Async execution with workqueues

Bhaktipriya Shridhar

About me

$whoami

- Outreachy Intern at the Linux Kernel with Tejun Heo as my mentor.
- Working on updating Legacy workqueue interface users in the Linux Kernel .
- Also, a 3rd year undergraduate student at IIIT Hyderabad, India

Linux

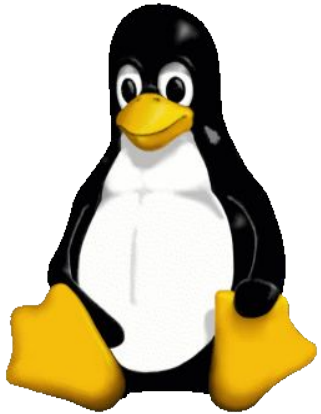
INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Introduction

Workqueues

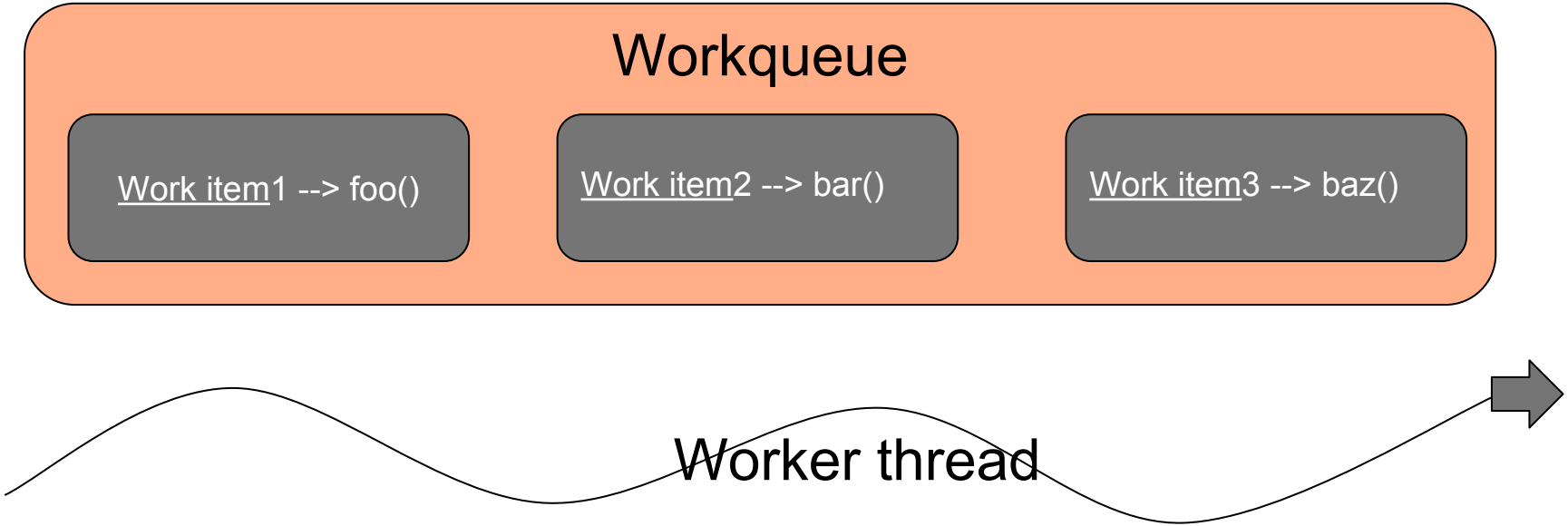
Workqueue is an **asynchronous execution mechanism** which is widely used across the kernel.

It's used for various purposes from simple context bouncing to hosting a persistent in-kernel service thread.



The design

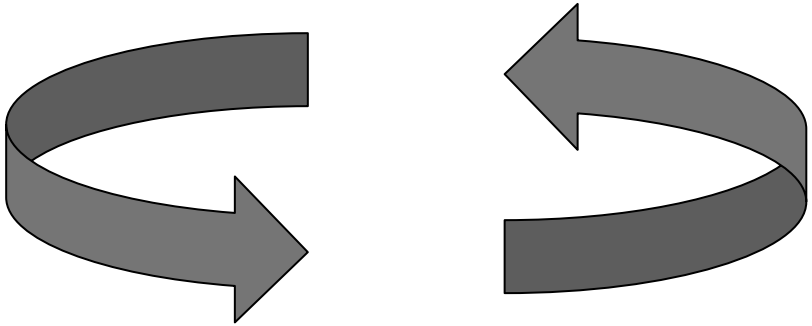
- **Work item** a simple struct that holds a pointer to the function that is to be executed asynchronously.
- **Work queue** a queue of work items
- **Worker threads** Special purpose threads that execute the functions off the queue, one after the other.
- **Workerpools** A thread pool that is used to manage the worker threads


