

ZOS 平台技术简介

December 10, 2009

Juphcon

菊风系统软件有限公司

版权所有

目录

1. ZOS 介绍.....	4
2. ZOS 有什么不同.....	5
3. ZOS 的技术优势.....	6
4. ZOS 的服务组件.....	7
5. ZOS 配套组件介绍.....	9
6. ZOS 组件性能指标.....	10
6.1 环矩阵计时器的性能分析	10
6.2 基本内存数据处理性能分析	11
7. ZOS 接口举例.....	13
7.1 汇聚缓冲区接口	13
7.2 BITMAP 内存池接口	16
7.3 动态 HASH 接口.....	18
7.4 双链表接口	19
7.5 任务接口	28
7.6 字符串接口	32

表格列表

表 2-1 ZOS 与操作系统、知名企业平台的对比说明	6
表 4-1 ZOS 系统服务组件说明	9
表 5-1 ZOS 配套组件说明	10
表 6-1 Timer 性能分析	11
表 6-2 RTimer 测试案例	11
表 6-3 总体性能对比	12

图片列表

图 1-1 ZOS 软件架构图	4
图 1-2 产品开发平台比较图	4

1. ZOS 介绍

ZOS(Zero Operating System)是菊风公司的操作系统服务平台，提供了支持多种操作系统环境下的统一抽象接口操作，使得软件产品能够独立于特定的处理机、编译器和操作系统等应用环境。此外，ZOS 增强了系统服务功能，提供任务管理、消息队列、计时器管理、内存管理、数据缓冲区管理、日志管理。

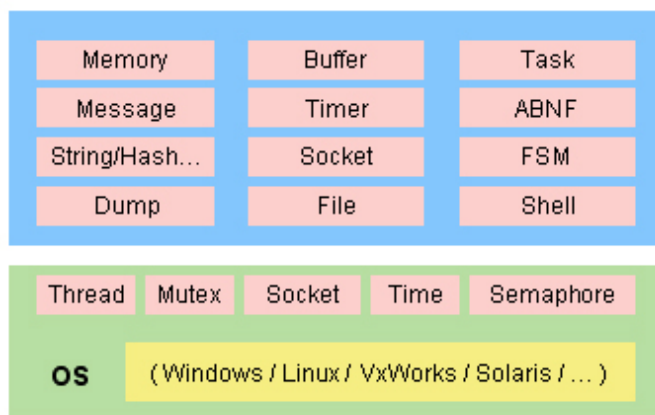


图 1-1 ZOS 软件架构图

对于需要支持多种通信产品的公司来说，采用 ZOS 的好处无疑是提供平台的通用性和开发效率，多种产品中可以大量复用成熟稳定的功能接口，有助于提高产品的研发速度，加快产品上市时间。

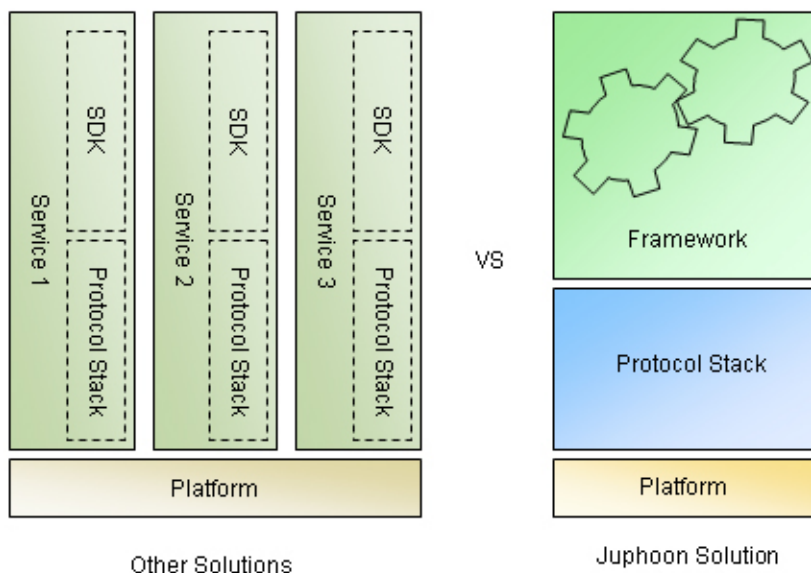


图 1-2 产品开发平台比较图

2. ZOS 有什么不同

ZOS 运行在操作系统之上，编译和运行环境跟操作系统相关，线程驱动、信号量、互斥、Socket 等功能依赖操作系统，但是大部分功能独立于操作系统实现，甚至有很多功能实现跟操作系统所提供的 SDK 有很大不同，菊风结合通信软件的业务特点，并且结合多年通信软件开发经验，对很多资源管理技术、基本数据结构做了丰富的细化处理，体现了对通信应用的专研性。

在做业务开发时，尤其在通信、嵌入式软件开发中，产品类型众多、软硬件平台选择也比较丰富，终端、服务器开发对资源容量和性能要求差距比较大，而不同操作系统之间的编译环境、系统接口有非常大的差异性，软件的系统通用性比较难。

同时，随着通信产品的研发进度随着市场的竞争激化，必然要求业务开发模式不能再走一个产品选择一种开发平台，僵化发展，尤其软件功能在产品中的比重越来越大，并且将决定其核心竞争力。当然像 iPhone 这种平台属于极端情况，苹果只支持一种开发平台，但是作为非系统提供商，往往需要考虑多平台的支持，比如客户端需要同时支持 iPhone, Android 手机终端，除非公司策略就是支持某一种平台或不同平台独立发展，否则产品竞争力无法跟采用统一的系统服务平台来支持业务开发来的有竞争力。

因此，全球一些知名的通信企业有其系统服务平台，如 CISCO 的 IOS, 华为的 DOPRA 平台等。同时，一些系统软件、协议栈软件提供商也会提供支持多平台的系统服务平台。但是把平台上升到公司的所有产品线，全球几乎没有。这也说明了，系统服务平台的重要性，但也说明了实现系统平台的复杂性。

ZOS 对比对象	相同性	不同性
操作系统 (如 Windows, Linux)	Thread, Time, Mutex, Semaphore, Socket 等功能跟操作系统的保持一致; 部分操作接口(比如字符串、打印函数等)跟 POSIX 接口参数保持一致。	ZOS 有 8 中缓冲区管理技术; 5 种内存池管理技术; 任务操作简单; 数据结构实现丰富, 参数化能力较强; 计时器策略丰富多样; 很多组件设计时是考虑到不同资源大小、数目、处理时间、性能要求的苛刻环境而独创设计的。 ZOS 没有图形开发界面等操作系统所提供的功能, 更多的是一个通信软件开发环境。
知名企业的系统服务平台 (如 CISCO 的 IOS)	在 Thread, Time, Mutex, Semaphore, Socket 方面有类型的概念, 往往也有较丰富的数据结构, 资源管理技术。	ZOS 有 8 中缓冲区管理技术; 5 种内存池管理技术; 资源管理的方式非常灵活。 同时, ZOS 是独立的第三方供应商, 业务功能通用性较强, 不同公司、不同产品适应弹性

		<p>较大。支持在 Windows 上开发，而部署在实际的操作系统上。</p> <p>知名公司的平台往往跟内部产品相关的功能有一定的相关性。基本不外卖，而且差异大的产品对平台往往需要很大的修改和移植工作量。</p>
--	--	---

表 2-1 ZOS 与操作系统、知名企业平台的对比说明

3. ZOS 的技术优势

ZOS 平台的技术优势主要体现在以下方面：

- 跨平台性

ZOS 平台可以快速移植到多种操作系统中，目前已经支持 Windows XP/Mobile, Linux(Redhat, uCLinux, ArmLinux 等), ThreadX, VxWorks 等操作系统。整个平台采用 C99 实现，操作系统依赖基本上涉及 Time, Thread, Mutex, Semaphore, Socket 等少数功能，操作系统移植工作量较小。
- 服务功能的完整性

产品的软件开发核心之一就是资源的使用和管理，掌握资源的生命周期的程度也就意味着产品的稳定性和高性能。ZOS 提供了丰富的功能来有效的管理各种不同需求和上下文环境的资源，并且提供资源的跟踪机制来完善和分析资源的使用情况。ZOS 平台提供任务管理、内存管理、消息队列、缓冲区管理等各种丰富的功能。
- 同一机制多样策略

ZOS 平台的主要创新性在于为某一类技术机制提供多种实现策略，使得业务实现者可以根据业务特点和使用场景灵活选择具体的实现策略，这对于挖掘系统性能有很大的帮助。比如不同业务对象对内存的需求会不一样，比如协议栈处理的内存的粒度会可以大一点，为了提高性能，尽量减少平台互斥锁的操作。又比如在有些数据结构或对象的 Hash 范围是固定，而有些数据结构或对象的 Hash 会是动态变化的。业务的复杂性导致对系统平台而言是很大的挑战，ZOS 对于通信产品开发而言，已经提供了相当大的弹性。
- 平台成熟稳定、适用于终端和服务器的开发

