1. Cyg\_Scheduler\_Implementation::Cyg\_Scheduler\_Implementation()

属于：Cyg\_Scheduler\_Implementation

调用：

1. Cyg\_Thread \* Cyg\_Scheduler\_Implementation::schedule(void)

属于：Cyg\_Scheduler\_Implementation

调用：

CYG\_ASSERT( \_bool\_, \_msg\_ )

cyg\_ass.h 宏定义函数替换

get\_current\_thread()

Sched.hxx (packages\kernel\current\include):

inline Cyg\_Thread \*Cyg\_Scheduler\_Base::get\_current\_thread()HAL\_LSBIT\_INDEX(index, pending\_map);

HAL层中若干硬件实现中均存在

#define HAL\_LSBIT\_INDEX(index, mask) index = hal\_lsbindex(mask)

get\_head()

Clist.hxx (packages\infra\current\include): Cyg\_DNode \*get\_head() { return head; };

get\_next()

Clist.hxx (packages\infra\current\include): Cyg\_DNode \*get\_next() { return next; };

1. Void Cyg\_Scheduler\_Implementation::add\_thread(Cyg\_Thread \*thread)

属于：Cyg\_Scheduler\_Implementation

调用：

CYG\_ASSERT( \_bool\_, \_msg\_ )

cyg\_ass.h 宏定义函数替换

remove(thread)

Clist.hxx (packages\infra\current\include): void remove( Cyg\_DNode \*node )

empty()

Clist.hxx (packages\infra\current\include): cyg\_bool empty() { return head == NULL; };

set\_need\_reschedule(thread);

Sched.hxx (packages\kernel\current\include):

inline void Cyg\_Scheduler\_Base::set\_need\_reschedule() Cyg\_Scheduler\_Base的静态内联函数

timeslice\_reset();

inline void Cyg\_SchedThread\_Implementation::timeslice\_reset()

1. Void Cyg\_Scheduler\_Implementation::rem\_thread(Cyg\_Thread \*thread)

调用：

1. void Cyg\_Scheduler\_Implementation::set\_need\_reschedule(Cyg\_Thread \*thread)

调用：

get\_state()

Thread.hxx (packages\kernel\current\include):inline cyg\_uint32 Cyg\_Thread::get\_state()

1. void Cyg\_Scheduler\_Implementation::set\_idle\_thread

( Cyg\_Thread \*thread, HAL\_SMP\_CPU\_TYPE cpu )

调用：

resume()

Void Cyg\_Thread::resume()

1. void Cyg\_Scheduler\_Implementation::timeslice(void)

调用：

timeslice\_cpu();

Void Cyg\_Scheduler\_Implementation::timeslice\_cpu(void)

1. Void Cyg\_Scheduler\_Implementation::timeslice\_cpu(void)

调用：

get\_current\_thread()

Sched.hxx (packages\kernel\current\include):

inline Cyg\_Thread \*Cyg\_Scheduler\_Base::get\_current\_thread()

get\_state()

Thread.hxx (packages\kernel\current\include):inline cyg\_uint32 Cyg\_Thread::get\_state()

rotate()

class Cyg\_CList：void rotate()

set\_need\_reschedule(thread);

Sched.hxx (packages\kernel\current\include):

inline void Cyg\_Scheduler\_Base::set\_need\_reschedule() Cyg\_Scheduler\_Base的静态内联函数

1. \_\_externC void cyg\_scheduler\_timeslice\_cpu(void)

调用：

Cyg\_Scheduler::scheduler.timeslice\_cpu();

1. Cyg\_SchedThread\_Implementation::Cyg\_SchedThread\_Implementation

(

CYG\_ADDRWORD sched\_info

}

1. Void Cyg\_SchedThread\_Implementation::yield(void)

调用：

Cyg\_Scheduler::lock()

get\_state()

Thread.hxx (packages\kernel\current\include):inline cyg\_uint32 Cyg\_Thread::get\_state()

get\_next()

to\_head( thread->get\_next()

get\_head()

rotate()

timeslice\_reset()

Mlqueue.hxx (packages\kernel\current\include)

inline void Cyg\_SchedThread\_Implementation::timeslice\_reset()

Cyg\_Scheduler::get\_need\_reschedule()

Cyg\_Scheduler::unlock\_reschedule()

1. Void Cyg\_SchedThread\_Implementation::rotate\_queue( cyg\_priority pri )

Cyg\_Scheduler::lock()

empty()

rotate()

set\_need\_reschedule()

Cyg\_Scheduler::unlock()

1. Void Cyg\_SchedThread\_Implementation::to\_queue\_head( void )

Cyg\_Scheduler::lock()

get\_current\_queue()

to\_head( thread )

in\_list()

Clist.hxx (packages\infra\current\include): cyg\_bool in\_list() { return next != this; };

set\_need\_reschedule( thread )

Cyg\_Scheduler::unlock()

1. Void Cyg\_ThreadQueue\_Implementation::enqueue(Cyg\_Thread \*thread)

get\_head()

add\_tail( thread )

Clist.hxx (packages\infra\current\include):

class Cyg\_CList：void add\_tail( Cyg\_DNode \*node )

get\_next()

insert( thread )

Clist.hxx (packages\infra\current\include):

class Cyg\_DNode：void insert( Cyg\_DNode \*node )

get\_tail()

get\_prev(

Clist.hxx (packages\infra\current\include):

class Cyg\_DNode ：Cyg\_DNode \*get\_prev() { return prev; };

append( thread )

class Cyg\_DNode：void append( Cyg\_DNode \*node )

1. Cyg\_Thread \* Cyg\_ThreadQueue\_Implementation::dequeue(void)

rem\_head()

1. Void Cyg\_ThreadQueue\_Implementation::remove( Cyg\_Thread \*thread )

Cyg\_CList\_T<Cyg\_Thread>::remove( thread )

1. Cyg\_Thread \* Cyg\_ThreadQueue\_Implementation::highpri(void)

get\_head()

总结:

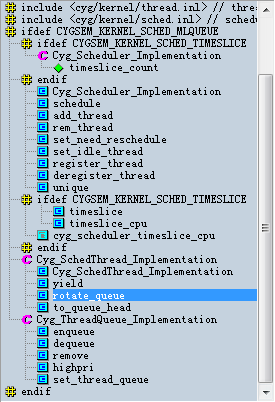
如下图所示，mlqueue.cxx文件中主要涉及到三个类的函数实现

即：

Cyg\_Scheduler\_Implementation

Cyg\_SchedThread\_Implementation

Cyg\_ThreadQueue\_Implementation



而这其中的函数调用到了如下类中的函数：

Cyg\_Scheduler\_Base：

get\_current\_thread()

set\_need\_reschedule()

set\_need\_reschedule(Cyg\_Thread \*thread)

Cyg\_Scheduler:

scheduler.timeslice\_cpu();

lock()

unlock()

unlock\_reschedule()

Cyg\_Thread:

get\_state()

resume()

除此之外还涉及到了Cyg\_DNode和Cyg\_CList两个类中提供的对节点和链表的操作函数，因为跟内核的核心函数关系不大，所以不再赘述

那么最后可以一张图的形式对此进行描述