QuecOpen EC2X&AG35 低功耗-休眠唤醒方案

QUECTEL PRELIMINARY

About the Document

本文档适用于 EC2X 和 AG35 平台

History

Revision	Date	Author	Description
1.0	2017-11-30	Gale	Initial
1.1	2018-01-30	Gale	Add the selection of driver(chapter 7)

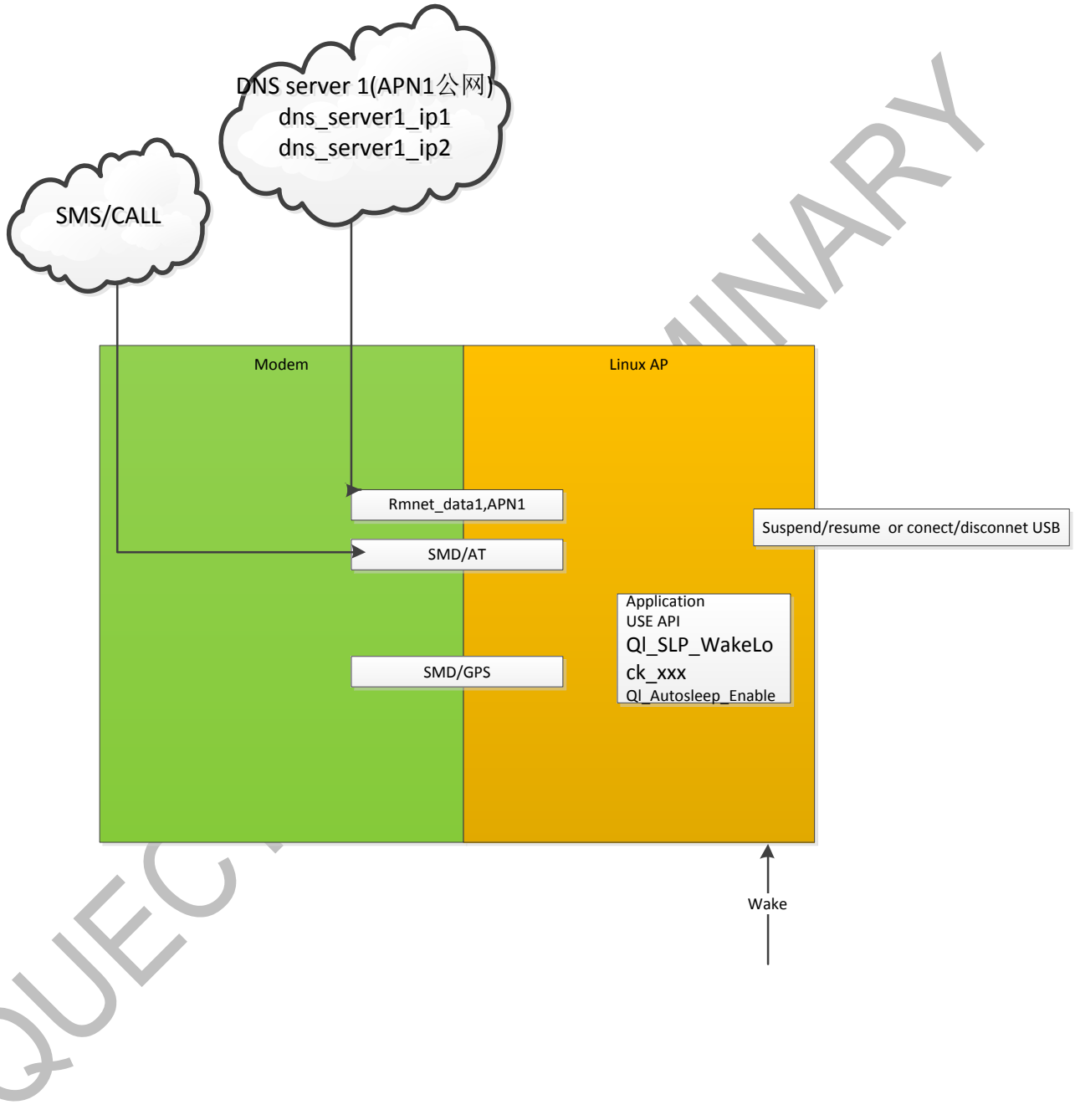
QUECTEL PRELIMINARY

目录

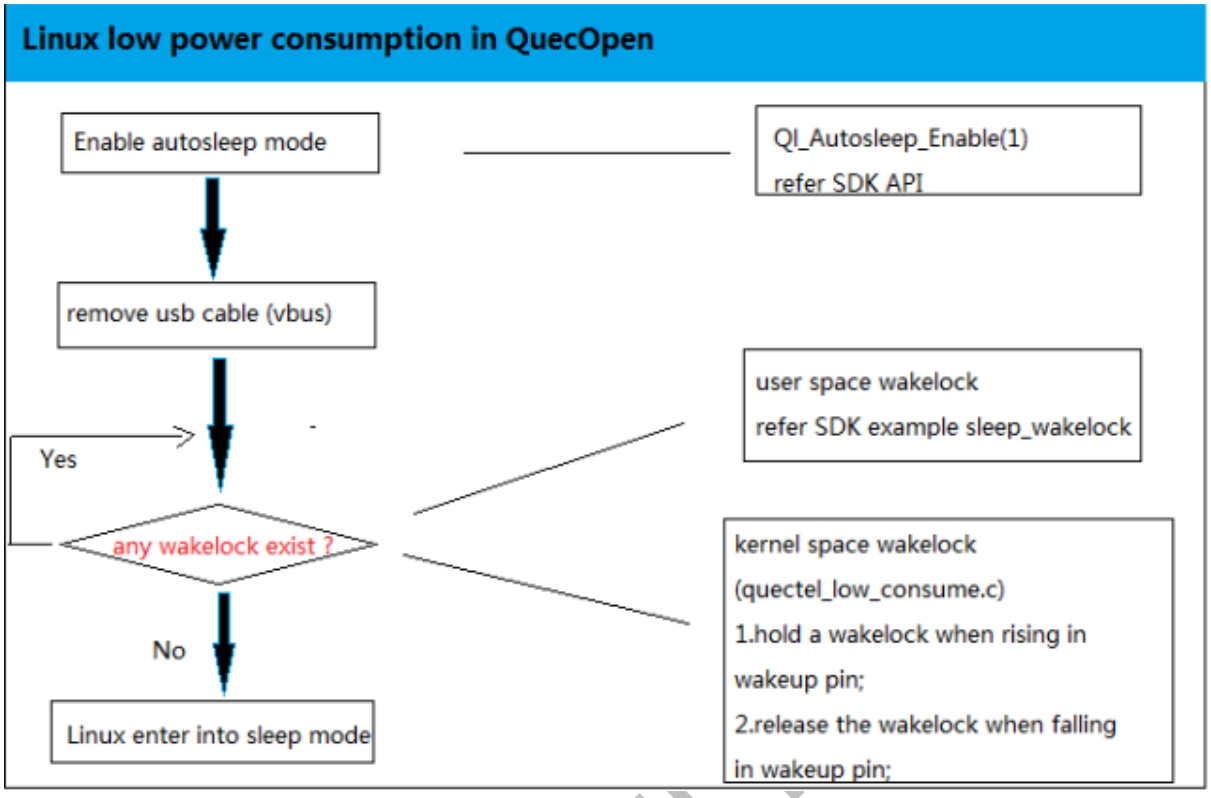
QuecOpen EC2X&AG35 低功耗-休眠唤醒方案 0
About the Document 1
1. 休眠结构 3
2. 概要 4
3. API 接口介绍 4
4. API 调试方法介绍 5
5. 用户层：low_power_consume_app.c 6
6. 驱动层：low_power_consume_driver.c 6
设备树配置 6
驱动实现 7
7. 调试 8
休眠唤醒过程 ..... **错误!未定义书签。**
8. 案例一： 10

QUECTEL PRELIMINARY

1.休眠结构



QUECTEL PRELIMINARY



2.概要

模块默认不休眠；
内核中 quectel 支持了 AutoSleep 和 wakelock 机制；
驱动层 quectel 提供了驱动 demo,供用户参考，二次开发，默认情况下低功耗 driver 未编译进内核，用户按需要自行打开（参考第 7 节）；
用户层 quectel 提供了 autosleep, 和 wakelock 的相关 API 用来控制休眠，参考 ql-ol-extsdk/example/low_power_consume_app；

3.API 接口介绍

Linux Appliction 使用下面的 API 来进行休眠控制。

int Ql_Autosleep_Enable(char enable);
使能 AutoSleep；使能后系统会在满足条件后自动进入休眠。

int Ql_SLP_WakeLock_Create(const char *name, size_t len);
否则返回-1，通过 errno 可以查询错误码。 最多可以创建 512 个 lock。参数 name 是 lock 的名字，len 是 name 的长度。名字长度最大允许 28 个字符。

