入式操作系统, 同时还可以提高操作系统的通用性和应用程序的可移植性.

1 相关工作

1.1 嵌入式操作系统

嵌入式系统是以应用为中心, 以计算机技术为基础, 并且软硬件可裁剪, 适用于应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机[4]. 如图 1 所示, 在传统的嵌入式系统组成结构中, 嵌入式系统包含硬件系统与软件系统两部分. 硬件系统包括处理器/微处理器、存储器、I/O 接口、外设器件; 软件系统主要包括嵌入式操作系统、嵌入式应用程序以及应用程序开发接口.

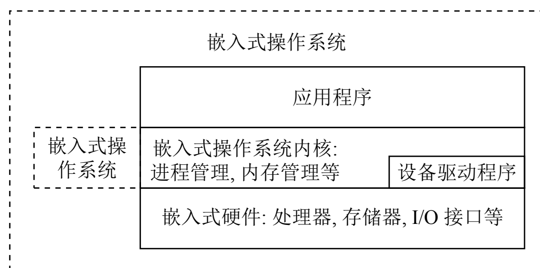

图 1 嵌入式系统结构图

实时嵌入式操作系统 (Real-Time embedded Operating Systems, RTOS) 是一种实时、支持嵌入式系统应用的操作系统软件, 在嵌入式软件系统中距离硬件最近, 与硬件联系最为紧密[5].

本文所选取的嵌入式实时操作系统——“锐华”是中国电子科技集团公司第 32 研究所自主研制的操作系统. 该操作系统的内核采用了“十二五”核高基成果——ReWorks. ReWorks 内核可抢占、中断可嵌套, 并且具备高效的中断管理机制和任务调度、上下文切换算法, 具有强实时性. 其采用微内核的体系架构和面向对象的设计方法, 具有良好的可裁剪性. ReWorks 的接口设计符合 POSIX 标准[6], 具有较强的可移植性. 此外, ReWorks 作为一款自主可控的操作系统, 还具有可持续性特点[7].

1.2 WindowsNT 硬件抽象层[8]

Windows NT 是一个被广泛应用的抢占式多任务操作系统. 为了提高其稳定性与兼容性, 微软公司首次提出了硬件抽象层 (Hardware Abstract Layer, HAL) 的概念, 将大部分与硬件相关的操作放在内核和硬件抽象层中, 由内核与硬件抽象层负责完成与硬件操作有关的细节.

硬件抽象层以动态链接库的形式 (HAL.DLL) 提供面向平台的函数. 这些函数将 Windows NT 操作系统与其所依赖的基本硬件进行了分离, 不需要对设备驱动程序做任何修改就可以支持同类处理机不同平台中的相同设备[9].

由于硬件抽象层的优化是针对 PC 的, 所以无法适用于嵌入式应用开发环境.

1.3 Linux 实时硬件抽象层[10]

实时硬件抽象层 (Real Time Hardware Abstract Layer, RTHAL) 是 Linux 操作系统和底层硬件之间的一个中间层. RTHAL 对硬件完全控制, 为 Linux 屏蔽了硬件细节并禁止其对硬件直接操作, 中断管理、任务管理、时钟管理等相关操作均由实时硬件抽象层完成. RTHAL 为硬件提供接口, Linux 和硬实时操作系统运行在其之上, 有效的避免了操作系统对 Linux 内核进行大量的修改, 减少对标准 Linux 内核可能带来的负面影响[11].

由于实时硬件抽象层是面向 Linux 系统的, 其结构、功能以及定义的接口都与 Linux 系统紧密相关, 所以无法为其他嵌入式系统所用.

1.4 操作系统抽象层[12]

文献[12]利用 C++的虚函数机制, 提出了操作系统抽象层 (Operating System Abstraction Layer, OSAL) 的概念. OSAL 封装具体操作系统实体为应用提供所需的各种服务, 使开发的应用与操作系统完全无关, 从而使应用的兼容性更强.

