30.08

(1)

Лекция 30.08

(от себя: smp (symmetrical multi-processing) – архитек-ра многопроцессорных систем, в которых 2 или более пр-сора имеют равный доступ к объекту памяти и устр-вам ввода-вывода. (все пр-соры равноправны)
*: Технологии позволили в чипе создать несколько пр-соров
В чип. ✓ Кэш L_1, L_2, L_3 – принадлежат каждому ядру, а кэш L_3 – общий кэш.

не прям важно:
- Кол-во используемых потоков ядра сократим, но увеличим параллелизм.
- Вот так лучше: кол-во используемых потоков ядра уменьшилось, но необходимая работа выполняется успешно.

- В совр. системах для выполнения очереди м. б. использован существующий рабочий поток (worker)

Сравнение тасклетов и очередей работ:

- Задача у очередей работ та же, что и у тасклетов; очереди работ также относятся к нижней половине, т. е. это возможность представления отложенных действий.
- 1) · Тасклет выполняется в контексте прогр. обеспечения прерывания, в рез. чего код тасклета д. б. неделимым
  · Очереди работ выполняются в контексте спец. потока ядра, в силу чего они явл. более гибкими и, в частности, могут блокироваться
- 2) · Тасклет всегда выполняется на пр-соре, на котором завершилась обработка аппар. прерывания (т. е. верхней половины, top half), который и поставил тасклет в очередь на выполнение
  · По умолч. очередь работ выполняется так же, т. е. на том пр-соре, на котором выполнялся обработчик прерывания, который проинициализировал очередь и поместил работу в неё. Но можно поменять пр-сор.
- 3) · Очереди могут блокироваться
  · Ф-ция обработчик тасклета или очереди работ спец. образом регистрируется. Выполнение этой ф-ции м. б. отложено на некот. время – для этого ∃ спец. ф-ции ядра

30.05 ②

• В более ранних версиях ядра для создания очереди работ надо было вызвать функцию:
(1) create_workqueue()

• В совр системах ↓ заменена на: alloc_workqueue()

При этом если вызвать (1) не д.б. ошибки, т.к. сделано так:

#define create_workqueue(name) alloc_workqueue(fmt, WQ_UNBOUND | WQ_ORDERED|flags, 1, ## args)

для ядра:
#define alloc_workqueue(fmt, flags, max_active, args, ...)

версия 6.3.5:
struct workqueue_struct * alloc_workqueue(const char *fmt, unsigned int flags, int max_active, ...);

print format for name

кол-во задач, кот. могут выполняться ||, находясь в одной очереди работ

В ядре объявлена стр-ра:

эта стр-ра описывает очередь работ

```
struct workqueue_struct {
    struct list_head pwqs;      /* WR: all pwqs of this wq */
    struct list_head list;      /* PR: list of all workqueues */
    ...
    struct wq_flusher *first_flusher;
    struct list_head flusher_queue;
    ...
    char name[WQ_NAME_LEN];
    ...
    struct pool_workqueue __percpu *cpu_pwqs;
    struct pool_workqueue __rcu *numa_pwq_tbl[];
};
```

pool workqueue

flush тут - это "слить на выполнение"

Очереди работ [redacted] - это общий механизм работы ядра по обработке прерываний. Прерывания - основа работы системы. При возникновении соотв-го прерывания, вызывается обработчик прер-я, кот. имеет наз-м „стандартный", и этот обраб-к [redacted] может создать очередь работ, передать работу существующему воркеру и т.д.

Речь идёт о механизме ядра, а для механизма ядра выполнение отложенных действий яв-я основными. (для того чтобы выполнились те работы, для кот. написано ядро)

30.05 · ③

struct work_struct — эта стр-ра представляет работу (work). Экземпляр этой стр-ры помещается в очередь работ.

```
struct work_struct {
    atomic_long_t data;
    struct list_head entry;
    work_func_t func;
#ifdef CONFIG_LOCKDEP
    struct lockdep_map lockdep_map;
#endif
};
```

· work-задача, кот. решает конкр. обработчик нижн. половины

① Проинициализировать/посмотреть работу можно стат-ки и динамически. во время компиляции

Статически: DECLARE_WORK(name, void (*func)(void *)); с пом. макроса

имя стр-ры work_struct

ф-ция, кот. будет выполняться конкр. очередью работ, т.е. это обработчик нижн. половины

Динамически:

1) INIT_work(struct work_struct *work, void (*func)(void *), void *data);

2) PREPARE_WORK(struct work_struct *work, void (*func)(void), void *data);

· Если первый раз инициализируем работу, то нужно использ. 1), т.е. - работает более детально (т.е. до этого не инициализ-ли)

