



Synopsis:

Titel:

Modulært jukebokssystem

Projekt Semester:

Afgangprojekt, Efteråret 2009

Forfatter:

Jesper Rohde Vestergaard

Vejleder:

Jens Dalsgaard Nielsen
Tatiana Kozlova Madsen

Virksomhedskontakt:

Mads Lange

Oplag: 6

Sider: 49

Appendix: 10

Afleveret: 7. Januar 2010

Denne rapport omhandler udviklingen af en modul baseret jukebokssystem. Projektet bygger på en use-case fra virksomheden Event-Danmark der har et repertoire af indspillede akkompagnementer de gerne vil have designet et jukebokssystem op omkring.

Rapportens primære fokus er at lave en prototype bestående af en klient og en vært hvor klienten står for brugerinteraktion og værten indeholder sangene og en afspiller.

I foranalysen blev der fundet frem til to krav for systemrespons-tiderne til systemet. Det første krav er 100 millisekunder for simple kommandoer og det andet er 720 millisekunder for avancerede kommandoer.

Systemet blev designet i overensstemmelse med kravspecifikation og stress-testet for at se hvordan det opførte sig under høj CPU belastning på både klient og vært, og under begrænset netværks båndbredde.

Det blev konkluderet at det er muligt at konstruere en sådant system men at systemrespons-tiderne kun kan overholdes under særlige forhold.

Læsevejledning

Igennem rapporten vil forkortelser første gang de optræder blive nævnt med forkortelsen efterfulgt af den fulde beskrivelse i parentes

Det kan eksempelvis være CPU (central Processing Unit)

Kildehenvisninger er placeret ved slutningen af en sætning ved reference til sætningen. Ved reference til et afsnit er henvisningen placeret i begyndelsen af afsnittet.

Indhold af CD-ROM

- Denne rapport i pdf format
- Kildekoden fra implementetation
- Opsamlet data fra tests
- Python scripts til generering af grafer

Forfatterens underskrift

Aalborg Øst, 7. Januar 2010

Jesper Rohde Vestergaard

Indhold

1	Indledning	7
1.1	Use-case	8
2	Foranalyse	11
2.1	Systemresponstid	11
2.2	Tynd eller tyk klient	12
2.3	Identificering af datakommunikation	13
2.4	Delkonklusion	14
3	Problemformulering	15
3.1	Problemformulering	15
4	Hardware og Software	17
4.1	Hardware	17
4.1.1	Vært	18
4.1.2	Klient	18
4.1.3	Display	19
4.1.4	Pulsgiver	20
4.1.5	Knapper	20
4.2	Software	20
4.2.1	Vært	20
4.2.2	Klient	21
5	Kravspecifikation	23
5.1	Kravspecifikation	23
5.1.1	Musikdata	23
5.1.2	Musikkontrol	24
5.1.3	Forbindelsesbevidsthed	24
5.1.4	Fejlmeddelelser	24
6	Design	25
6.1	Systemdesign	25
6.1.1	Komponenter	25
6.1.2	Grænseflader	25
6.2	Programdesign - Vært	27
6.2.1	Netværksprocessering	27
6.2.2	Applikationslogik	27
6.2.3	Musikafspilning	27
6.3	Procesdesign - Vært	27
6.3.1	JukeProtocol	27
6.3.2	JukeFactory	28
6.3.3	ApplicationLogic	28
6.3.4	Player	29
6.4	Programdesign - Klient	30
6.4.1	Udgående netværkskommunikation	30
6.4.2	Indgående netværkskommunikation	30
6.4.3	GPIO håndtering	30
6.4.4	Displayvisning	31

INDHOLD

6.5	Procesdesign - Klient	31
6.5.1	queue.c	31
6.5.2	gpio.c	32
6.5.3	network.c	32
6.5.4	disp.c	32
6.5.5	main.c	33
6.6	Implementation	33
7	Test	35
7.1	Testopstilling	35
7.2	Stress-test	36
7.2.1	Klient	36
7.2.2	Vært	38
7.2.3	Forbindelse	40
7.2.4	Brugerinteraktion	42
7.3	Delkonklusion	43
8	Konklusion	45
8.1	Konklusion	45
9	Perspektivering	47
A	Appendiks	49
A.1	Resultater	49
A.1.1	Tests	49
A.2	Vært	49
A.2.1	main.Player Class Reference	49
A.2.2	main.JukeProtocol Class Reference	51
A.2.3	main.JukeFactory Class Reference	52
A.2.4	main.Application Class Reference	52
A.3	Klient	53
A.3.1	main.c	53
A.3.2	Msgbuf Struct	54
A.3.3	queue.c	54
A.3.4	gpio.c	55
A.3.5	disp.c	56
A.3.6	network.c	57
	Litteratur	59

KAPITEL 1

Indledning

De færreste festmusikere kan spille det fulde repertoire af de danske festsangsmelodier. Det kan lede til at musikerne må undlade at akkompagnere numrene, eller prøve at gætte sig frem til hvordan en melodi skal spilles.

En anden mulighed er at benytte færdigindspillede akkompagnementer, hvilket giver en musiker eller dj mulighed for at give kunderne den melodi sangen er skrevet til.

Denne rapport vil omhandle udviklingen af et modulært jukebokssystem til at løse ovenstående problemstilling ved at stille færdigindspillede akkompagnementer, i en kompakt indpakning, til rådighed for brugerne.



