[FreeBSD taskqgroup 1](#_Toc559849018)

[qgroup\_softirq 1](#_Toc352517282)

[struct taskqgroup 1](#_Toc1636870899)

[struct grouptask 1](#_Toc1156847399)

[Taskqgroup APIs 2](#_Toc877986924)

[taskqgroup\_create() 2](#_Toc335852051)

[taskqgroup\_attach\_cpu() 2](#_Toc1972520383)

[taskqgroup\_detach() 3](#_Toc390557379)

[taskqgroup\_destroy() 3](#_Toc924079999)

[GROUPTASK\_INIT() 4](#_Toc1665887838)

[GROUPTASK\_ENQUEUE() 4](#_Toc574181986)

[Sample Program – softirq taskqgroup 5](#_Toc2092244523)

[In the sample program, task is attached to existing group task queue: qgroup\_softirq 5](#_Toc20775441)

[Sample Program – new taskqgroup 5](#_Toc704474876)

[linuxkpi : tasklet to taskqgroup Mapping 5](#_Toc578699761)

[struct tasklet\_struct 5](#_Toc76111603)

[struct tasklet\_worker 6](#_Toc1447715641)

[tasklet\_subsystem\_init() 6](#_Toc886600981)

[tasklet\_handler() 6](#_Toc748793466)

[tasklet\_subsystem\_uninit() 7](#_Toc2016988804)

[tasklet\_setup() 7](#_Toc2146940695)

[tasklet\_init() 8](#_Toc872523314)

[tasklet\_schedule() 8](#_Toc1471469784)

[tasklet\_kill() 9](#_Toc158867247)

[Sample Program – Tasklet 9](#_Toc2026369319)

[In the sample program, task is attached to existing group task queue: qgroup\_softirq 9](#_Toc241134976)

# FreeBSD taskqgroup

* Bottom half mechanism to manage group of tasks that can be executed concurrently across multiple CPUs
* Each taskqgroup can manage multiple tasks, which can be scheduled for execution.
* Task groups can be assigned to specific CPUs, enabling better load balancing and performance optimization.
* utilized for handling softirqs

### qgroup\_softirq

- taskqueue group specifically designed to handle software interrupts (softirqs)

- Utilized for managing tasks that need to be executed in the context of soft interrupts, ensuring that these tasks can be executed without blocking other critical kernel functions.

### struct taskqgroup

File: sys/kern/subr\_gtaskqueue.c

|  |
| --- |
| struct taskqgroup {  struct taskqgroup\_cpu tqg\_queue[MAXCPU];  struct mtx tqg\_lock;  const char \* tqg\_name;  int tqg\_cnt;  }; |

### struct grouptask

* Represents a task within a taskqgroup

File: sys/sys/gtaskqueue.h

|  |
| --- |
| struct grouptask {  struct gtask gt\_task;  void \*gt\_taskqueue;  LIST\_ENTRY(grouptask) gt\_list;  void \*gt\_uniq;  #define GROUPTASK\_NAMELEN 32  char gt\_name[GROUPTASK\_NAMELEN];  device\_t gt\_dev;  struct resource \*gt\_irq;  int gt\_cpu;  }; |

**gt\_task** – represents task itself

**gt\_taskqueue** – pointer to associated taskqueue

**gt\_cpu** – cpu affinity indicating the preferred CPU on which this task should execute

# Taskqgroup APIs

### taskqgroup\_create()

* Creates a task queue group

**Prototype:**

|  |
| --- |
| struct taskqgroup \*taskqgroup\_create(const char \*name, int cnt, int stride); |

**Parameters:**

**name**: name of the task group

**cnt**:no of threads to be used in the group

**stride**: configuration option related to scheduling and execution policies

**Return Value:**

On Success, returns pointer to struct taskqgroup

On Failure, returns NULL

### taskqgroup\_attach\_cpu()

* Attaches specific grouptask to a particular CPU within a taskqgroup

**Prototype:**

|  |
| --- |
| int taskqgroup\_attach\_cpu(struct taskqgroup \*qgroup,  struct grouptask \*grptask, void \*uniq, int cpu, device\_t dev,  struct resource \*irq, const char \*name); |

