Softasuunnittelusta

Raapaisu syvään päätyyn

Tavoite

Kehoittaa koodaamaan tietyllä tyylillä

Muistuttaa haasteista

Mitä on softa?

Koodia joka suoritetaan

Vaikka olisi tulkattu kieli, sekin suorittaa koodia

Mitä on koodi? Jotain fyysistä joka alustalla on, esim bittejä, tavuja, muistissa

Lineaarinen

Miten softaa suoritetaan

Raudalla

Luetaan käsky, suoritetaan käsky, kirjoitetaan tulokset

Ben Eater YouTubessa jne.

Mitä asioita halutaan (pitkän ajan kokemuksella tiedetään)

Osiointi

Modulaarisuus

Joustavuus

Muokattavuus

Samanaikaisuutta (rinnakkaisuutta, tai ainakin jotain sinne päin tai siltä vaikuttavaa)

Koodilla on muoto

Koodi suorittuu järjestyksessä

Lineaarisen rakenteen takia ~kaikkialla on oikeasti silmukka pohjalla

```
:start
instructions …
jmp .start
```

```
while(1)
{
	koodia…
}
```

Pieni vale

Jos ymmärtää hardiksen ja osaa koodata, koodin voi strukturoida toimimaan ilman silmukkaa, jos hardis sallii sen

Kuitenkin esim C:ssä määrätyn toiminnallisuuden takia ~pakollinen

Sen sijaan tila ja tilakierto on aina silmukoituva

Eli oikeasti aina silmukka sittenkin

Noin n “erilaista” tapaa strukturoida koodia

Synkroninen

Kooperatiivinen

Piilotettu kooperatiivinen

Tapahtumapohjainen

Asynkronisuus ja blokkaavuus

Jonkun muun hallitsema

Oikea rinnakkaisuus

Synkroninen

Helpoin ymmärtää

```
while(1)
{
	doX();
}
```

Kooperatiivinen

Helppo ymmärtää

```
while(1)
{
	doX();

	yield(); //delay(1000);
}
```

Konteksti tallennetaan

Piilotettu kooperatiivinen

Kompleksisuus kasvaa

```
async … {
        while(1)
        {
                doX();
                await doY();
        }
}
```

Konteksti tallennetaan

Tapahtumapohjainen

someObject.addEventListener(“eventName”, functionName);

```
ISR(NIMI_vect)
{
	koodia…
}
```

signal(SIG_PIPE, functionName);

Rekisteröidään koodia, joka suoritetaan sitten myöhemmin kun siihen on aihetta

Async-koodin ja synkronisen koodin sekoittaminen vaarallista

Asynkronisuus vs ei-blokkaava

Blokkaava: odottaa kunnes saa mitä halutaan tai voidaan antaa virhe

Ei-blokkaava: suoritus voi kestää, mutta ei odota jos ei ole antaa vastausta

Synkroninen: suoritetaan heti

Asynkroninen: palautuu heti, suoritus tapahtuu taustalla, jos ei virhettä

Asynkronisuus on yleensä “vain” hardistason synkronisuutta
    Tai vain piilotettu softassa

Monet asiat taiottu piiloon kehitysympäristön/kääntäjän puolesta

Jonkun muun hallitsema

Koodi missä vain muodossa, yleensä synkroninen

Mutta … ? Taikaa!

Koodi suorittuu taianomaisesti vain jonkin aikaa tai vain jossain tilanteissa

Rekisteröidään yleensä funktio, jota ajastussysteemi hallitsee ja ajaa tarvittaessa

Käytetään yleensä hardisresursseja

Tulkkauksessa ajastustarkastelu tulee “natiivisti”, koska joka käsky tarkastetaan

Linux, FreeRTOS, … (hosted vs barebones)

Jonkun muun hallitsema

```
func … (...)
{
	while(1)
	{
		doX();
		if(condition)
		{
			break;
		}
	}
	vTaskDelete(NULL);
}
```

```
xTaskCreatePinnedToCore (taskFunc, "taskName", taskStack, &taskArg, taskPriority, &taskHandle, taskCode);
```

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);
```

Ajastus (scheduling)

Tapahtumapohjainen

Tehtävä/ohjelma herätetään/ajetaan kun jokin tapahtuma, jota se odottaa, tapahtuu

Aikapohjainen

Annetaan ajaa n millisekuntia

Prioriteetti

Oikea rinnakkaisuus

API:n puolesta vähän kuin tapahtumapohjainen

Suositus yhdistelmä tapahtumia ja rajallisia pitkiä tehtäviä

Tapahtumakierto (event loop) jossa nukutaan/odotetaan tapahtumia

Luonnollisen tehokas

Joustava

Laajennettava

Ei “valehtele” vaan pitäytyy todellisuudessa (ajastettu pyörii n millisekuntia, jaetaan kaikkien kesken)

Raskas laskenta tai odotusta tarvitseva tehtävä loogisesti ryhmitelty

C (eri muodot) vs Js

C:ssä rakennettava itse asioita

Js annettu valmiiksi event loop

Todennäköisesti kirjoitatte paljon C/++

Ei ole pakko käyttää C/++, voitte myös käyttää muitakin kieliä

En silti suosittele

Raskas rajoitettu laskenta voidaan jakaa osiin

Strukturoi koodi ja algoritmit osioitaviksi ja jatkettaviksi

```
for(i=0; i<1000000000; ++i)
{
        doX();
}
```

vs

```
for(i=start; i<limit && timeNow()<stopTime; ++i)
{
        doX();
}
```

Vaatii hieman miettimistä rakenteen puolesta, mutta lähes aina tehtävissä pienellä vaivalla

Lisää epätehokkuutta

Virtaoptimisointi

Tehdään mahdollisimman vähän koodissa, ts. ei luupata loputtomasti luottaen pelkästään ajastukseen

Nukutaan aina kun mahdollista

Nukutaan mahdollisimman syvästi aina kun mahdollista

Tapahtumamalli sopii luontaisesti tähän, hardis toteuttaa myös tätä natiivisti

Helposti laajennettava :)

Helposti kompleksinen : (

Turvallisuus (integriteetti)

Moniajo luo huomattavan määrän kompleksisuutta

“Korruptio” ja korruptio

Synkronisointi

Muuttujien suojaaminen

Kriittiset sektiot

Jne hirveästi asioita

IPC

Tärkeimmät huomiot

Speksien perusteella toiminta

Määrittäkää tarkasti mitä pitää tehdä

Käyttäkää apia joka antaa yksinkertaisimman ja laajennettavimman mallin

Jos ei tarvitse miettiä virranhallintaa

Seuratkaa yleisiä parhaita toimintamalleja