURN_CODE=INVALID_PARAM;
    if(MSG_BUFFER 非法或 LENGTH 非法)
        RETURN_CODE=INVALID_PARAM;
    if(指定端口不是源队列端口)
        RETURN_CODE=INVALID_MODE;
//Normal:
    在端口的缓冲区中查找地址为输入参数的缓冲区:
    if(找到且缓冲区的控制权为应用所独占)
    {
        将缓冲区的控制权设置为该操作系统独占;
        RETURN_CODE=NO_ERROR;
    }
    else
```

```
        RETURN_CODE=INVALID_PARAM;
    return(RETURN_CODE);
}
```

输入值:

QUEUING_PORT_ID	队列端口标识
MSG_BUFFER,	缓冲区地址
LENGTH	消息长度

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功发送一个缓冲区
INVALID_PARAM	QUEUING_PORT_ID 标识的端口不存在, 缓冲区地址指针非法, 缓冲区长度非法, 队列端口没有参数中的端口缓冲区。
INVALID_MODE	QUEUING_PORT_ID 标识的不是源端口

6.4.2.5 释放端口缓冲区(UNLOCK_BUFFER)

释放端口缓冲区要求如下:

功能描述: 该服务用于释放端口缓冲区。当应用调用接收缓冲区服务获得一个缓冲区, 并从缓冲区复制完数据后, 必须调用该服务释放对该缓冲区的控制, 从而该缓冲区又可用于接收数据。该服务适应于接收端口。

调用格式:

```
RETURN_CODE_TYPE UNLOCK_BUFFER(
    QUEUING_PORT_ID_TYPE  QUEUING_PORT_ID,
    MESSAGE_ADDR_TYPE      *MSG_BUFFER
)
```

操作:

```
{
// Error:
    if(QUEUING_PORT_ID 无效)
        RETURN_CODE=INVALID_PARAM;
    if(MSG_BUFFER 非法)
        RETURN_CODE=INVALID_PARAM;
    if(QUEUING_PORT_ID 标识的端口不是接收队列端口)
        RETURN_CODE=INVALID_MODE;
// Normal:
    在接收端口的缓冲区中查找地址为 MSG_BUFFER 的缓冲区
    if(找到且缓冲区的控制权属于应用)
    {
        将缓冲区的控制权设置为操作系统所独占;
        RETURN_CODE=NO_ERROR;
    }
    else
        RETURN_CODE=INVALID_PARAM;
```

```
    return(RETURN_CODE);
}
```

输入值:

QUEUING_PORT_ID	端口标识
MSG_BUFFER	缓冲区地址

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功释放队列端口一个缓冲区。
INVALID_PARAM	QUEUING_PORT_ID 标识的端口不存在, 消息缓冲地址指针非法, 输入参数中的缓冲区在端口锁定链上不存在。
INVALID_MODE	QUEUING_PORT_ID 标识的不是目的端口。

6.5 健康监控

6.5.1 健康监控的数据类型

6.5.1.1 概述

健康监控的数据类型包括健康监控配置记录表类型、健康监控入口点配置记录表类型、系统健康监控事件结构类型、健康监控派遣级别类型和故障代码。

6.5.1.2 健康监控配置记录表类型

```
typedef struct
{
    APEX_INT32          crSize;          /* 配置记录的大小          */
    HM_ENTRY_CFG_RECORD * entries;       /* 分区健康监控表的入口点 */
    APEX_INT32          nEntries;        /* 分区健康监控表项的个数 */
    APEX_INT32          maxQueueDepth;   /* HM 队列深度            */
    APEX_INT32          queueThreshold;
    APEX_INT32          stackSize;       /* HM 任务栈大小          */
    APEX_INT32          maxLogEntries;
    APEX_INT32          logEntriesThreshold;
    APEX_INT32          attributeMask;    /* 属性屏蔽字              */
    APEX_UINT32         trustedPartitionsMask;
    APEX_INT32          maxErrorHandlerQueueDepth; /* 最大故障处理程序队列深度 */
    APEX_INT32          errorHandlerQueueThreshold; /* 故障处理程序队列深度门限 */
    SYSTEM_ADDRESS_TYPE hmCallback_Addr; /* callback 函数的入口地址 */
    NAME_TYPE           hmCallback_Object; /* callback 返回点(分区 name) */
} HM_TABLE_CFG_RECORD;
```

6.5.1.3 健康监控入口点配置记录表类型

```
typedef struct
{
    ERROR_CODE_TYPE     code;
    APEX_UINT32         sysStatus;
    Void                (* handlerEntry)(HM_EVENT * par);
} HM_ENTRY_CFG_RECORD;
```


6.5.1.4 系统健康监控事件结构类型

```
typedef struct
{
    APEX_UINT32          evtTag;          /* 事件标识符 */
    APEX_INT32           ParNumber;       /* 分区号 */
    enum HM_DISPATCH_LVL level;          /* 派遣级别 */
    SYSTEM_TIME_TYPE     timeStamp;       /* 时间戳 */
    APEX_UINT32          sysStatus;
    ERROR_CODE_TYPE      code;
    APEX_INT32           addInfo;         /* 附加信息 */
    SYSTEM_ADDRESS_TYPE  addr;
    PROCESS_ID_TYPE      taskId;
    NAME_TYPE            taskName;        /* 故障进程名 */
    APEX_INT32           msgLen;
    APEX_CHAR            msg[MAX_ERROR_MESSAGE_SIZE];
} HM_EVENT;
```

6.5.1.5 健康监控派遣级别类型

```
typedef enum
{
    HM_DIRECT=0,
    HM_NO_LVL,
    HM_PROCESS_LVL,
    HM_PARTITION_LVL,
    HM_MODULE_LVL,
} HM_DISPATCH_LVL;
```

6.5.1.6 故障代码

```
typedef enum
{
    DEADLINE_MISSED,
    APPLICATION_ERROR,
    NUMERIC_ERROR,
    ILLEGAL_REQUEST,
    STACK_OVERFLOW,
    MEMORY_VIOLATION,
    HARDWARE_FAULT,
    POWER_FAIL,
    KERNEL,
    CONFIG_ERROR,
    INIT_ERROR,
    PARTITION_OVERFLOW,
    PARTITION_MODE_SET,
    APEX_INTERNAL_ERROR,
    HARD_DEADLINE_MISSED,
```

```

HM_INTERNAL_ERROR,
PORT_INTERNAL_ERROR,
LOST_TICKS,
OTHER,
HM_ERROR,
HMQ_OVERFLOW,
DATA_LOSS,
HM_DEADLINE_MISSED,
HM_DEFAULT,
CBIT_ERROR,
MAX_CODES
}ERROR_CODE_TYPE;

```

6.5.2 健康监控服务

6.5.2.1 总要求

健康监控服务的总要求如下：

- a) 相对于 GJB 5357-2005 健康监控管理，本指导性技术文件增加了以下服务例程：
 - 1) 重新加载系统健康监控表(HM_SYSHMTABLE_REPLACE)；
 - 2) 重新加载模块健康监控表(HM_MODHMTABLE_REPLACE)。
- b) 相对于 GJB 5357-2005 健康监控管理，本指导性技术文件修改了以下服务例程：
 - 1) 创建故障处理进程(CREATE_ERROR_HANDLER)；
 - 2) 获得故障状态(GET_ERROR_STATUS)；
 - 3) 提交应用程序处理(RAISE_APPLICATION_ERROR)。

6.5.2.2 重新加载系统健康监控表(HM_SYSHMTABLE_REPLACE)

重新加载系统健康监控表要求如下：

功能描述：本服务提供应用重新加载系统健康监控表，使操作系统的故障处理按新的系统健康监控表进行派遣。本服务仅限系统分区使用。

调用格式：

```

RETURN_CODE_TYPE  HM_SYSHMTABLE_REPLACE(
    APEX_UINT32  *BUFFER_ADDR,
    APEX_INT32    LENGTH
)

```

操作：

```
{
```

```
// Error:
```

```
    if(检查本次替代是否会越界，否则返回 NO_ACTION。)
```

```
        RETURN_CODE=NO_ACTION;
```

```
// NORMAL:
```

```
    将原系统健康监控表清空；
```

```
    用新的系统 HM 表替代原有系统 HM 表，并初始化系统 HM 配置记录；
```

```
    RETURN_CODE=NO_ERROR;
```

```
    return (RETURN_CODE);
```

```
}
```

输入值：

BUFFER_ADDR	新的系统 HM 表首地址
LENGTH	替换长度，单位：字节

输出值：

无

返回值说明：

返回值	注释
NO_ERROR	成功完成
NO_ACTION	替换长度超出允许的范围

6.5.2.3 重新加载模块健康监控表(HM_MODHMTABLE_REPLACE)

重新加载模块健康监控表要求如下：

功能描述：本服务提供应用重新加载模块健康监控表，使操作系统的故障处理按新的模块健康监控表进行工作。本服务仅限系统分区使用。

调用格式：

```
RETURN_CODE_TYPE  HM_MODHMTABLE_REPLACE (
    APEX_UINT32  *BUFFER_ADDR,
    APEX_INT32  LENGTH
)
```

操作：

```
{
// Error:
    if(检查本次替代是否会越界，否则返回 NO_ACTION。)
        RETURN_CODE=NO_ACTION;
// NORMAL:
    将原模块健康监控表清空；
    用新的模块健康监控表替代原有模块健康监控表，并初始化模块健康监控配置记录；
    RETURN_CODE=NO_ERROR;
    return (RETURN_CODE);
}
```

输入值：

BUFFER_ADDR:	新的模块 HM 表首地址
LENGTH:	替换长度，单位：字节

输出值：

无

返回值说明：

返回值	注释
NO_ERROR	成功完成
NO_ACTION	替换长度超出允许的范围

6.5.2.4 创建故障处理进程(CREATE_ERROR_HANDLER)

创建故障处理进程要求如下：

功能描述：该服务为当前分区创建一个故障处理进程。该进程没有标识符，分区内的其它进程不能对其进行操作。故障处理进程是操作系统为了处理进程级故障(例如：截止期超时、数据错、栈溢出)而启动的一个特殊的、具有最高优先级的非周期进程；它在当前正在执行分区的时间窗口内执行，不能被其它进程挂起或停止，它的优先级也不能被修改。故障处理进程在抢占被禁止时(锁定级别不等于 0)，

仍然可抢占正在运行的进程。故障处理进程可通过调用 STOP_SELF 服务停止自己。

故障处理程序是应用程序，它能调用 STOP 服务和 START 服务去终止和重启出错进程；可调用设置分区模式服务的冷启动或热启动去重启(冷启动或热启动)整个分区；或调用设置分区模式服务的设置为空闲态来停止分区。

故障处理进程不能调用会发生系统阻塞的服务；如果故障处理进程中调用了锁定优先级抢占或解锁优先级抢占服务，则不做任何动作。

当故障处理进程正在执行时，诊断到一个故障(例如：存储区冲突、操作系统故障)，故障处理进程不能在分区上安全执行，分区健康监控表上所指定的恢复活动将自动执行。

调用格式：

```
RETURN_CODE_TYPE  CREATE_ERROR_HANDLER(
    SYSTEM_ADDRESS_TYPE  *ENTRY_POINT,
    STACK_SIZE_TYPE      STACK_SIZE
)
```

操作：

与 GJB 5357-2005 的不同之处是在对故障的判断中增加了：

```
if(没有足够的存储空间创建指定的进程)
    RETURN_CODE=INVALID_CONFIG。
```

6.5.2.5 获得故障状态(GET_ERROR_STATUS)

获得故障状态要求如下：

功能描述：本服务通常被故障处理进程调用来确定故障代码、发生故障进程的标识、故障发生的地址和与故障相关的信息。如果不止一个进程发生了故障，则需要在故障处理进程中循环调用此服务，直到所有的出错进程都处理完为止。

当多个故障同时发生时，这些故障将以一个 FIFO 队列临时存放，并且获得故障状态服务也是以 FIFO 次序处理的。如果故障是由应用程序利用 RAISE_APPLICATION_ERROR 服务提交的，那么消息是由应用来放置；其它故障的信息是与执行无关的。

调用格式：

