samansafn verkefna og svara

TÖL401G - Stýrikerfi

Sverrir Sigfússon Sigríður Birna Þorvaldur Tumi Bjarni Þór

Yfirlit

1. Umræða um stýrikerfi 1
2. Posix standardinn 1
2.1. Hvað er Posix? 1
2.2. Um hvað fjalla eftirfarandi kaflar í POSIX staðalinum? 1
2.3. Finndu dæmi um… 2
3. Hypervisors 2
3.1. Tímastýring 2
3.2. Sýndarvélar 2
4. Boot Process 2
5. Scheduler 3

1. Umræða um stýrikerfi

Verkefni: Ræðið hvort stýrikerfi séu nauðsynleg. Pælið til dæmis í tækjum sem keyra aðeins eitt forrit án nokkurs inntaks frá notanda. Er nauðsynlegt fyrir þessi tæki að hafa stýrikerfi?

Pínulitlar tölvur með jafnvel bara eitt forrit þurfa ekki endilega stýrikerfi.

Þar sem það er bara eitt forrit að keyra þarf ekki að halda vera með minnistýringu, það er bara einn aðili að nota minnið þannig það þarf ekki að passa að taka það frá og gefa það til baka.

Það þarf ekki að halda utan ferlavinnslu, *"process management"*, þar sem það er bara einn ferill í gangi á hverjum tíma

Segjum að tilgangur þessarar litlu tölvu okkar sé að mæla rakastig í herbergi, það eina sem hún gerir er að senda út hvort rakastigið sé yfir ákveðnu marki. Hér þarf aldrei að skrifa gögn þannig það þarf ekki að halda utan um "mass storage".

Í stuttu máli, þegar verið er að vinna á mjög láum level með kerfi sem eru vel hönnuð ætti ekki að þurfa stýrikerfi fyrir tölvur.

2. Posix standardinn

2.1. Hvað er Posix?

2.1.a. Hvaða stofnanir (fleiri en ein) sjá um skilgreiningu á POSIX staðalinum?

Stöðlunarhópar fyrir POSIX innihalda *Austin Group*, *ISO-hópinn* og *The Open Group*, sjá heimild

2.1.b. Hvað er nafn og númer núverandi POSIX staðals?

Nýjasti staðallinn er **POSIX.1-2017** líka þekktur sem **IEEE Std 1003.1-2017** og var gefinn út 2017.

2.1.c. Um hvað fjallar POSIX í stuttu máli?

POSIX er samansafn af stöðlum til þess að auðvelda tengingu á milli stýrikerfa, þá sérstaklega stýrikerfa byggð ofan á unix.

2.2. Um hvað fjalla eftirfarandi kaflar í POSIX staðalinum?

2.2.a. Base Definitions

“Base Definitions” fjalla um almenn kerfi og “interface” sem eru þekkt í langflestum kerfum. Þetta eru meðal annars staðlar um reglulegar segðir, hausa í C tungumálinu og skilgreiningar á gagnagrunnum.

2.2.b. System Interfaces

“System Interfaces” fjallar um, eins og nafnið gefur til kynna, viðmót sem kerfi sem uppfylla POSIX staðalinn geta gert ráð fyrir. Hér er skilgreint hvernig á að setja fram föll inn í forritum ásamt “macros” og mörgu fleira.

2.2.c. Shell & Utilities

“Shell and Utilities” kaflinn fjallar um skipanair og nytjagögn sem forrit á POSIX kerfum geta nýtt sér. Hér er talað um hvernig á að stýra skipanalínu, finna hvar í skráarkerfi maður er og fleira.

2.3. Finndu dæmi um...

2.3.a. Stýrikerfi sem uppfyllir POSIX staðalinn

Mac OS X 10.8 og seinni útgáfur af Mac OS eru POSIX vottuð

2.3.b. Stýrikerfi sem uppfyllir megnið af POSIX staðalinum

Flest linux stýrikerfi eru mestmegnis POSIX vottuð, mörg þeirra eru ábyggilega alveg POSIX vottuð en staðfesting á því er dýr og það stoppar flest distro í því að ná sér í vottun.

