CPU ütemezés virtualizációs környezetekben

Virtualizációs technológiák és alkalmazásaik
VIMIAV89

Salánki Ágnes
UC7QSD

2010. január 15.

1. Néhány szó az ütemezésről

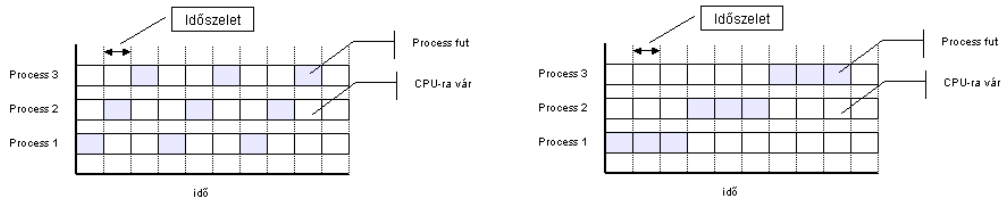
Az ütemező feladata nem más, mint meghatározni a folyamatok futási sorrendjét bizonyos algoritmus alapján. Virtualizált környezetben nyilván nem csak a guest operációs rendszerek ütmezőjéről beszélhetünk, hiszen maguk a virtuális gépek is osztoznak az erőforrásokon. A guest gépek egy virtuális CPU-t érzékelnek, ezeket a vCPU-kat kell ütemezés során fizikai CPUk-hoz rendelni. Ezt a leképezést nehezebbé teszi, hogy a virtuális CPU-k száma nem mindig egyezik meg a fizikaiak számával, ráadásul pl. emuláció esetén architektúrájuk is eltérhet a futtatható CPU-kétól.

A virtuális gépek ütemezése különböző stratégiákkal valósítható meg, az algoritmusok többsége a gépeket valamilyen statikusan meghatározott vagy dinamikusan változó súly, prioritás alapján rangsorolja be.

2. Ütemezéssel kapcsolatos fogalmak

2.1. Preemptív ütemező

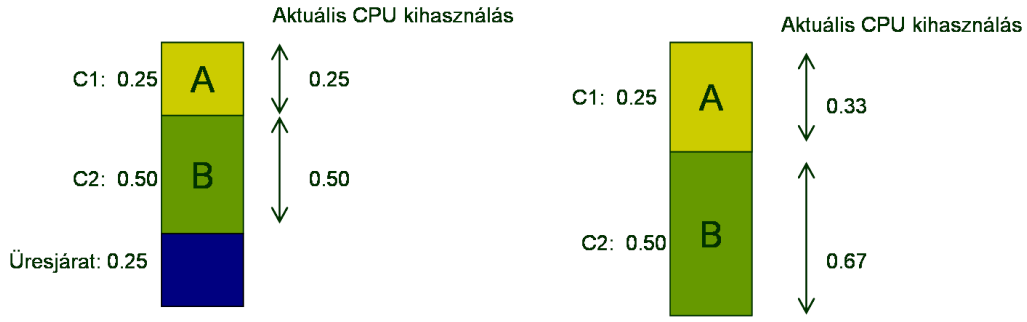
Az aktuális futó folyamat futását az ütemező felfüggesztheti. Nem preemptívnél újraütemezést ilyen esemény nem válthat ki, az újraütemezés akkor következik be, ha az aktuális folyamat lemond a CPU-ról, esetleg várakozni kényszerül a rendszer vagy valamilyen periféria válaszára stb. A 2.1 ábrán három folyamat ütemezése látszik, előbb preemptív, majd nem preemptív módban. [1]