```
RETURN_CODE_TYPE  GET_ERROR_STATUS(
    ERROR_STATUS_TYPE *ERROR_STATUS
)
```

操作：

与 GJB 5357-2005 的不同之处是在对故障的判断中增加了：

```
if(当前正在执行的进程不是一个故障处理进程)
    RETURN_CODE=INVALID_CONFIG；
```

6.5.2.6 提交应用程序处理(RAISE_APPLICATION_ERROR)

提交应用程序处理要求如下：

功能描述：提交应用程序处理服务允许当前分区以指定的故障代码(ERROR_CODE)激活故障处理进程。该故障处理进程将对出错进程实施恢复动作。如果故障处理进程未创建，则该故障将提交到分区级故障。

调用格式：

```
RETURN_CODE_TYPE  RAISE_APPLICATION_ERROR(
    ERROR_CODE_TYPE  ERROR_CODE,
    MESSAGE_AREA_TYPE  MESSAGE,
    MESSAGE_SIZE_TYPE  LENGTH
)
```

)

操作:

与 GJB 5357—2005 的不同之处是对故障的判断中增加了:

if(ERROR_CODE 不是一个应用故障)

RETURN_CODE=INVALID_PARAM;

6.6 文件管理

6.6.1 文件系统的数据类型

6.6.1.1 概述

文件系统的数据类型包括访问权限类型、打开文件属性类型、锁定状态类型、文件定位类型、删除选项类型、文件变量类型和日志文件记录结构类型。

6.6.1.2 访问权限类型

enum ACCESS_RIGHTS_TYPE(R, W, D, RW, WD, RWD, F);

句中:

R——表示读;

W——表示写;

D——表示删除;

F——表示缺省。

6.6.1.3 打开文件属性类型

typedef struct {

enum {

READ,

WRITE,

READWRITE

} USE_ACCESS; /*用户打开文件后访问文件的方式(读/写/读写)*/

enum {

SHARE,

EXCLUSIVE

} USE_CONCUR; /*多进程并发访问文件的方式(共享/独占)*/

} USE_OPTION_TYPE;

6.6.1.4 锁定状态类型

typedef enum {

LOCK, /* 文件锁定 */

UNLOCK /* 文件未锁定 */

} LOCK_STATUS_TYPE;

6.6.1.5 文件定位类型

typedef enum {

START_OF_FILE, /* 从文件头开始定位 */

CURRENT_POSITION /* 从当前位置开始定位 */

END_OF_FILE, /* 从文件尾开始定位 */

} SEEK_MODE_TYPE;

6.6.1.6 删除选项类型

typedef enum {

NORMAL, /*仅当所有对被删除的文件/目录访问均关闭时,才可删除的该文件/目录 */

IMMEDIATELY, /*立即删除文件/目录, 所有访问者均无法访问该文件/目录 */
 } DELETE_OPTION_TYPE;

6.6.1.7 文件变量类型

文件变量类型定义包括:

- a) 路径名类型 typedef APEX_CHAR PATH_NAME_TYPE[n]; PATH_NAME_TYPE 为一字符串, 其格式定义为: "<node>: <main-dir>[/<sub-dir>]/<file>", 其中<>内为 ASCII 字符(不能为冒号、斜杠);
- b) 目录名类型 typedef APEX_CHAR DIR_NAME_TYPE[n]; 目录名为一字符串, 其格式定义为: "<node>: <main-dir>[/<sub-dir>]", 其中<>内为 ASCII 字符(不能为冒号、斜杠);
- c) 文件描述符类型 typedef APEX_UINT32 FILE_HANDLE_TYPE;
- d) 地址类型 typedef APEX_UINT32 *ADDRESS_TYPE。

6.6.1.8 日志文件记录结构类型

日志文件记录的结构是:

typedef HM_EVENT LOG_RECORD_TYPE。

6.6.2 文件管理系统服务

6.6.2.1 总要求

文件管理提供一组对文件和目录的操作, 包括创建、删除、打开、关闭和文件定位等。

文件管理的服务例程包括:

- a) 创建文件(CREATE_FILE);
- b) 删除文件(DELETE_FILE);
- c) 打开文件(OPEN_FILE);
- d) 关闭文件(CLOSE_FILE);
- e) 锁定文件(LOCK_FILE);
- f) 解锁文件(UNLOCK_FILE);
- g) 获得文件属性(GET_FILE_ATTRIBUTES);
- h) 调整文件指针(SEEK_FILE);
- i) 读文件(READ_FILE);
- j) 写文件(WRITE_FILE);
- k) 获得文件缓冲区(GET_FILE_BUFFER);
- l) 释放文件缓冲区(RELEASE_FILE_BUFFER);
- m) 创建目录(CREATE_DIRECTORY);
- n) 删除目录(DELETE_DIRECTORY);
- o) 清空日志文件(CLEAR_LOG);
- p) 读日志记录(READ_LOG);
- q) 写日志记录(WRITE_LOG)。

6.6.2.2 创建文件(CREATE_FILE)

创建文件要求如下:

功能描述: 根据给定的名字创建文件, 如果目录不存在, 则创建对应的目录。访问权限指定访问本文件的其它进程的操作方式(文件所有者为创建文件的进程, 创建者具有完全访问权限), 如果未指定, 则继承目录的访问权限。

调用格式:

RETURN_CODE_TYPE CREATE_FILE (
 PATH_NAME_TYPE PATH_NAME,

```

        ACCESS_RIGHTS_TYPE    ACCESS_VALUE,
        APEX_UINT32            FILE_SIZE
    )
操作:
{
// Error:
    if(存在名为 PATH_NAME 的文件)
        RETURN_CODE=NO_ACTION;
// Normal:
    if(PATH_NAME 中未包含目录路径)
        设置文件路径为根目录;
    if(FILE_SIZE==0)
        设置文件长度属性=动态分配文件空间 //当访问文件的位置超过 end-of-file 时,
        操作系统将动态扩展文件的长度;
    else
        根据文件大小 FILE_SIZE 分配文件空间;
    if(创建文件成功)
        RETURN_CODE=NO_ERROR;
    else
        RETURN_CODE=NOT_AVAILABLE ;
    return(RETURN_CODE);
}
输入值:
    PATH_NAME:                文件名
    ACCESS_VALUE               访问权限
    FILE_SIZE                  文件大小
输出值:
    无
返回值说明:
    返回值                    注释
    NO_ERROR                  成功完成
    NO_ACTION                 文件已存在
    NOT_AVAILABLE            文件未能创建
```

6.6.2.3 删除文件(DELETE_FILE)

删除文件要求如下:

功能描述: 当所有对文件的 open 访问均关闭, 或给定的删除选项 DEL_OPT 为 IMMEDIATELY (仅在紧急状况下使用), 删除 PATH_NAME 指定的文件。

调用格式:

```
RETURN_CODE_TYPE  DELETE_FILE(
    PATH_NAME_TYPE    PATH_NAME,
    DELETE_OPTION_TYPE DEL_OPT,
    SYSTEM_TIME_TYPE  TIME_OUT
)
```

操作:

```

{
    // Error:
    if(不存在名为 PATH_NAME 的文件)
        RETURN_CODE=INVALID_PARAM;
    if(TIME_OUT 超出范围)
        RETURN_CODE=INVALID_PARAM;
    // Normal:
    if(DEL_OPT==IMMEDIATELY)
    {
        删除指定文件;
        RETURN_CODE=NO_ERROR;
        return (RETURN_CODE);
    }
    if(当前没有访问该文件的进程)
    {
        删除指定文件;
        RETURN_CODE=NO_ERROR;
        return (RETURN_CODE);
    }
    elseif(TIME_OUT 为零) // DEL_OPT==NORMAL
        RETURN_CODE=NOT_AVAILABLE
    elseif(TIME_OUT 是无穷大)
    {
        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        进程重调度;
        // 直到所有对文件的操作均完成, 该进程才会转为就绪态
        删除指定文件;
        RETURN_CODE=NO_ERROR;
    }
    else // TIME_OUT 大于零
    {
        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        按 TIME_OUT 初始化一个超时时间计数器;
        进程重调度;
        // 直到超时时间到或所有对文件的操作均完成, 该进程才会转为就绪态
        if(超时时间到)
            RETURN_CODE=TIME_OUT;
        else {
            删除指定文件;
            RETURN_CODE=NO_ERROR; }
    }
}

```



```
    }  
    return(RETURN_CODE);  
}
```

输入值:

PATH_NAME	文件名
DEL_OPT	删除标识
TIME_OUT	超时时间

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	指定的文件不存在
INVALID_PARAM	超时时间值超出允许的范围
NOT_AVAILABLE	指定的文件正在被使用, 删除模式为 NOMRMAL, 且 TIME_OUT=0
TIMED_OUT	指定的超时时间到

6.6.2.4 打开文件(OPEN_FILE)

打开文件要求如下:

功能描述: 打开 PATH_NAME 指定的文件, 设置文件访问的权限, 返回用于文件操作的文件描述符 FILE_HANDLE。

USE_OPTION 用于指定对文件的操作方式(读、写、读写), 同时指定于其它进程之间并发访问文件的方式(共享、独占)。

一个进程可使用不同方式多次打开同一文件。

调用格式:

```
RETURN_CODE_TYPE OPEN_FILE(  
    PATH_NAME_TYPE    PATH_NAME,  
    USE_OPTION_TYPE    USE_OPTION,  
    FILE_HANDLE_TYPE  *FILE_HANDLE  
)
```

操作:

```
{  
// Error:  
    if(不存在名为 PATH_NAME 的文件)  
        RETURN_CODE=NO_ACTION;  
// Normal:  
    分配用户文件描述符表  
    置当前文件位置=start-of-file;  
    设置打开属性;  
    FILE_HANDLE=文件描述符;  
    RETURN_CODE=NO_ERROR;  
    return(RETURN_CODE);  
}
```

输入值:

PATH_NAME	文件名
USE_OPTION	对文件的操作方式(读、写、读写)
输出值:	
FILE_HANDLE	文件描述符
返回值说明:	
返回值	注释
NO_ERROR	成功完成
NO_ACTION	文件已存在

6.6.2.5 关闭文件(CLOSE_FILE)

关闭文件要求如下:

功能描述: 关闭 FILE_HANDLE 指定的文件。

调用格式:

```
RETURN_CODE_TYPE CLOSE_FILE(  
    FILE_HANDLE_TYPE FILE_HANDLE  
)
```

操作:

```
{  
    // Error:  
    if(FILE_HANDLE 指定的文件未打开)  
        RETURN_CODE=NO_ACTION;  
    // Normal:  
    释放文件使用空间;  
    释放用户文件描述符表;  
    RETURN_CODE=NO_ERROR;  
    return(RETURN_CODE);  
}
```

输入值:

FILE_HANDLE	文件描述符
-------------	-------

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成
NO_ACTION	文件未打开

6.6.2.6 锁定文件(LOCK_FILE)

锁业文件要求如下:

功能描述: 锁定 FILE_HANDLE 指定的文件, 文件被锁定后, 除了拥有者外, 其它进程将不能访问它。

调用格式:

```
RETURN_CODE_TYPE LOCK_FILE(  
    FILE_HANDLE_TYPE FILE_HANDLE,  
    SYSTEM_TIME_TYPE TIME_OUT  
)
```

操作:

