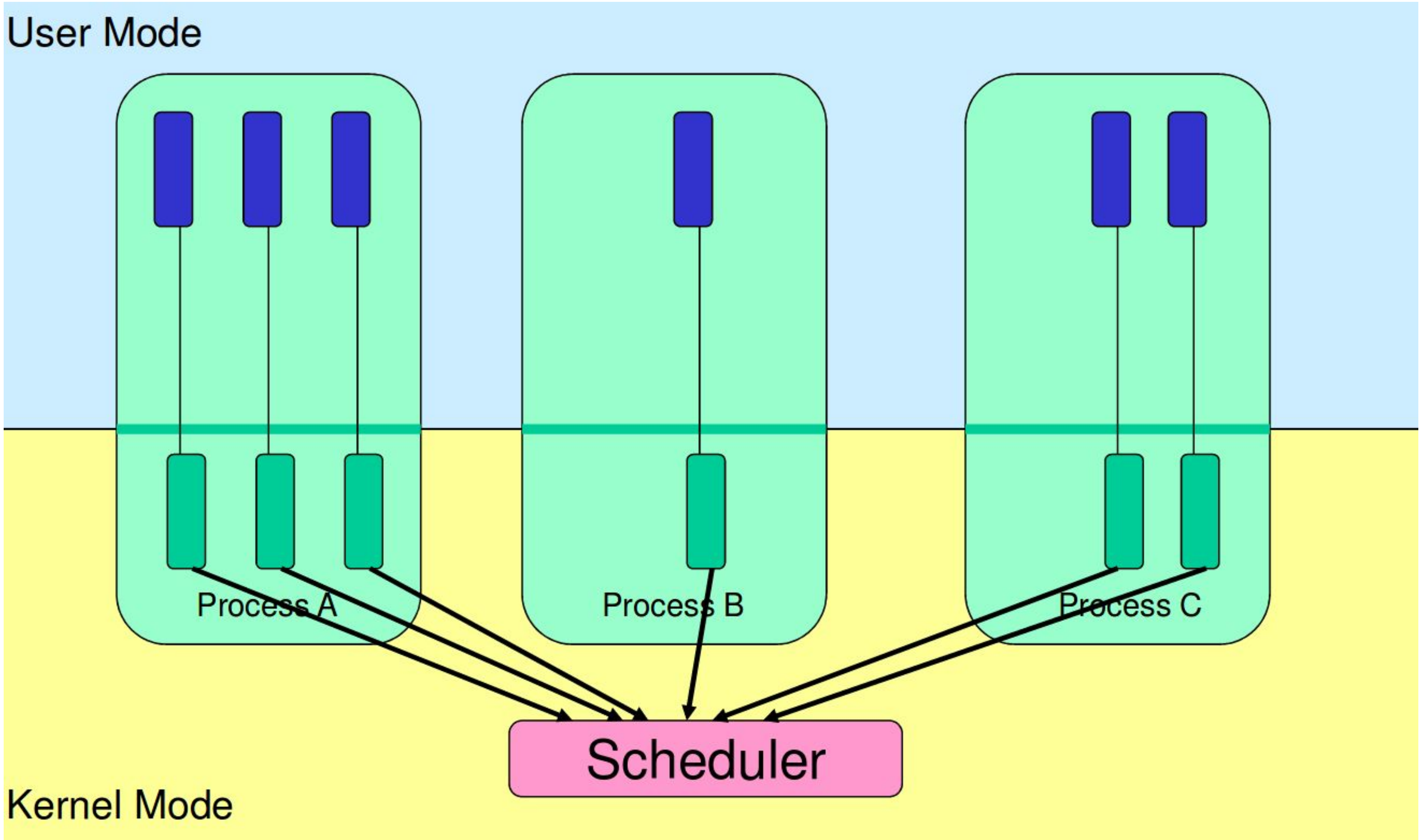
리눅스 스케줄러 기본 구조

국민대학교 임베디드 연구실
경주현

2017 LG MC 인재육성팀 교육

Kernel-level threads



Linux multiplexing

- **Linux multiplexes by two situations.**

Linux multiplexing

- **Linux multiplexes by two situations.**

- **Sleep and wakeup mechanism**
  - process wait for device or pipe I/O

Linux multiplexing

- **Linux multiplexes by two situations.**

- **Sleep and wakeup mechanism**
  - process wait for device or pipe I/O

- **Periodically forces a switch**
  - Preemption

- **Multiplexing creates the illusion**
  - Each process has its own CPU
  - Each process has its own memory

Linux scheduler design philosophy

- **Linux schedules user and kernel threads preemptively.**
- **Every 10ms a timer interrupt -> yield**
  - Except for tick-less status
- **Complete defense against CPU-hogging user programs, even bugs**
- **Helps with responsiveness**
  - Fairness

User-level threads - data structure

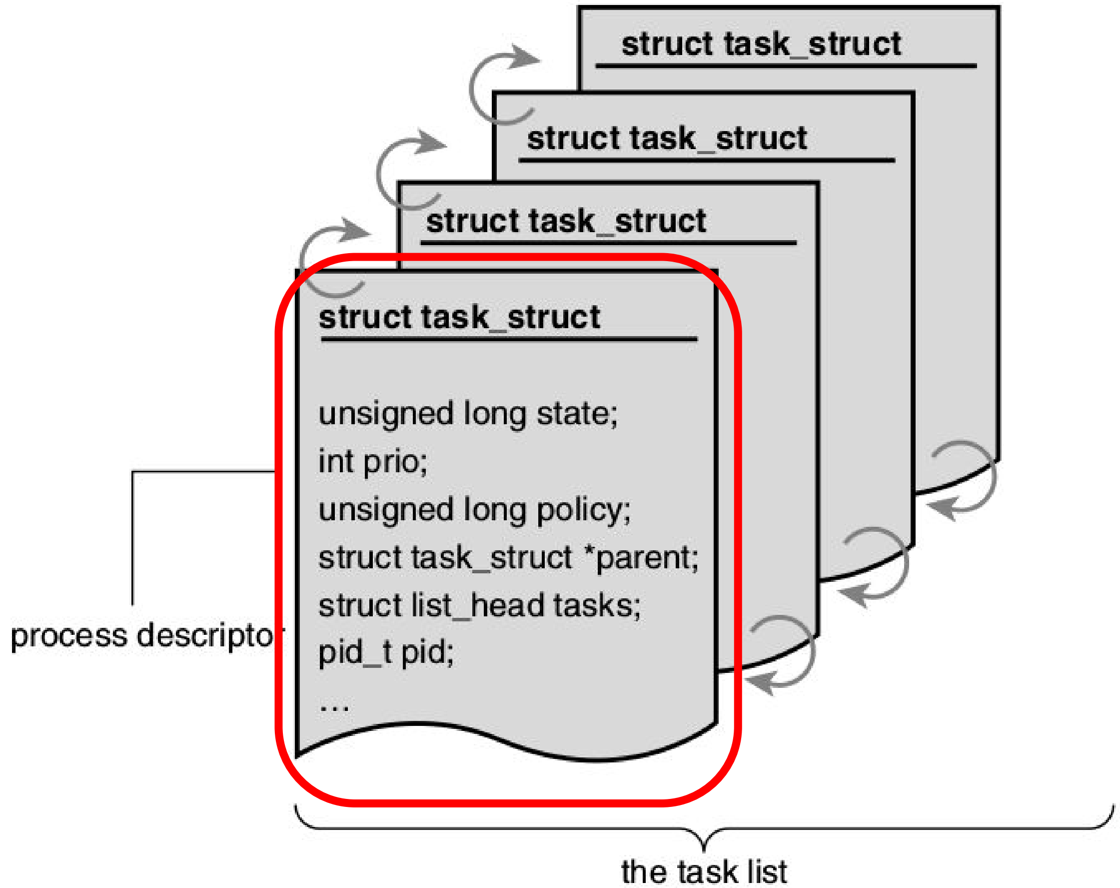
```
/* Possible states of a thread; */
#define FREE        0x0
#define RUNNING     0x1
#define RUNNABLE    0x2

#define STACK_SIZE  8192
#define MAX_THREAD  4

typedef struct thread thread_t, *thread_p;
typedef struct mutex mutex_t, *mutex_p;

struct thread {
  int        sp;                /* curent stack pointer */
  char stack[STACK_SIZE];       /* the thread's stack */
  int        state;             /* FREE, RUNNING, RUNNABLE */
};
static thread_t all_thread[MAX_THREAD];
thread_p  current_thread;
thread_p  next_thread;
extern void thread_switch(void);
```