Figur 1.1 Sir Juke jukeboksen fra Event-Danmark

1.1 Use-case

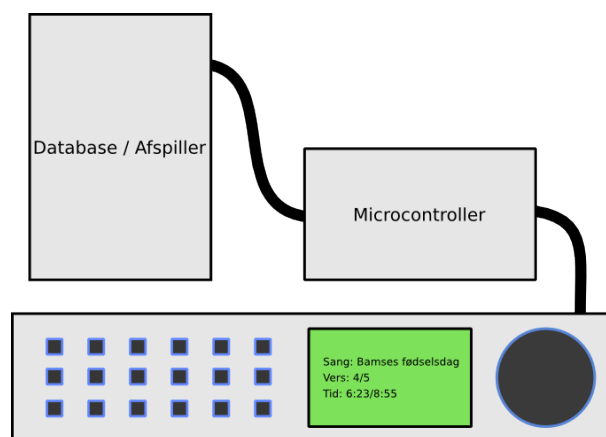
Virksomheden DoréDevelopment har til firmaet Event-Danmark udviklet et jukebokssystem kaldet Sir Juke. Sir Juke, som kan ses på figur 1.1, er en digital jukeboks med tusindvis af musiknumre liggende. Den har samtidig den unikke funktionalitet, at den også indeholder professionelt indspillet akkompagnement, til flere hundrede familiesange, så det er muligt at synge konfirmandens 12 vers lange takkesang til sin familie uden brug af musikere.

Dette repertoire af festsange vil Event-Danmark udnytte i andre sammenhænge. Derfor vil Event-Danmark have denne samling af akkompagnement integreret i et system på en et unit rack formfaktor, så de kan stille den til rådighed for eksempelvis musikere og dj's.

Det samlede system skal bestå af tre enheder; en musikafspiller med en tilknyttet database, et frontpanel og en mikrokontroller til at forbinde musikafspilleren med frontpanelet. En illustration af denne opstilling kan ses på figur 1.2.

Musikafspilleren vil have sangene inkorporeret, samt en lydudgang til afspilning. Frontpanelet vil bestå af et display, en række programmerbare knapper og en pulsgiver med push-to-enter funktionalitet, til at vælge sange med.

Musikafspilleren vil efterfølgende blive refereret til som vært og mikrokontrolleren med frontpanelet vil blive nævnt som klient.



Figur 1.2 *Produktopstilling*

Brugeren af systemet skal kunne bruge displayet og pulsgiveren til at browse og udvælge sangene der skal afspilles, samt vælge hvor mange vers afspilleren skal spille af den valgte sang. Dette skal kunne gøres uden at brugeren får følelsen af en synlig forsinkelse mellem brugerens input og systemets output på displayet.

På værten ligger alle sangene placeret og det er denne enhed der skal afspille dem. De respektive informationer som f.eks titel, om hver enkelt sang ligger i en database tilknyttet værten.

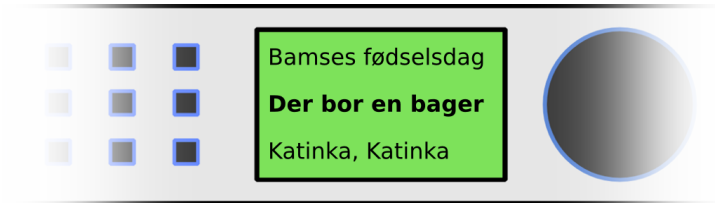
Forbindelsen mellem værten og klienten skal enten være af typen USB eller Ethernet og hvis forbindelsen forsvinder, skal det detekteres af både værten og klienten.

1.1.0.1 Scenario 1

Et typisk scenario for systemet er, at en bruger vil vælge en titel og starte afspilningen af den. Først vælges sangen ved at bladre mellem de titler der er til rådighed. På displayet vises tre titler af gangen og brugeren kan styre bladringen med pulsgiveren. En grafisk model af dette LCD (Liquid Crystal Display)

billede kan ses på figur 1.3.

Når brugeren har fundet den sang, der skal afspilles, trykker brugeren på knappen på pulsgiveren og en



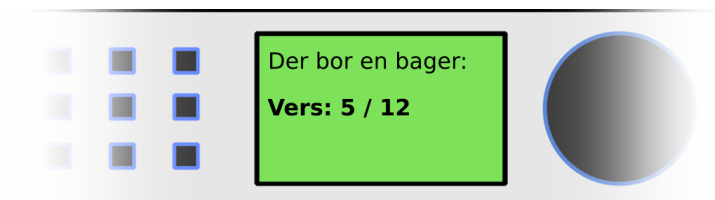
Figur 1.3 Bladring af sange på LCD.

ny menu vises på displayet. I denne menu er det muligt at se, hvor mange vers der er tilgængelige af den valgte sang og vælge, hvor mange vers der skal afspilles. Et konceptuelt billede af denne menu kan ses på figur 1.4. Efter at have valgt antallet af vers med pulsgiveren, trykkes på knappen på pulsgiveren og afspilningen af sangen begynder.

1.1.0.2 Scenario 2

Et andet scenario er, hvor en bruger vil programmere en af de mange knapper, som kan ses på figur 1.2, til at være en genvej til afspilning af en bestemt sang med et bestemt antal vers. Denne funktionalitet er kendt som speed-dial på telefoner.

Dette scenario har nøjagtig de samme trin som scenario 1, bortset fra, at der ikke trykkes på pulsgiverknappen i sidste trin. I stedet holdes en af de programmerbare knapper inde et kort øjeblik. Nu er knappen programmeret, og brugeren kan på et vilkårligt tidspunkt trykke kort på den for at afspilningen af den forprogrammerede sang med det forprogrammerede antal vers, begynder.



Figur 1.4 Valg af antal vers i LCD menu.

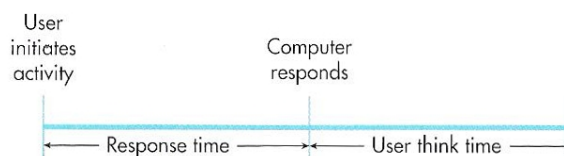
I dette kapitel vil der først blive undersøgt, hvilke krav en bruger har til responstiden i et system. De fundne krav vil blive benyttet i kravspecifikationen til systemet.

I afsnittet 2.2 på næste side, vil der blive vurderet på fordele og ulemper, ved en systemopbygning med klienten som en tynd klient¹ og valgt en passende model til use-casen tidligere beskrevet.