```
{
// Error:
    if(FILE_HANDLE 非法)
        RETURN_CODE=INVALID_PARAM;
    if(TIME_OUT 超出范围)
        RETURN_CODE=INVALID_PARAM;
    // Normal:
    if(当前文件未被锁定)
    {
        设置文件模式为 EXCLUSIVE;
        文件的所有者为当前进程;
        RETURN_CODE=NO_ERROR;
    }
    elseif(TIME_OUT 为零) // 文件已被其它进程锁定
        RETURN_CODE=NOT_AVAILABLE
    elseif(TIME_OUT 是无穷大)
    {
        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        进程重调度;
        // 直到文件变为 SHARE 模式, 该进程才会转为就绪态
        设置文件模式为 EXCLUSIVE;
        文件的所有者为当前进程;
        RETURN_CODE=NO_ERROR;
    }
    else // TIME_OUT 大于零
    {
        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        按 TIME_OUT 初始化一个超时时间计数器;
        进程重调度;
        // 直到超时时间到或文件变为 SHARE 模式, 该进程才会转为就绪态
        if(超时时间到)
            RETURN_CODE=TIME_OUT;
        else {
            设置文件模式为 EXCLUSIVE;
            文件的所有者为当前进程;
            RETURN_CODE=NO_ERROR; }
    }
    return(RETURN_CODE);
}
```

输入值:

FILE_HANDLE	文件描述符
TIME_OUT	等待时间

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	指定的文件不存在
INVALID_PARAM	TIME_OUT 值超出允许的范围
NOT_AVAILABLE	指定的文件已被锁定, 且 TIME_OUT=0
TIMED_OUT	指定的超时时间到

6.6.2.7 解锁文件(UNLOCK_FILE)

解锁文件要求如下:

功能描述: 解锁 FILE_HANDLE 指定的文件, 文件被解锁后, 进程可访问该文件。

调用格式:

```

RETURN_CODE_TYPE UNLOCK_FILE(
    FILE_HANDLE_TYPE FILE_HANDLE
)
    
```

操作:

```

{
// Error:
    if(FILE_HANDLE 非法)
        RETURN_CODE=INVALID_PARAM;
    // Normal:
    if(当前文件未被锁定)
        RETURN_CODE=NOT_AVAILABLE
    else
    {
        解锁文件, 设置文件的模式为 SHARE 模式;
        RETURN_CODE=NO_ERROER;
    }
    rcturn(RETURN_CODE);
}
    
```

输入值:

FILE_HANDLE	文件描述符
-------------	-------

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	指定的文件不存在
NOT_AVAILABLE	指定的文件未被锁定

6.6.2.8 获得文件属性(GET_FILE_ATTRIBUTES)

获得文件属性要求如下：

功能描述：获取 FILE_HANDLE 指定的文件的访问权限，文件是否被锁定。

调用格式：

```
RETURN_CODE_TYPE  GET_FILE_ATTRIBUTES (
    FILE_HANDLE_TYPE  FILE_HANDLE,
    ACCESS_RIGHTS_TYPE *ACCESS,
    LOCK_STATUS_TYPE  *LOCK_STATUS
)
```

操作：

```
{
// Error:
    if (FILE_HANDLE 非法)
        RETURN_CODE=INVALID_PARAM;
// Normal:
    获取文件的访问权限；
    获取文件的锁定状态；
    RETURN_CODE=NO_ERROER;
    return (RETURN_CODE);
}
```

输入值：

FILE_HANDLE 文件描述符

输出值：

ACCESS 访问权限
LOCK_STATUS 锁定状态

返回值说明：

返回值 注释
NO_ERROR 成功完成
INVALID_PARAM 指定的文件不存在

6.6.2.9 调整文件指针(SEEK_FILE)

调整文件指针要求如下：

表 8 文件位置调整关系表

定位模式 SEEK_MODE	SET_POS 的取值范围	NEW_POS 的值
START_OF_FILE	0...FILE_SIZE	SET_POS
CURRENT_POSITION	-Current_Position... (FILE_SIZE-Current_Position))	Current_Position+SET_POS
END_OF_FILE	-FILE_SIZE...0	FILE_SIZE +SET_POS

功能描述：调整文件的当前访问位置。根据文件定位模式 SEEK_MODE，以及位置调整参数 SET_POS，确定文件的新位置 NEW_POS(见表 8)。

调用格式：

```
RETURN_CODE_TYPE  SEEK_FILE (
    FILE_HANDLE_TYPE  FILE_HANDLE,
    SEEK_MODE_TYPE    SEEK_MODE,
```

```

    APEX_INT32          SET_POS,
    APEX_UINT32         *NEW_POS
)

```

操作:

```

{
    // Error:
    if(文件描述符 FILE_HANDLE 非法)
        RETURN_CODE=INVALID_PARAM;
    if(设置的位置 SET_POS 超过文件范围)
        RETURN_CODE=INVALID_PARAM;
    // Normal:
    if(定位模式 SEEK_MODE==START_OF_FILE)
    {
        if(0<=SET_POS<=FILE_SIZE) {
            移动文件指针;
            *NEW_POS=SET_POS;
            RETURN_CODE=NO_ERROR; }
        else
            RETURN_CODE=INVALID_PARAM;
    }
    elseif(SEEK_MODE==CURRENT_POSITION)
    {
        if(-Current_Position<=SET_POS<=( FILE_SIZE-Current_Position)) {
            移动文件指针;
            *NEW_POS=Current_Position+SET_POS;
            RETURN_CODE=NO_ERROR; }
        else
            RETURN_CODE=INVALID_PARAM;
    }
    elseif(SEEK_MODE==END_OF_FILE)
    {
        if(-( FILE_SIZE<=SET_POS<=0) {
            移动文件指针;
            *NEW_POS=FILE_SIZE | SET_POS;
            RETURN_CODE=NO_ERROR; }
        else
            RETURN_CODE=INVALID_PARAM;
    }
    return(RETURN_CODE);
}

```

输入值:

FILE_HANDLE	文件描述符
SEEK_MODE	定位模式

SET_POS	需设置的位置参数
输出值:	
NEW_POS	设置完成后返回的位置参数
返回值说明:	
返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	指定的文件不存在

6.6.2.10 读文件(READ_FILE)

读文件要求如下:

功能描述: 从 FILE_HANDLE 指定的文件中, 读取长度为 READ_COUNT 的内容放入 BUFFER_ADDRESS 指定的缓冲区中。

调用格式:

```
RETURN_CODE_TYPE  READ_FILE(  
    FILE_HANDLE_TYPE  FILE_HANDLE,  
    ADDRESS_TYPE      BUFFER_ADDRESS,  
    APEX_INT32        READ_COUNT,  
    APEX_INT32        *COUNT_READ,  
    SYSTEM_TIME_TYPE  TIME_OUT  
)
```

操作:

```
{  
// Error:  
    if(FILE_HANDLE 非法)  
        RETURN_CODE=INVALID_PARAM;  
    if(TIME_OUT 超出范围)  
        RETURN_CODE=INVALID_PARAM;  
    // Normal:  
    if(文件未被锁定)  
    {  
        设置文件模式为 EXCLUSIVE;  
        文件的所有者为当前进程;  
        if(READ_COUNT<文件当前位置到文件结束的长度){  
            将当前位置开始的 READ_COUNT 字节复制到 BUFFER_ADDRESS 的地址空间中;  
            *COUNT_READ=READ_COUNT; }  
        else {  
            将当前位置开始到文件结束的字节复制到 BUFFER_ADDRESS 的地址空间中;  
            *COUNT_READ=FILE_SIZE-Current_Position; }  
        设置文件模式为 SHARE;  
    }  
    else // 文件已被其它进程锁定  
    {  
        if(TIME_OUT 为零)  
            RETURN_CODE=NOT_AVAILABLE;
```

```
else //TIME_OUT 不为零
{
    if(TIME_OUT 是无穷大)
    {
        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        进程重调度;
        // 直到文件变为 SHARE 模式, 该进程才会转为就绪态
        设置文件模式为 EXCLUSIVE;
        文件的所有者为当前进程;
        读取文件内容; 设置读取计数 COUNT_READ;
        设置文件模式为 SHARE;
        RETURN_CODE=NO_ERROR;
    }
    else // TIME_OUT 大于零
    {
        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        按 TIME_OUT 初始化一个超时时间计数器;
        进程重调度;
        // 直到超时时间到或文件变为 SHARE 模式, 该进程才会转为就绪态
        if(超时时间到)
            RETURN_CODE=TIME_OUT;
        else
        {
            设置文件模式为 EXCLUSIVE;
            文件的所有者为当前进程;
            读取文件内容; 设置读取计数 COUNT_READ;
            设置文件模式为 SHARE;
            RETURN_CODE=NO_ERROR;
        }
    }
} // end TIME_OUT 不为零
} // end 文件已被其它进程锁定
return (RETURN_CODE);
}
```

输入值:

FILE_HANDLE	文件描述符
BUFFER_ADDRESS	缓冲区地址
READ_COUNT	需要读取的字节数
TIME_OUT	等待时间

输出值:

COUNT_READ	实际读取的字节数
------------	----------

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	指定的文件不存在
INVALID_PARAM	TIME_OUT 值超出允许的范围
NOT_AVAILABLE	指定的文件已被锁定, 且 TIME_OUT=0
TIMED_OUT	指定的超时时间到

6.6.2.11 写文件(WRITE_FILE)

写文件要求如下:

功能描述: 将 BUFFER_ADDRESS 指定地址开始的, 长度为 WRITE_COUNT 的数据内容写入 FILE_HANDLE 指定的文件。

调用格式:

```
RETURN_CODE_TYPE  WRITE_FILE (
    FILE_HANDLE_TYPE  FILE_HANDLE,
    ADDRESS_TYPE      BUFFER_ADDRESS,
    APEX_INT32        WRITE_COUNT,
    APEX_INT32        *COUNT_WRITTEN,
    SYSTEM_TIME_TYPE  TIME_OUT
)
```

操作:

```
{
// Error:
    if(FILE_HANDLE 非法)
        RETURN_CODE=INVALID_PARAM;
    if(TIME_OUT 超出范围)
        RETURN_CODE=INVALID_PARAM;
    // Normal:
    if(文件未被锁定)
    {
        设置文件模式为 EXCLUSIVE;
        文件的所有者为当前进程;
        if(WRITE_COUNT<文件当前位置到文件结束 || 文件长度可动态更改){
            将 BUFFER_ADDRESS 的地址的长度为 WRITE_COUNT 内容写入文件中;
            * COUNT_WRITTEN=WRITE_COUNT; }
    else {
        将 BUFFER_ADDRESS 所指的长度为 (FILE_SIZE-CurrentPosition) 内容写入文件中;
        * COUNT_WRITTEN=FILE_SIZE-CurrentPosition; }
        设置文件模式为 SHARE;
    }
    elseif(TIME_OUT 为零) // 文件已被其它进程锁定
        RETURN_CODE=NOT_AVAILABLE
    elseif(TIME_OUT 是无穷大)
    {
```

```

        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        进程重调度;
        // 直到文件变为 SHARE 模式, 该进程才会转为就绪态
        设置文件模式为 EXCLUSIVE;
        文件的所有者为当前进程;
        写文件; 设置写入计数 COUNT_WRITTEN;
        设置文件模式为 SHARE;
        RETURN_CODE=NO_ERROR;
    }
    else // TIME_OUT 大于零
    {
        设置当前进程状态为等待态;
        按排队原则, 将该进程插入到等待文件的进程队列中的指定位置;
        按 TIME_OUT 初始化一个超时时间计数器;
        进程重调度;
        // 直到超时时间到或文件变为 SHARE 模式, 该进程才会转为就绪态
        if(超时时间到)
            RETURN_CODE=TIME_OUT;
        else {
            设置文件模式为 EXCLUSIVE;
            文件的所有者为当前进程;
            写文件; 设置写入计数 COUNT_WRITTEN;
            设置文件模式为 SHARE;
            RETURN_CODE=NO_ERROR; }
    }
    return(RETURN_CODE);
}

```

输入值:

FILE_HANDLE	文件描述符
BUFFER_ADDRESS	缓冲区地址
WRITE_COUNT	需要些的字节数
TIME_OUT	等待时间

输出值:

COUNT_WRITTEN	实际写的字节数
---------------	---------

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	指定的文件不存在
INVALID_PARAM	TIME_OUT 值超出允许的范围
NOT_AVAILABLE	指定的文件已被锁定, 且 TIME_OUT=0
TIMED_OUT	指定的超时时间到

6.6.2.12 获得文件缓冲区(GET_FILE_BUFFER)

获得文件缓冲区要求如下：

功能描述：申请文件访问所使用的缓冲区，对写入文件的数据首先写入缓冲区中，然后调用 WRITE_FILE 服务写入对应的文件。读文件时，将文件读入到缓冲区中。

调用格式：

