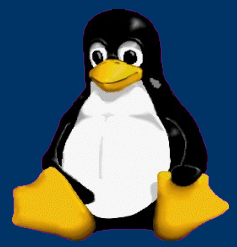
Linux O(1) CPU Scheduler

Amit Gud
amit (dot) gud (at) veritas (dot) com
http://amitgud.tk

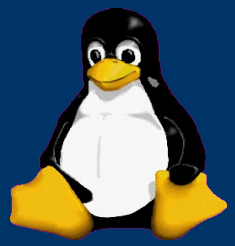
Agenda

- CPU scheduler basics
- CPU scheduler algorithms overview
- Linux CPU scheduler goals
- What is O(1)?
- Data structures of O(1) CPU scheduler
- Quick run through the task execution process
- Calculation of priorities
- Calculation of timeslices
- NUMA / SMP support
- Load Balancing



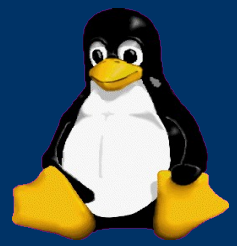
CPU Scheduler Basics

- Programs and processes
- Threads
- CPU timeslices
- Multiprogramming and preemption
- I/O bound and CPU bound processes
- Context switching



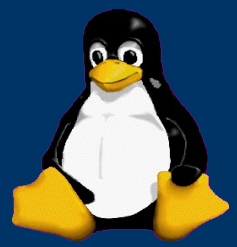
CPU Scheduler Algorithms Overview

- First Come First Server (FCFS)
- Shortest Job First (SJF)
- Priority scheduling
- Round robin
- Multilevel queue
- Multilevel feedback queue



Linux CPU Scheduler Goals

- Efficiency
- Interactivity
- Fairness
- SMT / SMP scheduling
- NUMA scheduling
- Soft real-time scheduling

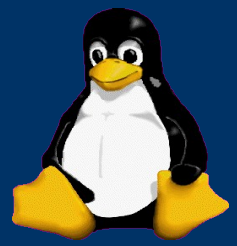


What is O(1)?

- Time complexity
- Big-O notation

- $O(n^2) > O(n)$
- $O(n) > O(1)$

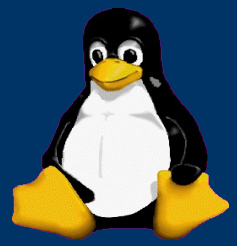
```
/* O(n) algorithm */
for (i = 0; i < n; i++) {
    printf("i: 5d\n", i);
}

/* O(n*m) = O(n*n) algorithm */
for (i = 0; i < n; i++) {
    for (j = 0; j < m; j++) {
        printf("i: %d, j: %d\n", i, j);
    }
}
```



Data Structures - runqueue

- For tracking all runnable processes
- Maintained per CPU

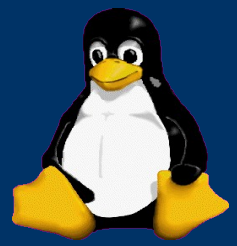
```
kernel/sched.c:
struct runqueue {
    spinlock_t lock; /* lock for this q */
    task_t *curr; /* current task */
    task_t *idle; /* the idle task */
    prio_array_t *active; /* active array */
    prio_array_t *expired; /* expired array */
    struct sched_domain *sd;/* for SMP/NUMA */
    task_t *migration_thread; /*migration task*/
    ...
    ...
};
```



Data Structures – priority arrays

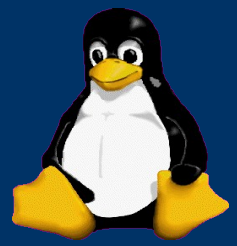
- Forms the basis of O(1) nature of the scheduler
- 140 priority levels
- 2 priority arrays per runqueue

```
kernel/sched.c:
struct prio_array {
    unsigned int nr_active;
    unsigned long bitmap[BITMAP_SIZE];
    struct list_head queue[MAX_PRIO];
};

include/linux/sched.h:
typedef struct prio_array prio_array_t;
#define MAX_PRIO 140
```



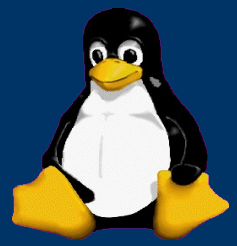
Priority arrays contd...