Workqueue
Work item1 --> foo()
Work item2 --> bar()
Work item3 --> baz()
Worker thread

Workqueue EMPTY

Worker thread IDLE

Work item queued



No queued work items

Workqueue QUEUED

Worker thread RUNNING

Presence in the kernel

Past and present...

$grep -r workqueue

Due to its development history, there currently are two sets of interfaces to create workqueues.

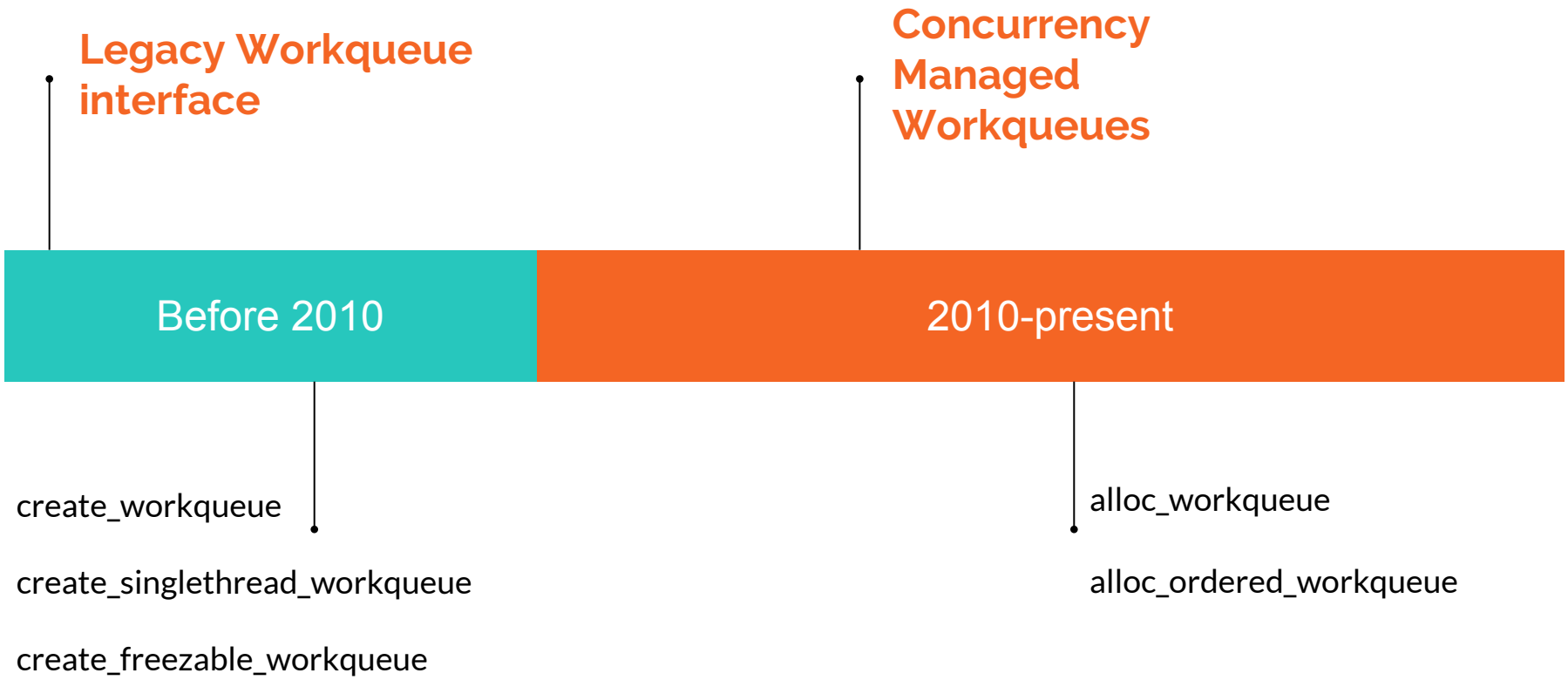
- **Old:** create[_singlethread|_freezable]_workqueue()
- **New:** alloc[_ordered]_workqueue()

Good to know...

