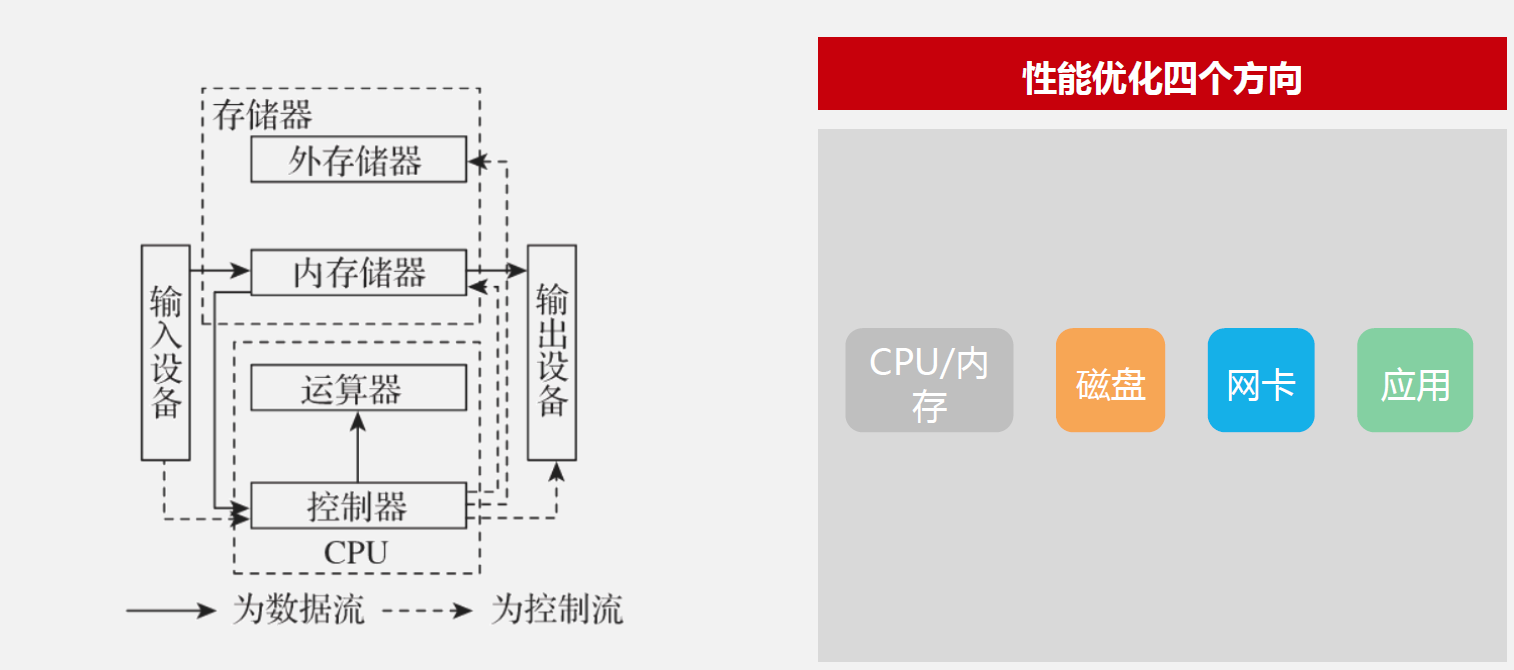
# 1 性能调优方向

从冯诺依曼架构看调优方向：



# 2 CPU/内存

## 2.1 指令优化

单独的指令优化对整体性能的提升不会有很大作用，可以放到编译优化的时候一起开展。

## 2.2 编译优化

编译优化涉及到整个微内核OS代码，需要对所有的汇编代码进行代码走查，并在优化的过程中，修改不合适的代码。

## 2.3 访存优化

1. memcpy/memset等函数的优化

目前memcpy的效率大约为270K/ms，pmon裸机程序的memcpy效率在500K/ms。考虑到Cache miss的情况，是否还有提升空间？

## 2.4 优化方案

这一部分的工作比较琐碎，需要对MIPS架构和指令集相当熟悉。而且对于微内核的底层机制理解透彻。

计划：后续微内核OS往ARM平台移植的时候，一起对这一部分的内容做一个深入的优化和调整。

# 3 操作系统

## 3.1 进程上下文切换优化

操作系统的任务切换时间至关重要，目前实际的任务切换时间还未可知（设计指标是0.2uS），需要测试该时间并进行优化，优化方向有：

1. 发生进程切换和中断响应时，进入内核态的汇编代码是否可以优化？

2. 进程上下文保护的汇编代码是否可以优化？

3. FPU寄存器上下文保护中使用Unix方式是否能够加速进程上下文切换？目前，采用的与通用寄存器相同方式。

## 3.2 调度算法优化

目前，微内核OS还是采用seL4的调度算法，其中，还涉及到虚拟化技术中Domain域的调度，是否去掉该内容？Domain域定义为1个，影响可能不大。

## 3.3 MCS算法优化

目前，微内核OS未使用，有待研究。

## 3.4 定时器

目前，微内核OS中的定时器实现是，一个高优先级的任务，可以提供的定时周期为500uS。而系统整体的调度节拍是100uS。这会非常影响系统整体性能。如果需求不需要500uS的定时器，是否可以调整为1mS？（需要和杭和西分协商）