1. ábra. Folyamatok ütemezése: előbb preemptív, majd nem preemptív módban

2.2. Work-conserving mód

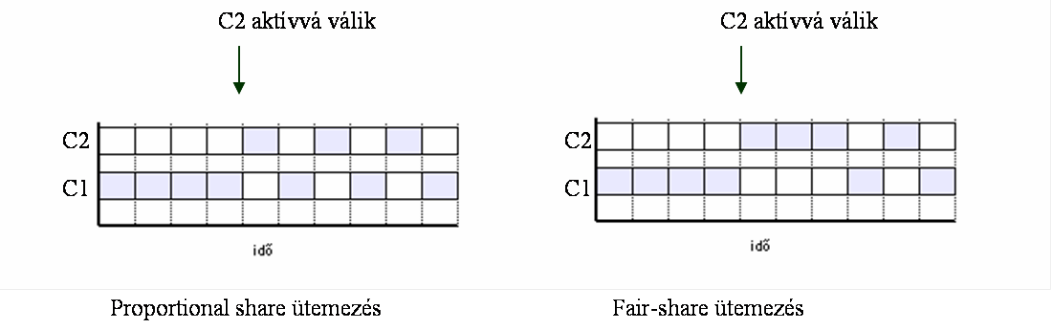
A munkakonzerváló *(wc)* és nem munkakonzerváló *(nwc)* ütemezők abban különböznek, milyen módon használják ki, ha valamilyen erőforrás nincs egészen leterhelve. Munkakonzerváló esetben a CPU csak akkor szabad, ha egyáltalán nincs futó kliens a rendszerben. Például, ha van egy aktív C_1, és egy passzív C_2 kliensünk, a munkakonzerváló ütemezésnél a C_1 megkapja a teljes CPU-időt, nem munkakonzerváló esetben csak a neki előre beállított időt használhatja fel, a C_2 idejében a CPU kihasználatlan marad. Az algoritmus elve jobban látszik akkor, ha a felhasználó által használt súlyok is relevánsak, például a 2.2 ábrán.



2. ábra. Kliensek ütemezése: előbb nwc, majd wc-módban. A felhasználó által beállított prioritás C_1: 0.25, C_2: 0.50. Látható, hogy wc-módban a prioritások csak azt mondják meg, a CPU-időt *milyen súlyozással* (1:2) osztja ki az ütemező, míg nwc-módban a *teljes CPU-időhöz való arányt* szabjuk meg.

2.3. Arányosan osztó ütemezési stratégia

Népszerű és egyszerű ütemezési stratégia. A CPU-időt a virtuális gépek előre beállított súlyozása szerint osztja fel. Abban az esetben, ha C_1 és C_2 kliens azonos ütemezési súllyal rendelkezik, minden esetben azonos módon lesznek kezelve. Ezzel ellentétben léteznek a teljes korrektséget biztosítani akaró ütemezők, amelyek bizonyos szituációkat máshogy kezelnek. Például, ha C_1 kliens éppen aktív, és a C_2 később vált azzá, a C_2 kliens nagyobb CPU-időt kap kezdetben, hogy „utol tudja érni" a C_1 klienst. A 2.3 ábrán mindkét ütemező stratégiája jól látszik.



3. ábra. Kliensek ütemezése: előbb arányosan osztó *(proportional-share)*, majd teljes korrektséget biztosító *(fair-share)* algoritmus szerint.

2.4. Terhelés-kiegyensúlyozás

Többprocesszoros rendszereknél előfordulhat, hogy bizonyos folyamatok adott CPU-hoz vannak kötve. Így előfordulhat, hogy az egyik processzorra több folyamat várakozik, míg a másik üresjáratban fut. Ezt megelőzendő, folyamatok migrációja következhet be. Ezáltal elkerülhető, hogy a várakozó folyamatok szükségtelen várakozási időt halmozzanak fel és a teljesítmény akár a felére csökkenjen. A migrációnak vannak negatív következményei, például a cache teljesítményéből származó munkamennyiségre a gyakori migráció rossz hatással lehet. Emiatt a migráció csak akkor kötvetkezik be, ha előreláthatólag a migráció valóban javítja a rendszer reakcióidejét és a CPU kihasználtságát.[2]

2.5. Real-time rendszerek

Vannak rendszerek, amikkel szemben valamilyen valós időskálához kötött időkövetelményeket támasztunk. Ezeket hívjuk valós idejű, vagy real-time rendszereknek. Két alapvető fajtája a hard real-time rendszer (biztosítjuk, hogy a kiritkus munkák befejeződjenek időben) és a soft real-time rendszer (azt garantáljuk, hogy a kritikus munkák prioritással futnak).

2.6. Időkezelés

A Xen a guest operációs rendszer felé három időt szolgáltat: a valós időt (processzor órajelével szinkron, nanoszekundumokban mért), a virtuális időt (csak akkor ketyeg, ha a CPU-t az adott virtuális gép birtokolja) és a falióra-időt. A falióra-idő mintegy offszetként van jelen a rendszerben, a kauzalitás megőrzését biztosítja.[3]

3. Virtualizált ütemezés a Xen esetében

3.1. Néhány szó a Xenről

A Xen egy x86-os processzor architektúrára készitett VMM, alapvető céljaként a biztonságos és igazságos erőforrás-menedzsmentet és a minimális plusz futtatási költséget jelölték meg, akár az operációs rendszerek százas nagyságrendben való futtatása esetén is.

