

# Prosessit ja tehtävät

Ohjelmoitavien logiikoiden tapaan myös Actiweb järjestelmän tärkeimpiä osiaon sovellusohjelmien ajaminen säännöllisesti ja luotettavasti. Jokaista tällaista itsenäistä ohjelmaa kutsutaan ohjausjärjestelmien yhteydessä usein tehtäväksi, mutta käyttöjärjestelmien tapauksessa samantapaista ohjelmistokokonaisuutta kutsutaan joskus prosessiksi

Actiweb **ei** seuraa IEC61131 standardia ohjausjärjestelmien ohjelmoinnista, vaan sovellusohjelmia kirjoitetaan Lua kielellä. Toisaalta, Lua kieli muistuttaa hieman edellämäintun standardin sisältämää ST-kieliltä. Muilta osin Actiweb-järjestelmä muistuttaa paljon IEC-standardin kuvaamaa ohjausjärjestelmää, jossa sovellusohjelmat käynnistetään toimimaan n.s. taskeina, joita suoritetaan tarkasti määrättyllä suoritusvälillä niin kauan kuin järjestelmä on toimintakykyinen.

Actiweb tukee nykyisessä versiossa kahta eri sovellusohjelman ajoitusmallia: "cyclic" ja "periodic", joissa on lopulta vain pieni ero.

**Periodic** tarkoittaa tasavälistä ajoitusta, siinä tehtävä käynnistetään aina samalla aikavälillä (mikäli prosessorin suorituskyky sen sallii). Eli mikäli tehtävän aikaväliksi määritetty 1000 ms (1 sekunti), ja tehtävän suorittaminen kestää 250 ms, odottaa kyseinen tehtävä 750 ms lepotilassa ennen seuraavaa käynnistystä. Tämä "lepoaika" lasketaan jokaisen suorituskerran jälkeen, ja järjestelmä koettaa pitää suorituskertojen (syklien) alkamishetket mahdollisimman tarkasti asetetun mukaisina.

**Cyclic** tarkoittaa että tehtävän suorittamisen jälkeen tehtävä asetetaan lepotilaan annetuksi ajaksi – esimerkiksi mainittu 1000 ms. Tällä tavalla voidaan varmistaa että prosessorille jää aikaa suorittaa muita toimia, ja sitä voidaan hyvin käyttää mikäli tehtävän tarkka ajoitus ei ole aivan kriittinen.

Useinkaan suoritustilan valinnalla ei ole kovin suurta merkitystä, mikäli itse ohjelmat on kirjoitettu niin että ne eivät oleta täysin tasaista suoritusväliä. Usein tärkeintä on, että ohjelma ajetaan riittävän usein ohjattavaan prosessiin nähden. Sovellusohjelmia kirjoitettaessa kannattaa huomata myös se, että esimerkiksi määrätyn pituisen viiveen tekeminen ei vaadi tarkkaa ohjelman suoritusväliä, vain riittävän sovelluksen kannalta riittävän tiheän suoritusvälin. Prosessori ja käyttöjärjestelmä pitävät taustalla käynnissä hyvin käyttökelpoista nanosekunti-tason tarkkuusajastinta ja reaaliaikakelloa, joiden avulla esimerkiksi ajastukset ja viiveet on helppo tehdä. Niitä voi hyödyntää sovellusohjelmissa API kutsuilla:

```
System.nanotimer ()  
os.time()  
os.date()
```

Jos tarkkaa ohjelmien ajoitusta tarvitaan, **periodic** -tilassa ohjelman ajoituksen huojunta (niin sanottu jitter) on Cortex A8 prosessorille käännettyllä versiolla suuruusluokassa +/- < 50 us

(mikrosekuntia) mikäli prosessorin kuormitus pysyy alle 80 %. Pitkän ajan vakaus riippuu täysin kellosignaalin lähteen vakaudesta (esimerkiksi AM3358 prosessorin tapauksessa sisäisen oskillaattorin stabiilius +/- 50 ppm).

Tavallinen tapa tehdä uusi ohjelmaprosessi, on luoda lua -päätteinen tiedosto /opt/slc/prg/run -hakemistoon. Käynnistyksen yhteydessä järjestelmä skannaa edellämainitun hakemiston lua-päätteiset tiedostot, ja käynnistää ne oletusasetuksilla, jotka ovat:

**cyclic** -suoritustapa, 1000ms suoritussväli.

Näitä asetuksia on mahdollista muuttaa käyttämällä API kutsuja:

**Slc.setSchedule (strMode)**

**Slc.setTiming (iDelay)**

Joskus on hyödyllistä käynnistää yhdestä ohjelmatiedostosta useampia ohjelmaprosesseja – joita kutsutaan usein PLC järjestelmien yhteydessä tehtäviksi, tai englanninkielisellä termillä "task". Uuden ohjelmaprosessin saa käynnistettyä käyttämällä API kutsua:

**Slc.createTask ( strFile, strName, strCall, strMode, delay)**

Jokainen task eli tehtävä on siis oma käyttöjärjestelmä prosessinsa, tai tarkemmin sanottuna säie. Jokaiselle säikeelle on varattu oma muistialueensa, ja Lua ohjelmat toimivat täysin toisistaan riippumatta. Se tarkoittaa sitä, että säikeet eivät suoraan näe toistensa muuttujia tai funktioita. Tämä tekee tavallisesti ohjelmista vakaampia, ja helpottaa niiden kirjoittamista. Helpoin keino tietojen vaihtamiseen Lua ohjelmien välillä on pistetietokanta. Monimutkaisempi mutta mahdollinen tapa on käyttää väliaikaistiedostoa tai käyttöjärjestelmän "nimettyjä putkia", jotka matalalla käyttöjärjestelmätasolla ovat tavallaan myös väliaikaisia tiedostoja.

Samanaikaisesti käynnissä olevien Lua ohjelmien maksimimäärä on sama kuin käyttöjärjestelmän prosessien maksimimäärä, joka selviää tiedostosta

*/proc/sys/kernel/pid\_max*

Raja on usein 32 768 prosessia, mutta tyypillisesti muut suorituskäkyyn liittyvät seikat, kuten muistin määrä tai prosessoriteho tulee vastaan ennen varsinaista prosessien määrää.

Käyttökelpoisia Linux komentoja:

**free** Ilmoittaa vapaan muistin määrän

**df** Ilmoittaa vapaan levytilan määrän

Sovelluksesta riippuen, sekä muistinkulutusta että vapaata levytilaa hyvä seurata, sillä mikäli sovellusohjelmisto kerää paljon esimerkiksi historia-dataa, voi levytila tai muisti loppua helposti kesken.