Til sidst vil der, i afsnit 2.3 på side 13, blive lavet en identificering af den nødvendige datakommunikationen for at kunne opfylde funktionaliteterne beskrevet i use-case afsnittet.

2.1 Systemresponstid

I use-case afsnittet beskrives, at brugeren ikke skal opleve en forsinkelse mellem input og output. Tiden mellem et system får et input, til det giver brugeren respons, kaldes SRT (systemresponstid). Det kan eksempelvis være tiden, fra brugeren trykker på en knap på fjernbetjenningen, til tv'et har skiftet kanal. På figur 2.1 ses en simpel grafisk forklaring af begrebet SRT.



Figur 2.1 Forklaring af SRT [Shneiderman B. & Plaisant C., 2005]

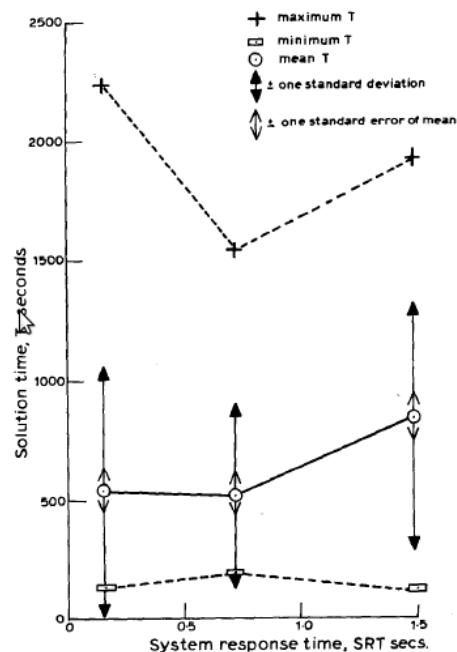
I 1968 udkom en klassisk artikel omkring SRT ved HCI (human-computer-interaction), hvor der ved estimation blev fremlagt en række SRT'er for forskellige HCI cases. Ved input fra f.eks. et tastatur blev det estimeret, at en SRT på under 100 millisekunder giver opfattelsen af, at responset er en del af den mekaniske reaktion på brugerens fysiske input [Miller, 1968].

¹ en tynd klient kaldes også for en dum klient

2. Foranalyse

Ved andre typer input forventer brugeren ikke et øjeblikkeligt respons. Hovedreglen er, at jo mere kompleks opgaven, computeren skal udføre for at give respons, er - jo længere SRT kan brugeren tolerere.

Brugerens forventning til og opfattelse af SRT er én side af sagen, men et andet vigtigt element er effektivitet. En undersøgelse lavet på Imperial College i London viste, at en lang SRT kan føre til, at brugeren løser en opgave langsommere og brugerens effektivitet falder [Goodman, T. & Spence, R., 1978].



Figur 2.2 Løsningsetid versus SRT [Goodman, T. & Spence, R., 1978]

I undersøgelsen blev 30 forsøgspersoner bedt om at løse en opgave ved hjælp af en lys-pen og en skærm. Målet for opgaven var at ændre på en kurves parametre, til den havde den rigtige form. Ved hver ændring af en parameter blev der indsat en SRT før den opdaterede kurve blev vist på skærmen. På figur 2.2 som stammer fra undersøgelsen, kan det ses at, ved SRT'er på 160 og 720 millisekunder, er gennemsnittet af løsningshastigheden hos forsøgspersonerne, nogenlunde den samme. Men ved den gennemsnitlige løsningshastighed ved en SRT på 1490 millisekunder, stiger løsningstiden for forsøgspersonerne.

Generelt er der ikke et lineært fald af effektivitet når SRT'en stiger. Der forekommer nærmere et effektivitetsdrop når SRT'en når over et vist punkt [Miller, 1968]. I dette eksperiment ligger dette punkt mellem en SRT på 720 og 1490 millisekunder. Udfra dette kan det udledes, at hvis en SRT ved en lignende opgavetype holder sig under 720 millisekunder, vil det ikke gå ud over brugerens effektivitet. Da opgaven i undersøgelsen må karakteriseres som mere kompliceret end opgaverne beskrevet i use-casen, må disse grænser som minimum også gælde for dette projekt.

2.2 Tynd eller tyk klient

Ved en hardwaremæssig opbygning med to logiske enheder, der tilsammen skal yde en funktionalitet, er et af de første spørgsmål der stiller sig, hvor meget programmellogik skal der ligge på hver enhed. I dette afsnit vil, fordelingen af logik mellem klient og vært, derfor blive diskuteret.

2.3 Identificering af datakommunikation

En fordeling af programmellogik, hvor klienten bliver tyndest, er hvor klienten agerer som en slags bro, der kun videreformidler input og output fra knapperne til værten og tilbage igen til displayet. Denne fordeling giver værten en slags forlænget arm, der ikke tænker selv, men kun videreformidler ordrer fra værten.

En af fordelene ved en fordeling, hvor alt logik ligger på værten, er at det er lettere at skifte til en anden microcontroller eller processorarkitektur, da kodemængden på denne enhed er minimeret.

Mindre logik på klienten betyder færre krav til klientens hardwareressourcer og derved en lavere produktionspris for denne del af systemet.

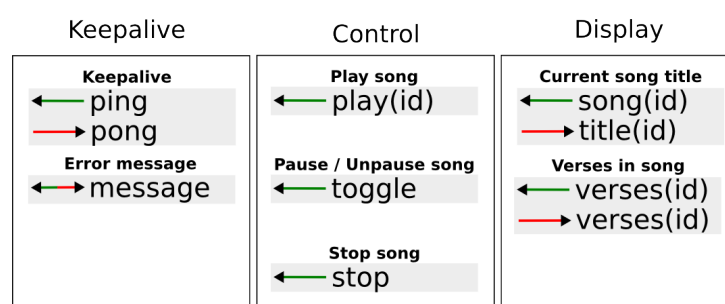
Fordelene ved at have meget programmellogik på klienten er, at det ikke er nødvendigt med, en nær så omfattende brug af kommunikation mellem klienten og værten. Det kan f.eks være ved inputs, hvor det ikke er nødvendig med ny data for at reagere, og klienten selv kan give respons på displayet. Da tykke klienter oftest har større hardwarekrav, er de dyrere, men samtidig har de en større fleksibilitet og giver større mulighed for udvidelser.