菊风公司对 ZOS 平台的研发时间持续 6 年，有近 50 种功能组件，稳定而大量的使用在公司的协议栈 SDK、终端 SDK、IMS 服务器等产品中。ZOS 平台被一些知名公司采用，获得良好的技术认可。

4. ZOS 的服务组件

ZOS 平台包括以下组件：

组件类型	组件描述
内存管理	<ul style="list-style-type: none"> • 动态 Bucket 组内存管理 • Brick 固定粒度内存管理 • Bucket 内存池管理 • Bitmap 内存池管理 • 二次幂内存池管理 <p>此 5 中内存管理策略，对象化使用。不同内存管理可以适用不同的应用场景，如下：</p> <p>Bucket 组内存管理可以对内存块按照不同大小动态创建，也可以动态调整数目。内存释放速度非常快。</p> <p>Brick 固定粒度内存管理可以对某种数据结构或对象进行统一的抽象管理，提高业务实现的复用率，资源查找和释放速度非常快。</p> <p>Bucket 内存池管理提供对象化管理，各个组件可以创建自主的内存池。比如 ZOS 内存池要求内存块的粒度比较细小，而且种类比较多。而对于业务组件来说，其所需内存粒度较大，而且种类有限。</p> <p>Bitmap 内存池管理可以动态的申请和释放内存块资源，对于对象中经常需要动态设置属性的，Bitmap 内存池是非常理想的选择。属性资源申请释放间几乎没有锁操作，大大提高性能。</p> <p>二次幂内存池可以通过 1 个逻辑操作就可以获得内存中数据所属的内存块的头地址。</p>
缓冲区管理	<ul style="list-style-type: none"> • 汇聚缓冲区管理 (ABUF) • 基本缓冲区管理 (BBUF) • 聚集缓冲区管理 (CBUF) • 数据缓冲区管理 (DBUF) • 编码缓冲区管理 (EBUF) • 管道缓冲区管理 (PBUF) • 结构缓冲区管理 (SBUF) • 交换缓冲区管理 (XBUF) <p>汇聚和聚集缓冲区管理非常适合对象的内部资源管理，是理想的对象内存控制点，使用中可以灵活派生孩子缓冲区，销毁对象是只需调用一个删除接口即可释放所有相关资源，使用非常高效和方便。</p>

	<p>基本缓冲区非常适合临时性的对象和通信数据资源管理，内存获取几乎没有锁，申请速度非常快，释放非常简便。</p> <p>数据缓冲区适合各种网络传输数据的管理，协议报文数据的管理，支持引用计数、克隆、多种拷贝方式。大量使用菊风的协议栈实现中。</p> <p>编码缓冲区是非常高效的协议栈中的数据结构的编码过程，存放报文单元性能非常高。</p> <p>管道缓冲区是一种先进后出的资源使用机制的实现策略。菊风的 XML 解析器采用此技术可以在只使用 512 字节的情况下完成任意文件大小的 XML 解析处理。对于需要大量 XML 解析的服务器来说无疑是大大的提高内存资源的消耗。</p> <p>结构缓冲区适合大型数据结构的资源管理，比如大文件的 XML 会产品几 M 甚至几十 M 的 XML 数据结构。</p> <p>交互缓冲区适合业务组件之间交换通信内容，从整体业务是实现来说，采用 XBUF 可以提高整体架构中消息处理的一致性。</p>
任务管理	支持预先规划线程管理和动态线程管理两种方式；任务可以创建所属的消息队列、计时器组；提供线程安全退出机制。
消息队列	优先级消息队列实现，消息队列长度可以动态调整，也可以固定范围。任务级消息实时性高，计时器相关消息永不丢失
计时器管理	<ul style="list-style-type: none"> • 队列实时计时器管理 • 环矩阵大容量计时器管理 <p>各个任务可以独立保持计时器，减少计时器资源干扰，提高性能。而且任务计时器保证无丢失、超时实时处理。</p> <p>环矩阵计时器非常适合大容量的计时器需求（比如几万、几十万），而且具有很高实时处理的平衡能力。</p>
日志管理	支持日志内存独立线程运行，日志级别管理，独立日志对象操作，日志打印重定向，日志实时开发控制等功能。
链表	支持单链表、双链表。可以通过宏自定义类型相关的单、双链表，此方式对调试时能方便查看链表中节点中的数据类型。有丰富的链表操作宏，比如遍历、判断是否为空等。
队列	支持普通队列和优先级队列。队列实现支持阻塞信号等待设计，优先级队列采用堆排序和数组索引等多种方式混合实现，内部设计灵活，性能良好。
哈希	支持普通 Hash 和 动态 Hash，动态 Hash 采用快速查找多目录 (FSMH) 的技术实现。用户可以自动 Hash Key 和比较函数，普通 Hash 的 Bucket 数目可以比较灵活。

状态机	统一的状态机模块，提供业务组件状态机实现的复用率。不同于一般的矩阵状态机、判断状态机实现，ZOS 状态机采用二级表，事件分散定义，统一驱动状态机的方式实现，业务状态机定义简单，易于理解。
对象 ID 隐射管理	根据对象 ID 建立隐射表。提供固定范围、循环定义、时间唯一、用户指定等多种方式来产生对象的 ID，ID 的隐射技术采用数组、链表、Hash、动态 Hash 等多种方式实现，用户可以根据不同对象类型、对象的数目范围、对象运行中的变化选择时效比最佳的方式来管理。
DUMP 跟踪内存资源	支持缓冲区、内存池中内存块、状态机处理过程进行过程跟踪，对调试和实时了解资源使用情况和对象生命周期非常有帮助。
字符串操作接口	丰富的字符串处理接口，安全的字符串比较、拷贝，前、后、中间去除空白符或线性空白符等几十个操作接口
信号量、互斥	支持操作系统所支持的信号量和互斥（关键区）接口
Socket、INET	支持 IPv4, v6 各种地址转换，完整的 POSIX 抽象 Socket 接口定义
系统参数配置	可以设置 ZOS 的各种组件参数，比如内存池的各种大小、数目，任务默认优先级、堆栈大小等。
其他工具接口	字符判断、字节对齐等各种小工具非常齐备。

表 4-1 ZOS 系统服务组件说明

5. ZOS 配套组件介绍

菊风的通信软件平台是采用组件化测试，ZOS 提供基本的通用的功能，其他功能根据应用独立设计和实现，整个开发平台有以下组件：

组件类型	组件描述
ZFILE	提供文件、目录、路径的跨平台的抽象接口。 有文件 Open, Close, Write, Read, Seek、Status 等接口； 有目录创建、读取、删除等接口； 有文件路径获取和设置等接口
ZSH	Shell 控制台，类似 Windows Command、Linux 的 Bash，但是功能没有他们丰富。可以添加业务相关的命令，查看 ZOS 等组件的统计数据，控制系统运行状态。
UTIL	提供 MD5, SHA, CRC, Digitmap, 随机数，字符串匹配，INI 文件格式、单元测试等多种工具接口
XML	支持 SAX, DOM 解析接口、XPath、菊风定义操作接口等功能，SAX 平均解析速度是 LibXML 的 2 倍以上；而且只需 512 字节内存可解析任意大小的 XML 文件。
DB	XML 内存数据库，支持 XML 定义数据库格式、XML 数据存储；数据库中支持嵌套表，支持 Hash, Tree 索引。

UTAL	客户端通用网络传输组件
DTE	大容量网络传输组件，不同的传输业务可以独立线程，非常适合服务器开发
ABNF	支持 ABNF 协议报文的解析和编码，处理速度快，协议栈编解码实现简单易懂，非常适合于新手和时间压力紧迫的项目
ASN.1	支持 BER, PER 协议报文的解析和编码，问题是成熟度不高。
ZSMS/ZSMC	远程接入 ZOS 系统平台，远程操作产品中的命令，本地显示执行结果

表 5-1 ZOS 配套组件说明

6. ZOS 组件性能指标

因为各种平台众多，而且没有行业标杆，此数据仅供参考，实际性能数据需要结合具体的平台的进行对比。

以下统计数据是跟某一国外系统平台进行对比，由于涉及第三方商业软件，在此无法指明。

以下统计数据均是在 Thinkpad T30 P4 2.0G, 512M 内存，Windows XP 操作系统、VC 开发环境测试所获得的数据，不同硬件配置、不同的操作系统，统计数据会有差异性，在此说明。