Linux Process Descriptor



Scheduling entity

- **task_struct is associated with a sched_entity data structure**
- **Scheduling information**
  - load, weight, a group or a single task
- **A single task becomes a scheduling entity on its own.**

```
struct task_struct {
    struct sched_entity se;
    /* ... */
};
```

```
struct sched_entity {
    struct load_weight load;
    struct sched_entity *parent;
    struct cfs_rq *cfs_rq;
    struct cfs_rq *my_rq;
    struct sched_avg avg;
    /* ... */
};
```

Linux Process States

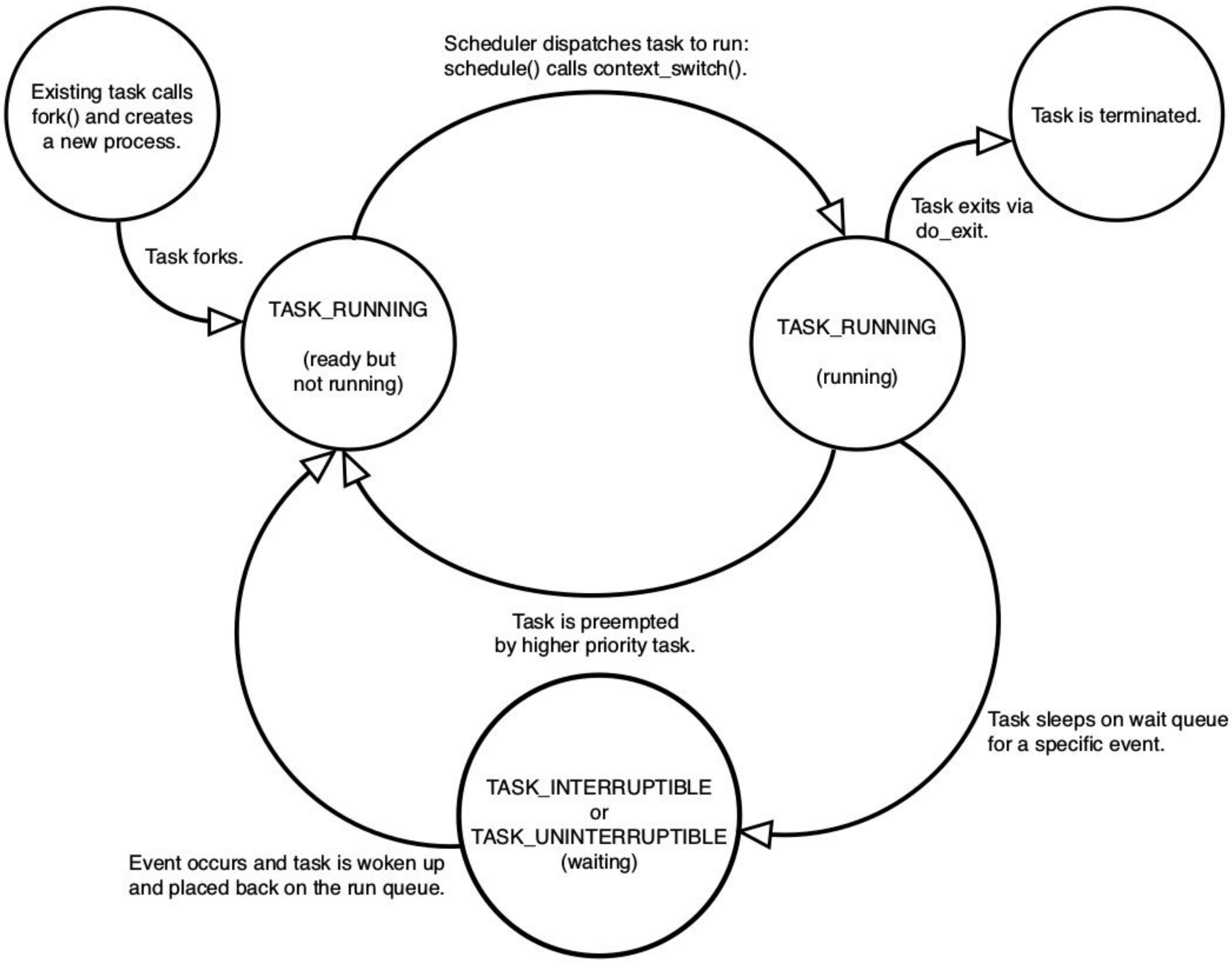


Figure 3.3 Flow chart of process states.

Sleep & Wake up

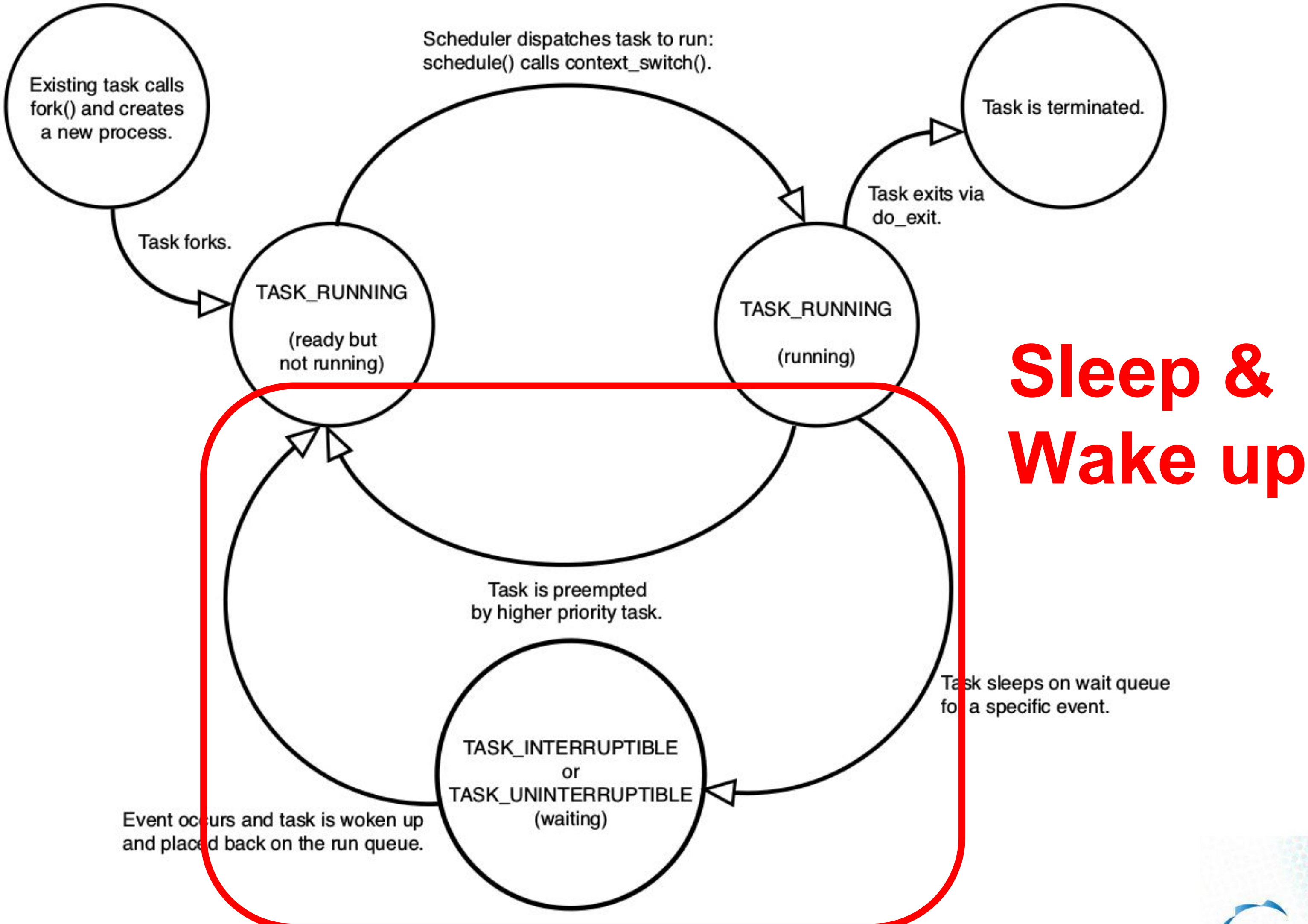


Figure 3.3 Flow chart of process states.