由于文献[12]使用的是面向对象的语言 C++, 执行过程中会引入大量临时对象, 导致性能方面损失较大. 同时实践证明, 仅对操作系统进行抽象是不够的, 还需考虑硬件平台的特殊性, 所以该方案仍有改进空间.

2 PAL 设计与实现

2.1 体系结构设计

PAL 由操作系统抽象、硬件平台抽象和设备驱动抽象 3 个部分构成, 如图 2 所示, 对操作系统和硬件平台进行了封装. 这 3 个部分可以形象的分为两层: 上层为应用程序提供基于 POSIX 标准的标准化接口, 这些接口不随宿主平台的改变而改变, 使得开发的嵌入式应用不再拘泥于某一平台; 下层是这些接口在操作系统与硬件平台上的具体实现.

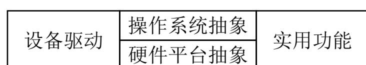

图 2 PAL 内部结构

加入 PAL 后, 软件体系结构在操作系统与应用程序之间多出一个独立接口层, 该接口层起到了衔接不同操作系统、硬件平台与应用程序的作用, 如图 3.

2.2 函数接口设计

本小节对平台抽象层中提供操作系统基础服务的接口进行了简单的描述, 具体如下:

任务接口: 任务接口用于提供任务的注册、创建、删除、查询以及延迟服务.

信号量接口: 信号量接口与经典操作系统中的设计思路一致, 主要提供二进制信号量、计数信号量、互斥信号量的创建、删除、释放、锁操作、等待以及查询服务.

消息队列接口: 消息队列接口用于提供队列的创建、删除、消息获取、消息输入、查询服务.

定时器接口: 定时器接口用于提供定时器的初始化、创建、设置、查询、删除服务.

文件系统接口: 文件系统接口除了用于提供文件和目录的创建、删除、打开、关闭服务以外, 还为基于 PAL 开发的应用程序提供了从文件中读取数据、向文件中写入数据和在文件中查询数据的服务.

中断/异常处理接口: 与传统的中断机制一致, 该模块接口用于提供中断和异常的使能、关闭、关联以及锁操作服务.

除了上述接口, PAL 还包括用于应用程序初始化、打印输出、等待的空闲任务接口等.

平台抽象层中部分设计的接口的具体实现如表 1 所示, 即 PAL 上层提供给应用程序的标准化接口.

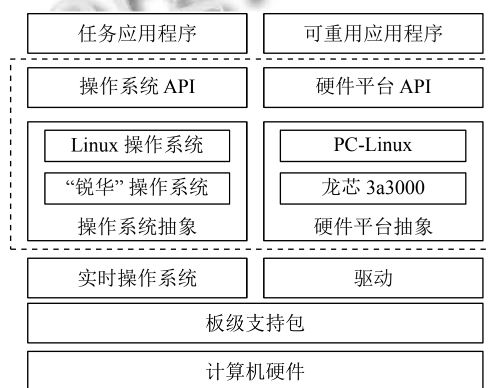

图 3 包含 PAL 的体系结构

表 1 PAL 部分接口及实现

接口	实现
任务接口	OS_TaskCreate、OS_TaskDelete、OS_TaskDelay、OS_TaskRegister、OS_TaskGetIdByName···
信号量接口	OS_BinSemCreate、OS_BinSemDelete、OS_MutSemGive、OS_MutSemTake、OS_CountSemTimedWait、OS_CountSemGetIdByName···
队列接口	OS_QueueCreate、OS_QueueDelete、OS_QueueGetInfo···
定时器接口	OS_TimerCreate、OS_TimerSet、OS_TimerDelete、OS_TimerGetIdByName···
文件系统接口	OS_creat、OS_open、OS_close、OS_read、OS_write、OS_remove、OS_rename、OS_mkdir、OS_opendir、OS_closedir···
中断/异常处理接口	OS_IntAttachHandler、OS_IntEnable、OS_IntDisable、OS_ExcAttachHandler、OS_ExcEnable、OS_ExcDisable···

