Heisrapport, gruppe 65

Aksel Sundal, Magnus Mæhlum

March 1, 2020

1 Beskrivelse av prosjektet

For å implementere tilstandsmaskinen brukes et funksjonspekerarray som består av fem funksjoner - én funksjon for hver tilstand. Med følgende enum blir dette ganske enkelt; funksjonene kan enkelt kalles med én linje kode.

```
typedef enum {
    BOOT,
    IDLE,
    MOVE,
    DOORS_OPEN,
    STOP,
} State;

void (state_functions[NUMBER_OF_STATES])(State *p_now_state) = {
    elevator_boot_state,
    elevator_idle_state,
    elevator_move_state,
    elevator_doors_open_state,
    elevator_stop_state,
};

// Kalles i en while(1) i main()
elevator_state_functions[state](&state);
```

Timer-modulen (brukes av: elevator.c)
Timer inneholder kun én funksjon, timer_check_timeout(), som sjekker om en timeout på en viss tid har funnet sted. Dette er å foretrekke framfor en delay-funksjon, da delay-funksjoner er busy waiting, og vi ønsker å kunne kalle andre funksjoner mens heisen venter på en timeout.

Queue-modulen (brukes av: elevator.c)
Queue inneholder en struct Request, som oppsummerer en forespørsel til heisen. Modulen har fem funksjoner, opplistet under. Hva de gjør kan entes forstås ut i fra navnene deres, eller leses mer om i tilhørende Doxygen-fil.

- queue_init()
- queue_check_and_add_requests()
- queue_remove_requests_on_floor()
- queue_get_next_floor()
- queue_flush()

Elevator-modulen (brukes av: main.c)
Elevator inneholder implementasjonen av den endelige tilstandsmaskinen, og blir derfor kjernen i programmet. Det er allerede beskrevet hvordan selve tilstandsmaskinen er implementert. I tillegg har modulen en rekke viktige lokale funksjoner og variabler.

- check_and_update_new_floor()
- transition_state()
- attempt_doors_open()
- elevator_last_movement
- elevator_current_floor
- elevator_next_floor

Av disse bør transistion_state() forklares nærmere. Funksjonen fungerer som et slags lookup-table for tilstandsmaskinen; dersom en transisjon som ikke er definert i tilstandsdiagrammet forsøkes vil det skrives ut en feilmelding og programmet vil terminere. På denne måten unngås feil i tranisjoner og utviklere/vedlikeholdere av programmet tvinges til å tenke over hvordan tilstandsmaskinen er tiltenkt.

2 Begrunnelser for viktige avgjørelser

Bruken av funksjonspekerarray

Det var to åpenbare metoder for å implementere tilstandsmaskinen: funksjonspekerarray og switch/case blokker. Til å begynne med brukte vi sistnevnte, men vi la merke til at blokkene med kode ble veldig store og uoversiktelige. Vi vurderte å lage én ny funksjon til hver case, men i det tilfellet ville det være mye lettere å bare legge de til i et funksjonspekerarray og kalle funksjonene basert på tilstands-enumen.

En stor fordel med denne løsningen er at koden blir langt ryddigere. En ulempe er at hver "tilstandsfunksjon" må returnere og ta inn akkurat det samme for å kunne legges til i samme funksjonspekerarray. For at hver tilstandsfunksjon skal ha tilgang til variablene de behøver kan det enten passes flere argumenter inn i alle tilstandsfunksjonene (som fort blir rotete!), eller bruke modulære variabler.

Design av moduler

Modulene er designet for å ta for seg en spesifikk del av programmet, og ha et så minimalt grensesnitt seg i mellom som mulig. Til å begynne med vurderte vi å ha flere moduler, som en utilities-modul, men fant ut at det ikke ble nødvendig og kun ville kludret til klassediagrammet vårt.

3 Unit testing

For å gjennomføre unit testing tok vi for oss hvert spesifikasjonskrav og testet om heisen oppfylte det. Her velger vi å kommentere hver kategori av spesifikasjonskrav.

1. Oppstart
    (a) O1 - helt sikkert.
    (b) O2 - helt sikkert.
    (c) O3 - OK.
2. Håndtering av bestillinger
    (a) H1 - høyst sannsynlig.
    (b) H2 - høyst sannsynlig.
    (c) H3 - OK.
    (d) H4 - helt sikkert.
3. Bestillingslys og etasjelys
    (a) L1 - helt sikkert.
    (b) L2 - helt sikkert.
    (c) L3 - helt sikkert.
    (d) L4 - helt sikkert.
    (e) L5 - helt sikkert.
    (f) L6 - helt sikkert.
4. Døren
    (a) D1 - helt sikkert.
    (b) D2 - helt sikkert.
    (c) D3 - høyst sannsynlig.
    (d) D4 - helt sikkert.
5. Sikkerhet
    (a) S1 - helt sikkert.
    (b) S2 - høyst sannsynlig.
    (c) S3 - høyst sannsynlig.
    (d) S4 - helt sikkert.
    (e) S5 - helt sikkert.
    (f) S6 - helt sikkert.