Legacy workqueue interface users are scheduled for removal..

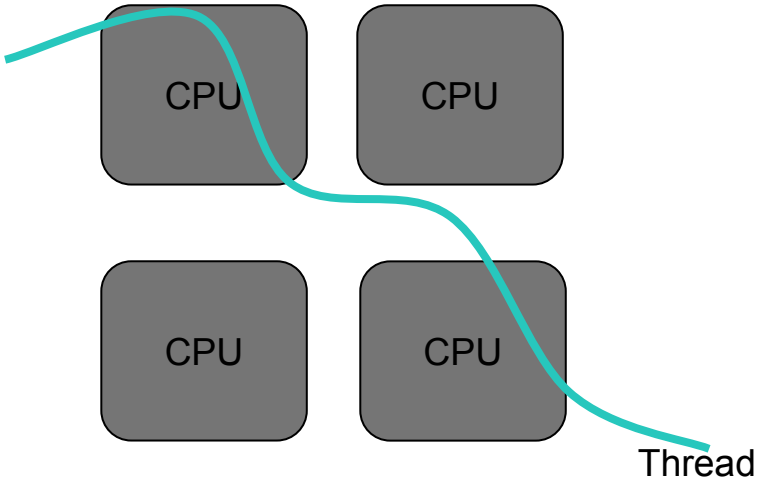
My Outreachy project was to remove 280 legacy workqueue interface users.

History


Legacy Workqueue interface
Concurrency Managed Workqueues
Before 2010
2010-present
create_workqueue
create_singlethread_workqueue
create_freezable_workqueue
alloc_workqueue
alloc_ordered_workqueue

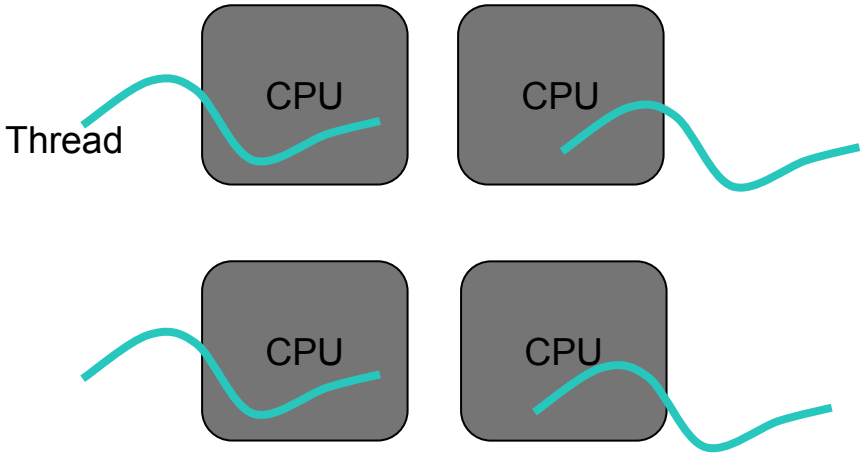
Legacy Workqueue interface

Single threaded workqueue



A single threaded workqueue had one worker thread system-wide.

Multi threaded workqueue



A multi threaded workqueue had one thread per CPU.

Legacy Workqueue interface needed a facelift...

Problems

- **Proliferation of kernel threads**

  The original version of workqueues could, on a large system, run the kernel out of process IDs before user space ever gets a chance to run.

- **Deadlocks** Workqueues could also be subject to deadlocks if locking is not handled very carefully

- **Unnecessary Context switches**

  Workqueue threads contend with each other for the CPU, causing more context switches than are really necessary.

Concurrency Managed Workqueues(CMWQ)- A better solution

Indeed! With CMWQ...

Maintains compatibility with the original workqueue API.

Uses per-CPU unified worker pools shared by all wq to provide flexible level of concurrency on demand without wasting a lot of resource.

Automatically regulates worker pool and level of concurrency so that the API users don't need to worry about such details.

CMWQ : A closer look

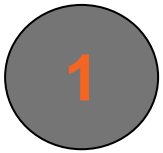
The richer, more expressive and better performing API...

Workqueue API

alloc_workqueue() allocates a wq.

Takes in 3 parameters:

- @name
- @flags
- @max_active



@name

is the name of the wq.

