igh移植步骤和igh主站应用

一、igh移植步骤

1、解压igh源码包

将源码包拷贝到/home/gree/igh目录下并解压。

```
cp XX/etherlabmaster-code.tar.gz /home/gree/igh/
cd /home/gree/igh
tar xvzf etherlabmaster-code.tar.gz
```

2、安装必要工具

安装libtool、autoconf、automake1.9。libtool和autoconf没有版本要求，因此可使用以下命令直接安装.。

```
sudo apt-get install libtool autoconf
```

而automake要求1.9版本，目前我知道的安装方法是，先上ftp://ftp.gnu.org下载automake1.9压缩包，然后使用离线安装的方法安装。

3、配置

调用configure进行配置。

```
cd etherlabmaster-code
./configure --enable-8139too=no --enable-generic=yes
```

4、编译

```
make
make modules
```

5、安装模块

(1) 安装EtherCAT头文件，初始化版本，系统配置文件和用户工具到默认路径。

```
make install
```

(2) 安装内核模块到内核模块路径。

```
make modules_install
```

(3) 生成模块依赖关系，之后modprobe会根据depmod产生的依赖关系决定加载哪些依赖模块。

```
depmod
```

6、配置主站

(1) 切换到/opt/etherlab目录。

```
cd /opt/etherlab
```

(2) 修改配置文件/opt/etherlab/etc/sysconfig/ethercat。

```
vim etc/sysconfig/ethercat
在MASTER0_DEVICE=""填写你的以太网卡的MAC地址，DEVICE_MODULES="generic"，保存退出。
```

(3) 新建/etc/sysconfig/文件夹，将上一步骤修改后的ethercat文件拷贝到新建的文件夹中，并创建一个连接。

```
mkdir /etc/sysconfig/
cp /opt/etherlab/etc/sysconfig/ethercat /etc/sysconfig/
ln -s /opt/etherlab/etc/init.d/ethercat /etc/init.d/
```

7、启动主站

完成上述步骤之后，执行命令：

```
/etc/init.d/ethercat start
```

如果显示以下说明启动成功：

```
Starting EtherCAT master 1.5.2 done
```

8、添加命令行工具

(1) 修改/root和/home/gree目录下的.bashrc，添加PATH=$PATH:/opt/etherlab/bin。

(2) 在对应目录执行 source .bashrc。

二、igh主站应用

1、主函数架构

- 请求一个主站实例
- 为从站创建过程数据域
- 获取从站的配置信息
- 根据获取的配置信息和同步管理器信息配置从站
- 为创建的域注册pdo条目
- 激活主站
- 获取过程数据区域地址，任务中直接使用来和从站进行数据交互
- 创建任务线程
- 等待任务线程终止来回收子线程资源

```
int main(int argc, char **argv)
{
    if (mlockall(MCL_CURRENT | MCL_FUTURE) == -1)
    {
        fprintf(stderr, "Failed to mlockall.\n");
        return EC_FALSE;
    }

    signal(SIGTERM, catch_signal);
    signal(SIGINT, catch_signal);

    /* step1. 请求一个master */
    master = ecrt_request_master(0);
    if (!master)
    {
        return EC_FALSE;
    }
    else
    {
        fprintf(stdout, "===========================\n");
        fprintf(stdout, "Initialize EtherCAT Master.\n");
        fprintf(stdout, "===========================\n");
    }

    /* step2. 为从站创建过程数据域domainIO */
    domainIO = ecrt_master_create_domain(master);
    if(!domainIO)
    {
        fprintf(stderr, "Failed to create domianIO.\n");
        return EC_FALSE;
    }

    /* step3. 获取从站的配置信息 */
    if (!(sc[IOPos] = ecrt_master_slave_config(master, 0, IOPos, IOINFO)))
    {
        fprintf(stderr, "Failed to get IO Slaves configuration.\n");
```

```
    return EC_FALSE;
}

fprintf(stdout, "Configuring IO Slave[%d] PDOs...\n", IOPos);

/* step4. 根据获取的配置信息和同步管理器信息配置从站 */
if(ecrt_slave_config_pdos(sc[IOPos], EC_END, io_syncs))
{
    fprintf(stderr, "Failed to configure IO Slaves[%d] PDOs.\n", IOPos);
    return EC_FALSE;
}


/* step5为域domainIO注册pdo条目 */
if(ecrt_domain_reg_pdo_entry_list(domainIO, io_regs))
{
    fprintf(stderr, "IO PDO entry registration failed!\n");
    return EC_FALSE;
}

/* END OF CONFIG */

/* step6 激活主站 */
if (ecrt_master_activate(master))
{
    return EC_FALSE;
}
else
{
    fprintf(stdout, "============================\n");
    fprintf(stdout, "Activating EtherCAT Master.\n");
    fprintf(stdout, "============================\n");
}

/* step7 获取域domainIO的过程数据区域，任务直接使用来和从站进行数据交互 */
if(!(domainIO_pd = ecrt_domain_data(domainIO)))
{
    return EC_FALSE;
}


/* create thread for command and cyclic job */
struct sched_param rtparam = { .sched_priority = 82 };
pthread_attr_t rtattr;

pthread_attr_init(&rtattr);
pthread_attr_setdetachstate(&rtattr, PTHREAD_CREATE_JOINABLE);
pthread_attr_setschedpolicy(&rtattr, SCHED_FIFO);
pthread_attr_setinheritsched(&rtattr, PTHREAD_EXPLICIT_SCHED);
pthread_attr_setschedparam(&rtattr, &rtparam);

/* create job task thread */
/* step7 创建任务线程cyclicJobTask */
```

```
    errno = pthread_create(&rt_jobtask, &rtattr, &cyclicJobTask, NULL);
    if (errno)
    {
        fprintf(stderr, "rt_cyclic_jobtask pthread_create: failed.\n");
    }


    /* step8 等待线程cyclicJobTask终止 */
    while(run)
    {
        sched_yield();
    }
    pthread_cancel(rt_jobtask);
    pthread_join(rt_jobtask, NULL);

    return 0;
}
```