2.3 宏设计

在 PAL 的设计中, 以宏的方式实现平台抽象层对不同操作系统和硬件平台的支持, 来满足一些平台的特殊需求. 例如, 对于基于 glibc 的 Linux 系统, 定义宏"_XOPEN_SOURCE=600"来启用 X/Open 6 标准. 因为 glibc 支持多个标准[13], 所以需要添加额外的宏定义以确定编译时采用的标准; 对于不同的操作系统, 定义宏"OS ="进行系统选择, 这种方法的好处在于随着需要支持的操作系统增多, 可以通过直接修改宏的方式增加选项, 避免重复的工作. 类似的, 也可以通过定义宏"BSP ="来选择不同硬件平台.

通过使用宏并采用条件编译的方法, 可以实现基于程序的透明性.

2.4 映射关系

由于自定义标识符可能会与 C 库提供的其他标识符重叠, 或与操作系统中的定义重叠, 所以在设计时为自定义标识符添加前缀, 避免命名空间带来的潜在冲突. 如对于"锐华"操作系统, 添加一个"RCS_"前缀, 将 C 库提供的所有标识符映射到一个带有"RCS_"前缀的命名空间, 以避免命名时的相互影响. 类似的, 将枚举型定义、类型定义、结构体定义和函数定义都映射到带有"RCS_"前缀的命名空间, 具体举例见表 2.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

表 2 PAL 中关于“锐华”系统的映射关系

	映射前	映射后
C 库	O_RDWR O_CREAT NULLDEV	RCS_O_RDWR RCS_O_CREAT RCS_NULLDEV
枚举型	ERROR OK SEM_FULL	RCS_ERROR RCS_OK RCS_SEM_FULL
类型	BOOL UINT UINT16	RCS_BOOL RCS_UINT RCS_UINT16
结构体	semaphore symbol stat	RCS_semaphore RCS_symbol RCS_stat
函数	write open errno	RCS_write RCS_open RCS_errno

2.5 数据类型

不同宿主平台上 C 语言对应的数据类型存在差异, PAL 并不直接使用 C 语言所定义的数据类型, 而是重新定义一套与平台无关的数据类型, 具体如表 3.

表 3 数据结构

名称	数据类型
palbool	unsigned char
int8	signed char
int16	short int
int32	long int
int64	long long int
uint8	unsigned char
uint16	unsigned short int
uint32	unsigned long int
uint64	unsigned long long int

2.6 实现

本小节选取“锐华”操作系统、龙芯 3a 硬件平台, 以任务相关接口为例, 对 PAL 的实现进行详细阐述.

在实现方式上, 受到平台独立[14]的启发, PAL 对各操作系统共有的系统特性和数据进行统一管理. 各平台的异构性则主要体现在接口的实现上.

平台共有的定义, 主要是指宏、数据结构等. 平台定义 1 定义了支持浮点运算的宏, 并给出了任务相关接口所需的结构体定义以及函数接口定义.

```
/*平台定义 1*/
#define OS_FP_ENABLED 1
typedef struct
  {
  Char
  name[OS_MAX_API_NAME];
  unit32 creator;
  unit32 stack_size;//任务栈大小
  priority;
  unit32 OStask_id;//任务 ID
  }OS_task_prop_t;
int32 OS_TaskCreate (uint32 *task_id,
        const char*task_name,
        osal_task_entry function_pointer,
        uint32*stack_pointer,
        uint32 stack_size,
        uint32 priority, uint32 flags);
int32 OS_TaskDelete (uint32 task_id);
void OS_TaskExit (void);
int32 OS_TaskDelay (uint32
millisecond);
```

