Sistem za rasporedjivanje računarskih procesa

September 5, 2025

1 Spisak članova tima

- Vladimir Popov SV29/2021

2 Motivacija

Efikasno rasporedjivanje jedinica rada na raspoložive resurse predstavlja problem i izazov ne samo u domenu operativnih sistema sa višeprocesorskim arhitekturama, već i u svim kompleksnijim softverskim sistemima, kao što su CI/CD sistemi, distribuirani sistemi, DBMS-ovi, mrežni ruteri. S obzirom na veliku složenost i primenljivost problema, ovaj projekat ima za cilj da kreira robustan sistem za rasporedjivanje zasnovan na pravilima. Iako će implementacija biti usmerena na rasporedjivanje računarskih procesa, sistem će biti dizajniran tako da omogući laku izmenu za primenu u drugim domenima.

3 Opis problema

Sistem treba da omogući efikasno rasporedjivanje sistemskih procesa na ograničene resurse CPU jezgara i memorije. Današnji operativni sistemi poseduju mehanizme za rasporedjivanje poput Completely Fair Scheduler-a u Linuxu. Medjutim, takvi rasporedjivači imaju svoje mane. Njihova impementacija leži u dubini kernela operativnog sistema, što ih čini nedovoljno transparentnim i nefleksibilnim. Ovo otežava njihovu dalju modifikaciju i prilagodjavanje specifičnim zahtevima.

Umesto fiksirane implementacije algoritama u kernelu, ovaj sistem će omogućiti laku modifikaciju ponašanja rasporedjivača upotrebom sistema baziranog na pravilima. Korisnici će moći da dodaju nova, ili modifikuju postojeća pravila kako bi ga dalje prilagodjavali svojim potrebama.

Iako rešenje neće imati peformanse na nivou rasporedjivača sa niskog, kernel nivoa, ono će ipak ponuditi značajnu fleksibilnost i modularnost. Sistem će raditi kao centralna instanca koja može da prima zahteve od različitih komponenti, što omogućava širok spektar primene i van operativnih sistema.

4 Metodologija rada

4.1 Ulazi u sistem

Ulaze u sistem predstavljale bi sledeće klase:

- Process
  - priority - prioritet procesa (1 - 10),
  - memoryRequirement - količina potrebne memorije za početak izvršavanja,
  - status - NEW / READY / RUNNING / EXIT / BLOCKED / SUSPENDED,
  - currentInstruction - index trenutno izvršavana instrukcije
  - insctructions - lista instrukcija u ovom procesu
  - lastStatusChange - vreme posledenje promene stanja
  - safeMemoryLimit - minimalan slobodan memorijski prostor da bi se proces vratio iz SUSPENDED stanja
- Instruction
  - type - I/O / NORMAL
- CpuCore
  - currentProcessId - id procesa koji se izvršava
  - status - IDLE / BUSY / PAGING
  - lastStatusChange - vreme poslednje promene stanja
  - enabled
- SystemState
  - availableMemory - dostupna količina memorije
  - totalMemory - ukupna količina memorije

4.2 Izlazi iz sistema

Redosled izvšavanja procesa po jezgrima, zajedno sa promenom stanja svih procesa.

4.3 Baza znanja

4.3.1 Pravila

```
when
    Process.state == NEW &&
    sufficientMemory
then
    Process.state = READY
    reduceAvailableMemory
```

```
when
    Process.state == READY &&
    processWithMaxPriority &&
    CpuCore.state == IDLE
then
    assignProcessToTheCore
```

```
when
    Process.state == RUNNING &&
    Process.currentInstruction == len(instructions)
then
    Process.status = TERMINATED
    freeCpuCore
```

```
when
    Process.state == RUNNING &&
    Process.currentInstruction == I/O
then
    Process.status = BLOCKED
    freeCpuCore
```

```
when
    Process.status == BLOCKED &&
    Process.currentInstruction == NORMAL
then
    Process.status = READY
```

```
when
    Process.status == RUNNING &&
    Process.currentInstruction != len(instructions) &&
    CpuCore.status == BUSY
then
    Process.currentInstruction++
```

```
when
    Process.status == RUNNING &&
    Process.currentInstruction != len(instructions) &&
    CpuCore.status == PAGING
then
    Process.currentInstruction += 0.5
```

```
when
    CpuCore.status == PAGING &&
    timeSince(CpuCore.lastStatusChange) > 2
then
    CpuCore.status = BUSY
```

```
when
    Process.status == RUNNING &&
    thereIsReadyProcessWithHigherPriority
then
    preemptTheProccess
    freeTheCore
```

```
when
    Process.status == SUSPENDED &&
    SystemState.availableMemory > Process.safeMemoryLimit
then
    Process.status = READY
```