6.1 环矩阵计时器的性能分析

RTimer 的设计主要是为了满足在系统需要大量的不规则的计时器的时候的需求，RTimer 综合平衡了时间复杂度和空间复杂度。

计时器数量为 m ，

最大计时器数量为 M ，

数组类方式的查找只是计算元素下标，其时间为 T_1 ，

链表类方式要逐个查找，访问每个节点时间为 O_1 ，

计时器结构大小 S_1 ，

RTimer 中，Ring Node 数量为 q ，

RTimer 中，LNode 结构大小为 S_2 ，

RTimer 中，Ring 结构大小 S_3 ，

资源	情况	数组类 Timer	RTimer	链表类 Timer
时间复杂度	最坏情况	$T_1 + O_1$	$T_1 + m * O_1$	$m * O_1$
	平均	$T_1 + O_1$	$T_1 + m / 2q * O_1$	$m / 2 * O_1$
	最好情况	$T_1 + O_1$	$T_1 + O_1$	O_1
空间复杂度	最坏情况	$m * S_1 + M * S_2$	$m * (S_1 + S_2) + S_3$	$m * S_1$
	平均	$m * S_1 + M * S_2$	$m * S_1 + m / 2 * S_2 + S_3$	$m * S_1$
	最好情况	$m * S_1 + M * S_2$	$m * S_1 + S_2 + S_3$	$m * S_1$

表 6-1 Timer 性能分析

由于极端情况在实际使用中极少出现，所以一般考察平均情况。从时间复杂度和空间复杂度上 RTimer 的效率都介于数组类Timer和链表类Timer中间；但为了保证Timer的实时性和时间效率，我们一般设定比较大的q值，比如128，这样便能在不浪费大量空间的情况下保证较高的性能，如果用户有更高的性能需求，即可设定 $m=2q$ ，即可保证最快的执行性能。同数组类方式相比，Ring Timer的最大计时长度不受最大计时器数量的限制，具有很强的伸缩性。

测试案例：

测试计时器系统在实际运行环境中的可用性，模拟3000个呼叫同时进行，总共申请接近两万个计时器。测试的平台是Centrino - M 1.5GHz，操作系统Windows XP professional。具体数量分配如下：

Timer Length(ms)	Quantity
500	10000
1000	10000
2000	5000
4000	5000
8000	5000
16000	2000
32000	2000
64000	1000

表 6-2 RTimer 测试案例

计时器超时处理为记录各种超时时间的次数，并再一次启动计时器。这样计时器部分的系统占用会比实际情况中高一些，因为实际应用中不会出现所有启动的计时器都超时这种情况。平均每秒钟超时的计时器数量为34000个左右。

CPU占有率DEBUG版本小于1%，RELEASE版本为也小于1%。数据结构占用内存约为630K。现有设计中，内存为初始化时静态分配。为提高空间利用率也可以采用动态分配。

测试表明，采用RTimer这种设计，在满足了苛刻的Timer需求（大量并发的、无规则的、空间有限的）的同时，也保证了Timer的执行性能。

6.2 基本内存数据处理性能分析

测试指标	基于 ZOS	Xxx 测试数据	说明
内存式的数据缓冲区 ⁽¹⁾	50ms	1282ms	
消息式的数据缓冲区	61ms	1822ms	
字符串解析性能 ⁽²⁾	270ms	18146ms	
数字解析性能 ⁽³⁾	821ms	7040ms	

SIP Header 解析性能 ⁽⁴⁾	370ms	7291ms	
SIP Method 解析性能 ⁽⁵⁾	411ms	8612ms	
唯一串产生的性能 ⁽⁶⁾	60ms	>460ms	Juphoon 使用 HrStamp 产生唯一串, Xxx 直接提供产生 BranchId 的函数。

表 6-3 总体性能对比

测试说明

(1) 内存性能

针对系统Buffer的各种大小的内存申请释放, 重复3万次所用时间

(2) 字符串性能

针对符合ABNF语法的各种字符串解析, 重复3万次所用时间

(3) 数字解析性能

针对符合ABNF语法的各种数字字符串解析, 重复3万次所用时间

(4) SIP Header 解析性能

对于各类典型SIP Header 的解析, 重复3千次所用时间

说明: 因为测试的Header有近90个, 如果重复3万次, 则测试时间会过长, 因此选定为3千次。

(5) SIP Method 解析性能

对于各类典型SIP Method 的解析, 重复3万次所用时间

(6) 唯一串产生的性能

Juphoon使用High Resolution Stamp产生唯一串, 用在类似于Call-Id, Tag, Branch-Id等需要时空唯一的字符串, 有些厂商直接提供产生类似于Branch-Id的函数, 重复3万次所用时间。

7. ZOS 接口举例

7.1 汇聚缓冲区接口

```
/* zos create aggregation buffer */
ZABUFID Zos_AbufCreate();

/* zos create aggregation buffer and alloc data */
ZABUFID Zos_AbufCreated(ZSIZE_T zSize, ZVOID **ppData);

/* zos create aggregation buffer and alloc data(clear data to 0) */
ZABUFID Zos_AbufCreateClrd(ZSIZE_T zSize, ZVOID **ppData);

/* zos create aggregation buffer in abuf */
ZABUFID Zos_AbufCreateX(ZABUFID zBufId);

/* zos create aggregation buffer and alloc data */
ZABUFID Zos_AbufCreateXD(ZABUFID zBufId, ZSIZE_T zSize, ZVOID **ppData);

/* zos create aggregation buffer and alloc data(clear data to 0) */
ZABUFID Zos_AbufCreateXClrd(ZABUFID zBufId, ZSIZE_T zSize,
                            ZVOID **ppData);

/* zos delete aggregation buffer */
ZVOID Zos_AbufDelete(ZABUFID zBufId);

/* zos clean up aggregation buffer */
ZVOID Zos_AbufClean(ZABUFID zBufId);

/* zos detach abuf from parent abuf */
ZINT Zos_AbufDetach(ZABUFID zBufId);

/* zos attach abuf into parent abuf */
ZINT Zos_AbufAttach(ZABUFID zDstBufId, ZABUFID zSrcBufId);

/* zos alloc memory block from an abuf */
ZVOID * Zos_AbufAlloc(ZABUFID zBufId, ZSIZE_T zSize);
```

```
/* zos alloc memory from abuf and clear data to 0 */
ZVOID * Zos_AbufAllocClrd(ZABUFID zBufId, ZSIZE_T zSize);

/* zos free a memory block in an abuf */
ZVOID Zos_AbufFree(ZABUFID zBufId, ZVOID *pMem);

/* zos size of all memory in abuf */
ZULONG Zos_AbufSize(ZABUFID zBufId);

/* zos check hold data in the abuf */
ZBOOL Zos_AbufHoldD(ZABUFID zBufId, ZVOID *pData);

/* zos deposit a data buffer(structure type) into abuf */
ZINT Zos_AbufDepositDbuf(ZABUFID zBufId, ST_ZOS_DBUF **ppstBuf);

/* zos deposit a data buffer into abuf */
ZINT Zos_AbufDepositDbufX(ZABUFID zBufId, ZUCHAR ucType,
                          ZULONG dwDftBlkSize, ST_ZOS_DBUF **ppstBuf);

/* zos resolve a data buffer from abuf */
ZINT Zos_AbufResolveDbuf(ZABUFID zBufId);

/* zos abuf copy string */
ZINT Zos_AbufCpyStr(ZABUFID zBufId, ZCHAR *pcSrcStr, ZCHAR **ppcDstStr);

/* zos abuf copy string */
ZINT Zos_AbufCpyNStr(ZABUFID zBufId, ZCHAR *pcSrcStr, ZUSHORT wSrcLen,
                    ZCHAR **ppcDstStr);

/* zos abuf copy data buffer structure string */
ZINT Zos_AbufCpyXStr(ZABUFID zBufId, ST_ZOS_SSTR *pstSrcStr,
                    ZCHAR **ppcDstStr);

/* zos abuf copy data buffer unsigned structure string */
ZINT Zos_AbufCpyUXStr(ZABUFID zBufId, ST_ZOS_USTR *pstSrcStr,
                    ZCHAR **ppcDstStr);
```

```
/* zos abuf copy data buffer string */
ZINT Zos_AbufCpyDStr(ZABUFID zBufId, ST_ZOS_DBUF *pstData,
                    ZCHAR **ppcDstStr);

/* zos abuf copy signed structure string */
ZINT Zos_AbufCpySStr(ZABUFID zBufId, ZCHAR *pcSrcStr ,
                    ST_ZOS_SSTR *pstDstStr);

/* zos abuf copy signed structure string */
ZINT Zos_AbufCpyNSStr(ZABUFID zBufId, ZCHAR *pcSrcStr, ZUSHORT wSrcLen,
                    ST_ZOS_SSTR *pstDstStr);

/* zos abuf copy signed structure string */
ZINT Zos_AbufCpyXSStr(ZABUFID zBufId, ST_ZOS_SSTR *pstSrcStr,
                    ST_ZOS_SSTR *pstDstStr);

/* zos abuf copy signed structure string */
ZINT Zos_AbufCpyUXSStr(ZABUFID zBufId, ST_ZOS_USTR *pstSrcStr,
                    ST_ZOS_SSTR *pstDstStr);

/* zos abuf copy data buffer string into signed structure string */
ZINT Zos_AbufCpyDSStr(ZABUFID zBufId, ST_ZOS_DBUF *pstData,
                    ST_ZOS_SSTR *pstDstStr);

/* zos abuf copy unsigned structure string */
ZINT Zos_AbufCpyUSStr(ZABUFID zBufId, ZUCHAR *pucSrcStr,
                    ST_ZOS_USTR *pstDstStr);

/* zos abuf copy unsigned structure string */
ZINT Zos_AbufCpyNUSStr(ZABUFID zBufId, ZUCHAR *pucSrcStr,
                    ZUSHORT wSrcLen, ST_ZOS_USTR *pstDstStr);

/* zos abuf copy unsigned structure string */
ZINT Zos_AbufCpyXUSStr(ZABUFID zBufId, ST_ZOS_SSTR *pstSrcStr,
                    ST_ZOS_USTR *pstDstStr);

/* zos abuf copy unsigned structure string */
```

```

ZINT Zos_AbufCpyUXUStr(ZABUFID zBufId, ST_ZOS_USTR *pstSrcStr,
                      ST_ZOS_USTR *pstDstStr);

/* zos abuf copy data buffer string into unsigned structure string */
ZINT Zos_AbufCpyDUSStr(ZABUFID zBufId, ST_ZOS_DBUF *pstData,
                      ST_ZOS_USTR *pstDstStr);

/* zos abuf copy formatted string */
ZINT Zos_AbufCpyFStr(ZABUFID zBufId, ZCHAR **ppcDstStr,
                    const ZCHAR *pcFormat, ...);

/* zos abuf copy object */
ZINT Zos_AbufCpyObject(ZABUFID zBufId, ZVOID *pSrcObject, ZINT iSize,
                      ZVOID **ppDstObject);