A Xen paravirtualizált megoldást valósít meg. Ennek lényege, hogy a guest operációs rendszert módosítjuk a problémát okozó rendszerhívások elkerülése érdekében. Az operációs rendszert úgy változtatjuk, hogy ezeket a kritikus rendszerhívásokat cserélje le valamilyen speciálisra, amely már teljesen biztonságos. A paravirtualizált módszer esetében minimális plusz futtatási költséggel kell csak számolnunk.[3]

3.2. A Xen ütemezése

A Xen azért van különleges helyzetben, mert esetében maga a felhasználó szabhatja meg az ütemezési stratégiát. Ez nyilván az ütemező teljes konfigurálásának felelősségét vonja maga után. A Xen ütemezője a kernel betöltésekor állítható

be. Ütemező alatt a továbbiakban mindig a Xen ütemezőjét értjük, ami a virtuális gépek CPU-birtoklását szabja meg.
A Xenre alapvetően a következő ütemezési stratégiák jellemzőek:

- BVT (`sched=bvt`)
- Atropos (`sched=atropos`)
- Round Robin (`sched=rrobin`)
- sEDF (`sched=sedf`)
- Credit (`sched=credit`)

Az Atropos és a Round Robin a Xen 2.0-s változatára volt jellemző, a 3.0-s változatba a BVT, az sEDF és a Credit ütemezőket építették be, ez utóbbit alapértelmezettként. [1], [6]

3.3. BVT - Borrowed Virtual Time

Az ütemezési stratégia lényege, hogy minden virtuális gép rendelkezik egy „előre foglalt" virtuális idővel. Az ütemező azt ütemezi be elsőnek, akinek a virtuális ideje a legkisebb. A virtuális gépek ideje folyamatosan változik, ennek megfelelően az újraütemezés mindig az aktuális súlyozás alapján történik. Ezen tulajdonságai alapján interaktív, késleltetés-érzékeny virtuális gépek esetén ideális.
Globális paramétere: ctx_ allow - beállítja a kontextusváltás gyakoriságát, a klasszikus ütemezők kvantum paraméteréhez hasonló. A BVT preemptív ütemező, kizárólag *wc*-módban működik. Mint egy-, mint többprocesszoros rendszerek esetén alacsony overheaddel működik. Sajnos azonban *nwc*-módban nem használható, ez virtuális gépek nagy száma esetén némiképp korlátozza használatát .

3.4. sEDF - Simple Earliest Deadline First

Az ütemező minden virtuális géphez hozzárendel két paramétert, ezek a d_i - az az idő, amikor az i. virtuális gép aktuális periódusa befejeződik és r_i - amennyi ideig a virtuális gép a CPU-t még birtokolja. Az ütemező a d_i paraméter alapján dönt, mindig a legkisebb értékűt ütemezi be következőnek (azaz amelyik futásából a legkevesebb van hátra). *Használata:*

```
xm sched-sedf <dom-id> <period> <slice>  <extra> <weight>
```

ahol a *period* és a *slice* értékeket nanoszekundumban kell megadni. Az *extra* paraméter egy flag, ami beállítja, kaphat-e a virtuális gép extra CPU-időt. A *weight* paraméter használatával a virtuális gép *slice* értékét dinamikusan változtathatjuk. Preemptív ütemező, *wc* és *nwc* módban is működik, az ütemezés korrektsége teljes egészében a beállított periódustól és az időszelettől függ. Az ütemezés a multiprocesszoros rendszerek terhelés-kiegyenlítésére alkalmazható.

3.5. Atropos

Az Atropos egy soft real-time ütemező. Garantálja a CPU-idő megfelelő felosztását, azzal a módosítással, hogy a terheletlen CPU-t is felosztja, valamilyen best-effort jelleggel. A késleltetés-érzékeny virtuális gépek pontosságát igyekszik megvalósítani. Minden virtuális gépre az alábbi paramétereket adjuk meg:

- *period* - az az idő, amely alatt a virtuális gép biztosan megkapja a CPU-t.
- *slice* - periódusonként az az időmennyiség, amely alatt a virtuális gép biztosan fut. (Mindkét paraméter nanoszekundomban értendő.)
- *latency* - meghatározza, a virtuális gép aktívvá válása után milyen hamar kell beütemezni.
- *xtratime* - jelzi, hogy üresjárat esetén a virtuális gép kaphat-e extra CPU-időt.

Az adminisztrátor a CPU-idő felosztását nanoszekundumokban és nem arányaiban adja meg, sőt, az ütemezés gyakoriságát is ő állítja be. Elkerülendő veszély tehát az Atropos használatánál, hogy az elérhetőn túl foglaljunk többlet CPU-időt.

