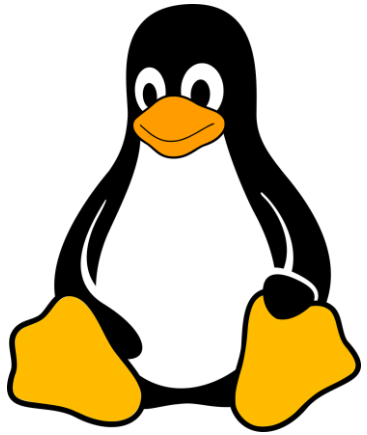


# Linux – raspoređivanje procesa

# PODSETNIK

- Jednoprocesorsko raspoređivanje
  - Klasično UNIX raspoređivanje
- Multiprocesesorsko raspoređivanje

## Klasično *UNIX* raspoređivanje

**Prioritet procesa $j$ iz grupe $k$**

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

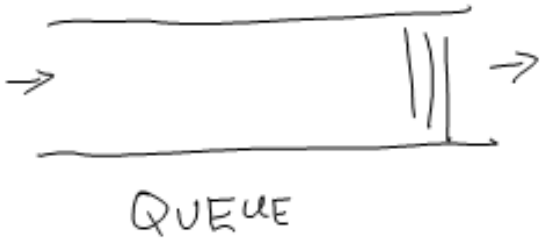
$$P_j(i) = Osnova_j + \frac{CPU_j(i)}{2} + nice_j$$

- $CPU_j(i)$ - mera iskorišćenja procesora od strane procesa $j$ u intervalu $i$
- $P_j(i)$ - prioritet. Manja vrednost - viši prioritet
- $Osnova_j$ - osnovni prioritet procesa $j$
- $nice_j$ - faktor za fino podešavanje

# Evolucija Linux raspoređivača

- O(n) scheduler
  - Linux 2.4 - 2.6
- O(1) scheduler
  - Linux 2.6 – 2.6.22
- CFS  scheduler
  - od Linux 2.6.23

# 1. O(n)




- Od prve verzije Kernela, 1991. – 2003.
- Za svaki vremenski kvantum, kernel je morao da prođe kroz listu svih procesa, da bi našao onaj sa najvećim brojačem
- Brojač se smanjuje svaki put kad se proces izvršava

```
void schedule(void)
{
    int i,next,c;
    struct task_struct ** p;

/* check alarm, wake up any interruptible tasks that have got a signal */

    for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
        if (*p) {
            if ((*p)->alarm && (*p)->alarm < jiffies) {
                    (*p)->signal |= (1<<(SIGALRM-1));
                    (*p)->alarm = 0;
                }
            if (((*p)->signal & ~(_BLOCKABLE & (*p)->blocked)) &&
            (*p)->state==TASK_INTERRUPTIBLE)
                (*p)->state=TASK_RUNNING;
        }

/* this is the scheduler proper: */

    while (1) {
        c = -1;
        next = 0;
        i = NR_TASKS;
        p = &task[NR_TASKS];
        while (--i) {
            if (!*--p)
                continue;
            if ((*p)->state == TASK_RUNNING && (*p)->counter > c)
                c = (*p)->counter, next = i;
        }
        if (c) break;
        for(p = &LAST_TASK ; p > &FIRST_TASK ; --p)
            if (*p)
                (*p)->counter = ((*p)->counter >> 1) +
                        (*p)->priority;
    }
    switch_to(next);
}
```