- Two priority arrays per runqueue
  - active array
  - inactive array
- Tasks are scheduled from active array
- When active array is empty, active array and expired arrays are swapped.



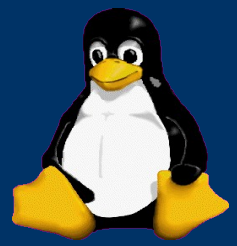
When task becomes runnable...

- Priority calculation
- Addition of the task to the priority queue of the active runqueue of some CPU (selection depends on the load balancing and cache warmth exploit)
- Setting the bit in the bitmask of the active priority queue



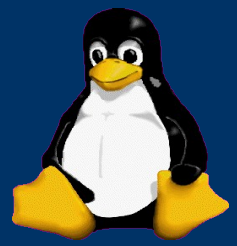
When a running task becomes unschedulable...

- Priority calculation
- Moving the task from active priority queue to inactive one
- Resetting the bit in the bitmask of the active priority queue



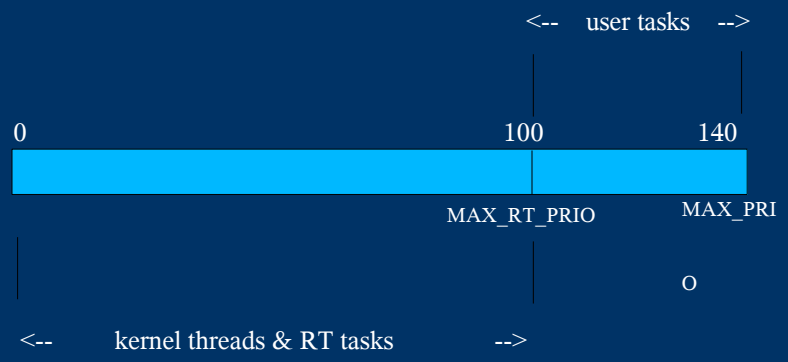
Selecting a task for scheduling

- Find first set bit in the bitmask of the active array (using __ffs() - O(1), length of bitmask is constant)
- Schedule the task of that priority from the linked list
- If more than one task of that priority, round robin is applied



Priority Calculation

- Static priority – nice value [-20, 19]
  - by default 0
  - used for timeslice calculation
- Dynamic priority [0, 139]
  - signifies the type of the task – I/O bound or CPU bound



lower the priority value – higher the priority

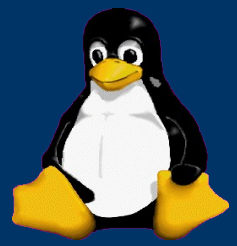
Dynamic Priority Calculation

- Penalty (addition) for CPU bound tasks and reward (subtraction) for I/O bound tasks [-5, 5]

- `p->sleep_avg` : average amount of time a task sleeps vs. average amount of time task uses CPU

```
p->sleep_avg += sleep_time

p->sleep_avg -= run_time
```

- Higher `sleep_avg` –> more I/O bound the task -> more reward. And vice versa.



Dynamic Priority Calculation contd...

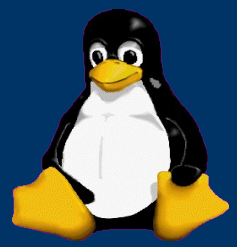
- Mapping sleep_avg to range `0 – MAX_BONUS`, i.e. 0 - 10

- `MAX_SLEEP_AVG : MAX_BONUS`

  `SLEEP_AVG : X (unknown)`

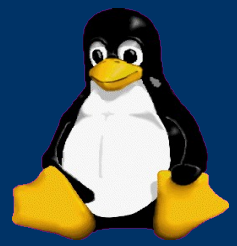
- `bonus = X – MAX_BONUS / 2;`

- `p->prio = p->static_prio - bonus;`



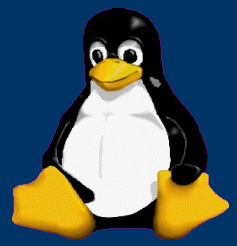
Timeslice Calculation

- Mapping of `static_prio` (nice value) onto `MIN_TIMESLICE - MAX_TIMESLICE`
- Directly proportional to `static_prio`
- i.e. `MIN_TIMESLICE + static_prio` scaled onto possible timeslice ranges