Overgangen mellem de to yderpunkter er glidende, så det er muligt at tilpasse fordelingen af programmellogik, så det passer til opgaven. Da dette projekt har fokus på SRT, vil det være fornuftigt at minimere dataudvekslingen mellem klient og vært. Derfor vil det være fordelagtigt at vælge en fordeling af programmellogik i dette system, hvor klienten bliver tyk.

2.3 Identificering af datakommunikation

I use-casen er der beskrevet en række funktionaliteter, der skal opfyldes af systemet. I dette afsnit vil, den nødvendige datakommunikation for at kunne opfylde disse funktionaliteter, blive blotlagt. Kommunikationen kan deles op i to kategorier, kontrol og display. Kontrol er den kommunikation, der skal til for at kunne styre musikafspilleren på værten. Display er den kommunikation, der er nødvendig for at kunne vise den relevante data på displayet.

Musikafspilleren skal, ifølge use-casen, kunne afspille en specificeret sang, pause og stoppe den. Den nødvendige datakommunikation, kan ses under Control på figur 2.3. De grønne pile på figuren viser, at det er kommunikation fra klient til vært. De røde pile er kommunikation i modsat retning.



Figur 2.3 Produkt opstilling

Til klienten skal der sendes informationer omkring de sange, der skal vises på displayet. Da det kun er akkompagnement, der skal afspilles, er det kun nødvendigt at sende titlen på numrene. Dataudvekslingen som disse funktionaliteter kræver, kan ses under Display på figur 2.3.

2.4 Delkonklusion

I foranalysen blev der først undersøgt, hvor stor en SRT, systemet beskrevet i use-casen, må have og der blev konkluderet at med en SRT på over 100 millisekunder forsvinder følelsen af, at systemets reaktion er en mekanisk respons på brugerens input og ved en SRT på over 720 millisekunder begynder brugerens effektivitet at dale. Kravet til at bladre igennem sangene vil derfor være 100 millisekunder og kravet til musikkontrol vil være 720 millisekunder.

I afsnit 2.2 på side 12 blev der vurderet fordele og ulemper ved at placere programmellogik på henholdsvis værten og klienten i systemet. Det blev konkluderet, at for at spare dataudveksling mellem de to enheder, er det hensigtsmæssigt at placere avanceret programmellogik på klienten. Derfor vil klienten i systemet blive såkaldt tyk enhed.

Til sidst blev mængden af nødvendig datakommunikation i systemet identificeret ved brug af en tyk enhed. Dataudvekslingen blev delt op i to kategorier, Control og Display, som kan ses afbilledet på figur 2.3 på forrige side. Hovedkategorien Control indeholder den nødvendige kommunikation for at kunne afspille, pause og stoppe sange på afspilleren fra klienten.

Kategorien Display indeholder kommunikation for visning af titel på nuværende, forrige og næste sang på databasen samt en dataudveksling af antallet af vers på en udspecificeret sang.

Problemformulering

3.1 Problemformulering

I et system med to enheder hvor en bruger kun benytter den ene enhed til interaktion, men alt relevant data skal hentes fra den anden enhed, er det vigtigt at kommunikationen mellem de to enheder har en kort reponstid. Det fører til følgende problemformulering:

Er det muligt at konstruere et system med to enheder der kan give systemet, beskrevet i use casen, en SRT på under 100 millisekunder ved bladring gennem numrene og 720 millisekunder ved musikkontrol, når der skal benyttes USB (Universal Serial Bus) protokolstacken eller TCP/IP (Transmission Control Protocol / Internet Protocol) protokolstacken til det de underliggende netværkslag?

På baggrund af denne problemstilling ønskes et system bestående af, en klient til bruger interaktion, og en vært med musikafspilningsfunktionalitet, specificeret, implementeret og testet.

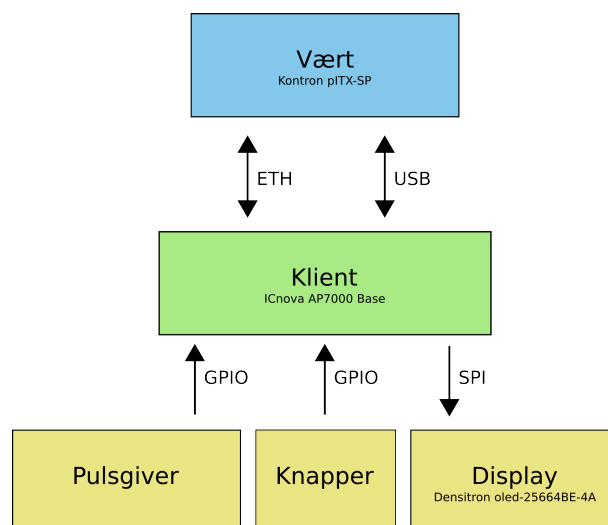
Hardware og Software

Dette kapitel beskriver de hardware og software komponenter, der vil blive brugt i forbindelse med projektet. Alle de delkomponenter der er nødvendige for et fungerende system, vil blive beskrevet.

4.1 Hardware

Alt hardware er blevet anskaffet i samarbejde med DoréDevelopment. Dette inkluderer både microcontrollerboardet, displayet og hardwaren til værten. Disse dele er blevet valgt af DoréDevelopments ud fra virksomhedens interne krav.

På figur 4.1 ses de forskellige hardware komponenter i systemet og interfaces mellem dem. Værten har



Figur 4.1 Diagram over hardware komponenterne og interfaces i systemet

4. Hardware og Software

mulighed for at være koblet til klienten med både Ethernet og USB. Herunder kommer en beskrivelse af de fem grupper af hardware komponenter.