The schedule()

- **Ready-to-run processes are maintained on a run queue.**
- **Once the timeslice of a running process is over**
  - Picks up another appropriate process from the run queue.
- **A process can go to sleep using the schedule() function.**

http://www.linuxjournal.com/article/8144?page=0,2

Ex) Go to sleep by schedule()

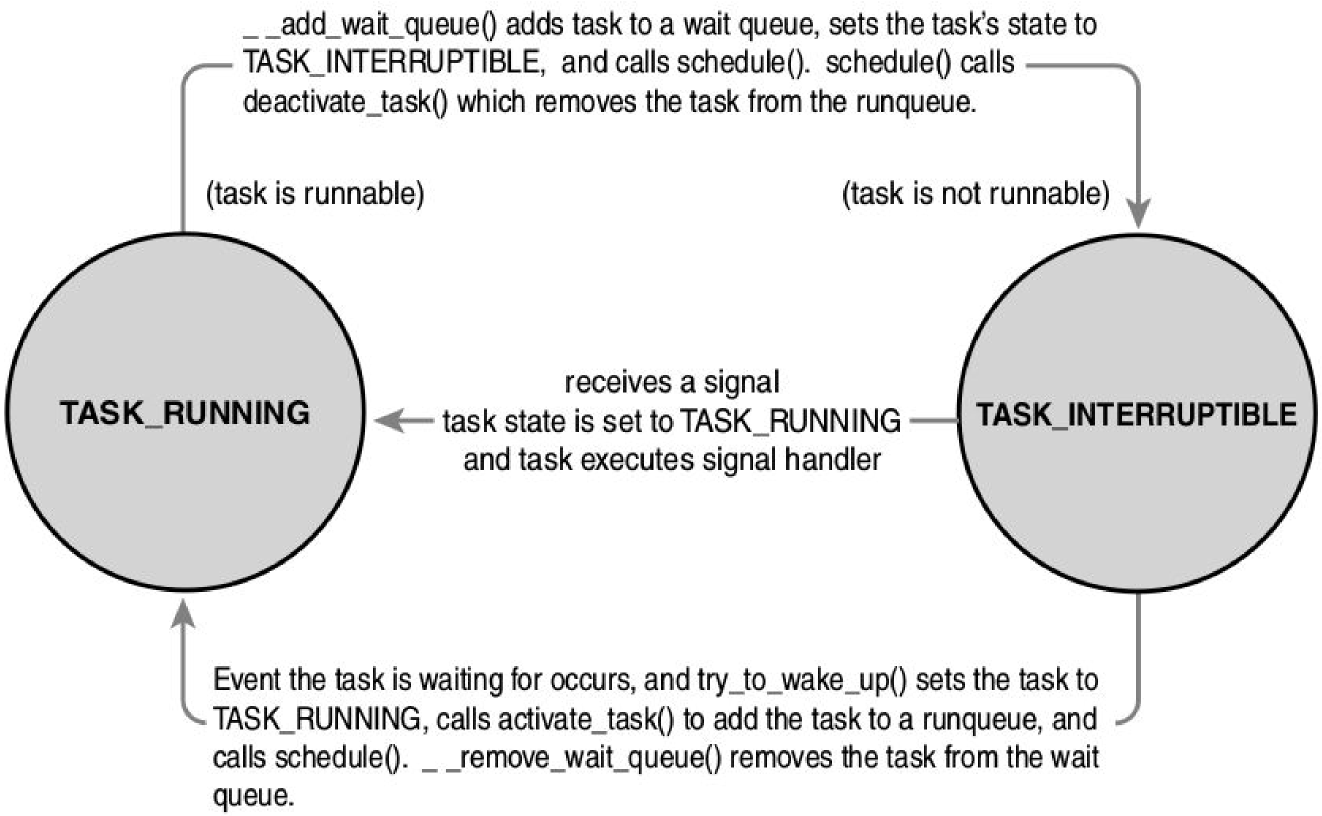
```
sleeping_task = current;  //A
set_current_state(TASK_INTERRUPTIBLE);  //B
schedule();  //C
func1();
/* The rest of the code */
```

- **Store a reference to this process' task structure**
- **Set current state changes from TASK_RUNNING to TASK_INTERRUPTIBLE**
- **The schedule() should schedule another process**

http://www.linuxjournal.com/article/8144?page=0,2

Sleep & Wake up

- **wake_up() , try_to_wake_up()**



try_to_wake_up()

kernel/sched/core.c

Sleep and wakeup

- **Two problems arise in design of sleep/wakeup**

Sleep and wakeup

- **Two problems arise in design of sleep/wakeup**

1. Lost wakeup

Linux Lost Wake-Up Problem

- Task A role : if list is empty -> goto sleep
- Task B role : insert item to list -> wake up Task A

Task A:

```
spin_lock(&list_lock);
if (list_empty(&list_head)) {
    spin_unlock(&list_lock);
    set_current_state(TASK_INTERRUPTIBLE);
    schedule();
    spin_lock(&list_lock);
}
/* Rest of the code ... */
spin_unlock(&list_lock);
```

Task B:

```
spin_lock(&list_lock);
list_add_tail(&list_head, new_node);
spin_unlock(&list_lock);
wake_up_process(Task A);
```

http://www.linuxjournal.com/article/8144?page=0,2

Linux Lost Wake-Up Problem

- Task A role : if list is empty -> goto sleep
- Task B role : insert item to list -> wake up Task A

Task A:

```
    spin_lock(&list_lock);
    if (list_empty(&list_head)) {
        spin_unlock(&list_lock);   //1
        set_current_state(TASK_INTERRUPTIBLE);  //2
        schedule();
        spin_lock(&list_lock);
    }
    /* Rest of the code ... */
   spin_unlock(&list_lock);
```

Task B:

```
    spin_lock(&list_lock);
    list_add_tail(&list_head, new_node);
    spin_unlock(&list_lock);
    wake_up_process(Task A); // already running
```

http://www.linuxjournal.com/article/8144?page=0,2

Lost Wake-Up Solution

```
Task A:
    set_current_state(TASK_INTERRUPTIBLE);
    spin_lock(&list_lock);
    if (list_empty(&list_head)) {
        spin_unlock(&list_lock);
        schedule();
        spin_lock(&list_lock);
    }
    set_current_state(TASK_RUNNING);
    /* Rest of the code ... */
   spin_unlock(&list_lock);
```

```
Task B:
    spin_lock(&list_lock);
    list_add_tail(&list_head, new_node);
    spin_unlock(&list_lock);
    wake_up_process(Task A); // already running
```

http://www.linuxjournal.com/article/8144?page=0,2

cpufreq_interactive_speedchange_task()

drivers/cpufreq/cpufreq_interactive.c

Sleeping in the Kernel

- Two problems arise in design of sleep/wakeup

1. Lost wakeup

2. Termination **running**

Termination while running

- **How does kill(target_pid) work?**

- **Problem: target process may be running**
  - Still using its kernel stack, page table, proc[] entry
  - might be in a critical section
    - needs to finish to restore invariants

Termination while running

- **How does kill(target_pid) work?**

- **Problem: target process may be running**
  - Still using its kernel stack, page table, proc[] entry might be in a critical section, needs to finish to restore invariants

- **Solution : Linux can't immediately terminate it**

Termination while running

- **Solution:**
- **Linux commits suicide**
  - Sets flag
  - Target thread checks flag in trap and exits
  - exit() closes FDs
  - Yields CPU -> parent process