# PREDNOSTI I MANE

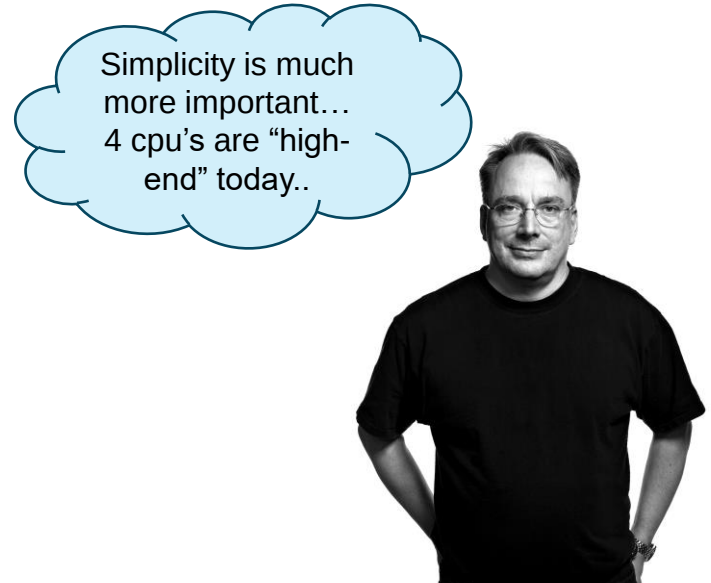
**PREDNOSTI**

- Lak za implementaciju i razumevanje

**MANE**

- U SMP, koristi se jedan globalni red sa procesima
- Kompleksnost
- Nije skalabilno – pogotovo kad se koristi JVM koji kreira mnogo procesa

- Na optimalan način grupisati procese u različite redove čekanja
- Da svaki procesor ima svoj red čekanja
- Work-stealing za balansiranje dužine redova



# 2. O(1)

- Ingo Molnar is a Hungarian Linux hacker.
- Uveden kako bi rešio problem skalabilnosti
- Dve liste zadataka
  - Aktivna
  - Istekla
- Zamena listi kad Aktivna postane prazna
- 2 koraka prilikom izbora sledećeg procesa :
  1. Naći onaj red sa najmanjim prioritetom koji ima bar 1 zadatak

     (-bsfl funkcija za Intelu)
  1. Izabrati prvi zadatak iz tog reda

```c
#include <stddef.h>

typedef struct {
  int nr_active;
  struct list_head queue[128];
} prio_array_t;

static prio_array_t active_array;
static prio_array_t expired_array;

void schedule(void) {
  prio_array_t *array;
  struct list_head *queue;
  task_t *next;

  // Switch the active and expired arrays if the active array is
empty.
  if (active_array.nr_active == 0) {
    array = &expired_array;
    expired_array = active_array;
    active_array = *array;
  }

  // Select the first task in the active array.
  queue = array->queue[0];
  next = list_entry(queue->next, task_t, run_list);

  // Remove the task from the active array.
  dequeue_task(next, array);

  // Context switch to the next task.
  context_switch(NULL, next);
}
```

# PREDNOSTI I MANE

**PREDNOSTI**

- Bolja skalabilnost
- Pogodan za sisteme sa dosta procesa
- Ne prolazi kroz listu svih procesa (bira prvi iz reda)
- Procesi ostaju na istom procesoru (bez ping-pong efekta)