4.1.1 Vært

Værten er baseret på et Kontron pITX-SP board, som er en lavpris pc og består af hardware komponenter, der er til at finde i f.eks. netbooks.

På figur 4.2 er der et billede af boardet med køleprofilen fjernet.



Figur 4.2 Kontron pITX-SP board - Kilde: www.kontron.com

De relevante specificationerne for boardet er [Kontron, 2009]:

CPU: Intel Atom Z530 1.6GHZ med HyperThreading¹ teknologi

RAM: 1GB

ETH: Intel 82574L Gigabit Ethernet

USB: SCH US15W chipset

4.1.2 Klient

Klienten består af et ICnova AP7000 Base microcontrollerboard, der også findes i en open-source udgave, kaldet Grasshopper. Begge bliver produceret af virksomheden In-circuit.

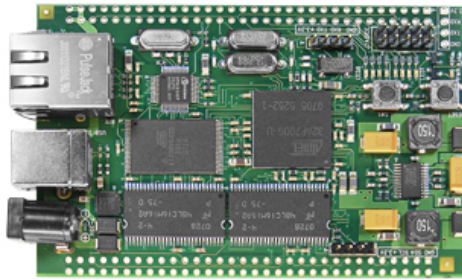
Et billede af ICnova AP7000 Base kan ses på figur 4.3 på næste side.

De relevante specifikationerne er [In-circuit, 2009] [Atmel, 2007]:

MPU: Atmel AT32AP7000

- 140 MHZ 32bit CPU
- MACB 10/100 Ethernet MAC (Media Access Control)
- SPI (Serial Peripheral Interface bus) Controller
- USBA High Speed USB interface

¹<http://ark.intel.com/Product.aspx?id=35463>



Figur 4.3 In-circuit ICnova AP7000 Base board - Kilde: www.avrfreaks.com

- 2x32 GPIO'er (General Purpose Input Output pins)

SDRAM: 2x32MB Samsung MT48LC16M16A2

ROM: 8MB Atmel AT49BV642D-70TU

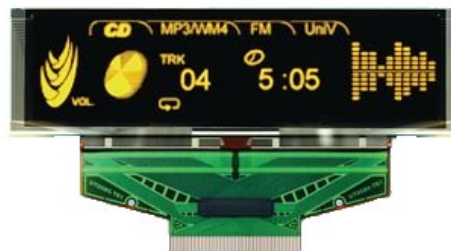
PHY: Davicom DM9161A

USB-to-UART: Silicon Labs CP2102

4.1.3 Display

Displayet valgt til dette projekt er et Densitron OLED 25664BE-4A, som er et OLED (Organic Light Emitting Diode) display. Denne type display har den egenskab, at det ikke er nødvendigt med baggrundsbelysning og kan bruges i alle typer belysning.

Et billede af displayet kan ses på figur 4.4.



Figur 4.4 Densitron OLED 25664BE-4A displayet - Kilde: www.farnell.com

De mest relevante informationer for displayet er [Densitron, 2008]:

Farve: Monokrom gul

Opløsning: 256x64

Controller: Solomon SSD1322

Interface: MPU Parallel / SPI

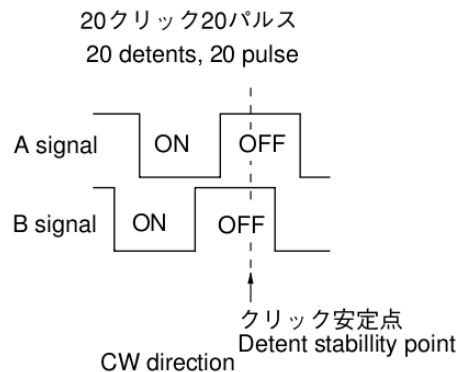


Figur 4.5 3d tegning af en rotary encoder[Alps, 2000]

4.1.4 Pulsgiver

På figur 4.5 ses en 3d model af en pulsgiver, som er en drejelig kontakt, der giver et bestemt antal pulser per omgang når den drejer rundt.

For at kunne vide hvilken vej rotary encoderen bliver drejet, udsender den to forskellige mønstre afhængig af, om det er med eller mod uret. På figur 4.6 er der en beskrivelse af mønstret, der bliver genereret under drejning i urets retning. Ved rotation mod urets retning er a og b signalerne byttet om.



Figur 4.6 Timing af signaler ved rotation i urets retning[Alps, 2000]

4.1.5 Knapper

Knapperne til systemet er sat til jord, så de går lav, når de bliver trykket på. Der er intern pull-up modstande på alle microcontrollerens GPIO'er så en ekstern modstand er ikke nødvendig.

4.2 Software

For at gøre udviklingen af en prototype så hurtig som muligt vil der blive implementeret et styresystem på både værten og klienten. Dette vil lette implementationsprocessen og sørge for at projektets mål hurtigere kan nås.

4.2.1 Vært

Kontron pITX-SP boardet minder hardwaremæssigt om en almindelig pc, og indeholder også en x86 kompatibel CPU, som findes i størstedelen af alle pc'er. Derfor er der blevet valgt at implementere Ubuntu Linux på boardet som er en desktop Linux distribution. Denne distribution indeholder blandt andet:

Kernel 2.6.31 Linux kernen

Python 2.6.4 Højniveau programmeringssprog

Python-twisted 8.2.0 Event baseret framework til python netværks applikationer

Python-pymad 0.5.4 Python modul til afspilning af mp3 filer

4.2.2 Klient

BSPen (Board Support Package) der følger med dette board benytter Buildroot, som er et byggesystem baseret på en række scripts. Buildroot genererer en cross-compiler, en kerne og et root filsystem til det valgte målsystem. I dette tilfælde er det AVR32 processor.

BusyBox 1.13.2 En samling UNIX værktøjer f.eks. ls, cd og mkdir.