```

7.2 Bitmap 内存池接口

```

/* zos bitmap memory type, size is maximum size of alloc memory
   small size is less than 65535 * iBitByteSize
   large size is more than or equal to 65535 * iBitByteSize*/
typedef enum EN_ZOS_BPOOL_TYPE
{
    EN_ZOS_BPOOL_SMALL_SIZE,          /* small size */
    EN_ZOS_BPOOL_SMALL_SIZE_QFREE,   /* small size with quick free memory */
    EN_ZOS_BPOOL_LARGE_SIZE,         /* large size */
    EN_ZOS_BPOOL_LARGE_SIZE_QFREE,   /* large size with quick free memory */
    EN_ZOS_BPOOL_UNKNOWN,            /* unknown type */
} EN_ZOS_BPOOL_TYPE;

/* zos bitmap memory pool id */
typedef ZVOID * ZBPOOLID;

/* zos create bitmap pool */
ZBPOOLID Zos_BpoolCreate(ZUCHAR ucType, ZUINT iBktSize, ZUINT iBitByte);

/* zos delete bitmap pool */
ZVOID Zos_BpoolDelete(ZBPOOLID zPoolId);

```

```
/* zos clean up bitmap pool */
ZVOID Zos_BpoolClean(ZBPOOLID zPoolId);

/* zos bitmap pool alloc memory */
ZVOID * Zos_BpoolAlloc(ZBPOOLID zPoolId, ZUINT iSize);

/* zos bitmap pool alloc memory with specific bucket size */
ZVOID * Zos_BpoolAllocX(ZBPOOLID zPoolId, ZUINT iBktSize,
                        ZUINT iMemSize);

/* zos bitmap pool alloc memory and zero memory */
ZVOID * Zos_BpoolAllocClr(ZBPOOLID zPoolId, ZUINT iSize);

/* zos bitmap pool alloc memory and zero memory */
ZVOID * Zos_BpoolAllocXClr(ZBPOOLID zPoolId, ZUINT iBktSize,
                           ZUINT iMemSize);

/* zos bitmap pool free memory block */
ZVOID Zos_BpoolFree(ZBPOOLID zPoolId, ZVOID *pMem);

/* zos bitmap pool get size */
ZINT Zos_BpoolGetSize(ZBPOOLID zPoolId, ZUINT *piSize);

/* zos bitmap pool get size by the memory address */
ZINT Zos_BpoolGetMemSize(ZBPOOLID zPoolId, ZVOID *pMem,
                        ZUINT *piSize);

/* zos bitmap pool check valid adress of memory block */
ZBOOL Zos_BpoolIsValid(ZBPOOLID zPoolId, ZVOID *pMem);

/* zos bitmap pool check data memory address is in the pool,
the memory address maybe in the middle of valid memory */
ZBOOL Zos_BpoolHoldD(ZBPOOLID zPoolId, ZVOID *pData);
```

7.3 动态 Hash 接口

```
/* zos dynamic hash key and compare funciton */
typedef ZINT (*PFN_ZDHASHKEY)(ZULONG dwParm1, ZULONG dwParm2, \
    ZULONG dwParm3, ZULONG *pdwHashKey);

typedef ZINT (*PFN_ZDHASHCMP)(ZULONG dwEntry, ZULONG dwParm1, \
    ZULONG dwParm2, ZULONG dwParm3);

/* zos dynamic hash id */
typedef ZVOID * ZDHASHID;

/* zos create a dynamic hash */
ZINT Zos_DhashCreate(PFN_ZDHASHKEY pfnKey, PFN_ZDHASHCMP pfnCmp,
    ZDHASHID *pzHashId);

/* zos create a dynamic hash in an abuf */
ZINT Zos_DhashCreateX(ZABUFID zMemBuf, PFN_ZDHASHKEY pfnKey,
    PFN_ZDHASHCMP pfnCmp, ZDHASHID *pzHashId);

/* zos delete a dynamic hash */
ZINT Zos_DhashDelete(ZDHASHID zHashId);

/* zos insert an entry into a dynamic hash */
ZINT Zos_DhashInsert(ZDHASHID zHashId, ZULONG dwEntry, ZULONG dwParm1,
    ZULONG dwParm2, ZULONG dwParm3);

/* zos remove an entry in a dynamic hash */
ZINT Zos_DhashRemove(ZDHASHID zHashId, ZULONG dwParm1,
    ZULONG dwParm2, ZULONG dwParm3);

/* zos remove an entry in a dynamic hash by entry with same hash key */
ZINT Zos_DhashRemoveX(ZDHASHID zHashId, ZULONG dwParm1,
    ZULONG dwParm2, ZULONG dwParm3, ZULONG dwEntry);

/* zos find an entry from a dynamic hash */
ZBOOL Zos_DhashFind(ZDHASHID zHashId, ZULONG dwParm1, ZULONG dwParm2,
    ZULONG dwParm3, ZULONG *pdwEntry);
```

```
/* zos find an entry from hash by index with same hash key and content */
ZBOOL Zos_DhashFindX(ZDHASHID zHashId, ZULONG dwParm1, ZULONG dwParm2,
                    ZULONG dwParm3, ZUINT iIndex, ZULONG *pdwEntry);

/* zos find an entry whit hash key from a dynamic hash */
ZBOOL Zos_DhashFindByKey(ZDHASHID zHashId, ZULONG dwHashKey,
                        ZULONG dwParm1, ZULONG dwParm2, ZULONG dwParm3,
                        ZULONG *pdwEntry);

/* zos find an entry whit hash key from hash by index with same content */
ZBOOL Zos_DhashFindByKeyX(ZDHASHID zHashId, ZULONG dwHashKey,
                        ZULONG dwParm1, ZULONG dwParm2, ZULONG dwParm3,
                        ZUINT iIndex, ZULONG *pdwEntry);

/* zos find entry size from hash with same hash key and content */
ZINT Zos_DhashFindSize(ZDHASHID zHashId, ZULONG dwParm1, ZULONG dwParm2,
                    ZULONG dwParm3, ZUINT *piSize);
```