· Если надо ещё раз проинициализ-ть работу, то вызывать 2), чтобы не повторять действия, уже выполненные при инициализации. 2) - сохраняет св-ва данных, связ. с данным соотв. стр-ры

② Поставить работу в очередь: ф-ции идут ∃ соотв-ующие

речь о конкр. cpu

· int queue_work(struct workqueue_struct *wq, struct work_struct *work);

· int queue_work_on(int cpu, struct workqueue_struct *wq, struct work_struct *work);

· int queue_delayed_work(struct workqueue_struct *wq, struct delayed_work *dwork, unsigned long delay);

возможность отложить выполнение запланир-ой работы

видимо тайм спец. поля есть

allxx = разместит 30.05.

Удаление очереди работ, кот. выполнила свою задачу: (1)

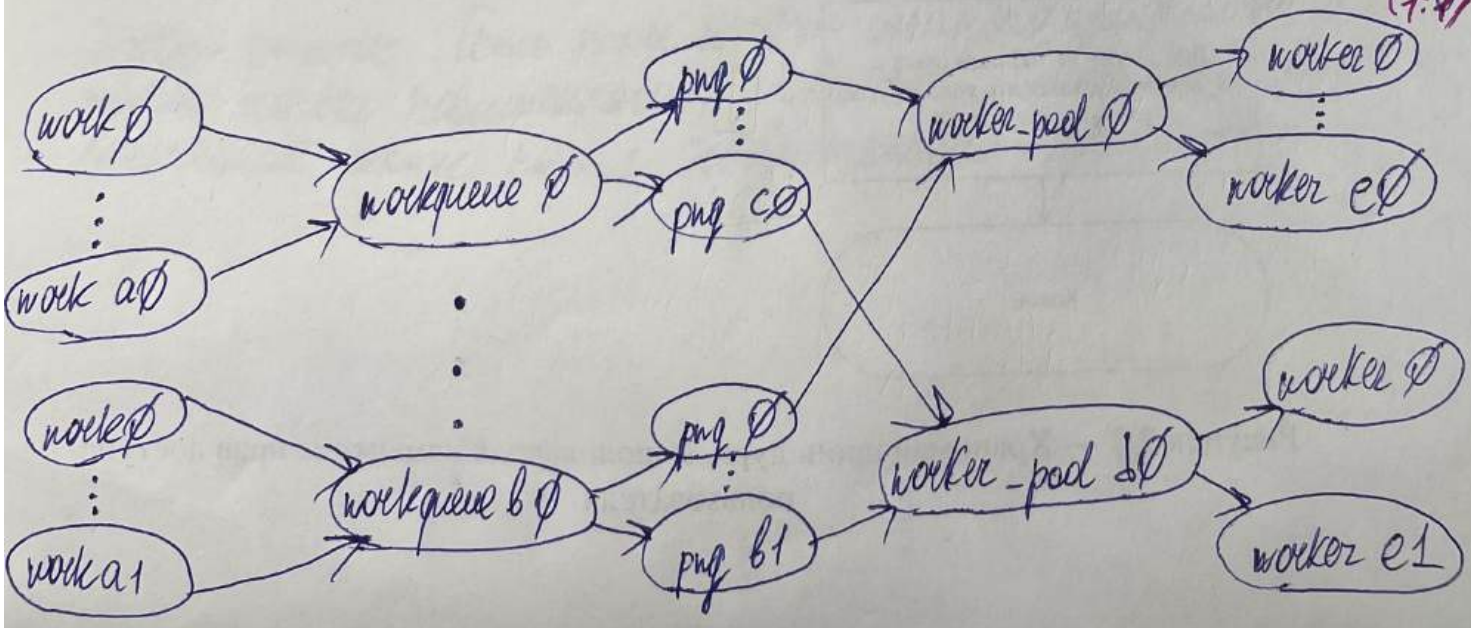
extern void destroy_workqueue(struct workqueue_struct *wq);

Осн. идея, с кот. очереди работ были написаны: одна параллельная множественная работа, в кот. работы не должны блокировать друг друга. При этом механизм строится с тем расчетом, чтобы экономить ресурсы, такие как поток, вовлекаемая память, планирование.

