How to Use TCSM in Linux

In JZ4750, TCSM is a kernel-friendly on-chip memory, that is, to access TCSM, programs must be running in kernel mode. We know that generic applications running on OSs like Linux just only possess user mode to prohibit them carelessly or intently intruding OS kernel. So how to let favorite applications take use of TCSM? This doc gives a tricky patch to work around the issue, however, it is not a perfect method.

In linux kernel for MIPS, there is only one way of identifying user mode. A macro function named user_mode(regs) is defined in the header file ptrace.h as following:

#define user_mode(regs) (((regs)->cp0_status & KU_MASK) == KU_USER)
Here KU_MASK equals 0x18 and KU_USER equals 0x10.

Let's think of several key constraints listed as following:
1. MIPS spec docs specifies the bit 5 named UM of STATUS regsiter of CP0 denoting user mode.
2. Many kernel codes would refer to **user_mode(regs)** to assist kernel to identify the corresponding mode of the process trapping in where. Note that *(regs)->cp0_status* contains the target process' status before trapping.
3. CPU of JZ4750 is compliant to MIPS spec.

So if we can clear the UM bit by some way, CPU of JZ4750 will grant access being issued from any generic process to TCSM. Meanwhile we must let kernel believe that the process resuming after clearing UM bit is still running in user mode. How to fulfill the two contradictive conditions? The key is the macro function **user_mode(regs)**. See following modification:

#define user_mode(regs) ((((regs)->cp0_status & KU_MASK) == KU_USER) || (((regs)->cp0_status & 0x08000000) == 0x08000000))

Obviously the modified function will cheat kernel, it tells kernel that another bit (bit 27 named RP) of STATUS also denotes user mode. In another word, value change of STATUS from UM=1 RP=0 to UM=0 RP=1 is transparent to the modified **user_mode(regs)**.
Why is RP? RP mode means reduced power mode, it is an implementation-defined field and may only absorb dedicated driver programmer for dedicated implementation, so kernel code does not care it. Furthermore, JZ4750 uses a different way to accomplish low power control, so using this bit is safe enough.

Finally, to make such modification, a dedicated driver is necessary. The driver must supply interface for generic applications to make such modification and to restore from such modification. Following are the key code of such a driver:

```
static int tcsm_open(struct inode *inode, struct file *filp)
{
   struct pt_regs *info = task_pt_regs(current);
   info->cp0_status &= ~0x10;
   info->cp0_status |= 0x08000000; // a tricky
   return 0;
}

static int tcsm_release(struct inode *inode, struct file *filp)
{
   struct pt_regs *info = task_pt_regs(current);
   info->cp0_status |= 0x10;
   info->cp0_status &= ~0x08000000; // a tricky
   return 0;
}
```