*\* kernel/workqueue.c - generic async execution with shared worker pool*

*\**

*/\**

*\* The externally visible workqueue. It relays the issued work items to*

*\* the appropriate worker\_pool through its pool\_workqueues.*

*\*/*

struct [**workqueue\_struct**](https://elixir.bootlin.com/linux/v4.8/C/ident/workqueue_struct) {

struct [**list\_head**](https://elixir.bootlin.com/linux/v4.8/C/ident/list_head) [**pwqs**](https://elixir.bootlin.com/linux/v4.8/C/ident/pwqs); */\* WR: all pwqs of this wq \*/*

struct [**list\_head**](https://elixir.bootlin.com/linux/v4.8/C/ident/list_head) list; */\* PR: list of all workqueues \*/*

struct [**mutex**](https://elixir.bootlin.com/linux/v4.8/C/ident/mutex) [**mutex**](https://elixir.bootlin.com/linux/v4.8/C/ident/mutex); */\* protects this wq \*/*

int [**work\_color**](https://elixir.bootlin.com/linux/v4.8/C/ident/work_color); */\* WQ: current work color \*/*

int [**flush\_color**](https://elixir.bootlin.com/linux/v4.8/C/ident/flush_color); */\* WQ: current flush color \*/*

[**atomic\_t**](https://elixir.bootlin.com/linux/v4.8/C/ident/atomic_t) [**nr\_pwqs\_to\_flush**](https://elixir.bootlin.com/linux/v4.8/C/ident/nr_pwqs_to_flush); */\* flush in progress \*/*

struct [**wq\_flusher**](https://elixir.bootlin.com/linux/v4.8/C/ident/wq_flusher) \***[first\_flusher](https://elixir.bootlin.com/linux/v4.8/C/ident/first_flusher)**; */\* WQ: first flusher \*/*

struct [**list\_head**](https://elixir.bootlin.com/linux/v4.8/C/ident/list_head) [**flusher\_queue**](https://elixir.bootlin.com/linux/v4.8/C/ident/flusher_queue); */\* WQ: flush waiters \*/*

struct [**list\_head**](https://elixir.bootlin.com/linux/v4.8/C/ident/list_head) [**flusher\_overflow**](https://elixir.bootlin.com/linux/v4.8/C/ident/flusher_overflow); */\* WQ: flush overflow list \*/*

struct [**list\_head**](https://elixir.bootlin.com/linux/v4.8/C/ident/list_head) [**maydays**](https://elixir.bootlin.com/linux/v4.8/C/ident/maydays); */\* MD: pwqs requesting rescue \*/*

struct [**worker**](https://elixir.bootlin.com/linux/v4.8/C/ident/worker) \*[**rescuer**](https://elixir.bootlin.com/linux/v4.8/C/ident/rescuer); */\* I: rescue worker \*/*

int [**nr\_drainers**](https://elixir.bootlin.com/linux/v4.8/C/ident/nr_drainers); */\* WQ: drain in progress \*/*

int [**saved\_max\_active**](https://elixir.bootlin.com/linux/v4.8/C/ident/saved_max_active); */\* WQ: saved pwq max\_active \*/*

struct [**workqueue\_attrs**](https://elixir.bootlin.com/linux/v4.8/C/ident/workqueue_attrs) \***[unbound\_attrs](https://elixir.bootlin.com/linux/v4.8/C/ident/unbound_attrs)**; */\* PW: only for unbound wqs \*/*

struct [**pool\_workqueue**](https://elixir.bootlin.com/linux/v4.8/C/ident/pool_workqueue) \***[dfl\_pwq](https://elixir.bootlin.com/linux/v4.8/C/ident/dfl_pwq)**; */\* PW: only for unbound wqs \*/*

#ifdef [**CONFIG\_SYSFS**](https://elixir.bootlin.com/linux/v4.8/K/ident/CONFIG_SYSFS)

struct [**wq\_device**](https://elixir.bootlin.com/linux/v4.8/C/ident/wq_device) \***[wq\_dev](https://elixir.bootlin.com/linux/v4.8/C/ident/wq_dev)**; */\* I: for sysfs interface \*/*

#endif

#ifdef [**CONFIG\_LOCKDEP**](https://elixir.bootlin.com/linux/v4.8/K/ident/CONFIG_LOCKDEP)

struct [**lockdep\_map**](https://elixir.bootlin.com/linux/v4.8/C/ident/lockdep_map) [**lockdep\_map**](https://elixir.bootlin.com/linux/v4.8/C/ident/lockdep_map);

#endif

char name[[**WQ\_NAME\_LEN**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_NAME_LEN)]; */\* I: workqueue name \*/*