标准化接口 OS_TaskCreate() 用于创建任务, 其参数定义了任务 ID、任务名、任务堆栈大小、优先级等. 算法 1 为“锐华”操作系统任务创建接口的实现代码, 具体实现的不同保证了平台的异构性.

```
/*算法 1*/
int32 OS_TaskCreate (uint32 *task_id,
          const char *task_name,
          osal_task_entry function_pointer,
          uint32 *stack_pointer,
          uint32 stack_size,
          uint32 prisority,
          uint32 flags)
{
  uint32 possible_taskid;
  uint32 i;
  int32 LocalFlags;
  int tmp_task_id = ERROR;
  /*检查空指针、优先级*/
  /*检查参数*/
  /*检查 id 是否溢出*/
  /*创建 ReWorks 任务*/
  if (flags == OS_FP_ENABLED)
  {
    LocalFlags = VX_FP_TASK;
  }
  else
  {
```


© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
        LocalFlags = 0;
    }
    /*ReWorks 任务创建函数*/
    tmp_task_id = taskSpawn(
    (char*)task_name, priority,
    LocalFlags, stack_size,
    (FUNCPTR) function_pointer,
    0,0,0,0,0,0,0,0,0,0);
    /* 创建任务是否成功 */
    if(tmp_task_id == ERROR)
    {
        OS_task_table[possible_taskid].free = TRUE
        return OS_ERROR;
    }
    /*设置任务的 ID、名称、堆栈大小和优先级*/
    *task_id = possible_taskid;
    OS_task_table[*task_id].id = tmp_task_id;
    strcpy(OS_task_table[*task_id].name, task_name);
    OS_task_table[*task_id].free = FALSE;
    OS_task_table[*task_id].creator = OS_FindCreator();
    OS_task_table[*task_id].stack_size = stack_size;
    OS_task_table[*task_id].priority = priority;
    return OS_SUCCESS;
  } /* end OS_TaskCreate */
```

3 PAL 测试与分析

本节首先对 PAL 提供的各类接口进行功能测试. 通过编写测试例程, 调用抽象完成的各类接口进行正确环境和错误环境测试; 然后对加入 PAL 之后的操作系统进行兼容性测试. 使用平台抽象层提供的 API 进行嵌入式开发, 在不修改代码的前提下, 该应用程序应当能够同时运行在 Linux 以及“锐华”操作系统上; 3.3 小节对加入 PAL 前后的实时性能进行比较分析. 因为任务调度延迟时间是评价操作系统实时性的重要指标之一[15], 所以选择对其进行测量; 最后, 3.4 小节对 PAL 的优势进行了分析与总结.

3.1 功能测试

以任务管理接口为例, 在测试例程初始化完成后, 分别进行正常情况和异常情况测试. 其中正常情况包括创建任务、根据 ID 查询任务信息、删除任务操作, 而异常测试则是指创建超过个数限制的任务, 创建两个同名任务, 删除未创建任务、删除已删除的任务操作. 测试结果表明, 设计的 PAL 能够完成正常情况下的需求, 并对异常情况做出符合预期的反馈.

3.2 兼容性测试

兼容性测试采用 PAL 提供的标准化接口编写测试例程, 分别在 Linux 系统与“锐华”系统上进行测试, 测试结果如图 4 所示. 这段测试程序在两个系统上均运行成功并输出正确, 这表明 PAL 的设计提高了应用程序的代码重用性.

3.3 实时性测试

在实时任务测试中, 获取测试任务执行前的时刻 t_1, 将实时任务休眠时间设置为 T; 获取实时任务执行完成时的时刻记为 t_2, t_2-t_1-T 就是所要测量的任务调度延迟时间.

使用上述方法对任务调度延迟时间进行测量, 共测试 1000 次, 对数据进行初步处理后, 得出结果如图 5.

对实验数据进行分析得出, 添加 PAL 前任务调度延迟的平均值为 0.875 ms, 添加 PAL 后任务调度延迟的平均值为 1.094 ms. 通过数据与图表对比可以看出, 添加 PAL 后的任务调度延迟时间要比添加 PAL 前略微增加. 这是由于 PAL 的引入致使系统多出了一个 PAL 接口与宿主平台接口间的函数调用, 但添加 PAL 后的时间指标仍可以保持在与原有指标相同的量级.

分析以上实验, 引入 PAL 的嵌入式操作系统在扩展了应用程序的代码重用性后仍具有可靠的实时性.

3.4 PAL 优势分析

PAL 是位于嵌入式操作系统与应用程序之间的独立接口层, 是一种衔接不同操作系统、硬件平台与应用程序的桥梁.

首先, 在平台抽象层上进行开发可以避免嵌入式应用与操作系统和底层硬件的直接交互, 有利于提高应用程序的健壮性. 其次, 平台抽象层可以根据具体应用所需的系统调用来构造相应的接口, 因此平台抽象层具有良好的可裁剪性和伸缩性. 最后, 平台抽象层按照系统的相关性对应用程序代码进行分类, 当应用适配新的宿主平台时, 只需修改少量甚至无需修改代码即可在新平台上运行应用程序, 大大提高了开发的效率. 总的来说, PAL 有利于实现软硬件协同设计, 促进可移植和可重用的实时嵌入式系统应用程序的创建.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
[BEGIN] PC-LINUX UNIT TEST
Making directories: ram0, ram1, eeprom1 for OSAL mount points

[BEGIN] 01 TimerTest
[ PASS] 01.001 timer-test.c:73 - Timer Test Task Created RC=0
[ PASS] 01.002 timer-test.c:103 - Timer 0 Created RC=0 ID=0
[ INFO] Timer 0 Accuracy = 0 microseconds

[ PASS] 01.003 timer-test.c:108 - Timer 0 programmed RC=0
[ PASS] 01.004 timer-test.c:103 - Timer 1 Created RC=0 ID=1
[ INFO] Timer 1 Accuracy = 0 microseconds

[ PASS] 01.005 timer-test.c:108 - Timer 1 programmed RC=0
[ PASS] 01.006 timer-test.c:103 - Timer 2 Created RC=0 ID=2
[ INFO] Timer 2 Accuracy = 0 microseconds

[ PASS] 01.007 timer-test.c:108 - Timer 2 programmed RC=0
[ PASS] 01.008 timer-test.c:103 - Timer 3 Created RC=0 ID=3
[ INFO] Timer 3 Accuracy = 0 microseconds

[ PASS] 01.009 timer-test.c:108 - Timer 3 programmed RC=0
[ INFO] Starting Delay loop.
```