```
RETURN_CODE_TYPE  GET_FILE_BUFFER(
    APEX_UINT32      BUFFER_SIZE,
    ADDRESS_TYPE      BUFFER_ADDRESS,
    SYSTEM_TIME_TYPE  TIME_OUT
)
```

操作：

```
{
    // Error:
    if(BUFFER_SIZE 为零)
        RETURN_CODE=INVALID_PARAM;
    if(TIME_OUT 超出范围)
        RETURN_CODE=INVALID_PARAM;
    // Normal:
    if(分配缓冲区成功){
        BUFFER_ADDRESS=分配的缓冲区地址;
        RETURN_CODE=NO_ERROR; }
    else    // 无缓冲区
    {
        if(TIME_OUT 为零)
            RETURN_CODE=NOT_AVAILABLE
        else //TIME_OUT 为非零
        {
            if(TIME_OUT 是无穷大)
            {
                设置当前进程状态为等待态;
                按排队原则，将该进程插入到等待缓冲区的进程队列中的指定位置;
                进程重调度;
                // 直到有可用的缓冲区，该进程才会转为就绪态
                BUFFER_ADDRESS=分配的缓冲区地址;
                RETURN_CODE=NO_ERROR;
            }
            else    //  TIME_OUT 大于零
            {
                设置当前进程状态为等待态;
                按排队原则，将该进程插入到等待缓冲区的进程队列中的指定位置;
                按 TIME_OUT 初始化一个超时时间计数器;
                进程重调度;
                // 直到超时时间到或有可用的缓冲区，该进程才会转为就绪态
                if(超时时间到)
```

```
        RETURN_CODE=TIME_OUT;
    else {
        BUFFER_ADDRESS=分配的缓冲区地址;
        RETURN_CODE=NO_ERROR; }
    } //end of    TIME_OUT 大于零
} // end of TIME_OUT 为非零
} // end of 无缓冲区
return(RETURN_CODE);
}
```

输入值:

BUFFER_SIZE	缓冲区大小
TIME_OUT	等待时间

输出值:

BUFFER_ADDRESS	缓冲区地址
----------------	-------

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	缓冲区大小非法
INVALID_PARAM	TIME_OUT 值超出允许的范围
NOT_AVAILABLE	无可用的缓冲区, 且 TIME_OUT=0
TIMED_OUT	指定的超时时间到

6.6.2.13 释放文件缓冲区(RELEASE_FILE_BUFFER)

释放文件缓冲区要求如下:

功能描述: 释放用于文件访问所使用的缓冲区。

调用格式:

```
RETURN_CODE_TYPE    RELEASE_FILE_BUFFER (
    ADDRESS_TYPE      BUFFER_ADDRESS
)
```

操作:

```
{
    // Error:
    if(BUFFER_ADDRESS 为空)
        RETURN_CODE=NO_ACTION;
    // Normal:
    if(释放 BUFFER_ADDRESS 指定的空间成功)
        RETURN_CODE=NO_ERROR;
    else
        RETURN_CODE=NOT_AVAILABLE ;
    return(RETURN_CODE);
}
```

输入值:

BUFFER_ADDRESS	缓冲区地址
TIME_OUT	等待时间

输出值:	
无:	
返回值说明:	
返回值	注释
NO_ERROR	成功释放空间
NOT_AVAILABLE	未能释放空间

6.6.2.14 创建目录(CREATE_DIRECTORY)

创建目录要求如下:
功能描述: 根据给定的名字创建目录, ACCESS_VALUE 指定访问本目录的其它进程的访问权限(目录创建者具有完全访问权限), 若 ACCESS_VALUE 的值为 DEFAULT, 则访问权限继承其父目录的访问权限。

调用格式:
RETURN_CODE_TYPE CREATE_DIRECTORY (
 DIR_NAME_TYPE DIR_NAME,
 ACCESS_RIGHTS_TYPE ACCESS_VALUE
)

操作:
{
 // Error:
 if (存在名为 DIR_NAME 的目录)
 RETURN_CODE=NO_ACTION;
 // Normal:
 if (DIR_NAME 中未包含目录路径)
 设置路径为根目录;
 根据目录名, 创建目录;
 if (创建目录成功)
 RETURN_CODE=NO_ERROR;
 else
 RETURN_CODE=NOT_AVAILABLE ;
 return (RETURN_CODE);
}

输入值:	
DIR_NAME	目录名
ACCESS_VALUE	访问权限

输出值:	
无	
返回值说明:	
返回值	注释
NO_ERROR	成功完成
NO_ACTION	目录已存在
NOT_AVAILABLE	目录未能创建

6.6.2.15 删除目录(DELETE_DIRECTORY)

删除目录要求如下:

功能描述：根据给定的名字删除目录，DEL_OPT 指定删除的方式是否为立即执行，TIME_OUT 给出超时等待时间。

调用格式：

```
RETURN_CODE_TYPE  DELETE_DIRECTORY (
    DIR_NAME_TYPE      DIR_NAME,
    DELETE_OPTION_TYPE  DEL_OPT,
    SYSTEM_TIME_TYPE    TIME_OUT
)
```

操作：

```
{
// Error:
if(不存在名为 DIR_NAME 的目录)
    RETURN_CODE=INVALID_PARAM;
if(TIME_OUT 超出范围)
    RETURN_CODE=INVALID_PARAM;
// Normal:
if(DEL_OPT==IMMEDIATELY)
{
    删除指定目录：
    RETURN_CODE=NO_ERROR;
}
elseif(当前没有访问该目录的进程)
{
    删除指定目录：
    RETURN_CODE=NO_ERROR;
}
elseif(TIME_OUT 为零) // DEL_OPT==NORMAL
    RETURN_CODE=NOT_AVAILABLE
elseif(TIME_OUT 是无穷大)
{
    设置当前进程状态为等待态；
    按排队原则，将该进程插入到等待目录的进程队列中的指定位置；
    进程重调度；
    // 直到所有对目录的操作均完成，该进程才会转为就绪态
    删除指定目录：
    RETURN_CODE=NO_ERROR;
}
else // TIME_OUT 大于零
{
    设置当前进程状态为等待态；
    按排队原则，将该进程插入到等待目录的进程队列中的指定位置；
    按 TIME_OUT 初始化一个超时时间计数器；
    进程重调度；
```

```
// 直到超时时间到或所有对目录的操作均完成，该进程才会转为就绪态
if(超时时间到)
    RETURN_CODE=TIME_OUT;
Else {
    删除指定目录;
    RETURN_CODE=NO_ERROR; }
}
return(RETURN_CODE);
}
```

输入值:

DIR_NAME	目录名
DEL_OPT	删除选项
TIME_OUT	等待时间

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成
INVALID_PARAM	指定的目录不存在
INVALID_PARAM	TIME_OUT 值超出允许的范围
NOT_AVAILABLE	指定的目录正在被使用，删除模式为 NOMRMAL，且 TIME_OUT=0
TIMED_OUT	指定的超时时间到

6.6.2.16 清空日志文件(CLEAR_LOG)

清空日志文件要求如下:

功能描述: 清空日志文件中的所有记录。

调用格式:

```
RETURN_CODE_TYPE CLEAR_LOG()
```

操作:

```
{
// Normal:
    清空日志文件中的所有记录;
    RETURN_CODE=NOT_ERROR;
    return(RETURN_CODE);
}
```

输入值:

无

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功完成

6.6.2.17 读日志记录(READ_LOG)

读日志记录要求如下:

功能描述：读日志文件中的记录信息。

调用格式：

```
RETURN_CODE_TYPE  READ_LOG (
    LOG_RECORD_TYPE  *LOG_RECORD
)
```

操作：

```
{
// Normal:
    将日志文件中的记录读到 LOG_RECORD 的地址空间中；
    RETURN_CODE=NOT_ERROR;
    return (RETURN_CODE);
}
```

输入值：

无

输出值：

日志记录

返回值说明：

返回值	注释
NO_ERROR	成功完成

6.6.2.18 写日志记录(WRITE_LOG)

写日志记录要求如下：

功能描述：将用户信息写入口志文件中。

调用格式：

```
RETURN_CODE_TYPE  WRITE_LOG (
    LOG_RECORD_TYPE  *LOG_RECORD
)
```

操作：

```
{
// Normal:
    将 LOG_RECORD 的内容写入口志文件中；
    RETURN_CODE=NOT_ERROR;
    return (RETURN_CODE);
}
```

输入值：

日志记录

输出值：

无

返回值说明：

返回值	注释
NO_ERROR	成功完成

6.7 数据传输管理

6.7.1 数据传输管理的数据类型

6.7.1.1 概述

数据传输管理的数据类型包括通道类型、通道配置类型、TC 传输方向类型、TC 配置类型、接口配置类型、通道配置表类型、TC 配置表类型和接口配置表类型。

6.7.1.2 通道类型

```
typedef enum {
    FIFO_channel,      /*先进先出*/
    HISTORY_channel,   /*后进先出*/
} CHANNEL_TYPE;
```

6.7.1.3 通道配置类型

```
typedef struct {
    APEX_INT32      channel_id;      /*通道标识          */
    CHANNEL_TYPE    which_type;      /*通道类型          */
    APEX_INT32      *tc_ids;         /*一个通道可对应多个 tc */
    APEX_INT16      tc_number;       /*tc 数目           */
    APEX_INT16      recv_port_num;   /*接收端口数目      */
} CHANNEL_CFG_RECORD;
```

6.7.1.4 TC 传输方向类型

```
typedef enum {
    SEND=0,         /*发送方向*/
    RECEIVE=1       /*接收方向*/
} TC_DIRECTION_TYPE;
```

6.7.1.5 TC 配置类型

```
typedef struct {
    APEX_INT32      tc_id;           /*传输连接标识      */
    APEX_INT32      interface_id;     /*该 TC 对应的接口  */
    APEX_UCHAR      *config_data;     /*TC 的配置数据     */
    APEX_INT32      config_data_length; /*配置数据长度      */
    APEX_INT16      channel_num;      /*该 TC 上的通道数目 */
    TC_DIRECTION_TYPE direction;      /*TC 方向           */
} TC_CFG_RECORD;
```

6.7.1.6 接口配置类型

```
typedef struct {
    APEX_INT32      interface_id;     /*接口标识          */
    APEX_INT32      network_id;       /*网络标识          */
    NAME_TYPE       dev_name;         /*设备名            */
    void            *config_data;     /*接口配置数据      */
    APEX_INT16      data_length;      /*配置数据长度      */
} INTERFACE_CFG_RECORD;
```

6.7.1.7 通道配置表类型

```
typedef struct {
    CHANNEL_CFG_RECORD *channels;     /*通道配置记录*/
    APEX_INT16      channel_num;      /*配置的通道数目*/
} CHANNEL_CFG_TABLE_TYPE;
```

6.7.1.8 TC 配置表类型

```
typedef struct {
    TC_CFG_RECORD *tcs;    /*传输连接配置记录*/
    APEX_INT16      tc_num; /*配置的传输连接数目*/
} TC_CFG_TABLE_TYPE;
```

6.7.1.9 接口配置表类型

```
typedef struct {
    INTERFACE_CFG_RECORD *interfaces; /*接口配置记录*/
    APEX_INT16      interface_num;    /*配置的接口数目*/
} INTERFACE_CFG_TABLE_TYPE;
```

6.7.2 数据传输管理服务

6.7.2.1 总要求

数据传输的服务例程应包括：

- a) 绑定传输连接和通道(ATTATCH_CHANNEL_TO_TC);
- b) 解除通道和传输连接的绑定(DETACH_CHANNEL_FROM_TC);
- c) 配置接口(CONFIGURE_INTERFACE);
- d) 配置网络(CONFIGURE_NETWORK);
- e) 创建传输连接(CREATE_TC);
- f) 配置所有通道(CONFIG_ALL_CHANNELS);
- g) 配置并绑定所有通道(CONFIG_ALL_CHANNELS_ATTACH);
- h) 删除传输连接(DESTROY_TC)。

6.7.2.2 绑定传输连接和通道(ATTATCH_CHANNEL_TO_TC)

绑定传输连接和通道要求如下：

功能描述：绑定通道和 TC。绑定以后，通道上发送端口的数据将通过 TC 来发送，TC 上接收到的数据传递给通道上的接收端口。本服务仅限系统分区使用。

调用格式：