Осн. понятия: CMWQ – Concurrency Managed Workqueue

Концепции CMWQ:

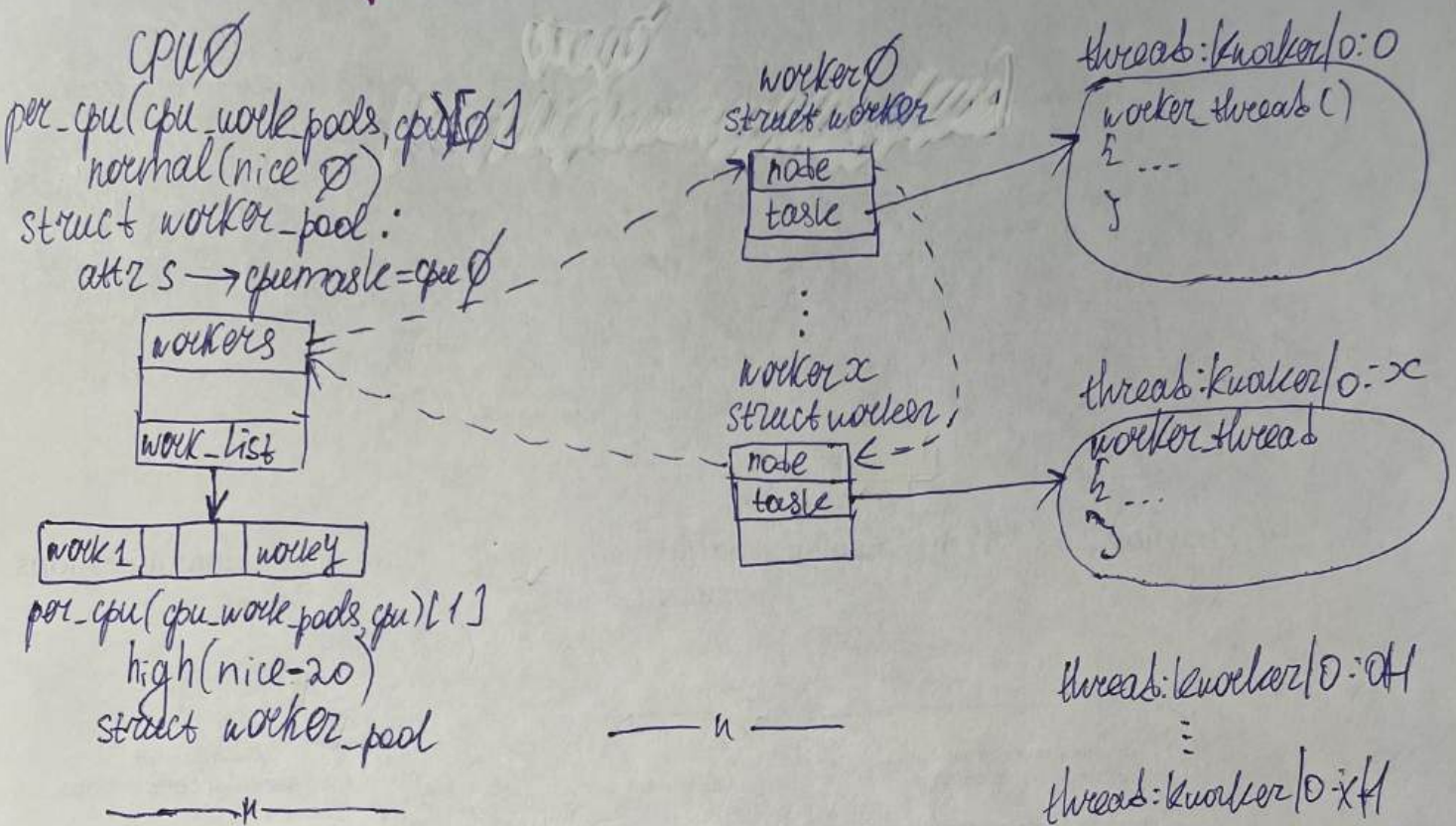
1) Work (работа)
2) Workqueue (очередь работ)
   - Очередь работ и работа представляют отношение один-ко-многим (1:N)
3) Worker
   При выполнении worker соответ-ет потоку ядра work_thread
4) pwq – pool_workqueue – посредник, отвечающий за установку отношений м/у workqueue и worker_pool (1:N)
   - workqueue и pwq → отношение один-ко-многим
   - pwq и worker_pool → отношение один-к-одному (1:1)



30.05

Для одного CPU нарисовано: речь идёт о потоках ядра (S)

workerов per cpu много разных, т.е. они действительно привязаны к пр-ссору, поэтому есть per_cpu (cpu_worker_pools)



— n —

— n —

- На каждый cpu имеются обычные потоки, и более приоритетные (high), т.е. 2 очереди: потоков обычного приоритета и потоков с повышенным приоритетом.
- Worker - это поток. Есть поток воркер обычного приоритета, а есть поток воркер повышенного приоритета. Сначала будут выполняться потоки воркер повышенного приоритета.