方案：只提供1mS周期的定时器。

# 3 文件系统

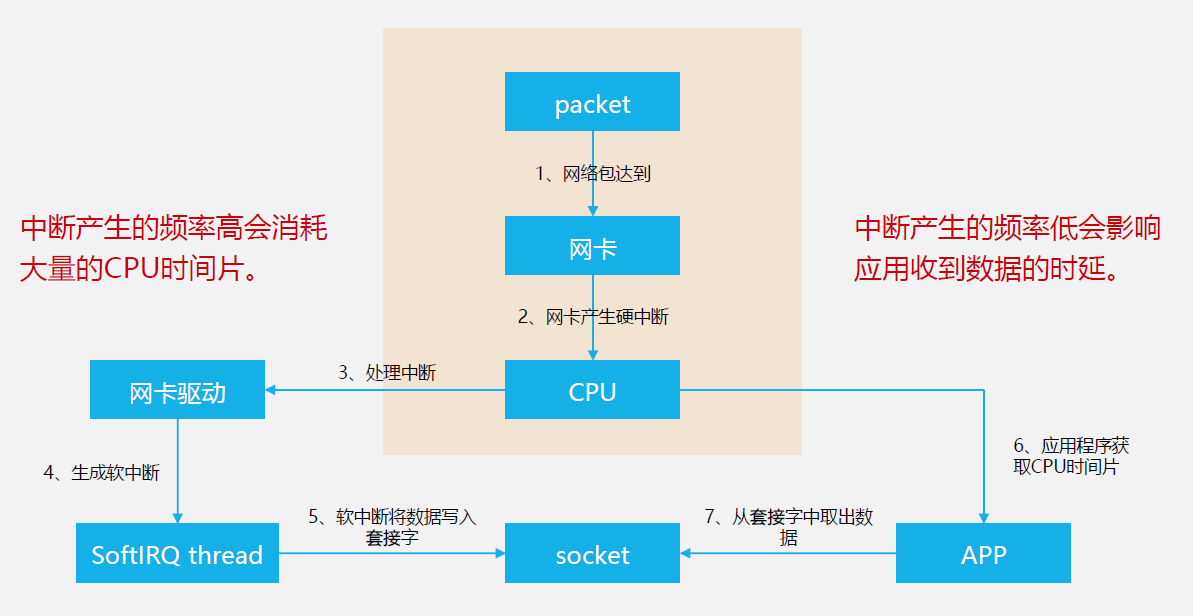
微内核OS的文件系统是基于Flash自定义的文件系统。如果加入读写缓存是否可以提高文件系统的性能？尤其是AT下装过程中和log日志的性能。

计划：目前因为flash芯片的问题，该部分的内容还在调整中。李丽娜李工还在持续优化代码中。等李工完成TSN的工作后，针对flash和文件系统再做一个深入的优化。

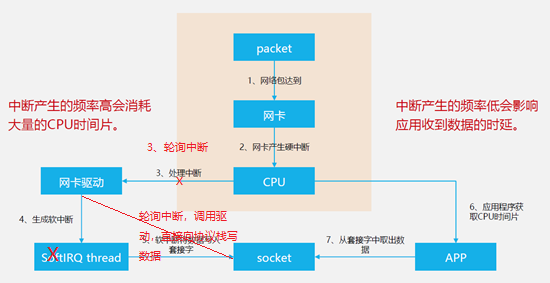
# 4 网卡

网络是RTS应用的最主要功能之一，也是影响整体性能的关键。因为它涉及到系统网A和B以及冗余网，数据量都不小。所以，其占用CPU时间片的时间也就比较长，也就是CPU负荷高的真正原因。

## 4.1 Linux网卡收发机制



## 4.2 微内核OS网卡收发机制



微内核OS中，没有真正意义上的中断。在网卡初始化的过程中，向内核注册相应的中断号，内核会记录该中断号，并使能相关的处理函数。当网卡产生硬中断的时候，微内核会通过该硬中断对应的注册函数发送消息给用户态的网卡中断处理任务（目前优先级为10）；网卡中断处理任务处理完成后，发送ack信号给内核态，告知可以继续响应中断。在此期间，内核一直阻塞该硬件中断。