```
RETURN_CODE_TYPE ATTATCH_CHANNEL_TO_TC(
    APEX_INT32      TC_ID,
    APEX_INT32      CHANNEL_ID
)
```

操作：

```
{
    //Error:
    if(当前分区不是系统分区)
        RETURN_CODE=NO_ACTION;
    if(当前分区不是系统分区)
        RETURN_CODE=NO_ACTION;
    if(TC_ID、通道标识非法)
        RETURN_CODE=INVALID_CONFIG;
    if(TC 上允许绑定的通道数目超过了配置值)
        RETURN_CODE=INVALID_CONFIG;
    if(通道可绑定的 TC 数超过了配置值)
        RETURN_CODE=INVALID_CONFIG
    //Normal:
```

```
        将通道标识记录到 TC 的控制表中；
        将 TC_ID 记录到通道的控制表中；
        RETURN_CODE=NO_ERROR;
    }
```

输入值:

TC_ID 传输连接标识
CHANNEL_ID 通道标识

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	绑定成功
NO_ACTION	调用该服务的分区不是系统分区
INVALID_CONFIG	TC_ID、通道标识非法，TC 上允许绑定的通道数超过了配置值，通道可绑定的 TC 数超过了配置值，通道没有配置或 TC 没有创建

6.7.2.3 解除通道和传输连接的绑定(DETACH_CHANNEL_FROM_TC)

解除通道和传输连接的绑定要求如下：
功能描述：解除通道和 TC 的绑定关系。本服务仅限系统分区使用。
调用格式：

```
RETURN_CODE_TYPE DETACH_CHANNEL_FROM_TC(  
    APEX_INT32        TC_ID,  
    APEX_INT32        CHANNEL_ID  
)
```

操作:

```
{  
//Error:  
    if(当前分区不是系统分区)  
        RETURN_CODE=NO_ACTION;  
    if(TC_ID 通道标识 非法)  
        RETURN_CODE=INVALID_CONFIG;  
    if(通道和 TC 不存在绑定关系)  
        RETURN_CODE=INVALID_PARAM;  
//Normal:  
    解除通道和 TC 的绑定;  
    RETURN_CODE=NO_ERROR;  
}
```

输入值:

TC_ID 传输连接标识
CHANNEL_ID 通道标识

输出值:

无

返回值说明:

返回值	注释
-----	----

NO_ERROR	操作成功
NO_ACTION	调用该服务的分区不是系统分区
INVALID_CONFIG	TC_ID 通道标识非法
INVALID_PARAM	通道和 TC 不存在绑定关系

6.7.2.4 配置接口(CONFIGURE_INTERFACE)

配置接口要求如下:

功能描述: 配置接口设备, 主要是对接口设备进行初始化。本服务仅限系统分区使用。

调用格式:

```
RETURN_CODE_TYPE CONFIGURE_INTERFACE(  
    APEX_INT32    INTERFACE_ID,  
    NAME_TYPE     DEV_NAME,  
    APEX_INT32    NETWORK_ID,  
    APEX_VOID     *CFG_DATA,  
    APEX_INT32    DATA_LENGTH  
)
```

操作:

```
{  
    //Error:  
    if(当前分区不是系统分区)  
        RETURN_CODE=NO_ACTION;  
    if(INTERFACE_ID|| NETWORK_ID 无效)  
        RETURN_CODE=INVALID_CONFIG;  
    if(CFG_DATA 非法||DATA_LENGTH 非法)  
        RETURN_CODE=INVALID_PARAM;  
    if(无法对设备进行初始化操作)  
        RETURN_CODE=INVALID_CONFIG;  
    //Normal:  
    初始化并打开接口设备;  
    RETURN_CODE=NO_ERROR;  
}
```

输入值:

INTERFACE_ID	接口标识
DEV_NAME,	设备名
NETWORK_ID	网络标识
CFG_DATA	配置数据
DATA_LENGTH	配置数据长度

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	成功配置 一个接口
NO_ACTION	调用该服务的分区不是系统分区
INVALID_PARAM	CFG_DATA 非法或 DATA_LENGTH 非法

INVALID_CONFIG INTERFACE_ID NETWORK_ID 无效，或无法对设备进行初始化和打开操作

6.7.2.5 配置网络(CONFIGURE_NETWORK)

配置网络要求如下：
功能描述：配置网络模块。该服务仅限系统分区使用。
调用格式：

```
RETURN_CODE_TYPE CONFIGURE_NETWORK (
    APEX_INT32      NETWORK_ID,
    APEX_INT32      TC_ID,
    APEX_VOID       *CFG_DATA,
    APEX_INT32      DATA_LENGTH
)
```

```
操作：
{
    //Error:
    if(当前分区不是系统分区)
        RETURN_CODE=NO_ACTION;
    if(TC_ID 非法)
        RETURN_CODE=INVALID_CONFIG;
    if(CFG_DATA 或 DATA_LENGTH 非法)
        RETURN_CODE=INVALID_CONFIG;
    //Normal:
    特定网络的配置函数；
    RETURN_CODE=NO_ERROR;
}
```

输入值：

TC_ID	传输连接标识
NETWORK_ID	网络标识
CFG_DATA	配置数据
DATA_LENGTH	配置数据长度

输出值：

无	
---	--

返回值说明：

返回值	注释
NO_ERROR	配置成功
NO_ACTION	调用该服务的分区不是系统分区
INVALID_CONFIG	TC_ID 非法，配置数据长度非法

6.7.2.6 创建传输连接(CREATE_TC)

创建传输连接要求如下：
功能描述：创建一个传输连接。该服务仅限系统分区使用。
调用格式：

```
RETURN_CODE_TYPE CREATE_TC(
    APEX_INT32      TC_ID,
```

```

        APEX_INT32          INTERFACE_ID,
        APEX_INT32          CHANNEL_NUMBER,
        TC_DIRECTION_TYPE   DIRECTION,
        APEX_UCHAR          *CFG_DATA,
        APEX_INT32          CFG_DATA_LENGTH
    )

```

操作:

```

{
    //Error:
    if(当前分区不是系统分区)
        RETURN_CODE=NO_ACTION;
    if(TC_ID、接口标识、通道数目、配置数据长度、TC 方向 非法)
        RETURN_CODE=INVALID_CONFIG;
    if(接口标识所标识的接口不存在或未配置)
        RETURN_CODE=INVALID_CONFIG;

    //Normal:
    记录 TC 的配置数据;
    配置 TC;
    RETURN_CODE=NO_ERROR;
}

```

输入值:

TC_ID	传输连接标识
INTERFACE_ID	接口标识
CHANNEL_NUMBER	通道数目
DIRECTION	传输连接方向
CFG_DATA	配置数据
CFG_DATA_LENGTH	配置数据长度

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	创建成功
NO_ACTION	调用该服务的分区不是系统分区
INVALID_CONFIG	TC_ID、接口标识、通道数目、配置数据长度、TC 方向非法，接口不存在，创建的 TC 数超过了最大数

6.7.2.7 配置所有通道(CONFIG_ALL_CHANNELS)

配置所有通道要求如下:

功能描述: 配置参数中指定的所有通道，但是并不进行通道和 TC 的绑定。该服务仅限系统分区使用。

调用格式:

```

RETURN_CODE_TYPE   CONFIG_ALL_CHANNELS(
    CHANNEL_CFG_TABLE_TYPE  *CHANNEL_CONFIG_TABLE
)

```

```
操作：
{
    //Error:
    if(当前分区不是系统分区)
        RETURN_CODE=NO_ACTION;
    if(CHANNEL_CONFIG_TABLE 非法)
        RETURN_CODE=INVALID_CONFIG;
    //Normal:
    记录每一个通道的配置数据；
    配置每一个通道；
    RETURN_CODE=NO_ERROR;
}
```

输入值：
CHANNEL_CONFIG_TABLE 通道配置表

输出值：
无

返回值说明：

返回值	注释
NO_ERROR	配置所有通道成功
NO_ACTION	调用该服务的分区不是系统分区
INVALID_CONFIG	通道配置非法

6.7.2.8 配置并绑定所有通道(CONFIG_ALL_CHANNELS_ATTACH)

配置并绑定所有通道要求如下：
功能描述：配置参数中指定的所有通道，并将通道和 TC 进行绑定。该服务仅限系统分区使用。
调用格式：

```
RETURN_CODE_TYPE CONFIG_ALL_CHANNELS_ATTACH (
    CHANNEL_CFG_TABLE_TYPE *CHANNEL_CONFIG_TABLE
)
```

```
操作：
{
    //Error:
    if(当前分区不是系统分区)
        RETURN_CODE=NO_ACTION;
    if(CHANNEL_CONFIG_TABLE 非法)
        RETURN_CODE=INVALID_CONFIG;
    //Normal:
    记录每一个通道的配置数据；
    配置每一个通道；
    绑定通道和 TC；
    RETURN_CODE=NO_ERROR;
}
```

输入值：
CHANNEL_CONFIG_TABLE 通道配置表

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	配置所有通道成功
NO_ACTION	调用该服务的分区不是系统分区
INVALID_CONFIG	通道配置非法

6.7.2.9 删除传输连接(DESTROY_TC)

删除传输连接要求如下:

功能描述: 删除一个传输连接。该服务仅限系统分区使用。

调用格式:

```
RETURN_CODE_TYPE DESTROY_TC (
    APEX_INT32    TC_ID
)
```

操作:

```
{
//Error:
    if(当前分区不是系统分区)
        RETURN_CODE=NO_ACTION;
    if(TC_ID 非法)
        RETURN_CODE=INVALID_CONFIG;
    if(TC 上绑定有通道)
        RETURN_CODE=INVALID_PARAM;
//Normal:
    删除传输连接;
    RETURN_CODE=NO_ERROR;
}
```

输入值:

TC_ID	传输连接标识
-------	--------

输出值:

无

返回值说明:

返回值	注释
NO_ERROR	删除成功
NO_ACTION	调用该服务的分区不是系统分区
INVALID_CONFIG	TC_ID 非法
INVALID_PARAM	TC 上绑定有通道

附录 A
(规范性附录)
蓝图配置要求

A.1 蓝图配置工具

航空电子系统人员应利用 XML 工具定义配置规则——XML-Schema，根据规则可生成不同的配置 XML 实例文件。根据 XML-Schema 开发转换工具，将 XML 实例文件生成操作系统可识别的蓝图配置格式。

A.2 蓝图配置的生成

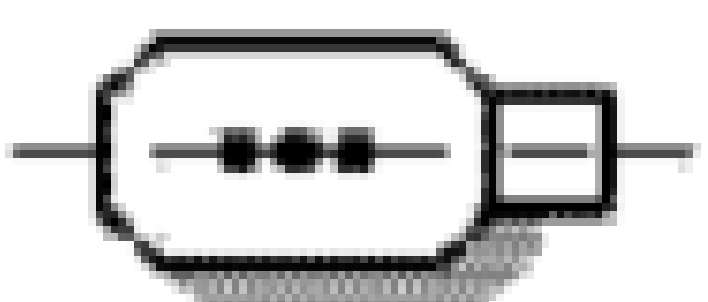



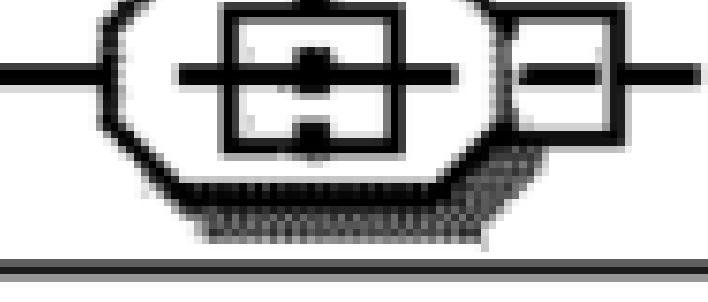
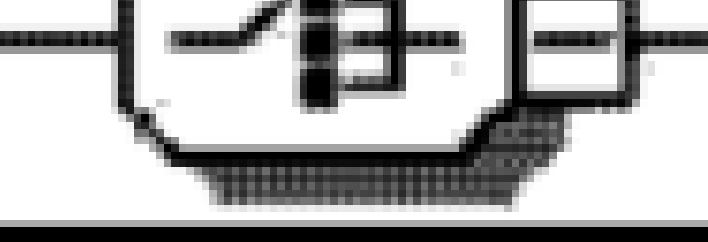
蓝图配置生成的步骤如下：

- a) 开发 XML-Schema，定义配置数据的结构；
- b) 操作系统开发者开发转换工具，该工具以 XML-Schema 为依据，可将 XML 配置文件转换成蓝图配置；
- c) 系统集成者和应用开发者依据 XML-Schema 生成 XML 配置文件；
- d) 系统集成者和应用开发者验证生成的 XML 文件的正确性；
- e) 系统集成者使用操作开发者提供的转换工具，将 XML 文件转换成操作系统可识别的蓝图配置格式。

A.3 基本图素

XML-Schema 中与蓝图配置相关的图素见表 A.1。

表 A.1 XML-Schema 中与蓝图配置相关的图素表

图素	说明
	等同于 XML 定义中的 ‘xsd: sequence’ 元素，所有的子元素必须按顺序出现
	等同于 XML 定义中的 ‘xsd: choice’ 元素，在所有子元素中只有一个可被选择
	等同于 XML 定义中的 ‘xsd: all’ 元素，所有子元素可按任意顺序出现
	等同于 XML 定义中的标记，用来标记 XML 文件中的数据
	被创建的类型定义，用来建立 SCHEMA 以及重用一些基
	等同于 XML 定义中的 ‘xsd: all’ 定义，允许用户定义 schema 的扩展

A.4 蓝图配置示例

蓝图配置示例参见附录 B。

A.5 XML-Schema 代码

能满足蓝图配置要求的 XML-Schema 代码参见附录 B。

附录 B
(资料性附录)
蓝图配置示例

B.1 系统配置结构

使用 XML-spy 生成 XML-schema，描述系统的配置结构(见图 B.1)，系统配置包括系统健康监控表、模块健康监控表、分区配置表、分区调度表和连接表(见图 B.2~图 B.6)。系统监控表配置特定的故障在特定的系统状态下的派遣级别；模块监控表定义了对模块级的故障采取的动作；分区配置主要是对分区资源配置，主要包括端口配置、进程配置、分区内存配置和分区健康监控配置等；分区调度表描述了分区的调度规则；连接表包括通道配置表、TC 表和接口配置表描述了系统的连接关系。

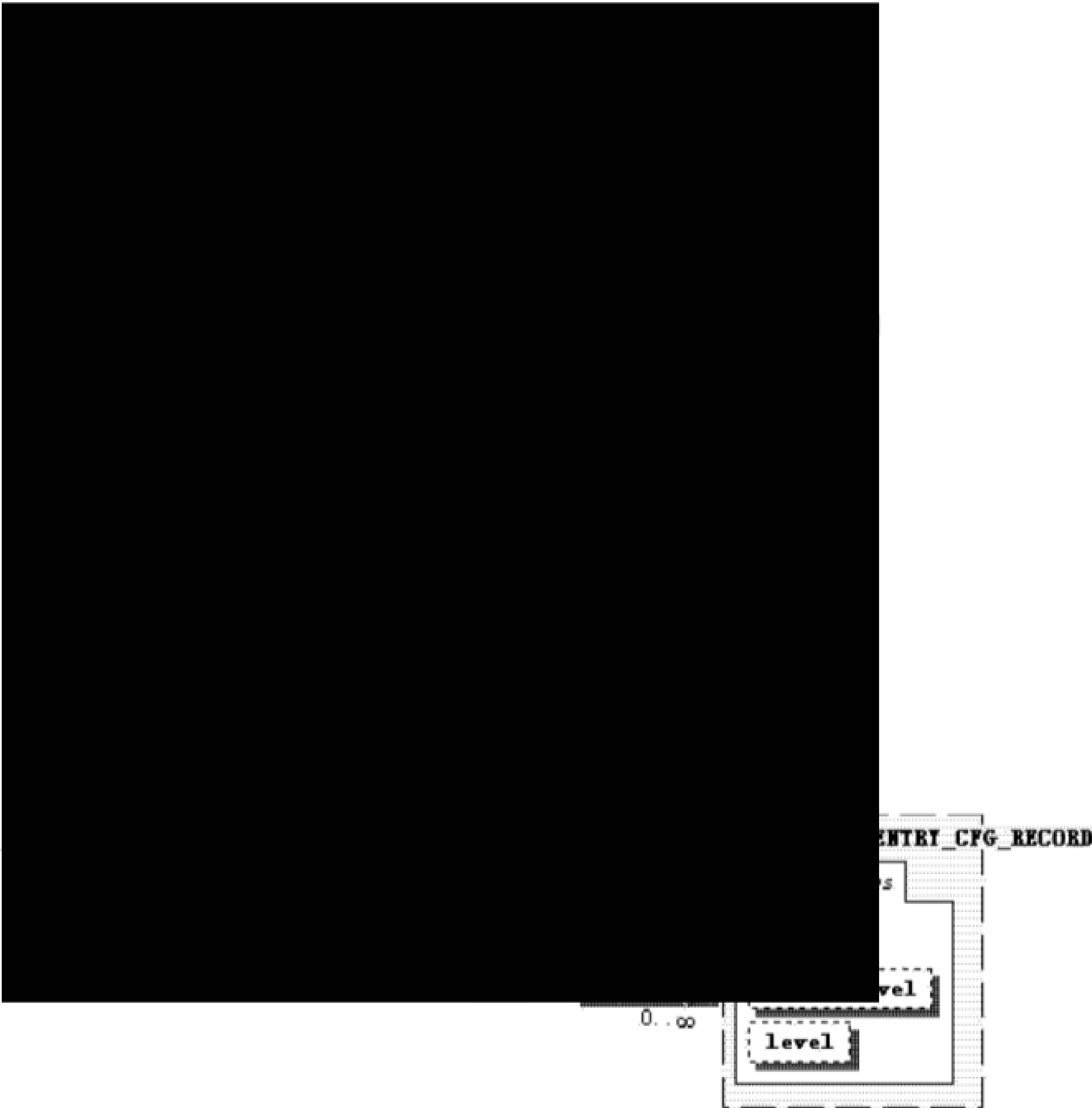


图 B.2 系统健康监控表结构图



图 B.3 模块健康监测表结构图



图 B.4 分区配置结构图



图 B.5 分区调度表结构图

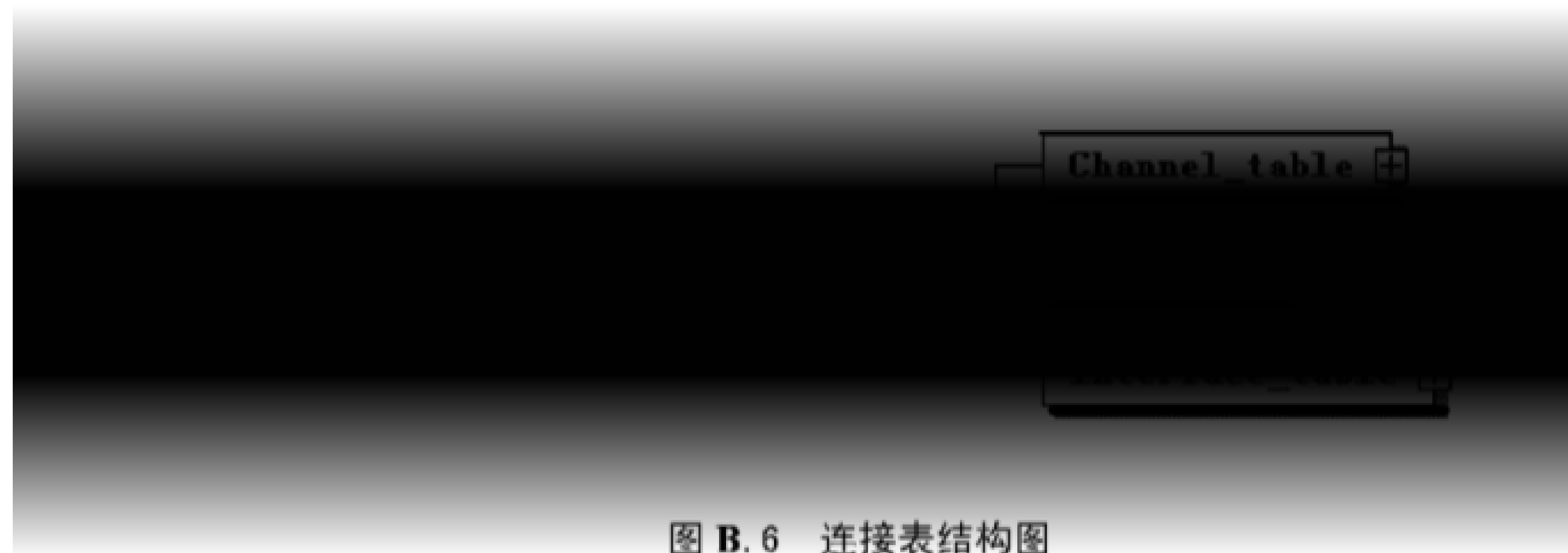


图 B.6 连接表结构图

B.2 XML 配置文件结构

根据 XML-schema，生成 XML 配置文件(见图 B.7)，XML 配置文件包括：系统健康监控表、模块健康监控表、分区配置表、分区调度表、通道配置表、传输连接配置表、接口配置表。其中各部分的配置见图 B.7~图 B.12。

GJB5357-2005	
xmlns:xdb	http://xmlns.oracle.com/xdb
xmlns:rsi	http://www.w3.org/2001/XMLSchema-instance
rsi:noNamespaceSchemaLocation	D:\GJB5357_CFG\GJB5357_CFG.xsd
System_HM_Table	nEntries=12 crSize=1024
Module_HM_Table	stackSize=2000 nEntries=13 trustedPartitionsMask=0 logEntriesThreshold=5
Partition (2)	
Module_Schedule	
Connection_Table	

图 B.7 XML 配置文件

System_HM_Table	nEntries	12		
	crSize	1024		
	entries (12)			
		level	StatusLevel	code
	1	HM_MODULE_LVL	HM_MODULE_MODE	KERNEL
	2	HM_MODULE_LVL	HM_PARTITION_MODE	KERNEL
	3	HM_PARTITION_LVL	HM_PROCESS_MODE	KERNEL
	4	HM_PARTITION_LVL	HM_PROCESS_MODE	APEX_INTERNAL_ERROR
	5	HM_PARTITION_LVL	HM_PARTITION_MODE	APEX_INTERNAL_ERROR
	6	HM_MODULE_LVL	HM_MODULE_MODE	APEX_INTERNAL_ERROR
	7	HM_PARTITION_LVL	HM_PROCESS_MODE	PORT_INTERNAL_ERROR
	8	HM_PARTITION_LVL	HM_PARTITION_MODE	PORT_INTERNAL_ERROR
	9	HM_MODULE_LVL	HM_MODULE_MODE	PORT_INTERNAL_ERROR
	10	HM_PROCESS_LVL	HM_PROCESS_MODE	HARDWARE_FAULT
	11	HM_MODULE_LVL	HM_PARTITION_MODE	HARDWARE_FAULT
	12	HM_MODULE_LVL	HM_MODULE_MODE	HARDWARE_FAULT

图 B.8 系统健康监控表

Module_HM_Table	stackSize	2000		
	nEntries	13		
	trustedPartitionsMask	0		
	logEntriesThreshold	5		
	hmCallback_Addr	0x00000000		
	attributeMask	1		
	crSize	1024		
	hmCallback_Object	0		
	maxQueueDepth	20		
	maxErrorHandlerQueueDepth	0		
	maxLogEntries	2		
	queueThreshold	5		
	errorHandlerQueueThreshold	5		
	entries (13)	handlerEntry	code	sysStatus
	1	(void (*)OEM_EVENT *)HM_Actions_Shutdown	KERNEL	HM_MODULE_INIT_STATUS
	2	(void (*)OEM_EVENT *)HM_Actions_FarSetIdle_ModHM	KERNEL	HM_PROCESS_MGMT_STATUS
	3	(void (*)OEM_EVENT *)HM_Actions_Ignore_NoHM	APEX_INTERNAL_ERROR	HM_MODULE_INIT_STATUS
	4	(void (*)OEM_EVENT *)HM_Actions_Ignore_NoHM	PORT_INTERNAL_ERROR	HM_PARTITION_SWITCH_STATUS
	5	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_SYSCALL_STATUS
	6	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_PARTITION_HM_STATUS
	7	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_PARTITION_INIT_STATUS
	8	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_PROCESS_MGMT_STATUS
	9	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_PARTITION_SWITCH_STATUS
	10	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_SYS_FUNC_STATUS
	11	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_MODULE_HM_STATUS
	12	void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	UNKNOWN_STATUS
	13	(void (*)OEM_EVENT *)HM_Actions_Reboot	HARDWARE_FAULT	HM_MODULE_INIT_STATUS

图 B.9 模块健康监控表

Partition	
WorkerTasknum	5
Period	400000000
Partition_entry	PART1_ENTRY
Semaphore_number	20
Criticality	1
Messagequeue_number	10
System_partition	1
Event_number	8
Partition_name	part1
Blackboard_number	0
Duration	400000000
Processes_number	10
CFGPath	""
Partition_MM (1)	
MemoryCFG (1)	
ProcessesPTR	
BASE_PRIORITY	1
ENTRY_POINT	0
NAME	USER_INIT
FLOAT_REQUIREMENT	1
TIME_CAPACITY	10000000
STACK_SIZE	4096
PERIOD	50000000
DEADLINE	SOFT
ProcessesPTR	
BASE_PRIORITY	15
ENTRY_POINT	0
NAME	p1
FLOAT_REQUIREMENT	0
TIME_CAPACITY	10000000
STACK_SIZE	4096
PERIOD	50000000
DEADLINE	SOFT
Ports_table	
sampling_port_num	1
queuing_port_num	1
ports	
refreshRate	100000000
msgSegmentLength	128
discipline	FIFO
port_name	samplingport
partition_name	part1
maxMsgSize	128
maxMsgNumber	1
direction	SOURCE
channel_id	1
mode	SAMPLING
ports	
refreshRate	0
msgSegmentLength	152
discipline	PRIORITY
port_name	queuingport
partition_name	part1
maxMsgSize	128
maxMsgNumber	6
direction	SOURCE
channel_id	2
mode	QUEUING

图 B.10 分区配置表

Module_Schedule	
Window	
Windows_start	0
Partition_name	"part1"
Windows_duration	40000000
Period_start_flag	1
Window	
Windows_start	40000000
Partition_name	"part2"
Windows_duration	40000000
Period_start_flag	0
Window	
Windows_start	0
Partition_name	0
Windows_duration	0
Period_start_flag	0

图 B.11 分区调度表

Connection_Table							
Channel_table							
channel_num		3					
channels (3)							
		tc number	which type	recv port num	channel id	tc ids	
1	1	FIFO_channel	0	1	1		
2	0	HISTORY_channel	1	2			
3	1	FIFO_channel	1	3	2		
TC_table							
tc num		2					
tes (2)							
		config data	interface id	config data length	tc id	channel num	direction
1	0	1	0	1	1		SEND
2	0	2	0	2	1		RECEIVE
Interface_table							
interface num		2					
interfaces (2)							
		dev name	config data	interface id	network id	data length	
1	"sio"	0	1	0	0	0	
2	"fei82557"	0	2	0x8010	0	0	

图 B.12 连接表

附 录 C

(资料性附录)

XML-Schema 代码

C.1 代码说明

本附录提供了满足 GJB 5357-2005 配置要求的 XML-Schema 的代码。使用商用 XML 工具 Altova XMLSpy 生成的该代码。

C.2 代码示例

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 U (http://www.xmlspy.com) by xxguang (631) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xdb="http://xmlns.oracle.com/xdb" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="GJB5357-2005">
        <xs:annotation>
            <xs:documentation>Comment describing your root element</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence>
                <xs:element name="System_HM_Table"
type="HM_SYSTEM_TABLE_CFG_RECORD"/>
                <xs:element name="Module_HM_Table"
type="HM_TABLE_CFG_RECORD"/>
                <xs:element name="Partition" type="PARTITION_CFG_RECORD"
maxOccurs="unbounded"/>
                <xs:element name="Module_Schedule" type="MODULE_SCHED_TYPE"/>
                <xs:element name="Connection_Table"
type="CONNECTION_TABLE_TYPE"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="APEX_CHAR">
        <xs:restriction base="DccOrHexValueType"/>
    </xs:simpleType>
    <xs:simpleType name="APEX_UCHAR">
        <xs:restriction base="DccOrHexValueType"/>
    </xs:simpleType>
    <xs:simpleType name="APEX_INT16">
        <xs:restriction base="DecOrHexValueType"/>
    </xs:simpleType>
    <xs:simpleType name="APEX_UINT16">
```



```

    <xs:restriction base="DecOrHexValueType"/>
</xs:simpleType>
<xs:simpleType name="APEX_INT32">
    <xs:restriction base="DecOrHexValueType"/>
</xs:simpleType>
<xs:simpleType name="APEX_UINT32">
    <xs:restriction base="DecOrHexValueType"/>
</xs:simpleType>
<xs:simpleType name="APEX_UINT64">
    <xs:restriction base="DecOrHexValueType"/>
</xs:simpleType>
<xs:simpleType name="NAME_TYPE">
    <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
        <xs:maxLength value="30"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="APEX_INT64">
    <xs:restriction base="DecOrHexValueType"/>
</xs:simpleType>
<xs:simpleType name="SYSTEM_ADDRESS_TYPE">
    <xs:restriction base="DecOrHexValueType"/>
</xs:simpleType>
<xs:simpleType name="MESSAGE_ADDR_TYPE">
    <xs:restriction base="APEX_UINT32"/>
</xs:simpleType>
<xs:simpleType name="MESSAGE_SIZE_TYPE">
    <xs:restriction base="APEX_INT32"/>
</xs:simpleType>
<xs:simpleType name="MESSAGE_RANGE_TYPE">
    <xs:restriction base="APEX_INT32"/>
</xs:simpleType>
<xs:simpleType name="SYSTEM_TIME_TYPE">
    <xs:restriction base="APEX_INT64"/>
</xs:simpleType>
<xs:simpleType name="PROCESS_ID_TYPE">
    <xs:restriction base="APEX_INT32"/>
</xs:simpleType>
<xs:simpleType name="LOCK_LEVEL_TYPE">
    <xs:restriction base="APEX_INT32"/>
</xs:simpleType>
<xs:simpleType name="STACK_SIZE_TYPE">
    <xs:restriction base="APEX_UINT32"/>