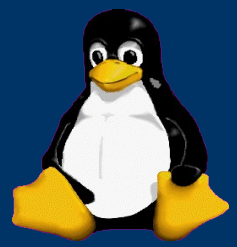
Interactive Credit

- Earned when a task sleeps for a 'long' time
- Spent when a task runs for a 'long' time
- IC is used to avoid loosing interactivity of the task very soon or vice versa
  - CPU hog sleeping occasionally for a long time
  - I/O bound task occasionally utilizing CPU for a long time



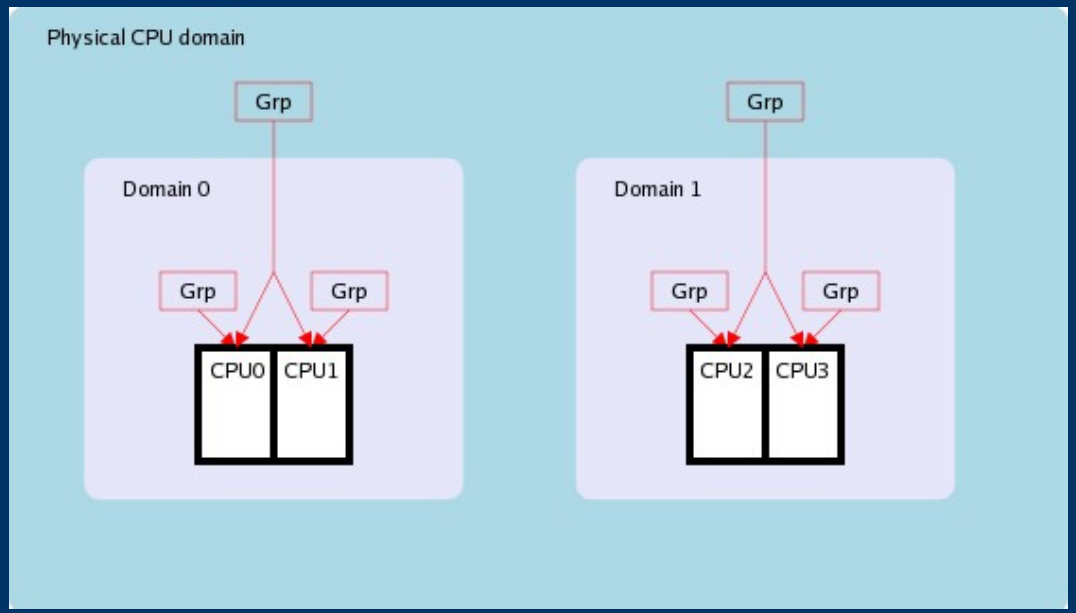
Real-Time Tasks

- Have priority between 0 – 99, so always preempt non-RT tasks
- No penalties or bonuses
- Scheduling schemes
  - `SCHED_FIFO` – preempts any other task, no timeslice limit, obeys priorities amongst other `SCHED_FIFO` tasks
  - `SCHED_RR` – preempts `SCHED_NORMAL` tasks, round robin scheduling amongst same priority `SCHED_RR` tasks



SMP / NUMA Support

- Virtual hierarchy
  - `sched_domain`
  - `sched_group`

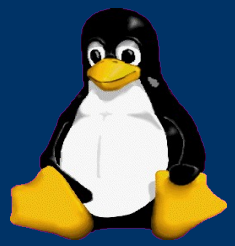


Load Balancing

- Balancing attempted after certain time interval
- `rq-> cpu_load` : represents load on the CPU
  - calculated every timer interrupt
  - calculated as average of previous load and current load

```
#define SCHED_LOAD_SCALE 128

current_load = rq->nr_running * SCHED_LOAD_SCALE;
```

- Bottom up load balancing
- Pulling of tasks instead of pushing



References

- Understanding the Linux 2.6.8.1 CPU Scheduler

  Josh Aas. `http://josh.trancesoftware.com/linux/`

- Linux on NUMA Systems

  Bligh, Dobson, Hart, Huizenga. Proceedings of the Linux Symposium 2004

- Linux Kernel Source Code

  http://lxr.linux.no/source