- **Problem:**
  - Task's kernel stack, page table

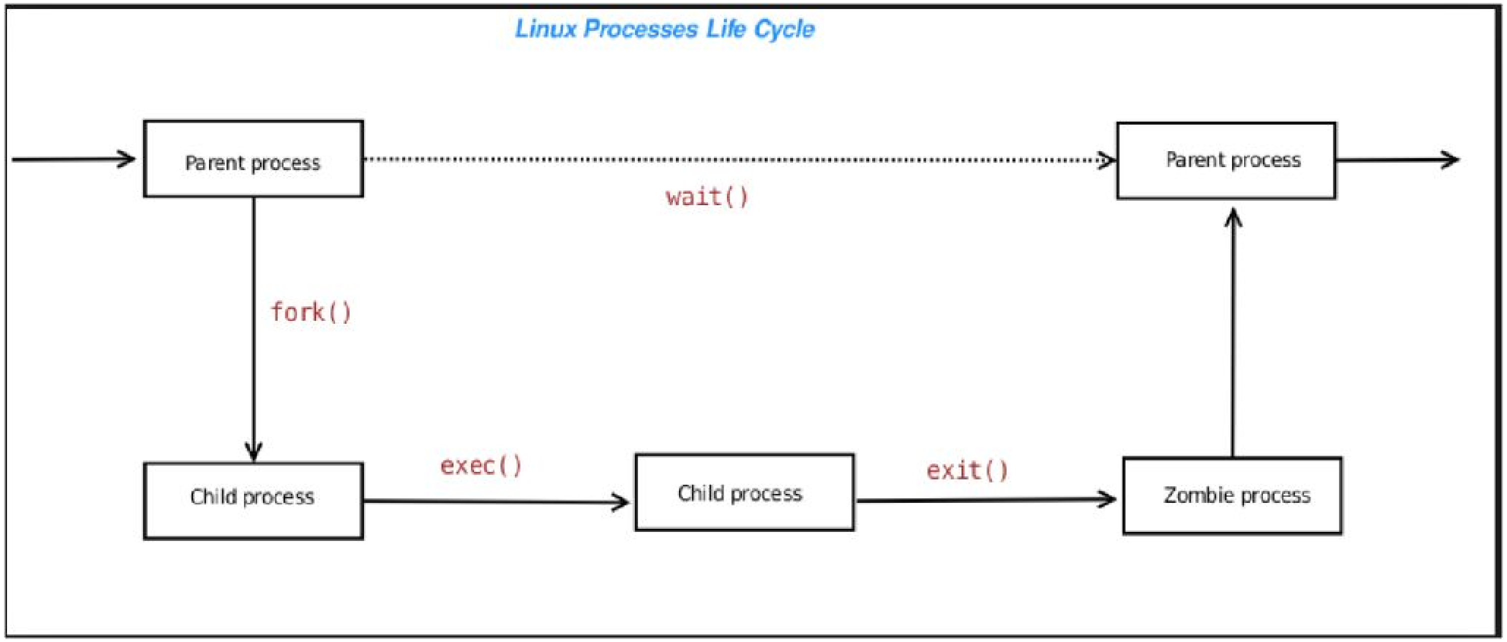
Termination while sleeping

- **Solution:**
- **Linux commits suicide**
  - Sets flag
  - Target thread checks flag in trap and exits
  - exit() closes FDs
  - Yields CPU -> parent process
- **Problem:**
  - Task's kernel stack, page table
- **Solution**
  - Sets state to ZOMBIE
  - Parent wait() frees kernel stack, page table

do_exit()
->exit_notify()

kernel/exit.c

Why defunct(zombie) process are created?



https://amitvashist.wordpress.com/2015/01/01/defunct-processes/

Linux Process States

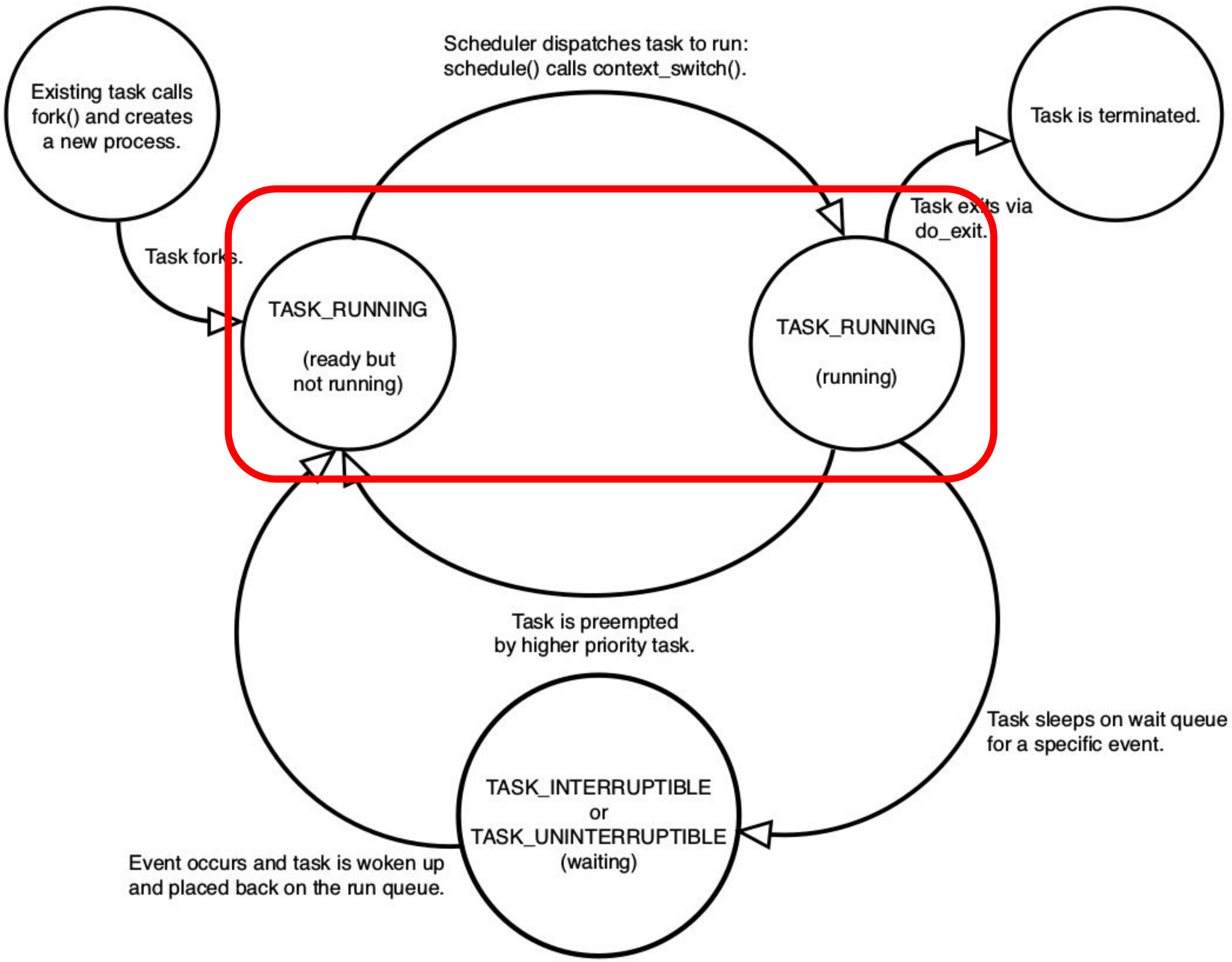


Figure 3.3 Flow chart of process states.

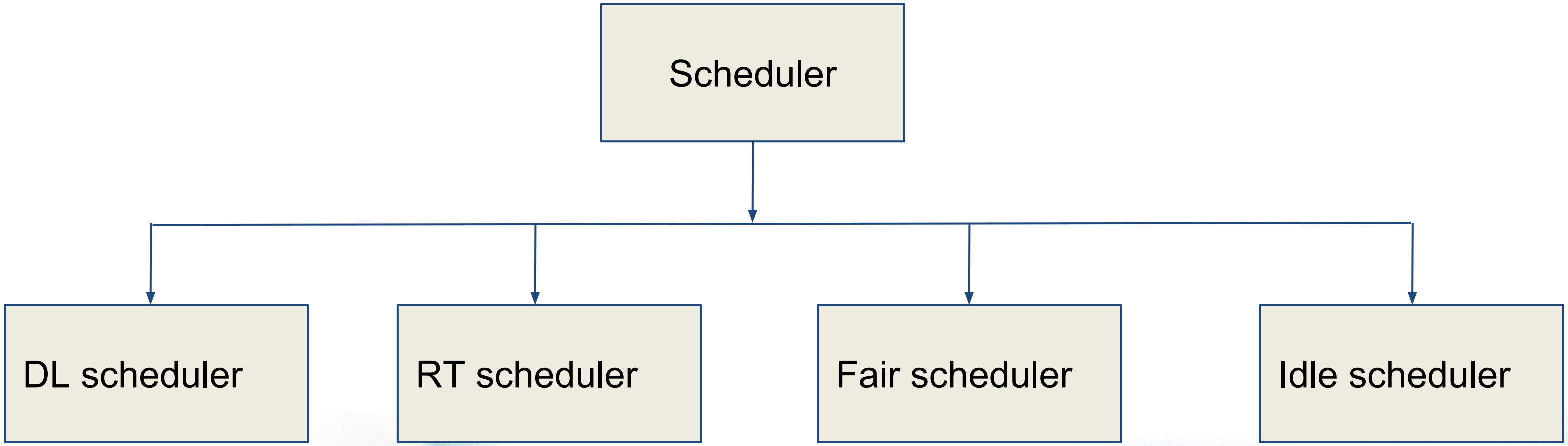
Scheduling policies in Linux Kernel

- The Linux scheduler has been made modular.