**Parameters:**

**qgroup**: pointer to struct taskqgroup representing the taskqgroup to which the task should be attached

**grptask**: pointer to struct grouptask representing the task to be attached

**cpu**: specifies the CPU to which the task should be bound

**name**: name for the task, often used for debugging purpose

### taskqgroup\_detach()

* Remove a task from task queue group
* Drains the pending task (ensures tasks pending in the queue are finished) before removing from the taskqgroup

**Prototype:**

|  |
| --- |
| void taskqgroup\_detach(struct taskqgroup \*qgroup, struct grouptask \*gtask); |

**Parameters:**

**qgroup**: pointer to struct taskqgroup representing the taskqgroup from which the task should be detached

**gtask**: pointer to struct grouptask representing the task to be detached

### taskqgroup\_destroy()

* Used to release resources associated with a taskqgroup when it is no longer needed

**Prototype:**

|  |
| --- |
| void taskqgroup\_destroy(struct taskqgroup \*qgroup); |

**Parameters:**

**qgroup**: pointer to struct taskqgroup which needs to be freed

### GROUPTASK\_INIT()

* Initialize a grouptask structure, which sets up a task for execution within a taskqgroup.

**Definition:**

|  |
| --- |
| #define GROUPTASK\_INIT(gtask, priority, func, context) \  GTASK\_INIT(&(gtask)->gt\_task, 0, priority, func, context) |
| #define GTASK\_INIT(gtask, flags, priority, func, context) do { \  (gtask)->ta\_flags = flags; \  (gtask)->ta\_priority = (priority); \  (gtask)->ta\_func = (func); \  (gtask)->ta\_context = (context); \  } while (0) |

**Parameters:**

**gtask**: pointer to struct grouptask that represents the task being initialized

**func**: function pointer to the task handler that will execute when the task is scheduled

**context**: context pointer that will be passed to the task function.

### GROUPTASK\_ENQUEUE()

* Schedule or enqueue a previously initialized grouptask for execution

**Definition:**

|  |
| --- |
| #define GROUPTASK\_ENQUEUE(gtask) \  grouptaskqueue\_enqueue((gtask)->gt\_taskqueue, &(gtask)->gt\_task) |

**Parameters:**

**gtask**: pointer to struct grouptask that represents the task to be enqueued.

# Sample Program – softirq taskqgroup

Refer to sample\_softirq\_group/bsd\_gtaskq\_softirq.c

### In the sample program, task is attached to existing group task queue: qgroup\_softirq

Output file: gtaskq\_softirq\_output.txt

# Sample Program – new taskqgroup

Refer to sample\_new\_group/bsd\_new\_gtaskq.c

In the sample program, new taskqgroup “my\_taskqgroup” is created and the task is attached to “my\_taskqgroup”.

Output file: gtaskq\_new\_output.txt

From the proctstat output, new thread “my\_taskqgroup\_0” is created.

Note: After module unload, “my\_taskqgroup\_0” is not destroyed as the “taskqgroup\_destroy()” API is not implemented in linuxkpi FreeBSD-14.1

# linuxkpi : tasklet to taskqgroup Mapping

### struct tasklet\_struct

defines tasklet, function to execute and associated data

**File**: sys/compat/linuxkpi/common/include/linux/interrupt.h

|  |
| --- |
| truct tasklet\_struct {  TAILQ\_ENTRY(tasklet\_struct) entry;  tasklet\_func\_t \*func;  /\* Our "state" implementation is different. Avoid same name as Linux. \*/  volatile u\_int tasklet\_state;  atomic\_t count;  unsigned long data;  tasklet\_callback\_t \*callback;  bool use\_callback;  }; |

### struct tasklet\_worker

responsible for handling execution of tasklets, typically by leveraging FreeBSD's taskqueue/gtaskqueue to enqueue and run deferred tasks at a low priority

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| struct tasklet\_worker {  struct mtx mtx;  TAILQ\_HEAD(tasklet\_list, tasklet\_struct) head;  struct grouptask gtask;  } \_\_aligned(CACHE\_LINE\_SIZE); |

### tasklet\_subsystem\_init()

Initialize the tasklet subsystem, providing a way to mimic Linux’s tasklet handling in FreeBSD.