*/\**

*\* Destruction of workqueue\_struct is sched-RCU protected to allow*

*\* walking the workqueues list without grabbing wq\_pool\_mutex.*

*\* This is used to dump all workqueues from sysrq.*

*\*/*

struct [**rcu\_head**](https://elixir.bootlin.com/linux/v4.8/C/ident/rcu_head) [**rcu**](https://elixir.bootlin.com/linux/v4.8/C/ident/rcu);

*/\* hot fields used during command issue, aligned to cacheline \*/*

unsigned int flags [**\_\_\_\_cacheline\_aligned**](https://elixir.bootlin.com/linux/v4.8/C/ident/____cacheline_aligned); */\* WQ: WQ\_\* flags \*/*

struct [**pool\_workqueue**](https://elixir.bootlin.com/linux/v4.8/C/ident/pool_workqueue) [**\_\_percpu**](https://elixir.bootlin.com/linux/v4.8/C/ident/__percpu) \***[cpu\_pwqs](https://elixir.bootlin.com/linux/v4.8/C/ident/cpu_pwqs)**; */\* I: per-cpu pwqs \*/*

struct [**pool\_workqueue**](https://elixir.bootlin.com/linux/v4.8/C/ident/pool_workqueue) [**\_\_rcu**](https://elixir.bootlin.com/linux/v4.8/C/ident/__rcu) \***[numa\_pwq\_tbl](https://elixir.bootlin.com/linux/v4.8/C/ident/numa_pwq_tbl)**[]; */\* PWR: unbound pwqs indexed by node \*/*

};

struct [**work\_struct**](https://elixir.bootlin.com/linux/v4.8/C/ident/work_struct) {

[**atomic\_long\_t**](https://elixir.bootlin.com/linux/v4.8/C/ident/atomic_long_t) data;

struct [**list\_head**](https://elixir.bootlin.com/linux/v4.8/C/ident/list_head) entry;

[**work\_func\_t**](https://elixir.bootlin.com/linux/v4.8/C/ident/work_func_t) [**func**](https://elixir.bootlin.com/linux/v4.8/C/ident/func);

#ifdef [**CONFIG\_LOCKDEP**](https://elixir.bootlin.com/linux/v4.8/K/ident/CONFIG_LOCKDEP)

struct [**lockdep\_map**](https://elixir.bootlin.com/linux/v4.8/C/ident/lockdep_map) [**lockdep\_map**](https://elixir.bootlin.com/linux/v4.8/C/ident/lockdep_map);

#endif

};

typedef void (\***[work\_func\_t](https://elixir.bootlin.com/linux/v4.8/C/ident/work_func_t)**)(struct [**work\_struct**](https://elixir.bootlin.com/linux/v4.8/C/ident/work_struct) \*[**work**](https://elixir.bootlin.com/linux/v4.8/C/ident/work));

*/\**

*\* initialize all of a work item in one go*

*\**

*\* NOTE! No point in using "atomic\_long\_set()": using a direct*

*\* assignment of the work data initializer allows the compiler*

*\* to generate better code.*

*\*/*

#ifdef [**CONFIG\_LOCKDEP**](https://elixir.bootlin.com/linux/v4.8/K/ident/CONFIG_LOCKDEP)

#define [**\_\_INIT\_WORK**](https://elixir.bootlin.com/linux/v4.8/C/ident/__INIT_WORK)(\_work, \_func, \_onstack) \

do { \

static struct [**lock\_class\_key**](https://elixir.bootlin.com/linux/v4.8/C/ident/lock_class_key) \_\_key; \

\

[**\_\_init\_work**](https://elixir.bootlin.com/linux/v4.8/C/ident/__init_work)((\_work), \_onstack); \

(\_work)->data = (**[atomic\_long\_t](https://elixir.bootlin.com/linux/v4.8/C/ident/atomic_long_t)**) [**WORK\_DATA\_INIT**](https://elixir.bootlin.com/linux/v4.8/C/ident/WORK_DATA_INIT)(); \

[**lockdep\_init\_map**](https://elixir.bootlin.com/linux/v4.8/C/ident/lockdep_init_map)(&(\_work)->**[lockdep\_map](https://elixir.bootlin.com/linux/v4.8/C/ident/lockdep_map)**, #\_work, &\_\_key, 0); \

[**INIT\_LIST\_HEAD**](https://elixir.bootlin.com/linux/v4.8/C/ident/INIT_LIST_HEAD)(&(\_work)->entry); \

(\_work)->**[func](https://elixir.bootlin.com/linux/v4.8/C/ident/func)** = (\_func); \

} while (0)

#else

#define [**\_\_INIT\_WORK**](https://elixir.bootlin.com/linux/v4.8/C/ident/__INIT_WORK)(\_work, \_func, \_onstack) \

do { \

[**\_\_init\_work**](https://elixir.bootlin.com/linux/v4.8/C/ident/__init_work)((\_work), \_onstack); \