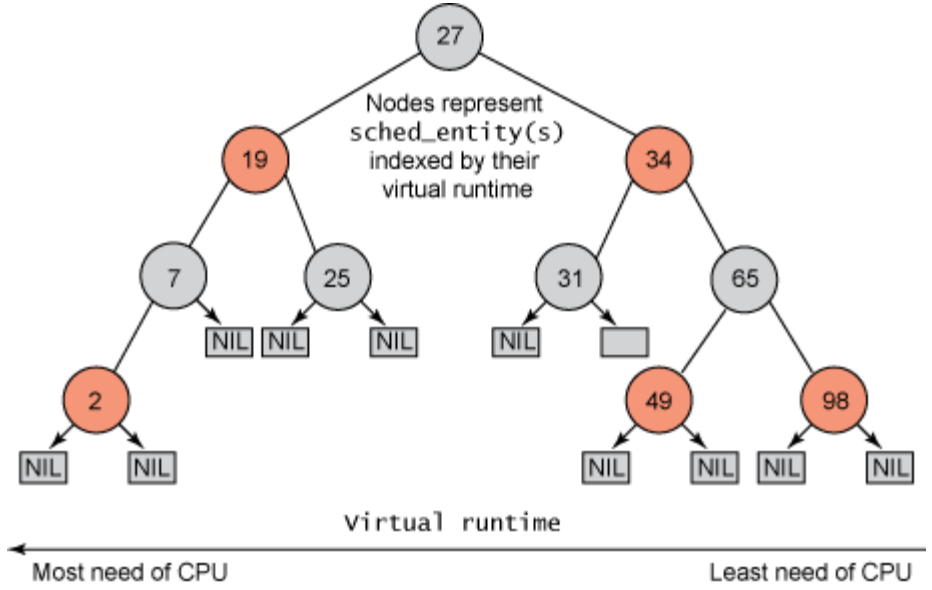
**MANE**

- Kad jednom proces uđe u listu isteklih, neće dobiti priliku da se ponovo pokrene sve dok se ne isprazni lista aktivnih
- Nije pravedan pri raspodeli CPU vremena za interaktivne procese, koji obično kraće traju
- Nije uzimao u obzir procese koji koriste U/I uređaje i čekaju na njih
- Na NUMA sistemu sa više memorijskih čvorova, gde svaki ima svoj CPU i memoriju, ima loše performanse

O(1) raspoređivač je neophodan za poboljšanje performansi i skalabilnosti. Spreman sam da žrtvujem pravičnost da bih postigao te ciljeve.

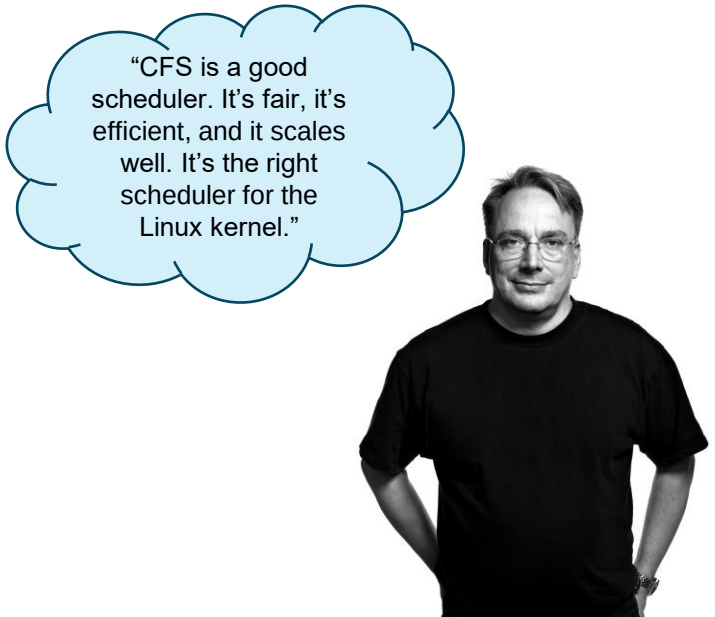

"The Linux Scheduler: a Decade of Wasted Cores"