3. Hypervisors

Í báðum verkefnum er verið að skoða *“VM hypervisor”* sem sér um að keyra tvö stýrikerfi OS_1 og OS_2 innan tveggja virtual véla á kerfi með einn kjarna.

3.1. Tímastýring

Verkefni: Verkraðari yfirstýrikerfisins *(scheduler, hypervisor)* hefur gefið OS_1 20ms af CPU tíma. Á meðan OS_1 er að keyra í sínum gefna tímaramma klárast biðtími OS_2 og það stýrikerfi lætur sinn CPU verkraða vita. Lýsið tveimur möguleikum fyrir yfirstýrikerfið til að takast á þessu.

Ein lausn væri fyrirbyggjandi plönun *(preemptive scheduling)*. Hver hlutur fær að keyra í ákveðin tíma, og eftir það er tekinn í burtu og settur í röð. Þannig gæti yfirstýrikerfið truflað keyrslu á OS_1 og farið að keyra OS_2.

Önnur lausn væri að tímastýra kerfunum *(time-sharing)*. Þá leyfir yfirstýrikerfið OS_1 að keyra í 20ms, en minnkar tíma sem það fær í næsta tímaramma til að bæta upp fyrir tímann sem OS_2 missti.

3.2. Sýndarvélar

Verkefni: Þegar verið er að vinna með sýndarvélar eru sýndarvélarnar ótengdar hvor annari. OS_1 keyrandi á vél VM_1 getur ekki nálgast skráarkerfi ytri vélarinnar né skránna á kerfi OS_2 keyrandi á VM_2. Lýsið mögulegri aðferð til að veita vélunum möguleika á að deila skrám

Það er mögulegt fyrir fleiri en eina sýndarvél að deila skrárkerfis-staðsetningu *(file-system volume)* sem gerir þessum sýndarvélum kleyft að deila skrám.

4. Boot Process

Verkefni: Lýsið skrefunum í uppstyllingu kerfisins *(boot process)* alveg þangað til að innskráningar gluggi stýrikerfisins birtist. Leggið áherslu á hvað gerist þegar innra ræsiforritið *(bootstrap)* hefur komið kjarna *(kernel)* inn í RAM

Við getum gert ráð fyrir því að búið sé að hlaða kernel inn í minni eftir skref tvö á bootloader ferlinu. Í hausnum á kernel myndinni er örlítill kóði sem setur upp lágmarks tengingu við vélbúnaðinn, þetta gerir vélinni kleift að uncompressa kernelinn og setja hann í high memory.

Eftir að búið er að klára vinnslu á kernel, er skráarkerfið fest sem leyfir kernelinu að sjá og nálgast nauðsynlegar skrár. Eftir það er keyrt frumstyllingarforrit sem setur upp kerfiseiningar, net, skráarkerfi og þesshättar. Að lokum keyrir frumstylliforritið upp notendaviðmótið á samt fleiri kerfiseiningum. Þegar öll þessi skref eru klárað er process 1 keyrt upp.

5. Scheduler

Verkefni: Útfærðu scheduler aðferð sem tekst á eftirfarandi aðstæðum:
- Timer interrupt - Aðferðin fær merki um að tímarammi hafi klárast
- I/O syscall - Aðferð sem er að keyra biður um **I/O**
- I/O interrupt - Tæki lætur vita að **I/O** hafi klárast

```
// kóðinn sem var gefinn
schedulerUnfinished() {
  if (called by timer interrupt) {
    // Time slice of current process expired
   } else if (called by I/O system call) {
    // I/O request by current process
  } else if (called by I/O interrupt) {
    // I/O of process ioCompleted completed
  }

  // Further code outside if statements (if required)
}

// kóðinn sem var skilað
schedulerFinished() {
    if (called by timer interrupt) { // Time slice of current process expired
        addToTail(running, ready);
    } else if (called by I/O system call) { // I/O request by current process
        addToTail(running, waiting);
    } else if (called by I/O interrupt) { // I/O of process ioCompleted completed
        addToTail(running, waiting);
        findAndRemove(ioCompleted, waiting);
    }

    // Further code outside if statements (if required)
    interruptOn()
    if (ready == null) {
        sleep()
    } else {
        running = removeFromHead(ready);
        switchTo(running)
    }
}
```