(\_work)->data = (**[atomic\_long\_t](https://elixir.bootlin.com/linux/v4.8/C/ident/atomic_long_t)**) [**WORK\_DATA\_INIT**](https://elixir.bootlin.com/linux/v4.8/C/ident/WORK_DATA_INIT)(); \

[**INIT\_LIST\_HEAD**](https://elixir.bootlin.com/linux/v4.8/C/ident/INIT_LIST_HEAD)(&(\_work)->entry); \

(\_work)->**[func](https://elixir.bootlin.com/linux/v4.8/C/ident/func)** = (\_func); \

} while (0)

#endif

#define [**INIT\_WORK**](https://elixir.bootlin.com/linux/v4.8/C/ident/INIT_WORK)(\_work, \_func) \

[**\_\_INIT\_WORK**](https://elixir.bootlin.com/linux/v4.8/C/ident/__INIT_WORK)((\_work), (\_func), 0)

*/\**

*\* Workqueue flags and constants. For details, please refer to*

*\* Documentation/workqueue.txt.*

*\*/*

enum {

[**WQ\_UNBOUND**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_UNBOUND) = 1 << 1, */\* not bound to any cpu \*/*

[**WQ\_FREEZABLE**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_FREEZABLE) = 1 << 2, */\* freeze during suspend \*/*

[**WQ\_MEM\_RECLAIM**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_MEM_RECLAIM) = 1 << 3, */\* may be used for memory reclaim \*/*

[**WQ\_HIGHPRI**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_HIGHPRI) = 1 << 4, */\* high priority \*/*

[**WQ\_CPU\_INTENSIVE**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_CPU_INTENSIVE) = 1 << 5, */\* cpu intensive workqueue \*/*

[**WQ\_SYSFS**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_SYSFS) = 1 << 6, */\* visible in sysfs, see wq\_sysfs\_register() \*/*

*/\**

*\* Per-cpu workqueues are generally preferred because they tend to*

*\* show better performance thanks to cache locality. Per-cpu*

*\* workqueues exclude the scheduler from choosing the CPU to*

*\* execute the worker threads, which has an unfortunate side effect*

*\* of increasing power consumption.*

*\**

*\* The scheduler considers a CPU idle if it doesn't have any task*

*\* to execute and tries to keep idle cores idle to conserve power;*

*\* however, for example, a per-cpu work item scheduled from an*

*\* interrupt handler on an idle CPU will force the scheduler to*

*\* excute the work item on that CPU breaking the idleness, which in*

*\* turn may lead to more scheduling choices which are sub-optimal*

*\* in terms of power consumption.*

*\**

*\* Workqueues marked with WQ\_POWER\_EFFICIENT are per-cpu by default*

*\* but become unbound if workqueue.power\_efficient kernel param is*

*\* specified. Per-cpu workqueues which are identified to*

*\* contribute significantly to power-consumption are identified and*

*\* marked with this flag and enabling the power\_efficient mode*

*\* leads to noticeable power saving at the cost of small*

*\* performance disadvantage.*

*\**

*\* http://thread.gmane.org/gmane.linux.kernel/1480396*

*\*/*

\* Рабочие очереди для каждого процессора, как правило, предпочтительнее, поскольку они, как правило,

\* демонстрируют более высокую производительность благодаря локальности кэша. Для каждого процессора

\* рабочие очереди исключают возможность выбора планировщиком процессора для

\* выполнение рабочих потоков, что имеет неприятный побочный эффект

\* увеличения энергопотребления.

\*

\* Планировщик считает, что процессор простаивает, если у него нет какой-либо задачи

\* для выполнения и пытается поддерживать простаивающие ядра в режиме ожидания для экономии энергии;

\* однако, например, рабочий элемент для каждого процессора, запланированный из

\* обработчика прерывания на неработающем процессоре заставит планировщик

\* исключить рабочий элемент на этом процессоре, нарушая бездействие, которое наоборот может привести к большему количеству вариантов планирования, которые являются неоптимальными с точки зрения энергопотребления.

\*

\* Рабочие очереди, помеченные символом WQ\_POWER\_EFFICIENT, по умолчанию предназначены для каждого процессора.

\* но становится несвязанным, если рабочая очередь.параметр ядра power\_efficient равен

\* указано. Рабочие очереди для каждого процессора, которые идентифицируются для

\* значительно увеличивают энергопотребление, идентифицируются и

\* помечаются этим флагом и включают режим power\_efficient.

\* приводит к заметной экономии электроэнергии за счет небольших

\* недостаток производительности.