## 4.3 微内核OS网卡收发机制能否改动

尝试过的方案：

1. 在网络中断处理任务中，将轮询中断的方式改为阻塞方式。但是，因为网卡中断产生太频繁，内核一直会发送消息过来，中断处理任务根本处理不过来。（未成功）
2. 将轮询中断的时间间隔由1ms改为2ms，占用CPU时间片的时间明显缩短，CPU负荷从70%左右降到30%左右（没有RTS程序）。带来的影响就是，数据包的响应时间下降，直观体现就是ping主控的时候，时间由1ms左右上升到了7ms左右。

## 4.4 GMAC和网讯驱动优化

方案1：延时方法（原驱动使用的方法）

理论分析：

GMAC驱动中，在读取中断类型寄存器的时候，为了确保数据是正确的，每次都添加了延时（10000次for循环），将其改为100之后，在测试系统网A口的时候，（大数据量测试），可以看出大约有10%的速度提升。

技术难点：无

实施：已经优化

方案2：sync指令同步

理论分析：

相比于网讯驱动代码，GMAC驱动在读写寄存器的时候，采用了延时的方法，保证寄存器数据正确交换到RAM中；而网讯驱动基本上采用的是同步指令 “sync”的方法。延时的方法，不太友好，延时时间长，影响读写网卡速度，延时短，可能无法保证寄存器读写的正确性。所以，需要将GMAC驱动中的延时方法，使用同步指令“sync”实现。

技术难点：不确定sync指令是否可以完全达到效果

实施：未进行

## 4.5 冗余网优化

冗余网的驱动就是GMAC驱动，所以优化GMAC驱动也可以提高冗余网的性能。但是，冗余网的使用还需要注意。存在的问题在[6.5小节](#_6.5_冗余网)有记录。

## 4.6 网络优化方案

方案1：将3个网络中断接收任务合并成1个任务。

理论依据：目前3个网络中断接收任务的任务优先级都是10（245），理论上分析，跟一个任务执行没有大的什么区别，因为都是顺序执行。改成一个任务之后，可以省掉2个任务的切换时间，当轮询频率很快的时候，系统整体性能会有提升。

技术难点：较容易实现，效果还得看测试结果。

实施：未进行。

方案2：尝试修改中断轮询方式为阻塞式。

理论依据： 目前之所以采用纯轮询的方式是因为中断阻塞不住（之前曾经尝试过，没有成功）。如果能改为阻塞式中断接收，空闲的时候就不需要执行轮询网络中断接收任务，会大大降低CPU负荷。

技术难点：阻塞式中断还未实现，目前还不清楚是因为网卡中断产生的太频繁，还是gmac的中断信号使用有问题。（可以查阅Linux内核下，gmac中断的实现方式）。

实施：未进行。

方案比较：方案1比方案2容易实现，方案2存在技术无法实现的风险。性能优化的效果上，理论上方案2由于方案1。

# 5 应用

杭和西分开发。

# 6 重要问题

针对杭和西分在使用微内核过程中出现的问题，整理汇总后，有如下问题需要解决。

## 6.1 定时器

杭和西分同事反映定时器不准，误差在10%左右。RTS代码要求，IEC任务的整体误差在10%左右。需要对定时器的实现机制进行分析，摸清真实的误差是多少，是否有提高的空间。