In FreeBSD, you need to explicitly initialize the tasklet subsystem before using it. This initialization is done in ‘tasklet\_subsystem\_init()’

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| static void  tasklet\_subsystem\_init(void \*arg \_\_unused)  {  ...  TAILQ\_INIT(&tw->head);  GROUPTASK\_INIT(&tw->gtask, 0, tasklet\_handler, tw);  ...  taskqgroup\_attach\_cpu(qgroup\_softirq, &tw->gtask,  "tasklet", i, NULL, NULL, buf);  }  SYSINIT(linux\_tasklet, SI\_SUB\_TASKQ, SI\_ORDER\_THIRD, tasklet\_subsystem\_init, NULL); |

### tasklet\_handler()

Function that handles deferrewd work. Executed when a tasklet is scheduled to run.

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| static void  tasklet\_handler(void \*arg)  {  ...  if (ts->use\_callback)  ts->callback(ts);  else  ts->func(ts->data);  ...  } |

### tasklet\_subsystem\_uninit()

Deinitialize the tasklet subsystem to clean up and release any resources associated with tasklet.

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| static void  tasklet\_subsystem\_uninit(void \*arg \_\_unused)  {  ...  taskqgroup\_drain\_all(qgroup\_softirq);  ...  taskqgroup\_detach(qgroup\_softirq, &tw->gtask);  ...  }  SYSUNINIT(linux\_tasklet, SI\_SUB\_TASKQ, SI\_ORDER\_THIRD, tasklet\_subsystem\_uninit, NULL); |

### tasklet\_setup()

Initialize tasklet structure.

Can be both statically and dynamically allocated

Used when tasklet\_struct may be dynamically allocated.

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| void  tasklet\_setup(struct tasklet\_struct \*ts, tasklet\_callback\_t \*c)  {  ...  ts->func = NULL;  ts->callback = c;  ts->data = 0;  ...  ts->use\_callback = true;  } |

### tasklet\_init()

Initialize tasklet with a handler function and data.

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| void  tasklet\_init(struct tasklet\_struct \*ts,  tasklet\_func\_t \*func, unsigned long data)  {  ...  ts->func = func;  ts->callback = NULL;  ts->data = data;  ...  ts->use\_callback = false;  } |

### tasklet\_schedule()

Schedule a previously initialized and configured tasklet for execution

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| void  tasklet\_schedule(struct tasklet\_struct \*ts)  {  ...  struct tasklet\_worker \*tw;  /\* enqueue tasklet \*/  TAILQ\_INSERT\_TAIL(&tw->head, ts, entry);  /\* schedule worker \*/  GROUPTASK\_ENQUEUE(&tw->gtask);  ...  } |

### tasklet\_kill()

Stop tasklet that has been scheduled or in process of execution.

**File**: sys/compat/linuxkpi/common/src/linux\_tasklet.c

|  |
| --- |
| void  tasklet\_kill(struct tasklet\_struct \*ts)  {  ...  /\* wait until tasklet is no longer busy \*/  while (TASKLET\_ST\_GET(ts) != TASKLET\_ST\_IDLE)  pause("W", 1);  } |

# Sample Program – Tasklet

Refer to sample\_lkpi\_tasklet/ linuxkpi\_tasklet.c

### In the sample program, task is attached to existing group task queue: qgroup\_softirq

Output: linuxkpi\_tasklet\_output.txt