7.4 双链表接口

```
/* zos dlist maximum infinite size */
#define ZDLIST_INFINITE_SIZE ZMAXULONG

#define ZOS_DLIST_SIZE(_list)\
    ((_list)->dwCount)

#define ZOS_DLIST_ISFULL(_list) \
    ((_list)->dwMaxNum && (_list)->dwCount >= (_list)->dwMaxNum)

#define ZOS_DLIST_ISEMPY(_list) \
    ((_list)->pstHead == ZNULL)
```

```
#define ZOS_DLIST_ISNOEMPTY(_list) \  
    ((_list)->pstHead != ZNULL)  
  
#define ZOS_DLIST_HEAD_NODE(_list) (_list)->pstHead  
#define ZOS_DLIST_TAIL_NODE(_list) (_list)->pstTail  
  
#define ZOS_DLIST_PREV_NODE(_node) ((_node) ? (_node)->pstPrev : ZNULL)  
#define ZOS_DLIST_NEXT_NODE(_node) ((_node) ? (_node)->pstNext : ZNULL)  
  
#define ZOS_DLIST_GET_DATA(_node) ((_node) ? (_node)->pData : ZNULL)  
#define ZOS_DLIST_SET_DATA(_node) (_node)->pData  
  
#define ZOS_DLIST_HEAD_DATA(_list) \  
    ((ZOS_DLIST_HEAD_NODE(_list)) ? ZOS_DLIST_HEAD_NODE(_list)->pData : ZNULL)  
  
#define ZOS_DLIST_TAIL_DATA(_list) \  
    ((ZOS_DLIST_TAIL_NODE(_list)) ? ZOS_DLIST_TAIL_NODE(_list)->pData : ZNULL)  
  
#define ZOS_DLIST_NEXT_DATA(_node) \  
    ((ZOS_DLIST_NEXT_NODE(_node)) ? ZOS_DLIST_NEXT_NODE(_node)->pData : ZNULL)  
  
#define ZOS_DLIST_PREV_DATA(_node) \  
    ((ZOS_DLIST_PREV_NODE(_node)) ? ZOS_DLIST_PREV_NODE(_node)->pData : ZNULL)  
  
#define FOR_ALL_NODE_IN_DLIST(_list, _node) \  
    for (_node = ZOS_DLIST_HEAD_NODE(_list); \  
        _node != ZNULL; \  
        _node = ZOS_DLIST_NEXT_NODE(_node))  
  
#define FOR_ALL_NODE_IN_DLISTX(_list, _node) \  
    for (_node = ZOS_DLIST_TAIL_NODE(_list); \  
        _node != ZNULL; \  
        _node = ZOS_DLIST_PREV_NODE(_node))  
  
#define FOR_ALL_NODE_IN_DLIST_NEXT(_list, _node, _nextnode) \  
    for (_node = ZOS_DLIST_HEAD_NODE(_list), \  
        _nextnode = ZOS_DLIST_NEXT_NODE(_node); \  
        _nextnode != ZNULL; \  
        _node = _nextnode, _nextnode = ZOS_DLIST_NEXT_NODE(_node))
```

```
_node != ZNULL; \  
_node = _nextnode, \  
_nextnode = ZOS_DLIST_NEXT_NODE(_node))  
  
#define FOR_ALL_NODE_IN_DLIST_PREV(_list, _node, _prevnode) \  
for (_node = ZOS_DLIST_TAIL_NODE(_list), \  
_prevnode = ZOS_DLIST_PREV_NODE(_node); \  
_node != ZNULL; \  
_node = _prevnode, \  
_prevnode = ZOS_DLIST_PREV_NODE(_node))  
  
#define FOR_ALL_DATA_IN_DLIST(_list, _node, _data) \  
for (_node = ZOS_DLIST_HEAD_NODE(_list), \  
_data = ZOS_DLIST_GET_DATA(_node); \  
_node != ZNULL; \  
_node = ZOS_DLIST_NEXT_NODE(_node), \  
_data = ZOS_DLIST_GET_DATA(_node))  
  
#define FOR_ALL_DATA_IN_DLISTX(_list, _node, _data) \  
for (_node = ZOS_DLIST_TAIL_NODE(_list), \  
_data = ZOS_DLIST_GET_DATA(_node); \  
_node != ZNULL; \  
_node = ZOS_DLIST_PREV_NODE(_node), \  
_data = ZOS_DLIST_GET_DATA(_node))  
  
#define FOR_ALL_DATA_IN_DLIST_NEXT(_list, _node, _nextnode, _data) \  
for (_node = ZOS_DLIST_HEAD_NODE(_list), \  
_data = ZOS_DLIST_GET_DATA(_node), \  
_nextnode = ZOS_DLIST_NEXT_NODE(_node); \  
_node != ZNULL; \  
_node = _nextnode, \  
_nextnode = ZOS_DLIST_NEXT_NODE(_node), \  
_data = ZOS_DLIST_GET_DATA(_node))  
  
#define FOR_ALL_DATA_IN_DLIST_PREV(_list, _node, _prevnode, _data) \  
for (_node = ZOS_DLIST_TAIL_NODE(_list), \  
_data = ZOS_DLIST_GET_DATA(_node), \  

```

```
_prevnode = ZOS_DLIST_PREV_NODE(_node); \  
_node != ZNULL; \  
_node = _prevnode, \  
_prevnode = ZOS_DLIST_PREV_NODE(_node), \  
_data = ZOS_DLIST_GET_DATA(_node))  
  
/* zos dlist create */  
#define ZOS_DLIST_CREATE(_list, _size) \  
    Zos_DlistCreate(_list, _size)  
  
/* zos dlist create */  
#define ZOS_DLIST_CREATEEX(_list) \  
    Zos_DlistCreate(_list, ZDLIST_INFINITE_SIZE)  
  
/* zos dlist delete */  
#define ZOS_DLIST_DELETE(_list) \  
    Zos_DlistDelete(_list)  
  
/* zos dlist node init */  
#define ZOS_DLIST_NODE_INIT(_node, _data) do { \  
    (_node)->pstPrev = ZNULL; \  
    (_node)->pstNext = ZNULL; \  
    (_node)->pData = (ZCHAR *)(_data); \  
} while (0)  
  
/* zos insert node before head in dlist, seemed as a function */  
#define Zos_DlistAdd2Head(_list, _node) \  
    Zos_DlistInsert(_list, ZNULL, _node)  
  
/* zos insert node after tail in dlist, seemed as a function */  
#define Zos_DlistAdd2Tail(_list, _node) \  
    Zos_DlistInsert(_list, (_list)->pstTail, _node)  
  
/* zos insert node before head in dlist */  
#define ZOS_DLIST_ADD2HEAD(_list, _node) \  
    Zos_DlistInsert(_list, ZNULL, _node)
```

```
/* zos insert node after tail in dlist */
#define ZOS_DLIST_ADD2TAIL(_list, _node) \
    Zos_DlistInsert(_list, (_list)->pstTail, _node)

/* zos insert node after the after previou node in dlist */
#define ZOS_DLIST_INSERT(_list, _prevnode, _node) \
    Zos_DlistInsert(_list, _prevnode, _node)

/*lint -save -e* */

/* zos dequeue node from the first node in dlist */
#define ZOS_DLIST_DEQUEUE(_list, _node) \
    _node = Zos_DlistDequeue(_list)

/* zos remove one node */
#define ZOS_DLIST_REMOVE(_list, _node) \
    Zos_DlistRemove(_list, _node)

/* zos find node by index */
#define ZOS_DLIST_FIND_BY_INDEX(_list, _index, _node) \
    _node = Zos_DlistFindByIndex(_list, _index)

/*lint -restore */

/* zos typedef dlist with specific name */
#define ZOS_TYPEDEF_DLIST(_name) \
    /* double list node */ \
    typedef struct tag##_name##_LST_NODE \
    { \
        struct tag##_name##_LST_NODE *pstNext; /* next dlist node */ \
        struct tag##_name##_LST_NODE *pstPrev; /* previous dlist node */ \
        ST_##_name *pData; /* dlist node data */ \
    } ST_##_name##_LST_NODE; \
    /* double list */ \
    typedef struct tag##_name##_LST \
    { \
        ZULONG dwMaxNum; /* maximum number of dlist nodes */ \