- SCHED_DEADLINE
- SCHED_FIFO
- SCHED_RR
- SCHED_NORMAL
- SCHED_BATCH
- SCHED_IDLE

Linux Scheduling Class

Class	Description	Policy
dl_sched_class	real-time task with deadline	SCHED_DEADLINE
rt_sched_class	real-time task	SCHED_FIFO SCHED_RR
fair_sched_class	time-sharing task	SCHED_NORMAL SCHED_BATCH
idle_sched_class	avoid to disturb other tasks	SCHED_IDLE



What is key decision points in the scheduler?

The key decisions

"how to determine a thread's ***timeslice****? and how to pick the* ***next thread*** *to run"*

Timeslice

- A time unit allowed for a task at a given time
- Affected by timer freq. (HZ)
- Hard to optimize

Why?

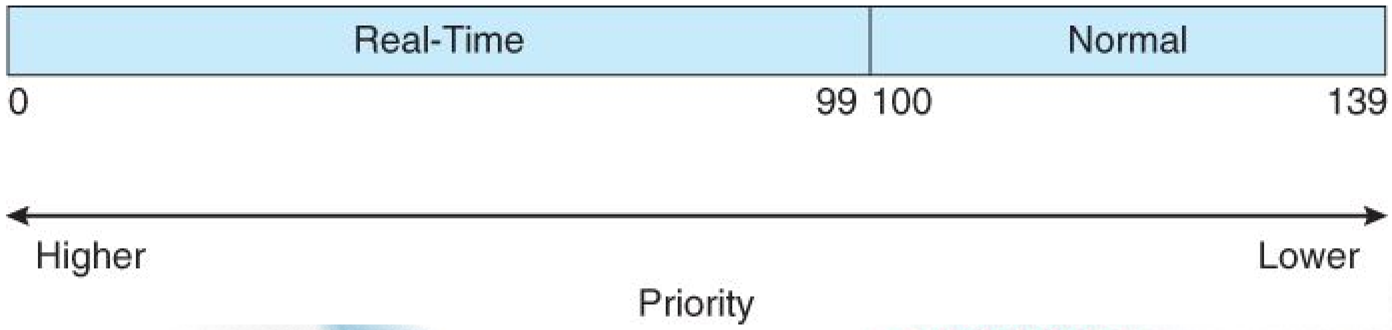
Timeslice

- **A time unit allowed for a task at a given time**
- **Affected by timer freq. (HZ)**
- **Hard to optimize since**
  - it should be small for less latency,
    - but poor throughput : C/W overhead
  - it should be large for better throughput
    - poor latency.

Nice Values and Task Priority

- **Non-real-time priority**
  - Nice value (-20~19, default 0)
  - A large nice value corresponds to a lower priority

- **Real-time priority**
  - Priority range: 0~99
  - Priority for real-time tasks
    - SCHED_FIFO, SCHED_RR
  - A smaller value corresponds to a higher priority



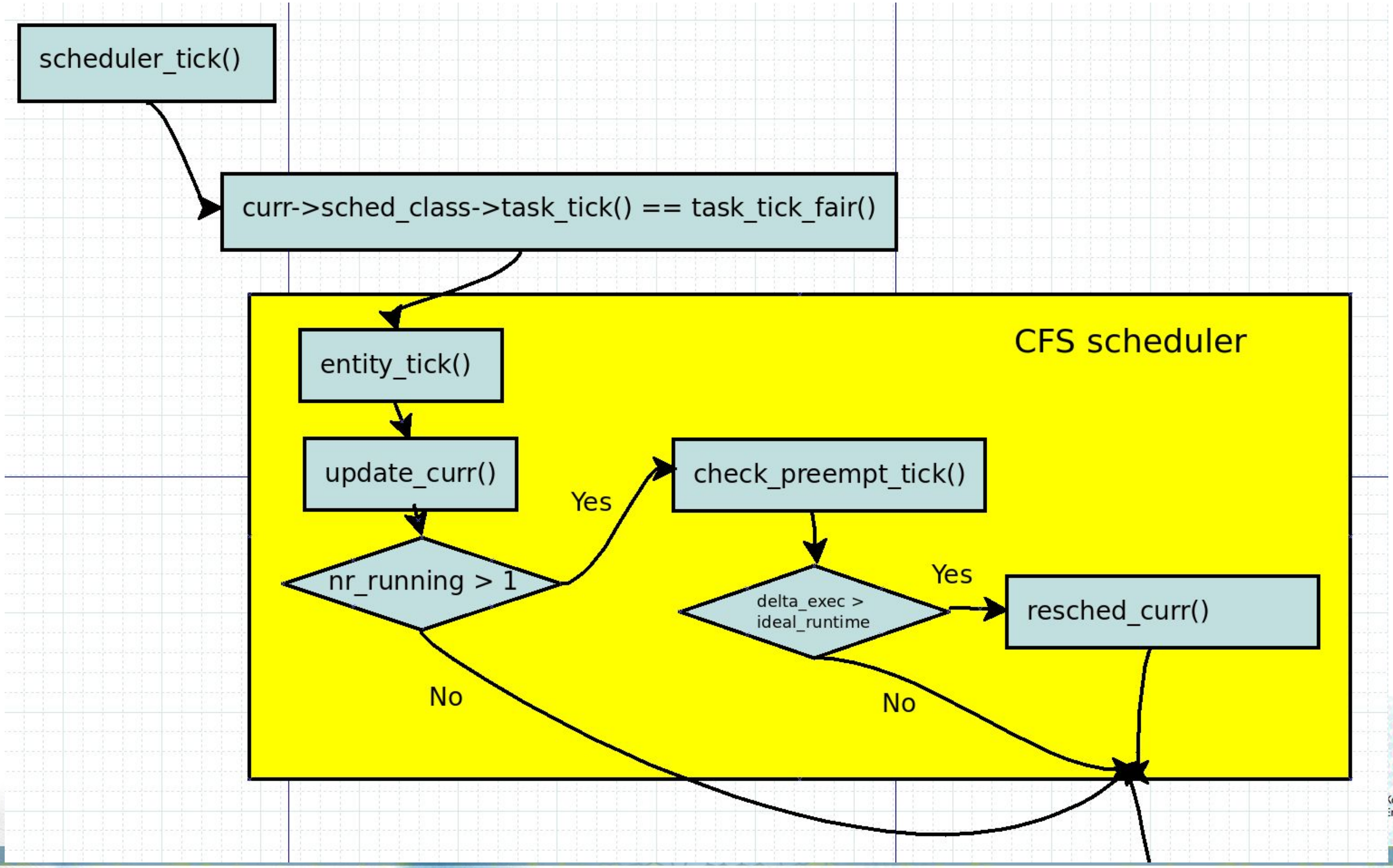
https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html

Functions in Scheduling Class

- **enqueue_task**
  - put the task into the run queue
  - increment the nr_running variable
- **dequeue_task**
  - remove the task from the run queue
  - decrement the nr_running variable
- **yield_task**
  - relinquish the CPU
- **check_preempt_curr**
  - check whether the currently running task can be preempted by a new task
- **pick_next_task**
  - choose the most appropriated task
- **load_balance**
  - trigger load balancing code