Linux 2.6.28.4 Linux kernen med drivere til USB, Ethernet og SPI

Avr32-linux-gcc 4.2.2 Gnu C Compiler kompileret med patches til AVR32 arkitekturen

Kravspekifikation

5.1 Kravspekifikation

Kravspekifikationen er udarbejdet ud fra use-case og foranalyse kapitlerne. Den forudsætter brugen af, hardwaren og softwaren beskrevet i kapitel 4 på side 17.

Dette afsnit beskriver, hvilke funktionaliteter systemet skal indholde for at opfylde use-casen. For hver funktionalitet vil kravene blive delt op i krav til klienten og krav til værten.

5.1.1 Musikdata

For at klienten kan præsentere sangtitel og tilgængelige antal vers for brugeren, er det nødvendigt at kunne hente denne gruppe data fra værten.

Det giver følgende funktionalitetskrav til klienten:

1. Hent information om tilgængelige antal vers fra værten og præsenter på display med en maksimum SRT på 100 millisekunder.
2. Hent information om sang titel fra værten og præsenter på display med en maksimum SRT på 100 millisekunder.

Og disse krav til værten:

1. Stille information om tilgængelige antal vers til rådighed for klienten.
2. Stille information om sangtitel til rådighed for klienten.
3. Stille information om antallet af sange i filsystemet til rådighed for klienten.

5. Kravspecifikation

5.1.2 Musikkontrol

Klienten skal kunne styre afspilleren på værten. De mulige funktionalitetskrav til klienten er:

1. Starte afspilningen af en sang på værten med en maksimums responstid på 720 millisekunder.
2. Pause afspilningen af den nuværende sang på værten med en maksimums responstid på 720 millisekunder.
3. Stoppe afspilningen af den nuværende sang på værten med en maksimums responstid på 720 millisekunder.

Ved alle tre funktionaliteter skal klienten have repons om kommandoen lykkedes.

Krav til værten er:

1. Stil adgang til musikkontrol til rådighed for klienten.

5.1.3 Forbindelsesbevidsthed

Klienten og værten skal være bevidste om at, der er en fungerende forbindelse mellem de to enheder. Det fører til følgende funktionalitetskrav til klienten:

1. Test forbindelse til værten med et interval på minimum et sekund.

Krav til værten:

1. Svar på forbindelsestest fra klient.

5.1.4 Fejlmeddelelser

Hvis der opstår en fejl, på klienten eller værten, skal det være muligt at videreformidle en fejlmeddelelse til den anden enhed.

Krav for klient:

1. Afsend fejlmeddelelse og præsenter fejlkode på display.
2. Modtag fejlmeddelelse og præsenter fejlkode på display.

Krav for værten:

1. Afsend fejlmeddelelse og gem i logfil.
2. Modtag fejlmeddelelse og gem i logfil.

Dette kapitel redegør for designet af softwaren i systemet. Fremgangsmåden til udarbejdelsen er baseret på SPU (Struktureret Program Udvikling) modellen og benytter UML (Unified Modeling Language) til grafisk fremstilling af softwaren i systemet.

Målet for designet er, at fremsætte strukturen for programmet i systemet, så det kan opfylde kravene i kravspecifikationen.

Da systemet er delt op i en vært og en klient, vil designet blive opdelt på samme vis. Designfaserne; programdesign og procesdesign vil gå igen for både vært og klient. Da systemet består af to simple programmer vil procesdesignet indeholde både procesdesign og moduldesign.

6.1 Systemdesign

Dette afsnit har til formål at identificere grænsefladen mellem de to programmer, og bestemme hvilke komponenter programmerne skal bestå af.

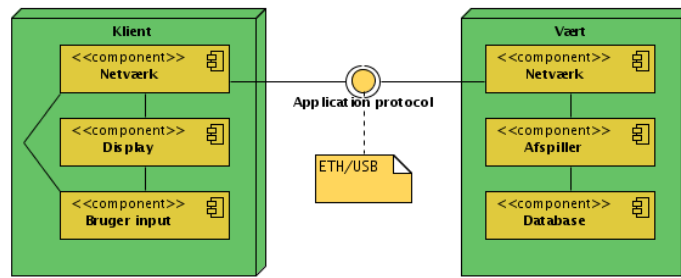
6.1.1 Komponenter

Til at beskrive komponenterne i systemet, er der på figur 6.1 på den følgende side, et deploymentdiagram over systemet. Hver komponent på figuren er en funktionalitet i programmet. Klientnoden består af tre komponenter netværk, display og brugerinput.

Brugerinput komponenten skal registrere og videreformidle de inputs brugeren giver systemet. Display komponenten skal fremvise informationerne fra systemet til brugeren. Netværks komponenten skal implementere den grænseflade, der vil blive beskrevet senere i dette afsnit og stille kommunikationen til rådighed for de andre komponenter i klientnoden.

6.1.2 Grænseflader

6. Design



Figur 6.1 Deploymentdiagram af systemet.

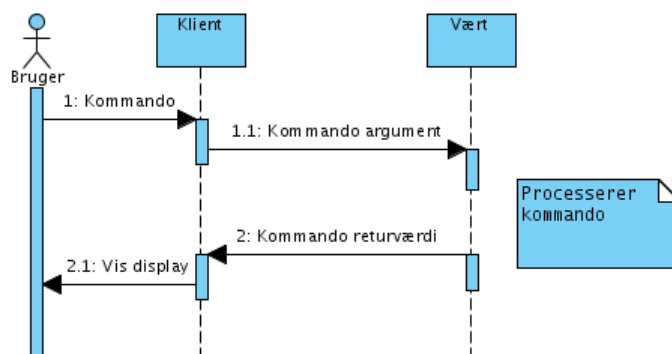
I afsnit 2.3 blev den nødvendige datakommunikation, for at opfylde use-casen og kravspecifikationen, identificeret. Denne datakommunikation kan generaliseres og et eksempel på dette, kan ses på sekvensdiagrammet for systemet på figur 6.2. Dette sekvensdiagram beskriver, hvordan en brugers input bliver behandlet i systemet.