4.3.2 Forward chaining

Postoji novi proces u stanju NEW, koji aktivira pravilo

```
when
    Process.status == NEW &&
    sufficientMemory
then
    Process.status = READY
    reduceAvailableMemory
```

Postoji slobodan core, aktivira se pravilo:

```
when
    Process.status == READY &&
    processWithMaxPriority &&
    CpuCore.status == IDLE
then
    assignProcessToTheCore
```

Proces se izvršava i aktivira pravilo

```
when
    Process.status == RUNNING &&
    Process.currentInstruction != len(instructions)
then
```

```
    Process.currentInstruction++
```

Proces stigne do kraja i aktivira se pravilo

```
when
    Process.state == RUNNING &&
    Process.currentInstruction == len(instructions)
then
    Process.status = TERMINATED
    freeCpuCore
```

Ovo je samo jedan standardni primer. Ulančavanje pravila može ostvariti i veću dubinu ukoliko je neka od instrukcija procesa I/O tipa i/ili se pojavi novi proces većeg prioriteta.

4.3.3 CEP

U sistemu se prate sledeći dogadjaji:

- CpuTemperatureEvent - prati temperaturu procesora i ne dozvoljava mu da se pregreva
  - value - temperatura procesora,

```
when
    CpuTemperatureEvent.value over 1min > dangerousTemp
then
    freeAllCores
    disableCpu for 5 minutes
```

- I/O Event - prati I/O dogadjaje i odblokira proces koji čeka na taj dogadjaj
  - processId - proces za koji je vezan I/O dogadjaj,

```
when
    I/O Event && Process.status == BLOCKED
then
    Process.currentInstruction++
    Process.status = READY
```

- PageFaultEvent - detektuje se page fault, što kao posledicu usporava rad jezgra koje mora da dobavi podatke sa diska
  - processId - proces koji je uzrokovao PageFault

```
when
    PageFaultEvent
then
    changeRelatedCpuCoreState
```

U cilju otkrivanja trashing šablona u izvršavanju procesa, posmatra se i količina javljanja ovih dogadjaja u odredjenom vremenskom intervalu i primenjuje se sledeće pravilo:

```
when
    count(PageFaultEvent) over 2s > 5 &&
    SystemState.availableMemory < $criticalLimit
then
    suspendProcessWithLowestPriority from PageFaultEvents
```

4.3.4 Template

Kako bi se sprečilo gladovanje procesa postoji pravilo koje će procesu pojačavati prioritet u zavisnosti od toga koliko dugo čeka bez izvršavanja.

```
when
    Process.priority < $highValue &&
    Process.priority > $lowValue &&
    Process.status == READY &&
    timeSinceLastStatusChange > $threshold
then
    Process.priority += $increment
```

Vrednosti highValue, lowValue, threshold i increment bi se uzimale kroz eksterni spreadsheet, što bi korisnicima omogućavalo proizvoljno definisanje pravila za sprečavanje gladovanja.

S namerom da se omogući fleksibilna politika upravljanja prijemom procesa (kako npr. procesi sa većim prioritetom ne bi patili od preterane količine PageFault-ova), definiše se template koji prima procese na osnovu procenta dostupne memorije u sistemu.

```
when
    Process.status == NEW &&
    Process.priority > $lowValue &&
    Process.priority < $highValue &&
    Process.memoryRequirement <= SystemState.availableMemory &&
    SystemState.availableMemory / totalMemory
    < $allowedMemoryUsage
then
    Process.status = READY
    reduceAvailableMemory
```

Zadavanjem vrednosti lowValue, highValue i allowedMemoryUsage potencijalno omogućavamo da procesi nižeg prioriteta mogu da se učitaju u memoriju i pri velikom ukupnom zauzeću same memorije, dok to možda neće biti slučaj za procese visokog prioriteta.