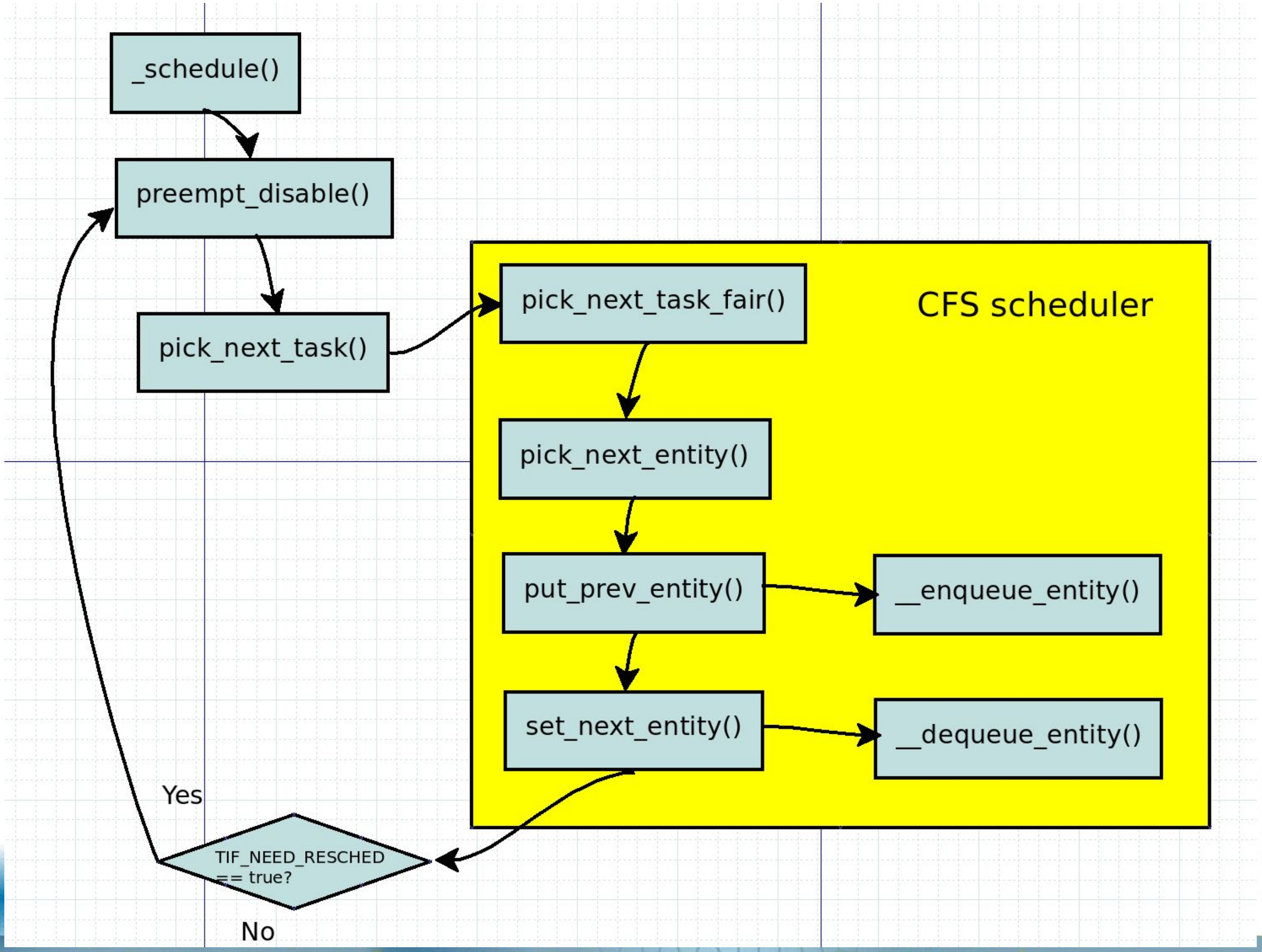
When schedule a task

- **Periodically forces a switch**
  - **Scheduler Tick**



Scheduling by schedule()

- TIF_NEED_RESCHED
- Sleep and wakeup - I/O wait, pipe, system call



KESL Kookmin Univ. Embedded System Lab

__schedule()

kernel/sched/core.c

```
static void __sched __schedule(void) {
     ...
need_resched:
    preempt_disable();
    cpu = smp_processor_id();
    //Save current task as prev
    rq = cpu_rq(cpu);
    prev = rq->curr;
    ...
    next = pick_next_task(rq, prev);
    //Enqueue and dequeue tasks
    ...
    if (need_resched())
        Check TIF_NEED_RESCHED
    goto need_resched;
}
```

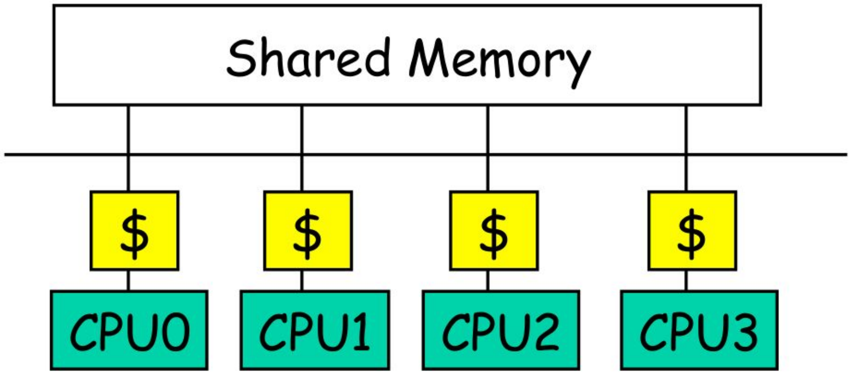
__schedule()