QUECTEL PRELIMINARY

int Ql_SLP_WakeLock_Lock(int fd);
锁定后，linux 无法进入 sleep。

int Ql_SLP_WakeLock_Unlock(int fd);
当所有 APP 的所有都解锁了，系统才有可能进入 sleep。

int Ql_SLP_WakeLock_Destroy(int fd);

Ql_Autosleep_Enable 使能 AutoSleep 后不要通过 Ql_Autosleep_Enable(0)来保持唤醒，而应该使用 Ql_SLP_WakeLock_Lock 使系统保持唤醒。Ql_SLP_WakeLock_Unlock 可以释放 wakelock，放弃休眠锁定。系统休眠后进程会被冻结，唤醒后进程会继续运行。

4.API 调试方法介绍

在控制台下面执行命令 cat /sys/kernel/debug/wakeup_sources

```
root@mdm9607-perf:~# cat /sys/kernel/debug/wakeup_sources
name            active_count    event_count     wakeup_count    expire_count
1883.quec50     0               0               0               0
1883.quec49     0               0               0               0
1883.quec48     0               0               0               0
1883.quec47     0               0               0               0
1883.quec46     0               0               0               0
1883.quec45     0               0               0               0
1883.quec44     0               0               0               0
1883.quec43     0               0               0               0
1883.quec42     0               0               0               0
1883.quec41     0               0               0               0
```

以“pid.name”的格式显示，其中 pid 是进程的 id，name 是 Lock 的名称。在上面的截图中，可以看到进程 1883 创建了名称为 quec50，quec49…的 lock。如果系统无法进入 sleep，可以通过这条命令来查看是否有一些 APP 的 lock 处于 lock 状态。这条命令同时也会打印 linux 内核中的其他 lock，在上面截图中没有显示出来。

awk -F " " '$6 != 0 {print $1"   "$6}' /sys/kernel/debug/wakeup_sources
这条指令可以查看系统当前持有的唤醒锁。

```
root@mdm9607-perf:~# awk -F " " '$6 != 0 {print $1"    "$6}' /sys/kernel/debug/w
akeup_sources
name    active_since
msm_otg    1616231
root@mdm9607-perf:~#
root@mdm9607-perf:~#
root@mdm9607-perf:~#
```

QUECTEL PRELIMINARY

5. 用户层：low_power_consume_app.c

实现了短信，电话，数据，gpio 的休眠唤醒；
具体参考 extsdk/example/low_power_consume_app；

6.驱动层：low_power_consume_driver.c

6.1 设备树配置

Demo 使用 9x07 芯片 gpio25(wakeup 脚), gpio75(notify 脚),gpio42(sleep_ind 脚可选)，用户也可以更换其他管脚；
ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607.dtsi

```
//2017-10-10,add by gale, for quectel low consume
quec,quectel-low-consume{
        compatible = "quec,quectel-low-consume";

        notify_gpio = <&tlmm_pinmux 75 0>;      /* which gpio can be controlled by ioctl to notify mcu */

        wakeup_gpio = <&tlmm_pinmux 25 0>;      /* wakeup pin */

        pinctrl-names = "default", "sleep", "wakeup_default";     /* status pin: module sleep indication gpio42 */
        pinctrl-0 = <&sleep_ind_active>;
        pinctrl-1 = <&sleep_ind_sleep>;
        pinctrl-2 = <&wakeup_in_default>;       /* wakeup pin default state */
        status = "disabled";                    /* disable by default */
};
```

ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-pinctrl.dtsi

QUECTEL CONFIDENTIAL PRELIMINARY

```
/* quectel low power consume */
wakeup_in_default: wakeup_in_default {
        config {
                pins = "gpio25";
                drive-strength = <2>;
                bias-pull-up;
        };
};

sleep_ind_active: sleep_ind_active {
        mux {
                pins = "gpio42";
                function = "gpio";
        };
        config {
                pins = "gpio42";
                drive-strength = <2>;
                bias-disable;
                output-low;
        };
};

sleep_ind_sleep: sleep_ind_sleep {
        mux {
                pins = "gpio42";
                function = "gpio";
        };
        config {
                pins = "gpio42";
                        drive-strength = <2>;
                        bias-disable;
                        output-high;
        };
};
```

6.2 驱动实现

驱动 demo 位于 ql-ol-kernel/drivers/quectel-drivers/low_power_consume_driver

1. 提供了一个 ioctl，供应用 app 调用，调用这个接口，把 Notify 引脚（9x07 gpio75）拉高可以用来唤醒 mcu，或者拉低通知 mcu 休眠；
2. 实现 wake 引脚（9x07 gpio25）的中断唤醒， 外部 mcu 拉高 wake 引脚唤醒 4G 模块，驱动会在中断子程序中异步通知用户 app 上升沿到来，开始处理用户业务；拉低 wakeup 脚，中断子程序中异步通知用户 app 下降沿到来，可以处理用户业务，再解锁进入休眠；
3. 休眠状态指示：实现 sleep ind（9x07 gpio42）引脚的休眠指示，系统 suspend 时， sleep_ind 配置为高电平，resume 时配置 sleep_ind 到低电平；
4. 客户可以基于这个驱动做二次开发；