2

@flags

control how work items are assigned execution resources, scheduled and executed.

- WQ_UNBOUND
- WQ_FREEZABLE
- WQ_MEM_RECLAIM
- WQ_HIGHPRI
- WQ_CPU_INTENSIVE

3

@max_active

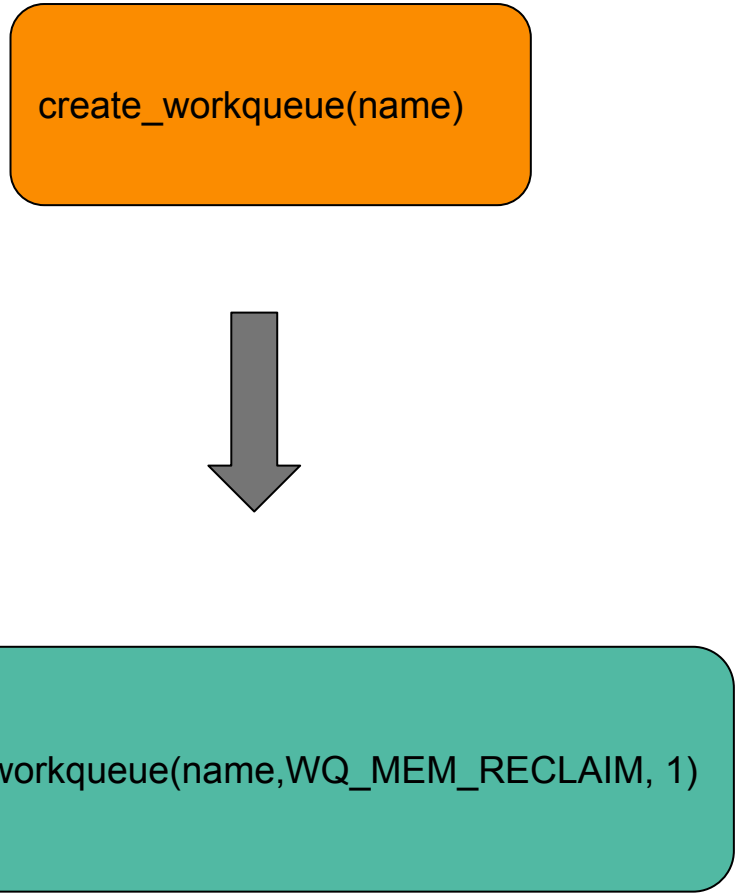
determines the maximum number of execution contexts per CPU which can be assigned to the work items of a wq.