Ved at tage den generaliserede datakommunikation og beskrive de enkelte kommandoer og returnværdier, fås den samlede grænsefladeprotokol der kan ses på den nedenstående tabel.

Kommando	Argument	Argument type	Returværdi	Returværdi type
play	sang id	heltal	status	streng
toggle			status	streng
stop			status	streng
getSongs			antal sange i databasen	heltal
getTitle	sang id	heltal	sang titel	streng
getVerses	sang id	heltal	antal vers	heltal
ping			"pong"	streng
message	besked	streng		

Tabel 6.1 Protokol til kommunikation mellem vært og klient.

Datakommunikationen mellem vært og klient foregår med strenge, og for at begge sider, er klar over hvilken datatype, argumenter og returnværdier har, er disse specificeret i protokollen.



Figur 6.2 Sekvens diagram for systemet.

6.2 Programdesign - Vært

Som beskrevet i afsnit 4.2.1 på side 20, er værten implementeret med styresystemet Ubuntu Linux, hvor der er installeret python og det eventbaserede netværksmodul Twisted. Python og Twisted vil blive benyttet ved implementationen af systemet på serveren.

Værten skal lytte til en forudbestemt port, behandle de indgående forespørgsler fra klienten og afspille musiknumrene. Programmet vil komme til at bestå af to processor der startes ved programstart og en tredje underproces der bliver affødt når netværksdata skal processeres. Disse tre processer bliver beskrevet i nedenstående afsnit.

6.2.1 Netværksprocessering

Denne process skal modtage og sende data til netværket. Når der indgår data vil den afføde en instans af applikationslogikprocessen, der behandler den indkomne data asynkront.

Mens dataen bliver behandlet skal denne process og vente på en tilbagemelding fra applikationslogikprocessen, som sendes tilbage til klienten på netværket.

6.2.2 Applikationslogik

Applikationslogikprocesserne er hjertet i programmet på værten. Disse processer skal hente den fornødne data ved kommandoerne getTitle, getVerses og getSongs fra databasen og returnere den til netværksprocessen der vil videresende dataen til klienten.

Ved kommandoerne play og toggle kommunikere disse processor med en kø til musikafspilningsprocessen. Status for udførelsen af afspilningskommandoerne skal sendes fra musikafspilningsprocessen til den retmæssige applikationslogikproces med en kø og returneres til netværksprocessen.

6.2.3 Musikafspilning

Denne process styrer musikafspilningen. Den vil lytte til musikafspilningskøen og vente på ændringer til musikafspilningen. Ved ændring hentes den retmæssige fil ind fra filsystemet og status for afspilningen sendes via applikationskøen tilbage applikationslogikprocesserne.

6.3 Procesdesign - Vært

Da værten vil blive implementeret i et objektorienteret programmeringssprog, vil modulerne i procesdesignet for værten være klasser. En oversigt over klasserne kan se på klassediagrammet på figur 6.3 på næste side. Funktionerne i klasserne bliver ifølge UML terminologi kaldt metoder.

6.3.1 JukeProtocol

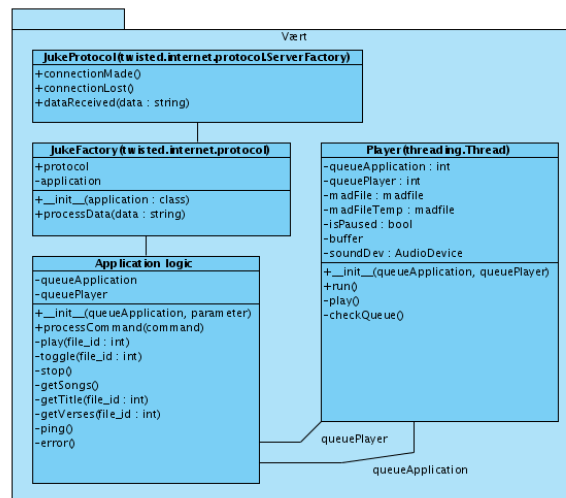
Denne klasse er netværksserveren og skal lytte efter og sende data til netværkporten, hvor kommunikationen med klienten foregår. På klassediagrammet på figur 6.3 ses det at denne klasse indeholder tre metoder, som vil blive beskrevet i nedenstående afsnit.

connectionMade

Denne metode skriver dato, tid og klient id til logfilen når der bliver oprettet en forbindelse til serveren.

connectionLost

6. Design



Figur 6.3 Klassesdiagram af værten.

Metoden skriver dato, tid og klient id til logfilen når en etableret forbindelse bliver lukket, frivilligt eller ufrivilligt, ned.

dataReceived(data)

Denne metode står for at modtage data fra netværket. Metoden bruger processData metoden fra klassen JukeFactory til at videresende den modtagne data i systemet.

Når dataen er blevet behandlet i systemet vil processData metoden, fra JukeFactory, returnere til dataReceived og status vil blive sendt via netværket til klienten.

6.3.2 JukeFactory

Da der kun er én proces til at håndtere netværket, er det vigtigt, at behandling af indgående data foregår asynkront. Denne klasse er en del af twisted frameworket og sørger for den asynkrone processering af data, og der derved kan være flere forespørgsler fra klienten i programmet på samme tid.

__init__(applikation)

Når en instans af JukeFactory klassen initialiseres bliver denne metode kørt. Metoden står for at koble JukeFactory klassen sammen med en anden klasse der skal processere den indgående data fra klassen JukeProtocol.

processData(data)

Denne metode sender data direkte videre til metoden processCommand i ApplikationsLogic klassen.

6.3.3 ApplicationLogic

Denne klasse skal sørge for logikken i programmet på værten. ApplicationLogic skal afgøre hvilken kommando, der er blevet sendt fra klienten, og udføre den pågældende hændelse, koblet til kommandoen. Ved kommandoerne play, toggle og stop sendes der via afspillerkøen kommandoer til processen med musikafspilleren. Ved kommandoerne getSongs, getTitle og getVerses hentes den relevante data fra den tilknyttede database.