QUECTEL

7.休眠唤醒的使能和调试

7.1 休眠唤醒的使能

7.1.1 使能设备节点

QuecOpen 默认使用的几个管脚来测试，用户也可以更换其他管脚；

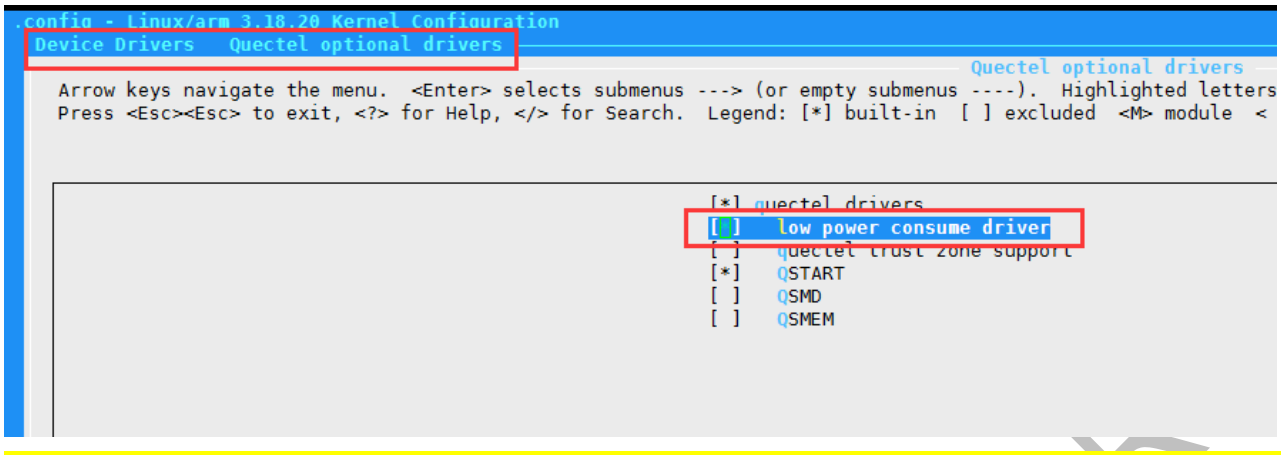
修改 ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607.dtsi　　status=”ok”;使能 device ；

```
    //2017-10-10,add by gale, for quectel low consume
    quec,quectel-low-consume{
            compatible = "quec,quectel-low-consume";

            notify_gpio = <&tlmm_pinmux 75 0>;      /* which gpio can be controlled

            wakeup_gpio = <&tlmm_pinmux 25 0>;      /* wakeup pin */

            pinctrl-names = "default", "sleep", "wakeup_default";     /* status pin
            pinctrl-0 = <&sleep_ind_active>;
            pinctrl-1 = <&sleep_ind_sleep>;
            pinctrl-2 = <&wakeup_in_default>;       /* wakeup pin default state */
            status = "disabled";                    /* disable by default */
    };
```

7.1.2 选中 driver

make kenel_menuconfig 选中 low_power_consume_driver　　使能 driver;

```
2:~/MDM9x07/SDK_FAG0130/ql-ol-sdk$ make kernel_menuconfig
)M9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-kernel ; make ARCH=arm mdm9607-perf_defconfig menuconfig O=build
ng directory `/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-kernel'
ng directory `/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-kernel/build'
efile
eless/bcmdhd/Kconfig:50:warning: defaults for choice values not supported
n/Kconfig:320:warning: choice value used outside its choice group
n/Kconfig:325:warning: choice value used outside its choice group
)C_QDSP6V2) selects SND_SOC_MSM_QDSP6V2_INTF which has unmet direct dependencies (SOUND && !M68K &&
)C_QDSP6V2) selects SND_SOC_MSM_QDSP6V2_INTF which has unmet direct dependencies (SOUND && !M68K &&

 written to .config
```