# 3. CFS (Completely Fair Scheduler)



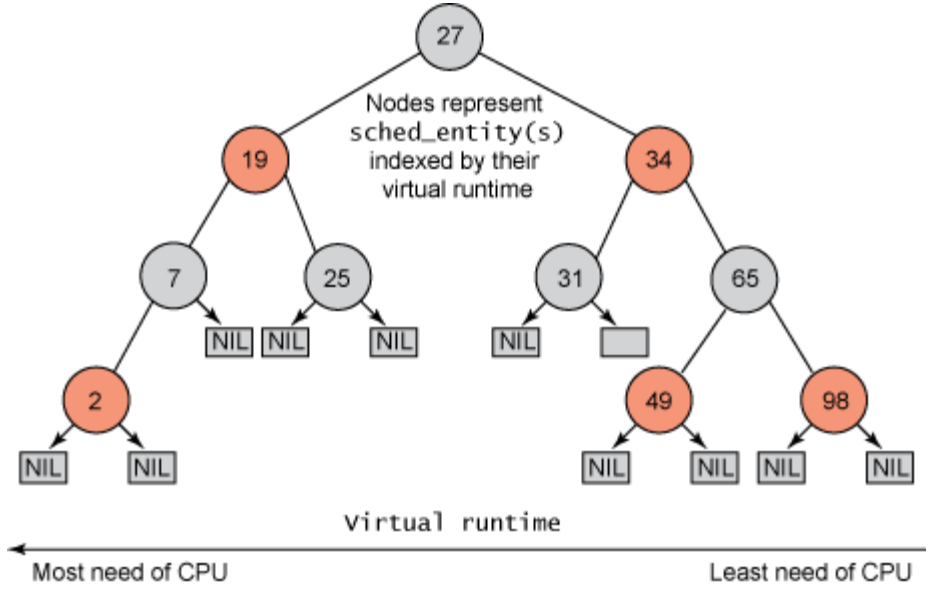

- Ingo Molnar
- Napravljeno po uzoru na RSDL (Rotating Staircase Deadline Scheduler)
- Napravljen sa ciljem da optimizuje generalnu iskorišćenost procesora i da poboljša interaktivne performanse

1. Bira se krajnji levi čvor stabla planiranja (pošto će imati najmanje potrošeno vreme izvršenja)
2. Ako proces završi, uklanja se iz sistema i stabla planiranja
3. Ako proces dostigne svoje maksimalno vreme izvršenja ili je na drugi način zaustavljen, ponovo se ubacuje u stablo (na osnovu novopotrošenog vremena izvršavanja)
4. Novi krajnji levi čvor će tada biti izabran iz stabla



# 3. CFS (Completely Fair Scheduler)

- CFS radi na grupama, a ne na pojedinačnim zadacima
- Grupa A ima 4 procesa, grupa B ima 6 procesa. Obe grupe dobijaju po 50% CPU vremena. Unutar grupe se vreme raspodeli na osnovu težine (prioriteta) i vremena koji čekaju na CPU.
- Jedan red izvršavanja može da ima više redova unutar sebe zbog grupa (kreirajući hijerarhiju)




| Operacija | Vremenska složenost |
|---|---|
| Pretraživanje | O(log n) |
| Ubacivanje | O(log n) |
| Brisanje | O(log n) |

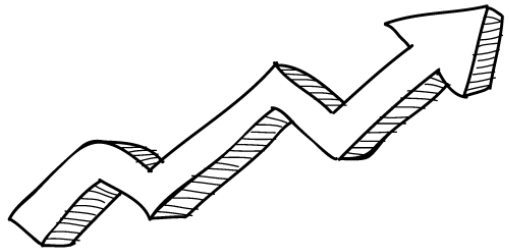
# PREDNOSTI

**PREDNOSTI**

- Pravednost – prema interaktivnim zadacima
- Efikasnost – kada uzmemo u obzir rukovanje zadacima vezanim za U/I
- Skalabilnost – sa povećanjem broja procesa, ne žrtvuju se performanse

- Ingo Molnar je pokazao u svom radu da se performanse i skalabilnost mogu uporediti sa O(1), a da pritom bude pravedniji i efikasniji.
- Pozitivna povratna informacija od korisnika Linux kernela

# DALJI NAPREDAK



- Podrška heterogenim procesorima koji imaju mešavinu različitih jezgara sa različitim brzinama ili skupovima instrukcija
- Aktivno istraživanje o značaju mašinskog učenja na poboljšanje odluka o rasporedu
  - Može se iskoristiti da se predvidi budućeg radnog opterećenja
- Upotreba distribuiranog planiranja za poboljšanje skalabilnosti