```

```
ZULONG dwCount;          /* actual count of dlist nodes */ \
ST_##_name##_LST_NODE *pstHead; /* dlist node head */ \
ST_##_name##_LST_NODE *pstTail; /* dlist node tail */ \
} ST_##_name##_LST

/* zos typedef dlist macros */
#define ZOS_TYPEDEF_DLIST_SIZE(_list) \
    ZOS_DLIST_SIZE((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_ISFULL(_list) \
    ZOS_DLIST_ISFULL((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_ISEMPY(_list) \
    ZOS_DLIST_ISEMPY((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_ISNOEMPTY(_list) \
    ZOS_DLIST_ISNOEMPTY((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_HEAD_NODE(_list) \
    ZOS_DLIST_HEAD_NODE((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_TAIL_NODE(_list) \
    ZOS_DLIST_TAIL_NODE((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_PREV_NODE(_node) \
    ZOS_DLIST_PREV_NODE(_node)

#define ZOS_TYPEDEF_DLIST_NEXT_NODE(_node) \
    ZOS_DLIST_NEXT_NODE(_node)

#define ZOS_TYPEDEF_DLIST_GET_DATA(_node) \
    ZOS_DLIST_GET_DATA(_node)

#define ZOS_TYPEDEF_DLIST_SET_DATA(_node) \
    ZOS_DLIST_SET_DATA(_node)

#define ZOS_TYPEDEF_DLIST_HEAD_DATA(_list) \
```

```
ZOS_DLIST_HEAD_DATA((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_TAIL_DATA(_list) \
    ZOS_DLIST_TAIL_DATA((ST_ZOS_DLIST *)(_list))

#define ZOS_TYPEDEF_DLIST_PREV_DATA(_node) \
    ZOS_DLIST_PREV_DATA((ST_ZOS_DLIST_NODE *)(_node))

#define ZOS_TYPEDEF_DLIST_NEXT_DATA(_node) \
    ZOS_DLIST_NEXT_DATA((ST_ZOS_DLIST_NODE *)(_node))

#define FOR_ALL_NODE_IN_TYPEDEF_DLIST(_list, _node) \
    FOR_ALL_NODE_IN_DLIST((ST_ZOS_DLIST *)(_list), _node)

#define FOR_ALL_NODE_IN_TYPEDEF_DLISTX(_list, _node) \
    FOR_ALL_NODE_IN_DLISTX((ST_ZOS_DLIST *)(_list), _node)

#define FOR_ALL_NODE_IN_TYPEDEF_DLIST_NEXT(_list, _node, _nextnode) \
    FOR_ALL_NODE_IN_DLIST_NEXT((ST_ZOS_DLIST *)(_list), _node, _nextnode)

#define FOR_ALL_NODE_IN_TYPEDEF_DLIST_PREV(_list, _node, _prevnode) \
    FOR_ALL_NODE_IN_DLIST_PREV((ST_ZOS_DLIST *)(_list), _node, _prevnode)

#define FOR_ALL_DATA_IN_TYPEDEF_DLIST(_list, _node, _data) \
    FOR_ALL_DATA_IN_DLIST((ST_ZOS_DLIST *)(_list), _node, _data)

#define FOR_ALL_DATA_IN_TYPEDEF_DLISTX(_list, _node, _data) \
    FOR_ALL_DATA_IN_DLISTX((ST_ZOS_DLIST *)(_list), _node, _data)

#define FOR_ALL_DATA_IN_TYPEDEF_DLIST_NEXT(_list, _node, _nextnode, _data) \
    FOR_ALL_DATA_IN_DLIST_NEXT((ST_ZOS_DLIST *)(_list), _node, _nextnode, _data)

#define FOR_ALL_DATA_IN_TYPEDEF_DLIST_PREV(_list, _node, _prevnode, _data) \
    FOR_ALL_DATA_IN_DLIST_PREV((ST_ZOS_DLIST *)(_list), _node, _prevnode, _data)

/* zos typedef dlist create */
#define ZOS_TYPEDEF_DLIST_CREATE(_list, _size) \
```

```
ZOS_DLIST_CREATE((ST_ZOS_DLIST *)(_list), _size)

/* zos typedef dlist create */
#define ZOS_TYPEDEF_DLIST_CREATEX(_list) \
    ZOS_DLIST_CREATE((ST_ZOS_DLIST *)(_list), ZDLIST_INFINITE_SIZE)

/* zos typedef dlist delete */
#define ZOS_TYPEDEF_DLIST_DELETE(_list) \
    ZOS_DLIST_DELETE((ST_ZOS_DLIST *)(_list))

/* zos typedef node init */
#define ZOS_TYPEDEF_DLIST_NODE_INIT(_node, _data) \
    ZOS_DLIST_NODE_INIT((ST_ZOS_DLIST_NODE *)_node, _data)

/* zos insert node before head in typedef dlist */
#define ZOS_TYPEDEF_DLIST_ADD2HEAD(_list, _node) \
    ZOS_DLIST_ADD2HEAD((ST_ZOS_DLIST *)(_list), _node)

/* zos insert node after tail in typedef dlist */
#define ZOS_TYPEDEF_DLIST_ADD2TAIL(_list, _node) \
    ZOS_DLIST_ADD2TAIL((ST_ZOS_DLIST *)(_list), _node)

/* zos insert node after the after previou node in typedef dlist */
#define ZOS_TYPEDEF_DLIST_INSERT(_list, _prevnode, _node) \
    ZOS_DLIST_INSERT((ST_ZOS_DLIST *)(_list), _prevnode, _node)

/* zos dequeue node from the first node in typedef dlist */
#define ZOS_TYPEDEF_DLIST_DEQUEUE(_list, _node) \
    ZOS_DLIST_DEQUEUE((ST_ZOS_DLIST *)(_list), _node)

/* zos remove one node */
#define ZOS_TYPEDEF_DLIST_REMOVE(_list, _node) \
    ZOS_DLIST_REMOVE((ST_ZOS_DLIST *)(_list), _node)

/* zos typedef dlist find node data on index location */
#define ZOS_TYPEDEF_DLIST_FIND_BY_INDEX(_list, _index, _node) \
    ZOS_DLIST_FIND_BY_INDEX((ST_ZOS_DLIST *)_list, _index, _node)
```

```
/* zos dlist node */
typedef struct tagZOS_DLIST_NODE
{
    struct tagZOS_DLIST_NODE *pstNext; /* next dlist node */
    struct tagZOS_DLIST_NODE *pstPrev; /* previous dlist node */
    ZVOID *pData; /* dlist node data */
} ST_ZOS_DLIST_NODE;

/* zos dlist node without data */
typedef struct tagZOS_DLIST_NODEX
{
    struct tagZOS_DLIST_NODE *pstNext; /* next dlist node */
    struct tagZOS_DLIST_NODE *pstPrev; /* previous dlist node */
} ST_ZOS_DLIST_NODEX;

/* zos dlist */
typedef struct tagZOS_DLIST
{
    ZULONG dwMaxNum; /* maximum number of dlist nodes */
    ZULONG dwCount; /* actual count of dlist nodes */
    ST_ZOS_DLIST_NODE *pstHead; /* dlist node head */
    ST_ZOS_DLIST_NODE *pstTail; /* dlist node tail */
} ST_ZOS_DLIST;

/* zos structure string dlist node */
typedef struct tagZOS_SSTR_DLST_NODE
{
    struct tagZOS_SSTR_DLST_NODE *pstNext; /* next dlist node */
    struct tagZOS_SSTR_DLST_NODE *pstPrev; /* previous dlist node */
    ST_ZOS_SSTR *pstData; /* dlist node data */
} ST_ZOS_SSTR_DLST_NODE;

/* zos structure string dlist */
typedef struct tagZOS_SSTR_DLST
{
    ZULONG dwMaxNum; /* maximum number of dlist nodes */
```

```

    ZULONG dwCount;                /* actual count of dlist nodes */
    ST_ZOS_SSTR_DLST_NODE *pstHead; /* dlist node head */
    ST_ZOS_SSTR_DLST_NODE *pstTail; /* dlist node tail */
} ST_ZOS_SSTR_DLST;

/* zos dlist create */
ZINT Zos_DlistCreate(ST_ZOS_DLST *pstList, ZULONG dwMaxNum);

/* zos dlist delete */
ZVOID Zos_DlistDelete(ST_ZOS_DLST *pstList);

/*
 * zos insert one dlist node to the after the previous node of this dlist
 * if pstPrevNode is ZNULL, then node will insert before the head as
 * the new head
 */
ZINT Zos_DlistInsert(ST_ZOS_DLST *pstList, ZVOID *pPrevNode, ZVOID *pNode);

/* zos dequeue one node from dlist */
ZVOID * Zos_DlistDequeue(ST_ZOS_DLST *pstList);

/* zos remove one dlist node from this dlist */
ZINT Zos_DlistRemove(ST_ZOS_DLST *pstList, ZVOID *pNode);

/* zos find node by the index number */
ZVOID * Zos_DlistFindByIndex(ST_ZOS_DLST *pstList, ZULONG dwIndex);

```

7.5 任务接口

```

#if ZOS_SUPT_PLATFORM2(WIN32, WINCE)

#define ZTASK_PRIORITY_MAX      (2)
#define ZTASK_PRIORITY_NORMAL  (0)
#define ZTASK_PRIORITY_MIN     (-2)
#define ZTASK_PRIORITY_INCREMENT (+1)