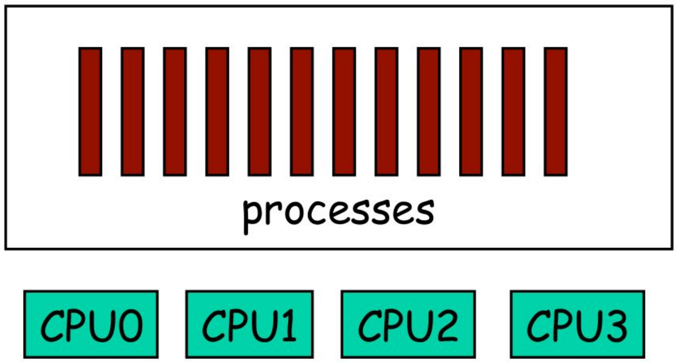
kernel/sched/core.c

Multi-core & Load balancer

Multiprocessor scheduling

- **Shared-memory Multiprocessor**

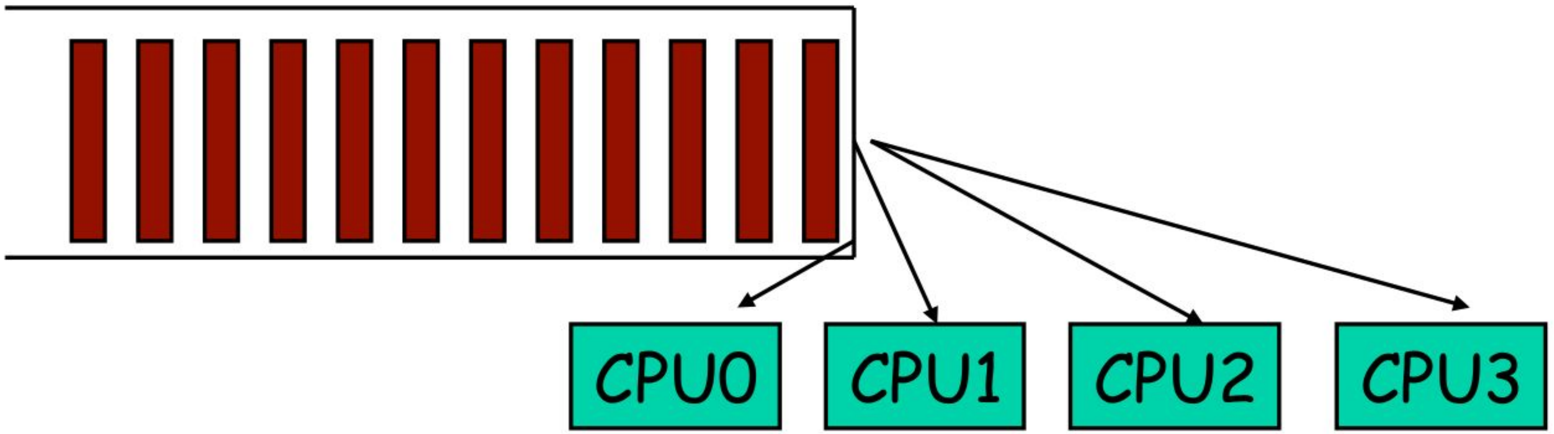




- **How to allocate processes to CPU?**

Global queue of processes

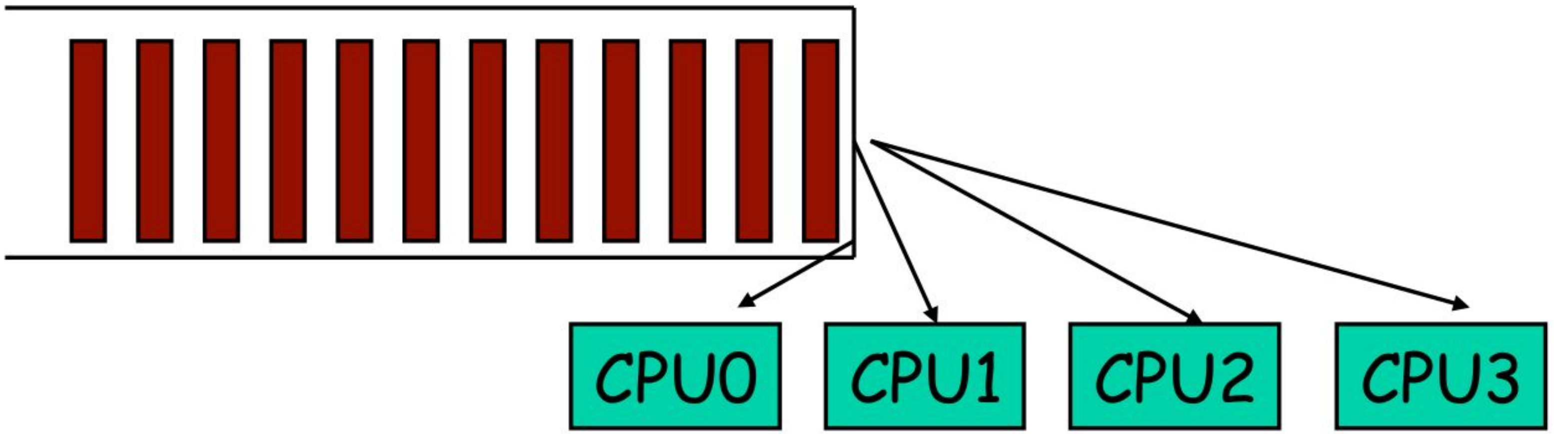
- One ready queue shared across all CPUs



- Advantages

Global queue of processes

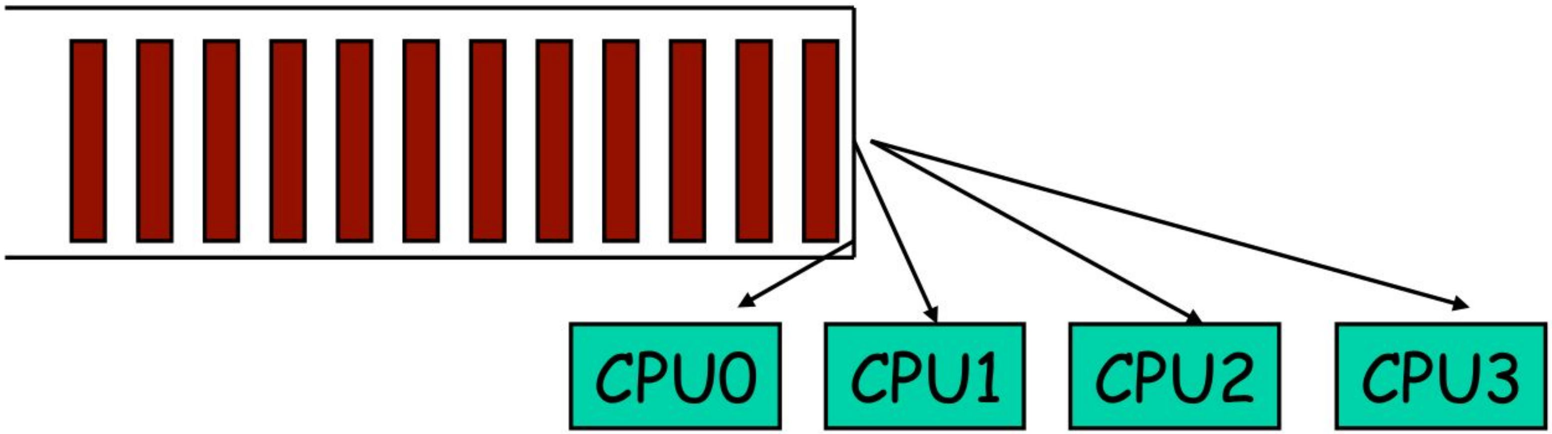
- **One ready queue shared across all CPUs**



- **Advantages**
  - Good CPU utilization
  - Fair to all processes
- **Disadvantages**

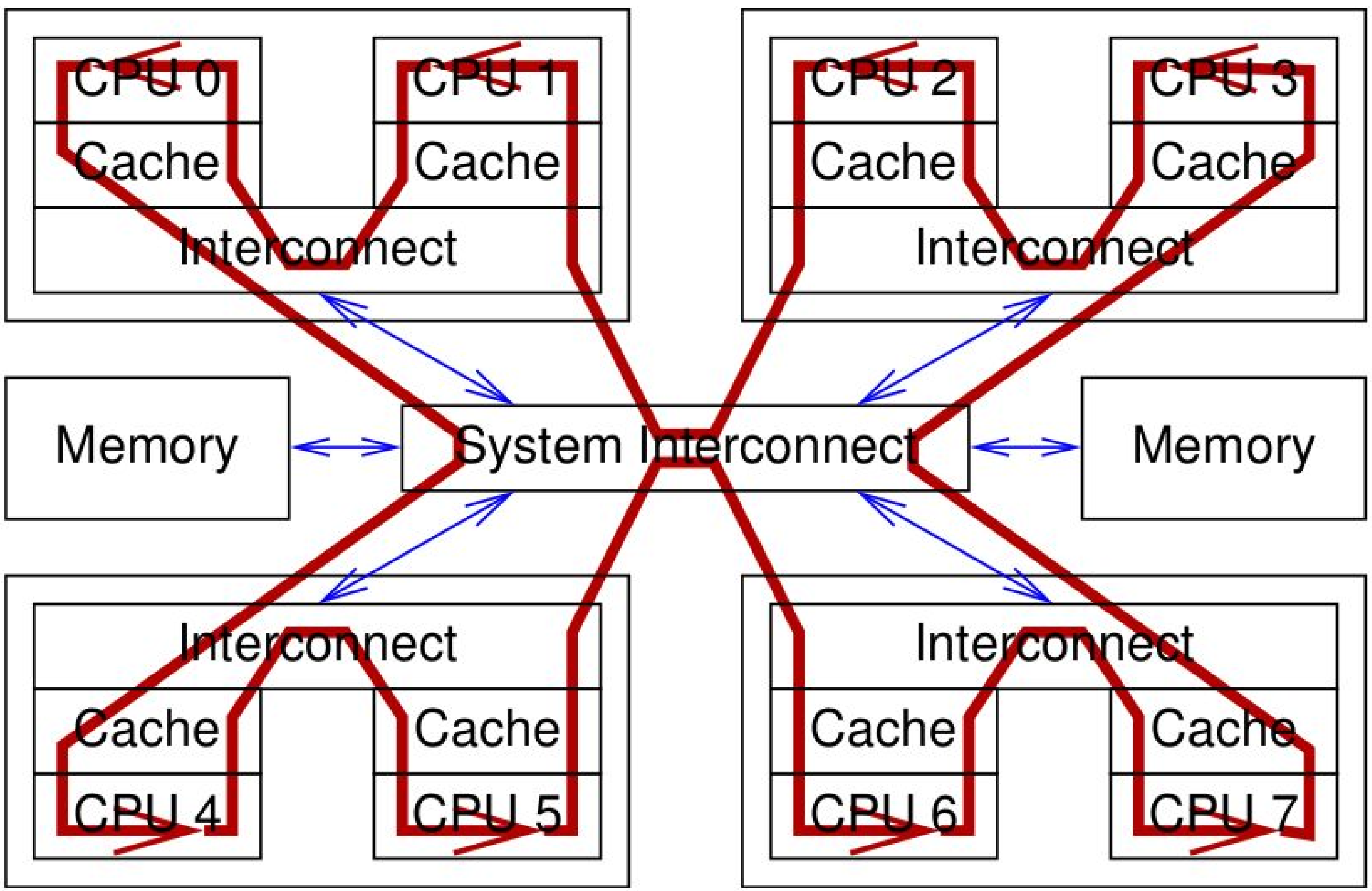
Global queue of processes

- **One ready queue shared across all CPUs**



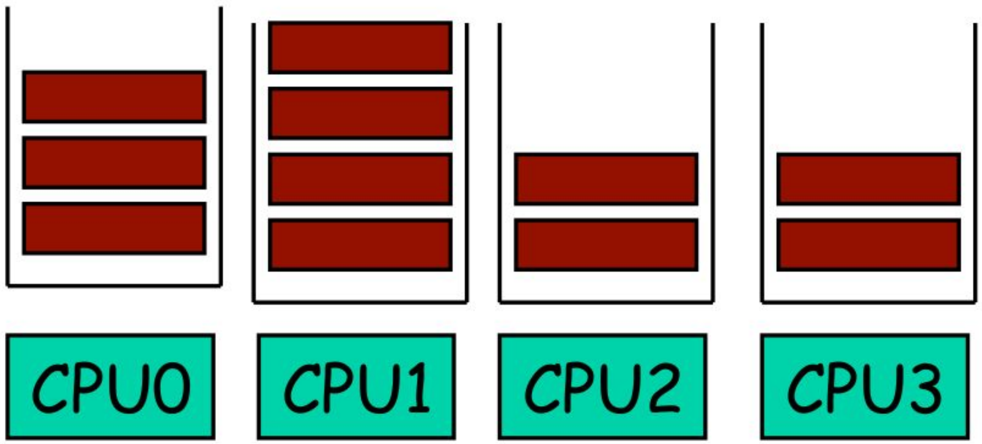
- **Advantages**
  - Good CPU utilization
  - Fair to all processes
- **Disadvantages**
  - Not scalable (contention for global lock)
  - Poor cache locality
- **Linux 2.4 uses global queue**