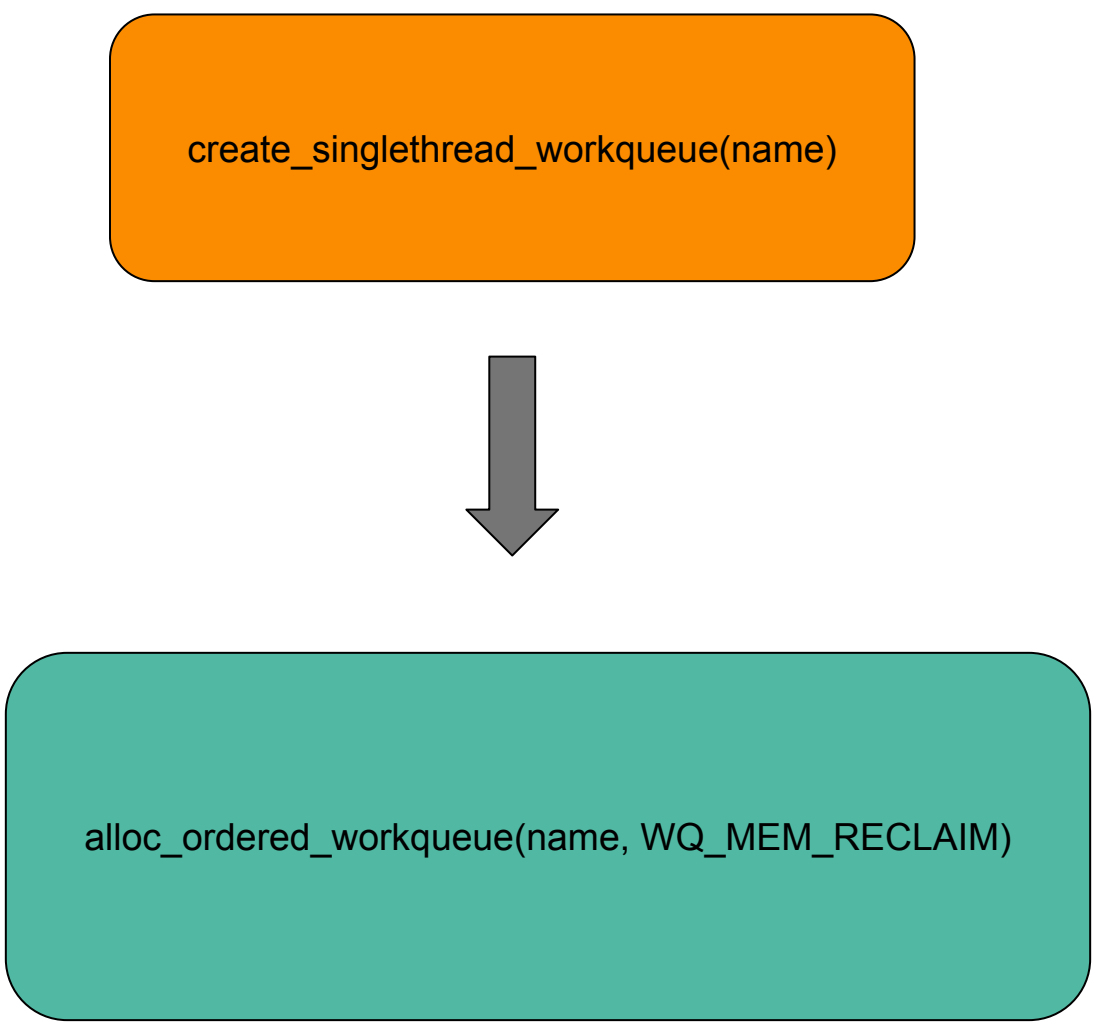
Example

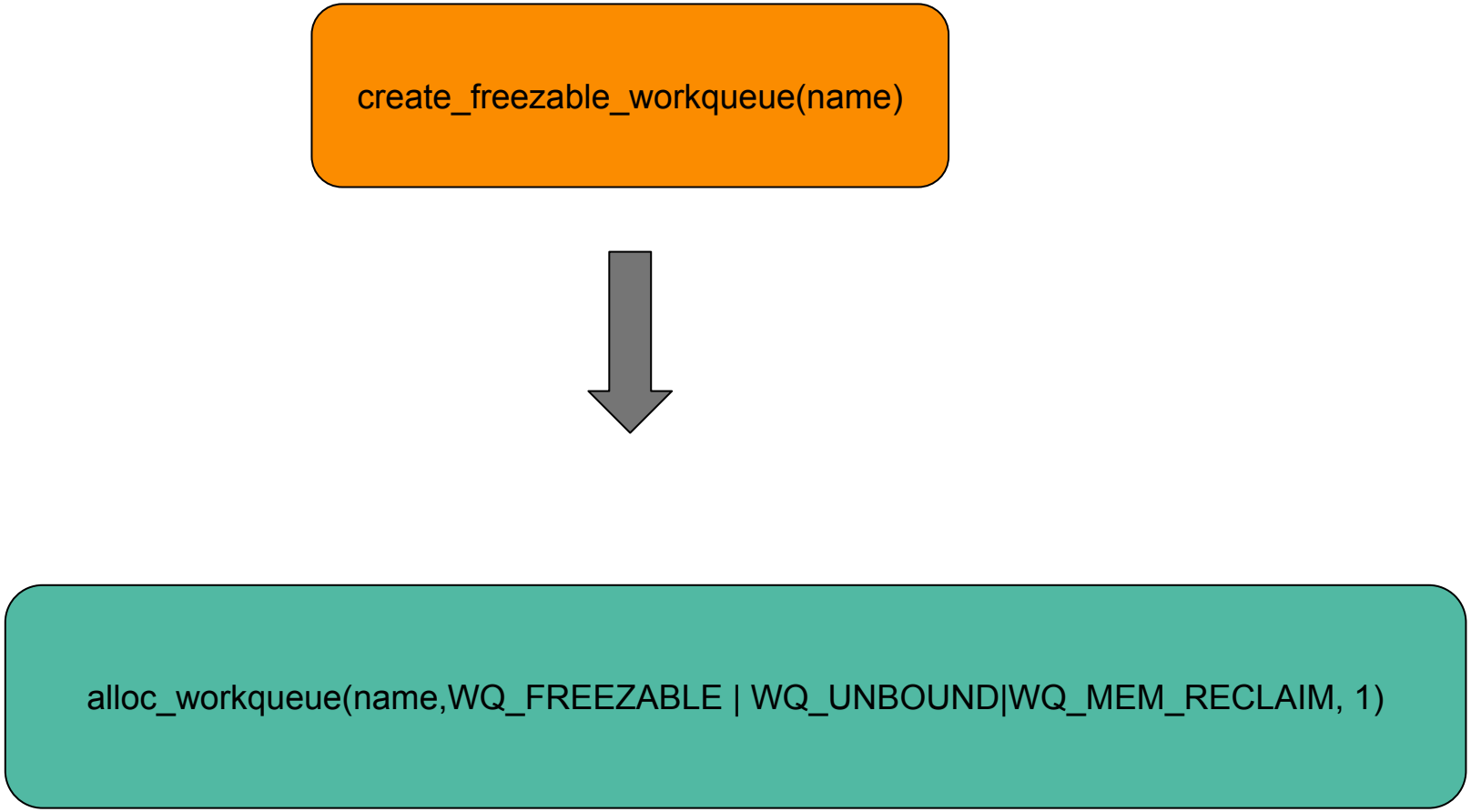
with @max_active of 16, at most 16 work items of the wq can be executing at the same time per CPU.