```

```
typedef DWORD ZTHREAD_ID;

#elif ZPLATFORM == ZPLATFORM_VXWORKS

#define ZTASK_PRIORITY_MAX      (2)
#define ZTASK_PRIORITY_NORMAL   (126)
#define ZTASK_PRIORITY_MIN      (254)
#define ZTASK_PRIORITY_INCREMENT (-1)

typedef ZINT ZTHREAD_ID;

#elif ZOS_SUPT_PLATFORM2(SOLARIS, LINUX)

#define ZTASK_PRIORITY_MAX      (0)
#define ZTASK_PRIORITY_NORMAL   (10)
#define ZTASK_PRIORITY_MIN      (20)
#define ZTASK_PRIORITY_INCREMENT (-1)

typedef pthread_t ZTHREAD_ID;

#elif ZPLATFORM == ZPLATFORM_ARENA

#define ZTASK_PRIORITY_MAX      (16)
#define ZTASK_PRIORITY_NORMAL   (18)
#define ZTASK_PRIORITY_MIN      (20)
#define ZTASK_PRIORITY_INCREMENT (-1)

typedef pthread_t ZTHREAD_ID;

#elif ZPLATFORM == ZPLATFORM_THREADX

#define ZTASK_PRIORITY_MAX      (0)
#define ZTASK_PRIORITY_NORMAL   (16)
#define ZTASK_PRIORITY_MIN      (31)
#define ZTASK_PRIORITY_INCREMENT (-1)

typedef TX_THREAD * ZTHREAD_ID;
```

```
#elif ZPLATFORM == ZPLATFORM_SYMBIAN

#define ZTASK_PRIORITY_MAX      (2)
#define ZTASK_PRIORITY_NORMAL  (0)
#define ZTASK_PRIORITY_MIN     (-2)
#define ZTASK_PRIORITY_INCREMENT (+1)

typedef ZULONG ZTHREAD_ID;

#endif

/* local cpu id */
#define ZCPUID_LOCAL 0

/* zos task id = ((MODID << 16) | (INSTID))*/
/* zos make the taks id from module id and instance id */
#define ZTASKID_MAKE(_modid, _inst) ZOS_MAKE_LONG((_modid), (_inst))

/* zos get moudle id from task id */
#define ZTASKID2MODID(_taskid) (ZOS_GET_HIGH_WORD(_taskid))

/* zos get instance id from task id */
#define ZTASKID2INSTID(_taskid) (ZOS_GET_LOW_WORD(_taskid))

/* zos task entry function */
typedef ZULONG (*PFN_ZTASKENTRY)(ZVOID *);

/* zos task state */
typedef enum EN_ZOS_TASK_STATE
{
    EN_TASK_STATE_FREE = 0,          /* free state */
    EN_TASK_STATE_DELETE,          /* delete state */
    EN_TASK_STATE_READY,           /* ready state */
    EN_TASK_STATE_RUNNING,         /* running state */
    EN_TASK_STATE_DELAY,           /* delay state */
    EN_TASK_STATE_SUSPEND,         /* suspend state */
}
```

```
    EN_TASK_STATE_SUSPEND_DELAY    /* suspend delay state */
} EN_ZOS_TASK_STATE;

/* zos spawn a task */
ZINT Zos_TaskSpawn(ZTASKID zTaskId, ZCHAR *pcName, ZINT iPriority,
                  ZULONG dwStackSize, ZULONG dwQueueSize, ZULONG dwTimerNum,
                  PFN_ZTASKENTRY pfnEntry, ZVOID *pEntryParm);

/* zos spawn a dyanmic task */
ZINT Zos_TaskSpawnX(ZCHAR *pcName, ZINT iPriority, ZULONG dwStackSize,
                  ZULONG dwQueueSize, ZULONG dwTimerNum, PFN_ZTASKENTRY pfnEntry,
                  ZVOID *pEntryParm, ZTASKID *pzTaskId);

/* zos delete task */
ZINT Zos_TaskDelete(ZTASKID zTaskId);

/* zos suspend task */
ZINT Zos_TaskSuspend(ZTASKID zTaskId);

/* zos resume task */
ZINT Zos_TaskResume(ZTASKID zTaskId);

/* zos task delay */
ZVOID Zos_TaskDelay(ZULONG dwMilliseconds);

/* zos task wait specific task to be deleted */
ZINT Zos_TaskWaitDelete(ZTASKID zTaskId);

/* zos set task priority */
ZINT Zos_TaskSetPriority(ZTASKID zTaskId, ZINT iPriority);

/* zos get task priority */
ZINT Zos_TaskGetPriority(ZTASKID zTaskId, ZINT *piPriority);

/* zos get current task id */
ZTASKID Zos_TaskGetSelfId();
```

```

/* zos task get real thread id */
ZTHREAD_ID Zos_TaskGetRealId(ZTASKID zTaskId);

/* zos get task name */
ZCHAR * Zos_TaskGetName(ZTASKID zTaskId);

/* zos get task state */
ZINT Zos_TaskGetState(ZTASKID zTaskId);

/* zos check whether task is deleted */
ZBOOL Zos_TaskIsDelete(ZTASKID zTaskId);

/* zos pool a task message, return immediately */
ZINT Zos_TaskPollMsg(ZTASKID zTaskId, ST_ZOS_MSG **ppstMsg);

/* zos peek a task message, return immediately */
ZINT Zos_TaskPeekMsg(ZTASKID zTaskId, ST_ZOS_MSG **ppstMsg);

```

7.6 字符串接口

```

#if ZOS_NOTUSE_STDLIB

ZVOID * Zos_MemSet(ZVOID *pMem, ZUINT iContent, ZSIZE_T zSize);
ZVOID * Zos_MemChr(const ZVOID *pMem, ZUCHAR ucContent, ZSIZE_T zSize);
ZVOID * Zos_MemCpy(ZVOID *pDest, const ZVOID *pSrc, ZSIZE_T zSize);
ZVOID * Zos_MemMove(ZVOID *pDest, const ZVOID *pSrc, ZSIZE_T zSize);
ZINT Zos_MemCmp(const ZVOID *pMem1, const ZVOID *pMem2, ZSIZE_T zSize);

ZINT Zos_StrLen(const ZCHAR *pcStr);
ZCHAR * Zos_StrCpy(ZCHAR *pcDst, const ZCHAR *pcSrc);
ZCHAR * Zos_StrNCpy(ZCHAR *pcDst, const ZCHAR *pcSrc, ZINT iCount);
ZCHAR * Zos_StrCat(ZCHAR *pcDst, const ZCHAR *pcAppend);

#else /* use the standard library */

```

```

#define Zos_MemSet(_p, _n, _size)      memset((_p), (_n), (_size))
#define Zos_MemChr(_p, _n, _size)     memchr((_p), (_n), (_size))
#define Zos_MemCpy(_p1, _p2, _size)   memcpy((_p1), (_p2), (_size))
#define Zos_MemMove(_dst, _src, _size) memmove((_dst), (_src), (_size))
#define Zos_MemCmp(_p1, _p2, _size)   memcmp((_p1), (_p2), (_size))

#define Zos_StrLen(_p)                 strlen((_p))
#define Zos_StrCpy(_dst, _src)         strcpy((_dst), (_src))
#define Zos_StrNCpy(_dst, _src, _size) strncpy((_dst), (_src), (_size))
#define Zos_StrCat(_str1, _str2)      strcat((_str1), (_str2))

#endif /* ZOS_NOTUSE_STDLIB */

#define Zos_StrChr(_p, _c)             strchr((_p), (_c))
#define Zos_StrStr(_p1, _p2)          strstr((_p1), (_p2))

#define Zos_StrToL(_p1, _p2, _p3)     strtol((_p1), (_p2), (_p3))

/* zos strlen with maximum length,
   the retrun value is same with standard return value */
ZINT Zos_StrNLen(const ZCHAR *pcStr, ZINT iCount);
ZINT Zos_StrCmp(const ZCHAR *pcStr1, const ZCHAR *pcStr2);
ZINT Zos_StrNCmp(const ZCHAR *pcStr1, const ZCHAR *pcStr2, ZSIZE_T zSize);
ZINT Zos_StrICmp(const ZCHAR *pcStr1, const ZCHAR *pcStr2);
ZINT Zos_StrNICmp(const ZCHAR *pcStr1, const ZCHAR *pcStr2, ZSIZE_T zSize);

ZCHAR * Zos_StrTok(ZCHAR *pcStr, const ZCHAR *pcDelim, ZCHAR **ppLast);

/* zero the memory by byte */
ZVOID Zos_Bzero(ZCHAR *pcMem, ZINT iCount);

/* zero the memory quick */
ZVOID Zos_ZeroMem(ZVOID *pMem, ZSIZE_T zSize);

/* zos quick fill memory content */
ZVOID Zos_Qfill(ZCHAR *pcMem, ZUCHAR chr, ZINT iCount);