(a) Linux 系统测试输出

```
COMPLETE: 1 tests Segment(s) executed

[BEGIN] 01 TimerTest
[ PASS] 01.001 timer-test.c:73 - Timer Test Task Created RC=0
[ PASS] 01.002 timer-test.c:104 - Timer 0 Created RC=0 ID=0
[ INFO] Timer 0 Accuracy = 1000 microseconds

[ PASS] 01.003 timer-test.c:109 - Timer 0 programmed RC=0
[ PASS] 01.004 timer-test.c:104 - Timer 1 Created RC=0 ID=1
[ INFO] Timer 1 Accuracy = 1000 microseconds

[ PASS] 01.005 timer-test.c:109 - Timer 1 programmed RC=0
[ PASS] 01.006 timer-test.c:104 - Timer 2 Created RC=0 ID=2
[ INFO] Timer 2 Accuracy = 1000 microseconds

[ PASS] 01.007 timer-test.c:109 - Timer 2 programmed RC=0
[ PASS] 01.008 timer-test.c:104 - Timer 3 Created RC=0 ID=3
[ INFO] Timer 3 Accuracy = 1000 microseconds

[ PASS] 01.009 timer-test.c:109 - Timer 3 programmed RC=0
[ INFO] Starting Delay loop.
```

(b) “锐华” 系统测试输出

图 4　兼容性测试结果

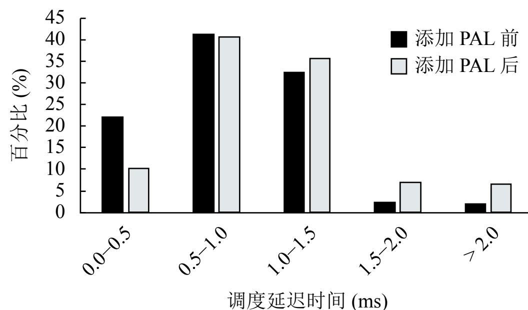

图 5　实时性测试结果

4　结束语

本文为嵌入式实时操作系统设计了一个平台抽象层, 着重介绍平台抽象层的设计方案, 并针对 Linux 系统与“锐华”操作系统给出实现方案, 最后对平台抽象层的功能, 兼容性和实时性给出测试方案. 实验证明, 引入本文设计的平台抽象层不仅能提高应用程序的可移植性, 同时操作系统还保留有可靠的实时性.