此时会生成隐藏文件在 ql-ol-kernel/build/.config ， 此文件是 make kernel 过程中所依赖的内核配置文件，当用户确定此修改后的.config 文件要保存提交代码仓库，则需要 cp ql-ol-kernel/build/.config ql-ol-kernel/arch/arm/configs/mdm9607-perf_defconfig， 这样之前对内核选项的修改就更新到 ql-ol-kernel/arch/arm/configs/mdm9607-perf_defconfig（另外若用户使用 git 维护代码，那么需要追踪此文件，而不要追踪.config）

```
ql-ol-sdk$ cp ql-ol-kernel/build/.config ql-ol-kernel/arch/arm/configs/mdm9607-perf_defconfig
ql-ol-sdk$
```

7.2 调试

1.编译 make kernel，烧录此内核镜像；

```
:~/MDM9x07/SDK_FAG0130/ql-ol-sdk$ make kernel
M9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-kernel ; [ ! -f build/.config ] && echo -e "\033[31m.co
H=arm CC=arm-oe-linux-gnueabi-gcc LD=arm-oe-linux-gnueabi-ld.bfd -j 4 O=build || exit ; \
/arch/arm/boot/zImage build/arch/arm/boot/dts/qcom/mdm9607-mtp.dtb /home/gale/MDM9x07/SDK
g directory `/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-kernel'
g directory `/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-kernel/build'
file
```

2.编译 low_power_consume_app 上传并执行，拔出 usb(或者 usb 进入 suspend 模式)，在控制台下面执行命令 cat /sys/kernel/debug/wakeup_sources 可以看到 app 当前持有唤醒锁；

```
ql-ol-sdk/ql-ol-extsdk/example/low_power_consume_app$ make
a -mfloat-abi=softfp -mfpu=neon  -O2 -fexpensive-optimizations -frename-registers
0130/ql-ol-sdk/ql-ol-extsdk/example/low_power_consume_app/../../lib/interface/inc
I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neo
oe-linux-gnueabi/usr/include/data -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-o
k/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include/qmi -I/ho
mework  -c  -I./ -I./inc -I../../include -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-
l-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include -I/home/gale
```

```
root@mdm9607-perf:~# cat /sys/kernel/debug/wakeup_sources
name            active_count    event_count     wakeup_count    expire_count    active_since    tota
1284.low_power_consume  1               1               0               0               413486
ipc00000058_pdc_daemon  1               1               0               0               0
ipc00000057_pdc_daemon  0               0               0               0               0
ipc00000056_pdc_daemon  0               0               0               0               0
ipc00000055_atfwd_daemon        79              79              0               0               0
```

3.此时 sleep_ind 脚低电平；

4. wakeup 引脚输入下降沿，通知到用户层 app 拉低 notify 脚通知 mcu，并解锁进入休眠，同时会拉高 sleep_ind；

```
[gpio_cb_handler  301]: gpio irq: Have caught signal 29 from driver
[gpio_cb_handler  327]: edge: falling
[gpio_cb_handler  331]: do something before enter sleep mode
[gpio_cb_handler  332]: module will enter sleep mode
```

5.wakeup 引脚输入上升沿，唤醒 4G 模块，并通知到用户层 app，拉低 sleep_ind 脚，app 开始处理业务。

```
root@mdm9607-perf:~# [gpio_cb_handler  301]: gpio irq: Have caught signal 29 from driver
[gpio_cb_handler  327]: edge: rising
```

参考 extsdk/example/low_power_consume_app，也可以在 app 中打开短信，电话，数据一起测试；

8.案例一：

概要：使用 3 个 gpio，gpio25(wakeup 脚), gpio75(notify 脚),gpio42(sleep_ind 脚可选)
wakeup(输入脚，边沿触发，触发 4G 模块进入休眠或者唤醒)
notify(输出脚，mcu 端使用边沿触发，触发 mcu 进入休眠或者唤醒)
sleep_ind(输出脚，仅作为 4G 模块状态指示，可选)

默认 4G 模块和 MCU 开机正常工作时，wakeup 脚输入高，notify 脚输出高，app 使能 autosleep，并 lock ；

1. **休眠方式：（需要 mcu 通知 4G 模块休眠）**

Mcu 满足休眠条件后，拉低 wakeup 脚通知 4G 模块，触发用户层回调，unlock，并 ioctl 拉低 notify 脚，4G 模块进入休眠，mcu 进入休眠。

2. **唤醒方式:**

A) 4G 模块唤醒 MCU

4G 模块收到电话短信 tcp 数据后，进入回调，首先 wakelock, ioctl 拉高 notify 脚，唤醒 MCU，并处理客户业务；

B) MCU 唤醒 4G 模块

MCU 通过拉高 wakeup 脚，唤醒 4G 模块，触发用户层回调，首先 wakelock，开始处理客户业务。

QUECTEL PRELIMINARY