实施：微内核OS小组已经对定时器进行测试，实际测试结果最大误差为2%（基于0.5.9版本）。

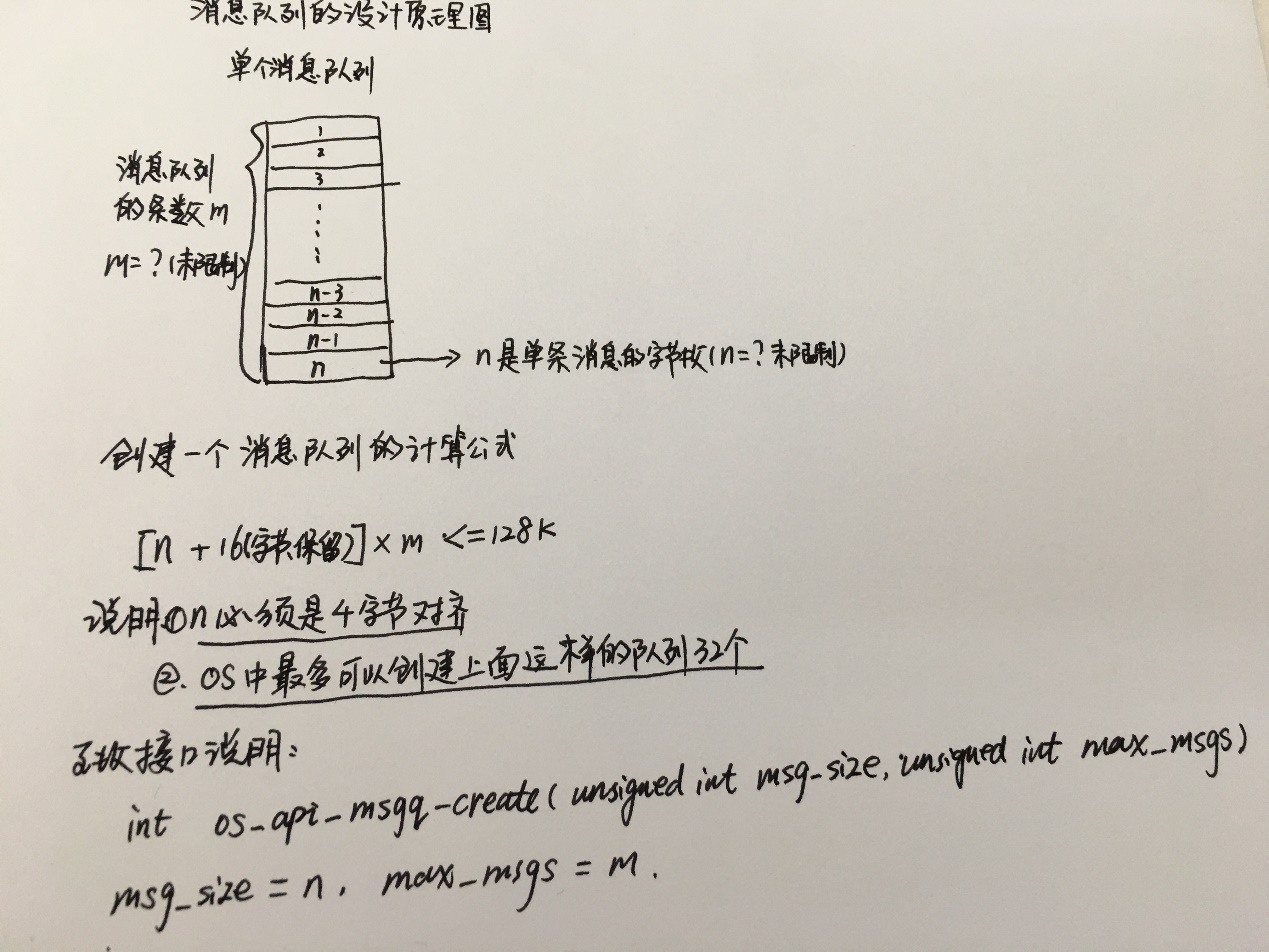
问题解决：经过沟通，是杭和西分同事的测试代码有问题。

## 6.2 消息队列

消息队列目前使用中老是出现bug，具体问题有两个：

1. 单条消息发送（频繁）过程中，出现丢消息的问题；
2. 多任务消息处理过程中，出现崩溃问题。

消息队列的设计原理和规格：



分析和实施：分析了代码，排除消息队列的问题。

使用请按照上面图片的说明进行设计。

## 6.3 文件系统

文件系统单独使用没有问题。加入log日志功能，会有崩溃现象出现。已经定位到跟消息队列有关（董工、李工正在处理中）。

另外，文件系统加上日志系统需要进行可靠性测试。（技术中心自己进行）

1. 2021-04-02

文件系统目前支持1024+100个文件。

Log日志支持的最大消息长度是128个字符。

## 6.4 校时功能

RTS在调用os\_api\_gettimeofday和os\_api\_settimeofday函数中有问题。

os\_api\_settimeofday写的比较频繁时，系统指示灯会有全灭的情况。目前，怀疑跟系统TICK中断频繁关闭有关，需要应用使用这个函数的时候注意一下。

另外，需要针对时间相关函数进行测试，并进行代码走查。（技术中心自己进行）

问题解决：测试代码有问题。

## 6.5 冗余网

目前技术中心自己测试的情况是：在没有其它任务的情况下，测试了冗余网的发送速率是90K/ms左右。杭和西分的杨（银霞）工测试最大速率是10ms周期内发送130K可以。数据量170K的时候，会发生备机接收不到数据的情况（也就是好像卡住了）。

优先级顺序：2、3、5、1、4

# 7 改善项

7.1 如果出现崩溃，是否可以将崩溃信息打印到日志系统？或者自复位？

7.2 pmon命令行flash烧写有时候会有问题

7.3 CPU负荷优化

7.4 网卡驱动中有时候会打印CRC错误？

7.5 pmon版本号的问题