平台抽象层对不同操作系统与硬件平台的应用程序开发接口进行抽象, 给用户开发各种嵌入式应用提供了标准化接口, 极大减少移植带来的冗余工作, 并可高效重用已开发的软件组件, 有效实现了资源复用.

参考文献

1 王龙飞. 嵌入式系统的应用现状及发展趋势. 中国新通信, 2018, 20(23): 95–96. [doi: 10.3969/j.issn.1673-4866.2018.23.073]

2 胡德鹏. 嵌入式系统技术发展趋势探析. 数字技术与应用, 2018, 36(10): 233–234. [doi: 10.19695/j.cnki.cn12-1369.2018.10.121]

3 翟阳阳. 浅论新时期计算机软件开发技术的应用及发展趋势. 计算机产品与流通, 2019, (5): 12.

4 夏小翔. 嵌入式系统的特点及应用. 鄂州大学学报, 2015, 22(1): 110–112. [doi: 10.3969/j.issn.1008-9004.2015.01.040]

5 王福刚, 杨文君, 葛良全. 嵌入式系统的发展与展望. 计算机测量与控制, 2014, 22(12): 3843–3847, 3863. [doi: 10.3969/j.issn.1671-4598.2014.12.001]

6 ISO/IEC. 14515-1: 2000 /Amd.1-2003 ISO/IEC Standard for information technology – Portable Operating System Interface (POSIX(R)) – Test methods for measuring conformance to POSIX – Part 1: System interfaces – Amendment 1: Realtime extension (C Language). IEEE. [doi: 10.1109/IEEESTD.2003.7840066]

7 李健. 高可靠嵌入式实时操作系统的研究与实现[硕士学位论文]. 上海: 上海交通大学, 2008.

8 蒋句平. Windows NT HAL 的结构与移植. 计算机工程与设计, 1997, 18(1): 16–21.

9 郭静寰, 孟祥迪, 郭丽虹, 等. Windows NT 硬件抽象层 HAL 功能分析. 计算机应用, 2002, 22(7): 86–88.

10 Zhang GY, Chen LY, Yao AH. Study and comparison of the RTHAL-based and ADEOS-based RTAI real-time solutions for Linux. Proceedings of the 1st International Multi-Symposiums on Computer and Computational Sciences. Hangzhou, China. 2006. 771–775. [doi: 10.1109/IMSCCS.2006.272]

11 苏曙光, 刘云生. 基于 RTHAL 的 Linux 实时性研究和实现. 计算机科学, 2009, 36(7): 270–272. [doi: 10.3969/j.issn.1002-137X.2009.07.066]

12 尚海忠, 朱培彦, 王霞, 等. 操作系统抽象层——一种支持跨平台的新技术. 计算机工程, 2002, 28(2): 109–111. [doi: 10.3969/j.issn.1000-3428.2002.02.044]

13 雷鸿. 基于虚拟机架构下嵌入式开发环境搭建的研究与实现. 信息通信, 2011, (4): 11–13. [doi: 10.3969/j.issn.1673-1131.2011.04.006]

14 高佳羽. 模型驱动体系结构的研究及其应用[硕士学位论文]. 杭州: 浙江大学, 2007.

15 余兵, 黎忠文. Linux 操作系统实时性分析. 计算机技术与发展, 2007, 17(9): 41–44, 47. [doi: 10.3969/j.issn.1673-629X.2007.09.013]

© 中国科学院软件研究所 http://www.c-s-a.org.cn