Global queue of processes



Per-CPU queue of processes

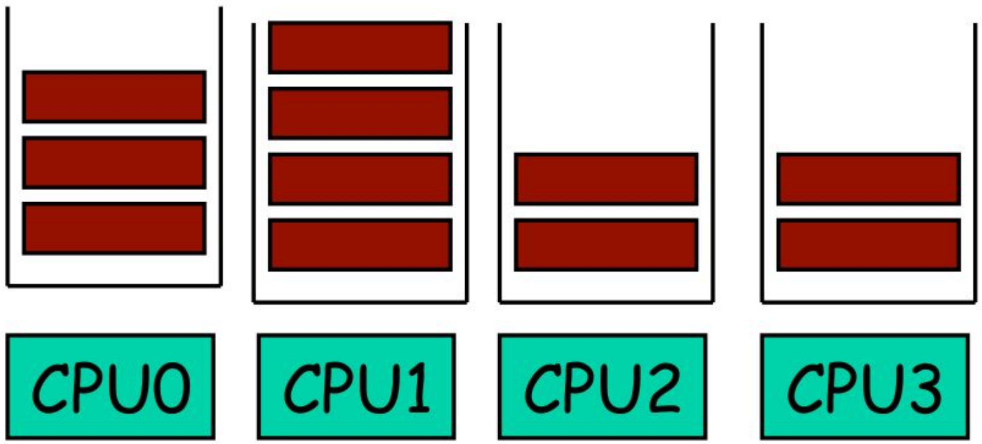
- **Static partition of processes to CPUs**



- **Advantages**

Per-CPU queue of processes

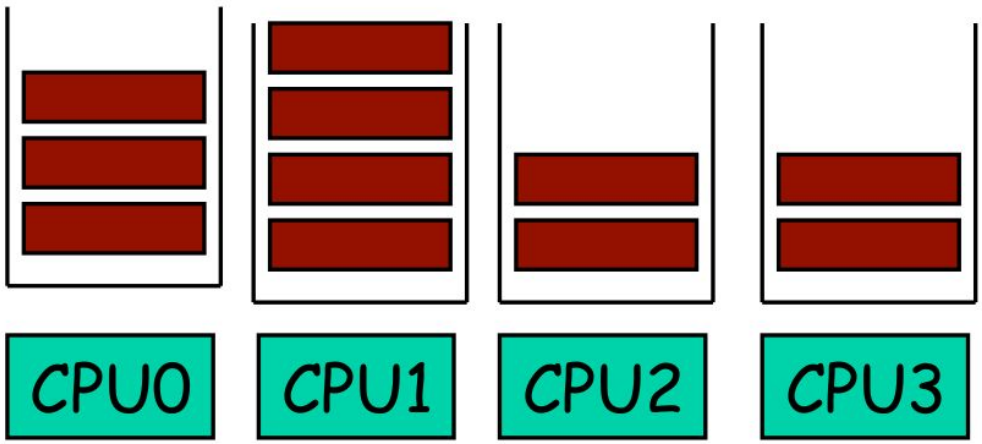
- **Static partition of processes to CPUs**



- **Advantages**
  - Easy to implement
  - Scalable (no contention on ready queue)
  - Better cache locality
- **Disadvantages**

Per-CPU queue of processes

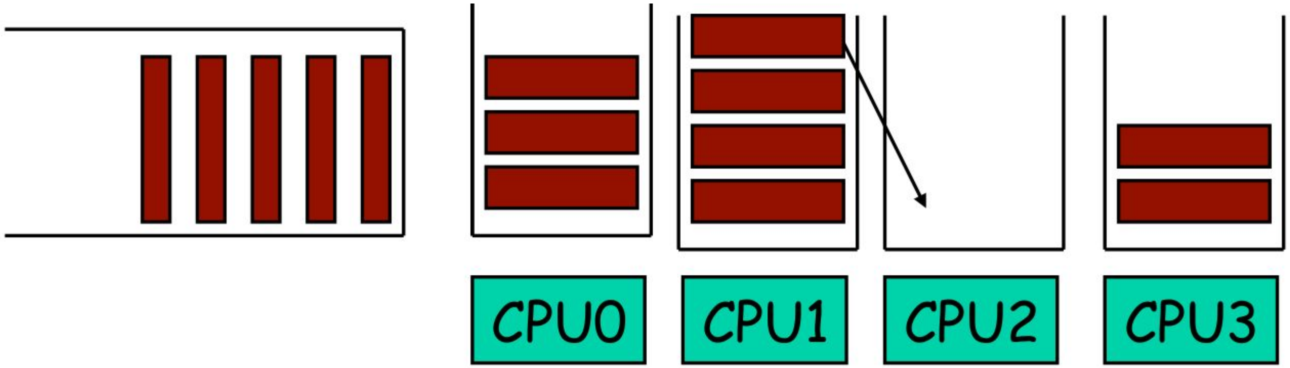
- **Static partition of processes to CPUs**



- **Advantages**
  - Easy to implement
  - Scalable (no contention on ready queue)
  - Better cache locality
- **Disadvantages**
  - Load-imbalance (some CPUs have more processes)

Hybrid approach

- **Use both global and per-CPU queues**
- **Balance jobs across queues**



- **Processor Affinity**
  - Add process to a CPU's queue if recently run on the CPU
  - Cache state may still present
- **Linux 2.6 uses a very similar approach**

load_balance()
select_task_rq()

kernel/sched/core.c

Chances to balance

- **fork & exec**
  - spread to any idlest cpu
- **wake up**
  - keep prev cpu or migrate to current cpu or its idle sibling
- **idle**
  - migrate to current cpu
- **periodic**
  - migrate to current cpu (or its siblings)

Next Step.

Energy-aware scheduling: EAS

1. **CFS scheduler - Kernel level**
2. **Load Balancer(Group Scheduling, Bandwidth Control, PELT)**
3. **EAS features**

Reference

- https://pdos.csail.mit.edu/6.828/2016/schedule.html
- http://web.mit.edu/6.033
- http://www.rdrop.com/~paulmck/
- “**Is Parallel Programming Hard, And If So, What Can You Do About It?”**
- Davidlohr Bueso. 2014. Scalability techniques for practical synchronization primitives. *Commun. ACM* 58

http://queue.acm.org/detail.cfm?id=2698990

- **"CPUFreq and The Scheduler Revolution in CPU Power Management", Rafael J. Wysocki**
- **https://sites.google.com/site/embedwiki/oses/linux/pm/pm-qos**
- **https://intl.aliyun.com/forum/read-916**
- **User-level threads : co-routines**

  http://www.gamedevforever.com/291

  https://www.youtube.com/watch?v=YYtzQ355_Co

- **Scheduler Activations**
  - https://cgi.cse.unsw.edu.au/~cs3231/12s1/lectures/SchedulerActivations.pdf
- **https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)**
- **http://jake.dothome.co.kr/**
- **http://www.linuxjournal.com/magazine/completely-fair-scheduler?page=0,0**
- **https://www2.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/6_CPU_Scheduling.html**
- **"Energy Aware Scheduling", Byungchul Park, LG Electronic**
- **“Update on big.LITTLE scheduling experiments”, ARM**
- **“EAS Update” 2015 september ARM**
- **"EAS Overview and Integration Guide”, ARM TR**
- **“Drowsy Power Management”, Matthew Lentz, SOSP 2015**
- **https://www.slideshare.net/nanik/learning-aosp-android-hardware-abstraction-layer-hal**
- https://www.youtube.com/watch?v=oTGQXqD3CNI
- https://www.youtube.com/watch?v=P80NcKUKpuo
- https://lwn.net/Articles/398470/
- “SCHED_DEADLINE: It’s Alive!”, ARM, 2017