2、测试任务线程

- 周期性数据交互
- 运行完毕释放主站

```
void *cyclicJobTask(void *arg)
{
    fprintf(stdout, "Starting Cyclic Function.\n");
    cyclic_task();  /* 进入周期性数据交互任务 */

#if USE_CTI
    endsignal();
#endif

    /* 运行完了任务，释放主站 */
    ecrt_release_master(master);
    return NULL;
}
```

2.1、周期性数据交互

- 先睡眠一段时间
- 醒来后接收从站的过程数据
- 更新主站的过程数据
- 时钟同步相关操作
- 发送过程数据给从站

```
void cyclic_task(void)
{
    .....

    /* 获取当前时间到wakeupTime */
    clock_gettime(CLOCK_REALTIME, &wakeupTime);
```

```
/* 周期性任务循环 */
while(run)
{
    /* 重新设置wakeupTime为原来的wakeupTime加上cycletime */
    wakeupTime = timespec_add(wakeupTime, cycletime);

    /* step1. 先睡眠cycletime */
    clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, &wakeupTime, NULL);

    .....

    /* step2. 接收从站发送的过程数据 */
    ecrt_master_receive(master);
    ecrt_domain_process(domainIO);

    if(counter)     /* 第一次循环 */
    {
        counter--;  /* 更新counter，每次循环更新一次 */
    }
    else        /* 第一次循环或者counter减少到了0 */
    {
        counter         = FREQUENCY;
       .....
        if(init_flag == 216)
        {
            init_flag = Circle_Init();  /* 初始化过程数据都为0 */
            fprintf(stdout, "Circle init.\n" );
            isflag    = 1;
        }
    }

    if(isflag)
    {
        if(init_flag)
        {
            /* step3. 更新过程数据IOSlaveOutPut */
            Circle_main();
        }

       /* step4. 将过程数据写到过程数据域对应的区域 */
        EC_WRITE_U16(domainIO_pd + off_dig_out1, IOSlaveOutPut.DO1);
        EC_WRITE_U16(domainIO_pd + off_dig_out2, IOSlaveOutPut.DO2);
        EC_WRITE_U16(domainIO_pd + off_dig_out3, IOSlaveOutPut.DO3);

    }


    /* step5. 时钟同步相关操作 */
    clock_gettime(CLOCK_REALTIME, &time);
    ecrt_master_application_time(master, TIMESPEC2NS(time));
    if (sync_ref_counter)
    {
        sync_ref_counter--;
```

```
        }
        else
        {
            sync_ref_counter = 1; // sync every cycle
            ecrt_master_sync_reference_clock(master);
        }
        ecrt_master_sync_slave_clocks(master);


        /* step6. 发送过程数据，即将过程数据域中的pdo映射到从站 */
        ecrt_domain_queue(domainIO);
        ecrt_master_send(master);
        .....
    }
}
```