```

```
/* zos string trim left and right whitespaces,
   if liner whitespace is true, it will trim whitespace, '\r', '\n',
   it will reset string end('\0') if the string length is ZNULL, */
ZVOID Zos_Trim(ZCHAR **ppcStr, ZUSHORT *pwStrLen, ZBOOL bLWS);
ZVOID Zos_TrimX(ZCHAR **ppcStr, ZULONG *pdwStrLen, ZBOOL bLWS);

/* zos string trim left whitespace,
   if liner whitespace is true, it will trim whitespace, '\r', '\n',
   it will reset string end('\0') if the string length is ZNULL, */
ZVOID Zos_TrimLeft(ZCHAR **ppcStr, ZUSHORT *pwStrLen, ZBOOL bLWS);
ZVOID Zos_TrimLeftX(ZCHAR **ppcStr, ZULONG *pdwStrLen, ZBOOL bLWS);

/* zos string trim right whitespace,
   if liner whitespace is true, it will trim whitespace, '\r', '\n',
   it will reset string end('\0') if the string length is ZNULL, */
ZVOID Zos_TrimRight(ZCHAR **ppcStr, ZUSHORT *pwStrLen, ZBOOL bLWS);
ZVOID Zos_TrimRightX(ZCHAR **ppcStr, ZULONG *pdwStrLen, ZBOOL bLWS);

/* zos string trim left and right whitespaces and replaces
   each contiguous set of whitespace within the string by one whitespace,
   if liner whitespace is true, it will trim whitespace, '\r', '\n',
   it will reset string end('\0') if the string length is ZNULL, */
ZVOID Zos_TrimAll(ZCHAR **ppcStr, ZUSHORT *pwStrLen, ZBOOL bLWS);
ZVOID Zos_TrimAllX(ZCHAR **ppcStr, ZULONG *pdwStrLen, ZBOOL bLWS);

/* zos conver string to lower case string */
ZINT Zos_Str2Lower(ZCHAR *pcStr);

/* zos conver string to upper case string */
ZINT Zos_Str2Upper(ZCHAR *pcStr);

/* zos conver string with length to lower case string */
ZINT Zos_NStr2Lower(ZCHAR *pcStr, ZUSHORT wLen);

/* zos conver string with length to upper case string */
ZINT Zos_NStr2Upper(ZCHAR *pcStr, ZUSHORT wLen);
```

```
/* zos case compare the head or tail of a string,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrHTCmp(ZCHAR *pcStr, ZCHAR *pcPart, ZBOOL bHeadCmp);

/* zos ignore case compare the head or tail of a string,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrHTICmp(ZCHAR *pcStr, ZCHAR *pcPart, ZBOOL bHeadCmp);

/* zos case compare the head or tail of a string with length,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_NStrHTCmp(ZCHAR *pcStr, ZUSHORT wLen,
                  ZCHAR *pcPart, ZBOOL bHeadCmp);

/* zos ignore case compare the head or tail of a string with length,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_NStrHTICmp(ZCHAR *pcStr, ZUSHORT wLen,
                  ZCHAR *pcPart, ZBOOL bHeadCmp);

/* zos string with length case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrCmpL(ZCHAR *pcStr1, ZCHAR *pcStr2);

/* zos string with length ignore case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrICmpL(ZCHAR *pcStr1, ZCHAR *pcStr2);

/* zos string with length case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrCmpN(ZCHAR *pcStr1, ZCHAR *pcStr2, ZUSHORT wLen2);

/* zos string with length ignore case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrICmpN(ZCHAR *pcStr1, ZCHAR *pcStr2, ZUSHORT wLen2);

/* zos string with length case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrCmpX(ZCHAR *pcStr1, ST_ZOS_SSTR *pstStr2);
```

```
/* zos string with length ignore case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_StrICmpX(ZCHAR *pcStr1, ST_ZOS_SSTR *pstStr2);

/* zos string with length case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_NStrCmp(ZCHAR *pcStr1, ZUSHORT wLen1,
                 ZCHAR *pcStr2, ZUSHORT wLen2);

/* zos string with length ignore case compare,
   it will return ZOK if string is equal, otherwise return ZFAILED */
ZINT Zos_NStrICmp(ZCHAR *pcStr1, ZUSHORT wLen1,
                  ZCHAR *pcStr2, ZUSHORT wLen2);

/* zos string copy string to a limited buffer,
   it will set string null end and return ZOK if pcSrcStr is ZNULL */
ZINT Zos_NStrCpy(ZCHAR *pcDstStr, ZUSHORT wMaxSize, ZCHAR *pcSrcStr);

/* zos string copy string with length to a limited buffer,
   it will set string null end and return ZOK if pcSrcStr is ZNULL
   or wSrcLen is 0 */
ZINT Zos_NStrNCpy(ZCHAR *pcDstStr, ZUSHORT wMaxSize,
                  ZCHAR *pcSrcStr, ZUSHORT wSrcLen);

/* zos string copy signed structure string to a limited buffer,
   it will set string null end and return ZOK if pstSrcStr is ZNULL
   or pstSrcStr length is 0 */
ZINT Zos_NStrXCpy(ZCHAR *pcDstStr, ZUSHORT wMaxSize,
                  ST_ZOS_SSTR *pstSrcStr);

/* zos string copy unsigned structure string to a limited buffer,
   it will set string null end and return ZOK if pstSrcStr is ZNULL
   or pstSrcStr length is 0 */
ZINT Zos_NStrUXCpy(ZCHAR *pcDstStr, ZUSHORT wMaxSize,
                  ST_ZOS_USTR *pstSrcStr);
```

```
/* zos string copy data buffer to a limited buffer,
   it will set string null end and return ZOK if pstMsgBuf is ZNULL */
ZINT Zos_NStrDCpy(ZCHAR *pcDstStr, ZUSHORT wMaxSize,
                 ST_ZOS_DBUF *pstMsgBuf);

/* zos string forward copy function */
ZINT Zos_ChRFCpy(ZCHAR **ppcDstStr, ZUSHORT *pwBufLen, ZCHAR chr);
ZINT Zos_StrFCpy(ZCHAR **ppcDstStr, ZUSHORT *pwBufLen, ZCHAR *pcSrcStr);

ZINT Zos_StrNFCpy(ZCHAR **ppcDstStr, ZUSHORT *pwBufLen,
                 ZCHAR *pcSrcStr, ZUSHORT wSrcLen);

ZINT Zos_StrSFCpy(ZCHAR **ppcDstStr, ZUSHORT *pwBufLen,
                 ST_ZOS_SSTR *pstSrcStr);

ZINT Zos_SStrCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_SSTR *pstDstStr,
                 ZCHAR *pcSrcStr);

ZINT Zos_SStrNCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_SSTR *pstDstStr,
                 ZCHAR *pcSrcStr, ZUSHORT wLen);

ZINT Zos_SStrFCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_SSTR *pstDstStr,
                 const ZCHAR *pcFormat, ...);

ZINT Zos_SStrXCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_SSTR *pstDstStr,
                 ST_ZOS_SSTR *pstSrcStr);

ZINT Zos_SStrUXCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_SSTR *pstDstStr,
                 ST_ZOS_USTR *pstSrcStr);

ZINT Zos_SStrDCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_SSTR *pstDstStr,
                 ST_ZOS_DBUF *pstMsgBuf);

ZINT Zos_LSStrCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_LSSTR *pstDstStr,
                 ZCHAR *pcSrcStr);

ZINT Zos_LSStrNCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_LSSTR *pstDstStr,
```

```
ZCHAR *pcSrcStr, ZULONG dwLen);

ZINT Zos_LSStrXCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_LSSTR *pstDstStr,
                  ST_ZOS_LSSTR *pstSrcStr);

ZINT Zos_LSStrDCpy(ST_ZOS_DBUF *pstMemBuf, ST_ZOS_LSSTR *pstDstStr,
                  ST_ZOS_DBUF *pstMsgBuf);

/* zos string cat string to a limited buffer,
   it will set string null end and return ZOK if pcSrcStr is ZNULL */
ZINT Zos_NStrCat(ZCHAR *pcDstStr, ZUSHORT wMaxSize, ZCHAR *pcSrcStr);

/* zos string cat string to a limited buffer,
   it will set string null end and return ZOK if pcSrcStr is ZNULL */
ZINT Zos_NStrNCat(ZCHAR *pcDstStr, ZUSHORT wMaxSize, ZCHAR *pcSrcStr,
                 ZUSHORT wSrcLen);

/* zos convert boolean data to a string */
ZINT Zos_BoolToStr(ZBOOL bTrueFalse, ZCHAR *pcStr, ZUSHORT wMaxSize);

/* zos convert a string to boolean data
 * if the wStrLen is 0, it will get length by Zos_StrLen */
ZINT Zos_StrToBool(ZCHAR *pcStr, ZUSHORT wStrLen, ZBOOL *pbTrueFalse);

/* zos convert unsigned long data to a string */
ZINT Zos_UlToStr(ZULONG dwData, ZCHAR *pcStr, ZUSHORT wMaxSize);

/* zos convert a string to decimal unsigned long data
 * if the wStrLen is 0, it will get length by Zos_StrLen */
ZINT Zos_StrToUl(ZCHAR *pcStr, ZUSHORT wStrLen, ZULONG *pdwData);

/* ZOS converts decimal int data into a string */
ZINT Zos_IntToStr(ZINT iData, ZCHAR *pcStr, ZUSHORT wMaxSize);

/* ZOS converts a string into decimal int data */
ZINT Zos_StrToInt(ZCHAR *pcStr, ZUSHORT wStrLen, ZINT *piData);
```

```
/* zos convert unsigned short data to a string */
ZINT Zos_UsToStr(ZUSHORT wData, ZCHAR *pcStr, ZUSHORT wMaxSize);

/* zos convert a string to decimal unsigned short data
 * if the wStrLen is 0, it will get length by Zos_StrLen */
ZINT Zos_StrToUs(ZCHAR *pcStr, ZUSHORT wStrLen, ZUSHORT *pwData);

/* zos convert unsigned char data to a string */
ZINT Zos_UcToStr(ZUCHAR ucData, ZCHAR *pcStr, ZUSHORT wMaxSize);

/* zos convert a string to unsigned char data
 * if the wStrLen is 0, it will get length by Zos_StrLen */
ZINT Zos_StrToUc(ZCHAR *pcStr, ZUSHORT wStrLen, ZUCHAR *pucData);

/* zos convert unsigned long data to a hexadecimal string */
ZINT Zos_UlToXStr(ZULONG dwData, ZCHAR *pcStr, ZUSHORT wMaxSize);

/* zos convert a hexadecimal string to unsigned long data
 * if the wStrLen is 0, it will get length by Zos_StrLen */
ZINT Zos_XStrToUl(ZCHAR *pcStr, ZUSHORT wStrLen, ZULONG *pdwData);

/* zos check string is a hexadecimal string */
ZBOOL Zos_IsXStr(ZCHAR *pcStr, ZUSHORT wLen);
```