```

```

</xs:simpleType>
<xs:simpleType name="WAITING_RANGE_TYPE">
  <xs:restriction base="APEX_INT32"/>
</xs:simpleType>
<xs:simpleType name="PRIORITY_TYPE">
  <xs:restriction base="APEX_INT32"/>
</xs:simpleType>
<xs:simpleType name="ERROR_MESSAGE_SIZE_TYPE">
  <xs:restriction base="APEX_INT32"/>
</xs:simpleType>
<xs:simpleType name="ERROR_MESSAGE_TYPE">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:complexType name="ERROR_STATUS_TYPE">
  <xs:attribute name="ERROR_CODE" type="ERROR_CODE_TYPE"/>
  <xs:attribute name="ERROR_MESSAGE" type="ERROR_MESSAGE_TYPE"/>
  <xs:attribute name="LENGTH" type="ERROR_MESSAGE_SIZE_TYPE"/>
  <xs:attribute name="FAILED_PROCESS_ID" type="PROCESS_ID_TYPE"/>
  <xs:attribute name="FAILED_ADDRESS" type="SYSTEM_ADDRESS_TYPE"/>
</xs:complexType>
<xs:simpleType name="PORT_DIRECTION_TYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="SOURCE"/>
    <xs:enumeration value="DESTINATION"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ERROR_CODE_TYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="DEADLINE_MISSED"/>
    <xs:enumeration value="APPLICATION_ERROR"/>
    <xs:enumeration value="NUMERIC_ERROR"/>
    <xs:enumeration value="ILLEGAL_REQUEST"/>
    <xs:enumeration value="STACK_OVERFLOW"/>
    <xs:enumeration value="MEMORY_VIOLATION"/>
    <xs:enumeration value="HARDWARE_FAULT"/>
    <xs:enumeration value="POWER_FAIL"/>
    <xs:enumeration value="KERNEL"/>
    <xs:enumeration value="CONFIG_ERROR"/>
    <xs:enumeration value="INIT_ERROR"/>
    <xs:enumeration value="PARTITION_OVERFLOW"/>
    <xs:enumeration value="PARTITION_MODE_SET"/>
    <xs:enumeration value="APEX_INTERNAL_ERROR"/>
    <xs:enumeration value="HARD_DEADLINE_MISSED"/>
  </xs:restriction>

```

```

    <xs:enumeration value="HM_INTERNAL_ERROR"/>
    <xs:enumeration value="PORT_INTERNAL_ERROR"/>
    <xs:enumeration value="LOST_TICKS"/>
    <xs:enumeration value="OTHER"/>
    <xs:enumeration value="HM_ERROR"/>
    <xs:enumeration value="HMQ_OVERFLOW"/>
    <xs:enumeration value="DATA_LOSS"/>
    <xs:enumeration value="HM_DEADLINE_MISSED"/>
    <xs:enumeration value="HM_DEFAULT"/>
    <xs:enumeration value="CBIT_ERROR"/>
    <xs:enumeration value="MAX_CODES"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="RETURN_CODE_TYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="NO_ERROR"/>
    <xs:enumeration value="NO_ACTION"/>
    <xs:enumeration value="NOT_AVAILABLE"/>
    <xs:enumeration value="INVALID_PARAM"/>
    <xs:enumeration value="INVALID_CONFIG"/>
    <xs:enumeration value="INVALID_MODE"/>
    <xs:enumeration value="TIMED_OUT"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TC_DIRECTION_TYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="SEND"/>
    <xs:enumeration value="RECEIVE"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="QUEUEING_DISCIPLINE_TYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="FIFO"/>
    <xs:enumeration value="PRIORITY"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="PORT_MODE_TYPE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="SAMPLING"/>
    <xs:enumeration value="QUEUEING"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="HM_DISPATCH_LVL">

```

```

    <xs:restriction base="xs:string">
      <xs:enumeration value="HM_DIRECT"/>
      <xs:enumeration value="HM_NO_LVL"/>
      <xs:enumeration value="HM_PROCESS_LVL"/>
      <xs:enumeration value="HM_PARTITION_LVL"/>
      <xs:enumeration value="HM_MODULE_LVL"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="PARTITION_MEM">
    <xs:attribute name="SizeBytes" type="APEX_INT32"/>
    <xs:attribute name="PhysicalAddress" type="SYSTEM_ADDRESS_TYPE"/>
    <xs:attribute name="StackSize" type="APEX_INT32"/>
    <xs:attribute name="HeapSize" type="APEX_INT32"/>
    <xs:attribute name="MemoryType" type="APEX_CHAR"/>
    <xs:attribute name="Access" type="APEX_CHAR"/>
  </xs:complexType>
  <xs:complexType name="HM_EVENT">
    <xs:attribute name="evtTag" type="APEX_UINT32"/>
    <xs:attribute name="ParNumber" type="APEX_INT32"/>
    <xs:attribute name="level" type="HM_DISPATCH_LVL"/>
    <xs:attribute name="timeStamp" type="SYSTEM_TIME_TYPE"/>
    <xs:attribute name="sysStatus" type="SYSTEM_STATUS_TYPE"/>
    <xs:attribute name="code" type="ERROR_CODE_TYPE"/>
    <xs:attribute name="addInfo" type="APEX_INT32"/>
    <xs:attribute name="addr" type="SYSTEM_ADDRESS_TYPE"/>
    <xs:attribute name="taskId" type="PROCESS_ID_TYPE"/>
    <xs:attribute name="taskName" type="NAME_TYPE"/>
    <xs:attribute name="msgLen" type="APEX_INT32"/>
    <xs:attribute name="msg">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:minLength value="1"/>
          <xs:maxLength value="64"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
  <xs:simpleType name="SYSTEM_STATUS_TYPE">
    <xs:restriction base="xs:string">
      <xs:enumeration value="HM_UNKNOWN_STATUS"/>
      <xs:enumeration value="HM_MODULE_INIT_STATUS"/>
      <xs:enumeration value="HM_MODULE_HM_STATUS"/>
      <xs:enumeration value="HM_SYS_FUNC_STATUS"/>
    </xs:restriction>
  </xs:simpleType>