(g) S7 - helt sikkert.

6. Robusthet

(a) R1 - helt sikkert.

(b) R2 - høyst sannsynlig.

(c) R3 - helt sikkert.

4 UML-diagrammer

Merk at klassediagrammet ligger vedlagt som en seperat .pdf-fil, da det ble veldig vanskelig å lese hvis det var lagt til her.

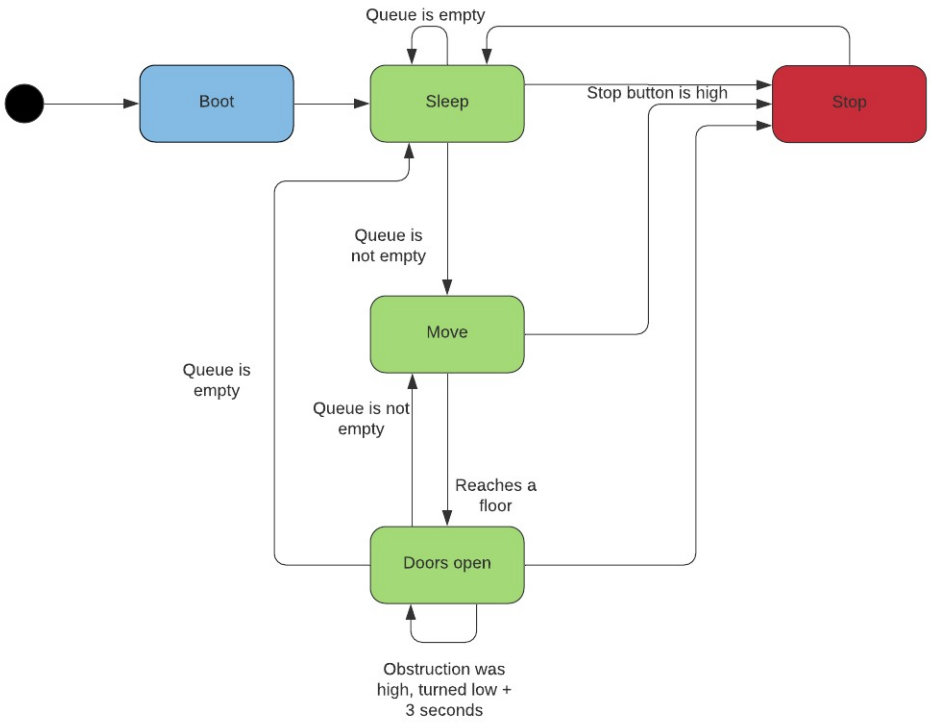

Figure 1: Tilstandsdiagram

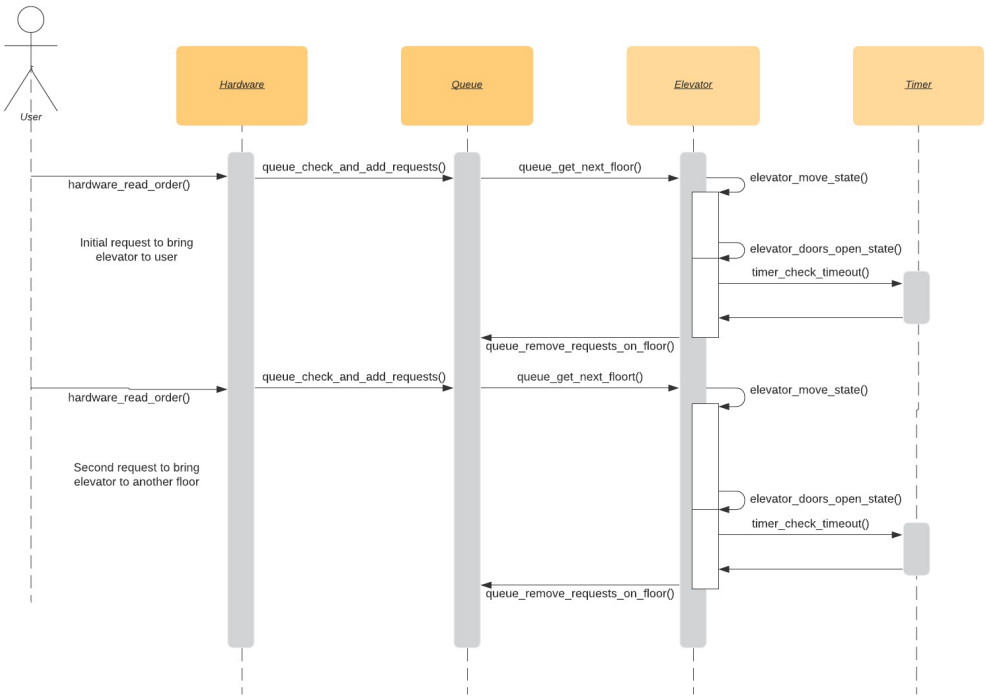

Figure 2: Sekvensdiagram