Mappings

Identity conversions…..


create_workqueue(name)
alloc_workqueue(name,WQ_MEM_RECLAIM, 1)


create_singlethread_workqueue(name)
alloc_ordered_workqueue(name, WQ_MEM_RECLAIM)


create_freezable_workqueue(name)
alloc_workqueue(name,WQ_FREEZABLE | WQ_UNBOUND|WQ_MEM_RECLAIM, 1)

Examples most common workqueue usages

Understanding from the context of the legacy workqueue interface….

alloc_workqueue() (Vanilla)

/drivers/platform/x86/asus-laptop.c

```
-       asus->led_workqueue = create_singlethread_workqueue("led_workqueue");
+        asus->led_workqueue = alloc_workqueue("led_workqueue", 0, 0);
        if (!asus->led_workqueue)
                return -ENOMEM;
```

Tip..

Used when the queued work items can be run concurrently.

No special flags required

- led_workqueue is involved in updating LEDs queues &led->work per asus_led.
- The led_workqueue has **multiple work items which can be run concurrently.**
- The dedicated workqueue is kept so that the work items can be **flushed as a group**.
- Since it is **not being used on a memory reclaim path**, WQ_MEM_RECLAIM has not been set.
- Since there are only a **fixed number of work items**, explicit concurrency limit is unnecessary here.

alloc_workqueue() + WQ_MEM_RECLAIM

/drivers/net/ethernet/synopsys/dwc_eth_qos.c

```
-       lp->txtimeout_handler_wq = create_singlethread_workqueue(DRIVER_NAME);
+       lp->txtimeout_handler_wq = alloc_workqueue(DRIVER_NAME,
+                                                  WQ_MEM_RECLAIM, 0);
```

Tip..

Used when the work items are on a memory reclaim path.

- A dedicated workqueue has been used since the work item viz lp->txtimeout_reinit is involved in **packet TX/RX path** .

- As a network device can be used during memory reclaim, the workqueue **needs forward progress guarantee under memory pressure**. WQ_MEM_RECLAIM has been set to ensure this.

- Since there is **only a single work item**, explicit concurrency limit is unnecessary here.

alloc_workqueue() + WQ_HIGHPRI

/drivers/gpu/drm/radeon/radeon_display.c

```
-       radeon_crtc->flip_queue = create_singlethread_workqueue("radeon-crtc");
+       radeon_crtc->flip_queue = alloc_workqueue("radeon-crtc", WQ_HIGHPRI, 0);
```

Tip..

Used for workqueues that queue work items that require high priority for execution..

Each hardware CRTC has a single flip work queue.
When a radeon_flip_work_func item is queued, it needs to be executed ASAP because **even a slight delay may cause the flip to be delayed by one refresh cycle**.