3.6. Round Robin

Mint Xen-beli belső ütemező van beépítve. Egyetlen globális paramétere: *rr_slice* , az az időszelet, amelyet minden virtuális gép garantáltan megkap, mielőtt újraütemezés történne.

3.7. Credit

A legújabb ütemezési megoldás. Az ütemezés 30 milliszekundumonként történik meg, a virtuális gépeknél felhalmozott ún. kreditek alapján. Kezdetben minden virtuális gép ugyanannyi kredittel indul, csakhogy számuk a futás során változik: a nagyobb prioritású virtuális gépeké alapértelmezett sebességgel, míg a kisebb prioritású gépeké ennél gyorsabban. Ha valamelyik virtuális gép kreditszáma nulla alá csökken, akkor újra beáll minden vCPU az alapértelmezett értékre, ezzel mintegy újraindítva a folyamatot.
Jellemzője, hogy egyetlen fizikai CPU sem válik szabaddá mindaddig, amíg van futó virtuális gép. Az ütemező *wc* és *nwc* módban is használható: *nwc* módban egy paraméter beállításával meghatározhatjuk, hogy kaphat-e az adott virtuális gép extra CPU-időt, és ha igen, mennyit. A Credit nem preemptív ütemező. *Használata:* `xm sched-credit -d <domain> -w <weight> -c <cap>` . [4]

3.8. Néhány ütemezéssel kapcsolatos rendszerhívás

A leggyakoribb, ütemezéssel kapcsolatos rendszerhívások a következőek:

```
/schedule.c
SCHEDOP_yield
SCHEDOP_block
SCHEDOP_shutdown
getpriority( )
```

```
setpriority( )
sched_getscheduler( )
sched_setscheduler( )
sched_getparam( )
sched_setparam( )
sched_yield( )
sched_get_priority_min( )
sched_get_priority_max( )
sched_rr_get_interval( )
```

A SCHEDOP_ yield hívás hatására az ütmezés azonnal megtörténik, amikor egy új virtuális gép lép be az ütemezői sorba, de az ütemező az aktuálisan aktív virtuális gépet nem állítja le.
A SCHEDOP_ block hatására a hívott virtuális gép kikerül az ütemezői sorból és vissza sem kerül mindaddig, amíg az általa várt esemény be nem következik.
A SCHEDOP_ shutdown a virtuális gép befejezi a munkáját.
A getpriority( ) és setpriority( ) a virtuális gépek prioritását kérdezi le és állítja be. A sched_ getscheduler( ), valamint a sched_ setscheduler( ) a folyamatok ütemezésének beállítására és lekérdezésére szolgál.
A sched_ yield( ) az éppen futó virtuális gépet arra kényszeríti, hogy lemondjon a processzorról. A virtuális gép paraméterei megmaradnak, de mindaddig várnia kell, míg újra be nem ütemezik.
A sched_ getparam( ) és sched_ setparam( ) lekérdezi és beállítja a folyamat prioritását.
A sched_ rr_ get_ interval( ) a folyamat kvantumjának értékét kérdezi le.[5] [6]

3.9. Összefoglalás

A Xen akár több- akár egyprocesszoros rendszerek esetén is nyújt megfelelő ütemezési stratégiát, amelyek viszonylag könnyen felparaméterezhetők, de csak a pontos algoritmus ismeretében. A felhasználó tehát feladatkörének megfelelően optimális ütemezést választhat.

Hivatkozások

[1] „Comparison of the Three CPU Schedulers in Xen”, http://cseweb.ucsd.edu/ dgupta/papers/per07-3sched-xen.pdf, 2009.december 11. 11:00

[2] „The CPU Scheduler in VMware® ESX^{TM} 4”, http://www.vmware.com/files/pdf/perf-vsphere-cpu_ scheduler.pdf, 2009.december 11. 11:15

[3] https://wiki.inf.mit.bme.hu/twiki/pub/InfInf/ InfInfMenHaziVirtualizacio/Virtualizáció.pdf, 2009.december 11. 11:20

[4] CreditScheduler, http://wiki.xensource.com/xenwiki/CreditScheduler, 2009.december 11. 11:22

[5] System calls, http://www.informit.com/articles/article.aspx?p=101760&seqNum=5, 2009.december 11. 11:50

[6] XenSource, http://wiki.xensource.com/xenwiki/Scheduling, 2009.december 11. 12:00