[**WQ\_POWER\_EFFICIENT**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_POWER_EFFICIENT) = 1 << 7,

[**\_\_WQ\_DRAINING**](https://elixir.bootlin.com/linux/v4.8/C/ident/__WQ_DRAINING) = 1 << 16, */\* internal: workqueue is draining \*/*

[**\_\_WQ\_ORDERED**](https://elixir.bootlin.com/linux/v4.8/C/ident/__WQ_ORDERED) = 1 << 17, */\* internal: workqueue is ordered \*/*

[**\_\_WQ\_LEGACY**](https://elixir.bootlin.com/linux/v4.8/C/ident/__WQ_LEGACY) = 1 << 18, */\* internal: create\*\_workqueue() \*/*

[**WQ\_MAX\_ACTIVE**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_MAX_ACTIVE) = 512, */\* I like 512, better ideas? \*/*

[**WQ\_MAX\_UNBOUND\_PER\_CPU**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_MAX_UNBOUND_PER_CPU) = 4, */\* 4 \* #cpus for unbound wq \*/*

[**WQ\_DFL\_ACTIVE**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_DFL_ACTIVE) = [**WQ\_MAX\_ACTIVE**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_MAX_ACTIVE) / 2,

};

*/\* unbound wq's aren't per-cpu, scale max\_active according to #cpus \*/*

#define [**WQ\_UNBOUND\_MAX\_ACTIVE**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_UNBOUND_MAX_ACTIVE) \

[**max\_t**](https://elixir.bootlin.com/linux/v4.8/C/ident/max_t)(int, [**WQ\_MAX\_ACTIVE**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_MAX_ACTIVE), [**num\_possible\_cpus**](https://elixir.bootlin.com/linux/v4.8/C/ident/num_possible_cpus)() \* [**WQ\_MAX\_UNBOUND\_PER\_CPU**](https://elixir.bootlin.com/linux/v4.8/C/ident/WQ_MAX_UNBOUND_PER_CPU))

*/\**

*\* System-wide workqueues which are always present.*

*\**

*\* system\_wq is the one used by schedule[\_delayed]\_work[\_on]().*

*\* Multi-CPU multi-threaded. There are users which expect relatively*

*\* short queue flush time. Don't queue works which can run for too*

*\* long.*

*\**

*\* system\_highpri\_wq is similar to system\_wq but for work items which*

*\* require WQ\_HIGHPRI.*

*\**

*\* system\_long\_wq is similar to system\_wq but may host long running*

*\* works. Queue flushing might take relatively long.*

*\**

*\* system\_unbound\_wq is unbound workqueue. Workers are not bound to*

*\* any specific CPU, not concurrency managed, and all queued works are*

*\* executed immediately as long as max\_active limit is not reached and*

*\* resources are available.*

*\**

*\* system\_freezable\_wq is equivalent to system\_wq except that it's*

*\* freezable.*

*\**

*\* \*\_power\_efficient\_wq are inclined towards saving power and converted*

*\* into WQ\_UNBOUND variants if 'wq\_power\_efficient' is enabled; otherwise,*

*\* they are same as their non-power-efficient counterparts - e.g.*

*\* system\_power\_efficient\_wq is identical to system\_wq if*

*\* 'wq\_power\_efficient' is disabled. See WQ\_POWER\_EFFICIENT for more info.*

*\*/*

\* Общесистемные рабочие очереди, которые всегда присутствуют.

\* system\_wq - это тот, который используется schedule[\_delayed]\_work[\_on]().

\* Многопроцессорная многопоточность. Есть пользователи, которые ожидают относительно

\* короткое время промывки очереди *queue flush*. Не ставьте в очередь работы, которые могут выполняться слишком долго длинный.

\* system\_highpri\_wq аналогичен system\_wq, но для рабочих элементов, которые

\* требуется WQ\_HIGHPRI.

\* system\_long\_wq похож на system\_wq, но может содержать длительное время

\* работает. Очистка очереди может занять относительно много времени.

\* system\_unbound\_wq - это несвязанная рабочая очередь. Работники не обязаны

\* любой конкретный процессор, не управляемый параллелизмом, и все работы в очереди выполняются

\* выполняется немедленно до тех пор, пока не будет достигнут предел max\_active и

\* ресурсы доступны.

\* system\_freezable\_wq эквивалентен system\_wq, за исключением того, что он

\* возможность замораживания.

\* \*\_power\_efficient\_wq склонны к экономии энергии и преобразованы

\* в варианты WQ\_UNBOUND, если включен параметр 'wq\_power\_efficient'; в противном случае

\* они такие же, как и их неэнергосберегающие аналоги - например

\* system\_power\_efficient\_wq идентичен system\_wq, если

\* 'wq\_power\_efficient' отключен. Дополнительную информацию см. в разделе WQ\_POWER\_EFFICIENT.