Hence, a dedicated workqueue with WQ_HIGHPRI set, has been used here since a delay can cause the outcome to miss the refresh cycle.

Since there are only **a fixed number of work items**, explicit concurrency limit is unnecessary here.


MAKE AN API CALL THEY SAID
IT'LL BE EASY THEY SAID
makeameme.org

alloc_ordered_workqueue()

/drivers/net/caif/caif_hsi.c

```
-       cfhsi->wq = create_singlethread_workqueue(cfhsi->ndev->name);
+       cfhsi->wq = alloc_ordered_workqueue(cfhsi->ndev->name, WQ_MEM_RECLAIM);
```

Tip..

Used when the queued work items require strict execution ordering...

An ordered workqueue has been used since workitems &cfhsi->wake_up_work and &cfhsi->wake_down_work **cannot be run concurrently**.

Since the work items are being used on a packet tx/rx path, WQ_MEM_RECLAIM has been set to guarantee forward progress under memory pressure.

System workqueue

/drivers/android/binder.c

```
-       binder_deferred_workqueue = create_singlethread_workqueue("binder");


-       queue_work(binder_deferred_workqueue, &binder_deferred_work);
+       schedule_work(&binder_deferred_work);
```

Tip..

Used when the work items don't take very long and can be run concurrently.

No special flags required..

BEST option in these cases!

- Binder is the RPC mechanism used on androids. The workqueue is being used to run deferred work for the android binder.
- The "binder_deferred_workqueue" queues only a single work item and hence **does not require ordering**.
- Also, this workqueue is **not being used on a memory reclaim path**.
- Hence, it has been converted to use sytem_wq.

System wq with multiple work items

drivers/staging/octeon/ethernet.c

```
-       queue_delayed_work(cvm_oct_poll_queue,
-                          &cvm_oct_rx_refill_work, HZ);
+       schedule_delayed_work(&cvm_oct_rx_refill_work, HZ);

-       queue_delayed_work(cvm_oct_poll_queue,
-                                  &priv->port_periodic_work, HZ);
+      schedule_delayed_work(&priv->port_periodic_work, HZ);

-       cvm_oct_poll_queue = create_singlethread_workqueue("octeon-ethernet");
-       destroy_workqueue(cvm_oct_poll_queue);

+       cancel_delayed_work_sync(&cvm_oct_rx_refill_work);
+       cancel_delayed_work_sync(&priv->port_periodic_work);
```

- cvm_oct_poll_queue was used for polling operations.
- There are multiple work items per cvm_oct_poll_queue (viz. cvm_oct_rx_refill_work, port_periodic_work) and different cvm_oct_poll_queues need not be be ordered. Hence, concurrency can be increased by switching to system_wq.
- All work items are sync canceled so it is guaranteed that no work is in flight by the time exit path runs.
- With concurrency managed workqueues, use of dedicated workqueues can be replaced by system_wq.

system_long_wq

/drivers/gpu/drm/ttm/ttm_memory.c

```
-       glob->swap_queue = create_singlethread_workqueue("ttm_swap");

-       flush_workqueue(glob->swap_queue);
-       destroy_workqueue(glob->swap_queue);


-       queue_work(glob->swap_queue, &glob->work);
+       schedule_work(glob->swap_queue, &glob->work);

+       flush_work(&glob->work);
```

Tip..

Used when the queued work items are long running and don't require any special flags.

- swap_queue was created to handle shrinking in low memory situations.
- Earlier, a separate workqueue was used in order to avoid other workqueue tasks from being blocked since work items on swap_queue spend a lot of time waiting for the GPU.
- Since these **long-running work items aren't involved in memory reclaim** in any way, system_long_wq has been used.
- Work item has been flushed in ttm_mem_global_release() to ensure that nothing is pending when the driver is disconnected.

Summary....

Benefits

CMWQ extends workqueue such that it can serve as robust async mechanism.

- ➔ Less to worry about causing deadlocks around execution resources.
- ➔ Far fewer number of kthreads.
- ➔ More flexibility without runtime overhead.
- ➔ Richer and far more expressive

Many thanks to....

Tejun Heo

Outreachy Team

Organizing Committee, LinuxCon NA 2016


"Aw, I'm going to miss the whole summer."
"Don't worry, boy. When you get a job like me, you'll miss every summer."


Thank you!

Questions?


GO EASY ON ME,
IT'S MY FIRST TIME
memes.com