```

```

        <xs:enumeration value="HM_PARTITION_SWITCH_STATUS"/>
        <xs:enumeration value="HM_PARTITION_INIT_STATUS"/>
        <xs:enumeration value="HM_PARTITION_HM_STATUS"/>
        <xs:enumeration value="HM_SYSCALL_STATUS"/>
        <xs:enumeration value="HM_PROCESS_EXEC_STATUS"/>
        <xs:enumeration value="HM_PROCESS_MGMT_STATUS"/>
        <xs:enumeration value="HM_TOS_EXEC_STATUS"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="HM_ENTRY_CFG_RECORD">
    <xs:attribute name="code" type="ERROR_CODE_TYPE"/>
    <xs:attribute name="sysStatus" type="SYSTEM_STATUS_TYPE"/>
    <xs:attribute name="handlerEntry" type="xs:string"/>
</xs:complexType>
<xs:complexType name="HM_TABLE_CFG_RECORD">
    <xs:sequence>
        <xs:element name="entries" type="HM_ENTRY_CFG_RECORD" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="crSize" type="APEX_INT32"/>
    <xs:attribute name="nEntries" type="APEX_INT32"/>
    <xs:attribute name="maxQueueDepth" type="APEX_INT32"/>
    <xs:attribute name="queueThreshold" type="APEX_INT32"/>
    <xs:attribute name="stackSize" type="APEX_INT32"/>
    <xs:attribute name="maxLogEntries" type="APEX_INT32"/>
    <xs:attribute name="logEntriesThreshold" type="APEX_INT32"/>
    <xs:attribute name="attributeMask" type="APEX_INT32"/>
    <xs:attribute name="trustedPartitionsMask" type="APEX_UINT32"/>
    <xs:attribute name="maxErrorHandlerQueueDepth" type="APEX_INT32"/>
    <xs:attribute name="errorHandlerQueueThreshold" type="APEX_INT32"/>
    <xs:attribute name="hmCallback_Addr" type="SYSTEM_ADDRESS_TYPE"/>
    <xs:attribute name="hmCallback_Object" type="NAME_TYPE"/>
</xs:complexType>
<xs:complexType name="PORT_CFG_RECORD">
    <xs:attribute name="port_name" type="NAME_TYPE"/>
    <xs:attribute name="partition_name" type="NAME_TYPE"/>
    <xs:attribute name="direction" type="PORT_DIRECTION_TYPE"/>
    <xs:attribute name="mode" type="PORT_MODE_TYPE"/>
    <xs:attribute name="maxMsgSize" type="APEX_INT16"/>
    <xs:attribute name="maxMsgNumber" type="APEX_INT16"/>
    <xs:attribute name="msgSegmentLength" type="APEX_INT16"/>
    <xs:attribute name="discipline" type="QUEUEING_DISCIPLINE_TYPE"/>
    <xs:attribute name="refreshRate" type="SYSTEM_TIME_TYPE"/>

```

```

        <xs:attribute name="channel_id" type="DecOrHexValueType"/>
    </xs:complexType>
    <xs:complexType name="PORT_CFG_TABLE_TYPE">
        <xs:sequence>
            <xs:element name="ports" type="PORT_CFG_RECORD" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="sampling_port_num" type="APEX_INT16"/>
        <xs:attribute name="queuing_port_num" type="APEX_INT16"/>
    </xs:complexType>
    <xs:complexType name="PARTITION_CFG_RECORD">
        <xs:sequence>
            <xs:element name="Partition_HM" type="HM_TABLE_CFG_RECORD"/>
            <xs:element name="MemoryCFG" type="PARTITION_MEM"/>
            <xs:element name="ProcessesPTR" type="PROCESS_ATTRIBUTE_TYPE"
maxOccurs="unbounded"/>
            <xs:element name="Ports_table" type="PORT_CFG_TABLE_TYPE"/>
        </xs:sequence>
        <xs:attribute name="Partition_name" type="NAME_TYPE"/>
        <xs:attribute name="Partition_entry" type="NAME_TYPE"/>
        <xs:attribute name="CFGPath">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:minLength value="1"/>
                    <xs:maxLength value="64"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="Period" type="SYSTEM_TIME_TYPE"/>
        <xs:attribute name="Duration" type="SYSTEM_TIME_TYPE"/>
        <xs:attribute name="Criticality" type="APEX_INT32"/>
        <xs:attribute name="System_partition" type="APEX_CHAR"/>
        <xs:attribute name="WorkerTasksnum" type="APEX_UINT32"/>
        <xs:attribute name="Messagequeue_number" type="APEX_UINT32"/>
        <xs:attribute name="Event_number" type="APEX_UINT32"/>
        <xs:attribute name="Semaphore_number" type="APEX_UINT32"/>
        <xs:attribute name="Blackboard_number" type="APEX_UINT32"/>
        <xs:attribute name="Processes_number" type="APEX_UINT32"/>
    </xs:complexType>
    <xs:complexType name="PROCESS_ATTRIBUTE_TYPE">
        <xs:attribute name="NAME" type="PROCESS_NAME_TYPE"/>
        <xs:attribute name="ENTRY_POINT" type="SYSTEM_ADDRESS_TYPE"/>
        <xs:attribute name="STACK_SIZE" type="STACK_SIZE_TYPE"/>

```

```

    <xs:attribute name="BASE_PRIORITY" type="PRIORITY_TYPE"/>
    <xs:attribute name="PERIOD" type="SYSTEM_TIME_TYPE"/>
    <xs:attribute name="TIME_CAPACITY" type="SYSTEM_TIME_TYPE"/>
    <xs:attribute name="DEADLINE" type="DEADLINE_TYPE"/>
    <xs:attribute name="FLOAT_REQUIREMENT" type="APEX_CHAR"/>
</xs:complexType>
<xs:simpleType name="PROCESS_NAME_TYPE">
    <xs:restriction base="NAME_TYPE"/>
</xs:simpleType>
<xs:simpleType name="DEADLINE_TYPE">
    <xs:restriction base="xs:string">
        <xs:enumeration value="SOFT"/>
        <xs:enumeration value="HARD"/>
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="USER_CONFIGURE_TIME_FRAME">
    <xs:attribute name="Partition_name" type="NAME_TYPE"/>
    <xs:attribute name="Widows_start" type="SYSTEM_TIME_TYPE"/>
    <xs:attribute name="Windows_duration" type="SYSTEM_TIME_TYPE"/>
    <xs:attribute name="Period_start_flag" type="APEX_BOOL"/>
</xs:complexType>
<xs:simpleType name="APEX_BOOL">
    <xs:restriction base="xs:unsignedInt"/>
</xs:simpleType>
<xs:complexType name="HM_SYSTEM_ENTRY_CFG_RECORD">
    <xs:attribute name="code" type="ERROR_CODE_TYPE"/>
    <xs:attribute name="StatusLevel" type="SYSTEM_STATUS_LEVEL_TYPE"/>
    <xs:attribute name="level" type="HM_DISPATCH_LVL"/>
</xs:complexType>
<xs:complexType name="HM_SYSTEM_TABLE_CFG_RECORD">
    <xs:sequence>
        <xs:element name="entries" type="HM_SYSTEM_ENTRY_CFG_RECORD"
minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="crSize" type="APEX_INT32"/>
    <xs:attribute name="nEntries" type="APEX_INT32"/>
</xs:complexType>
<xs:simpleType name="CHANNEL_TYPE">
    <xs:restriction base="xs:string">
        <xs:enumeration value="FIFO_channel"/>
        <xs:enumeration value="HISTORY_channel"/>
    </xs:restriction>
</xs:simpleType>

```

```

<xs:complexType name="CHANNEL_CFG_RECORD">
  <xs:sequence>
    <xs:element name="tc_ids" type="DecOrHexValueType" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="channel_id" type="DecOrHexValueType"/>
  <xs:attribute name="which_type" type="CHANNEL_TYPE"/>
  <xs:attribute name="tc_number" type="APEX_INT16"/>
  <xs:attribute name="recv_port_num" type="APEX_INT16"/>
</xs:complexType>
<xs:complexType name="TC_CFG_RECORD">
  <xs:attribute name="tc_id" type="DecOrHexValueType"/>
  <xs:attribute name="interface_id" type="DecOrHexValueType"/>
  <xs:attribute name="config_data" type="APEX_UINT32"/>
  <xs:attribute name="config_data_length" type="APEX_INT32"/>
  <xs:attribute name="channel_num" type="APEX_INT16"/>
  <xs:attribute name="direction" type="TC_DIRECTION_TYPE"/>
</xs:complexType>
<xs:complexType name="INTERFACE_CFG_RECORD">
  <xs:attribute name="interface_id" type="DecOrHexValueType"/>
  <xs:attribute name="network_id" type="DecOrHexValueType"/>
  <xs:attribute name="dev_name" type="NAME_TYPE"/>
  <xs:attribute name="config_data" type="APEX_INT32"/>
  <xs:attribute name="data_length" type="APEX_INT16"/>
</xs:complexType>
<xs:complexType name="CHANNEL_CFG_TABLE_TYPE">
  <xs:sequence>
    <xs:element name="channels" type="CHANNEL_CFG_RECORD"
minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="channel_num" type="APEX_INT16"/>
</xs:complexType>
<xs:complexType name="TC_CFG_TABLE_TYPE">
  <xs:sequence>
    <xs:element name="tcs" type="TC_CFG_RECORD" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="tc_num" type="APEX_INT16"/>
</xs:complexType>
<xs:complexType name="INTERFACE_CFG_TABLE_TYPE">
  <xs:sequence>
    <xs:element name="interfaces" type="INTERFACE_CFG_RECORD"
minOccurs="0" maxOccurs="unbounded"/>

```



```

    </xs:sequence>
    <xs:attribute name="interface_num" type="APEX_INT16"/>
</xs:complexType>
<xs:complexType name="MODULE_SCHED_TYPE">
    <xs:sequence>
        <xs:element name="Window" type="USER_CONFIGURE_TIME_FRAME"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="CONNECTION_TABLE_TYPE">
    <xs:sequence>
        <xs:element name="Channel_table" type="CHANNEL_CFG_TABLE_TYPE"/>
        <xs:element name="TC_table" type="TC_CFG_TABLE_TYPE"/>
        <xs:element name="Interface_table" type="INTERFACE_CFG_TABLE_TYPE"/>
    </xs:sequence>
</xs:complexType>
<xs:simpleType name="SYSTEM_STATUS_LEVEL_TYPE">
    <xs:restriction base="xs:string">
        <xs:enumeration value="HM_MODULE_MODE"/>
        <xs:enumeration value="HM_PARTITION_MODE"/>
        <xs:enumeration value="HM_PROCESS_MODE"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="DecOrHexValueType">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
</xs:schema>

```

中华人民共和国航空行业标准
航空电子应用软件接口应用指南
HB/Z 360—2008

*

中国航空综合技术研究所出版
(北京东外京顺路 7 号)
中国航空综合技术研究所印刷车间印刷
北京 1665 信箱发行
版权专有 不得翻印

*

开本 880×1230 1/16 印张 6¼ 字数 198 千字
2008 年 9 月第一版 2008 年 9 月第一次印刷
印数 1—200

*

书号: 标 301.2421 定价 63.00 元