På klassesdiagrammet 6.3 ses metoderne for denne klasse.

__init__(queueApplication, queuePlayer)

Når en instans af ApplicationLogic initialiseres bliver denne metode kørt. Her gemmes de to køer til at kommunikere med musikafspillerprocessen. Den første kø, queueApplication, bruges til at modtage data fra musikafspillerprocessen og queuePlayer bruges til at sende data.

processCommand(command)

Denne metode splitter command strengen op i to dele. Den ene del er selve kommandoen og den anden del er det medfølgende argument. Efterfølgende kalder metoden den respektive metode til processering af kommandoen.

play(file_id)

Sender via afspillerkøen besked til musikafspilningsprocess om at et nummer skal afspilles. Status for udførsel skal returneres.

toggle

Sender via afspillerkøen besked til musikafspilningsprocess om at det nuværende nummer skal toggles. Status for udførsel skal returneres.

stop

Sender via afspillerkøen besked til musikafspilningsprocess om at et nuværende nummer skal stoppes. Status for udførsel skal returneres.

getSongs

Denne metode henter antallet af sange i databasen. Databasens svar returneres.

getTitle(file_id)

Metoden henter titlen på nummeret svarende til file_id. Den fundne titel returneres.

getVerses(file_id)

Henter antallet af vers tilknyttet til musiknummeret med file_id. Databasens svar returneres.

ping

Skal returnerer "pong".

error(message)

Skriver tidsstempel og fejlkode til logfilen.

6.3.4 Player

Denne klasse står for afspilningen af musikfilerne. Den henter den respektive fil ind fra filsystemet, afspille, toggle og stoppe filen.

__init__(queueApplication, queuePlayer)

Når en instans af Player initialiseres bliver denne metode kørt. Her gemmes de to køer til at kommunikere med applikationslogikprocessen. Den første kø, queueApplication, skal bruges til at sende data til applikationslogikprocessen og queuePlayer skal bruges til at modtage data.

run

Denne metode er nedarvet fra Thread klassen og er den metode der kører uendeligt når processen startes. Metoden kalder play metoden.

play

Denne metode er selve afspilleren i Player klassen. Det er denne metode der skal hente en specificeret fil fra filsystemet, og afspille, pause eller stoppe den. Kommandoerne bliver sendt til denne metode fra applikationslogikprocessen med køen queuePlayer og vil bruge metoden, checkQueue, til at undersøge om der er kommet ændringer fra applikationslogikprocessen.

checkQueue

Bruges af play metoden til at undersøge om der er kommet ny data fra i queuePlayer køen. Hvis der er kommet ny data returneres dette ellers returneres False.

6.4 Programdesign - Klient

Klienten er implementeret med styresystemet Linux kompileret med byggesystemet Buildroot. I kernen er det bygget drivere til SPI, GPIO og netværk ind.

Dette program skal stå for at registrere når en bruger giver inputs til klienten fra knapperne og pulsgiveren. Disse inputs skal sendes videre til værten og resultaterne vises på displayet tilkoblet klienten.

Programmet skal bestå af fire processer:

- Udgående netværskommunikation
- Indgående netværskommunikation
- GPIO-håndtering
- Displayvisning

Kommunikationen mellem disse fire processor vil bestå af to køer. En kø kaldet displaykø, til kommunikation med displayvisningsprocessen og en kø, kaldet netværkskø, til at kommunikere med den udgående netværskommunikationsprocess.

Alle processorne er baseret på løkker og vil kun stoppe, hvis programmet bliver lukket ned.

6.4.1 Udgående netværskommunikation

Denne proces står for at sende kommandoer til værten via netværket. Når der kommer data i køen til denne proces vil den videresende det til netværkskernemodulet.

6.4.2 Indgående netværskommunikation

Denne proces lytter til netværket og vil videresende indgående data til displaykøen.

6.4.3 GPIO håndtering

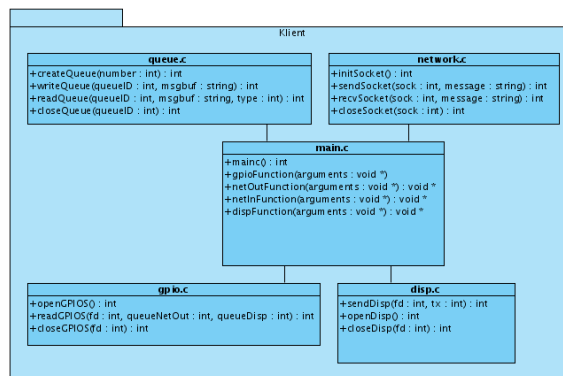
Processen skal oversætte input fra event kernemodulet, der registrerer inputs fra knapperne og pulsgiveren, til kommandoer. Disse kommandoer skal sendes til netværkskøen. Processen sørger for at åbne og lukke event interfacet til kernemodulet.

6.4.4 Displayvisning

Displayvisningsprocessen modtager data på displaykøen og oversætter det til data der kan sendes med SPI kernemodul til displayet. Processen skal også håndtere åbning og nedlukning af SPI interfacet.

6.5 Procesdesign - Klient

Dette afsnit vil beskrive både procesdesign og moduldesign af programmet på klienten. Klienten vil blive implementeret i programmeringssproget C og terminologien vil følge SPU modellen.



Figur 6.4 Klassediagram af klienten.

På figur 6.4 ses et klassediagram over klienten hvor alle moduler og funktioner er listet.

6.5.1 queue.c

Dette modul opretter, skriver, læser og lukker en specificeret kø. Modulet vil blive brugt af alle processerne til at kommunikere med hinanden.

createQueue(number)

Funktionen createQueue opretter en uendelig kø, svarende til identifikations nummeret i variabelen number og returnere køen. Hvis køen allerede er oprettet vil den eksisterende kø blive returneret.

writeQueue(queueID, messagebuf, type)

Beskedkøerne implementeret i styresystemet understøtter typer. Disse typer kan bruges til eksempelvis at prioritere beskederne i køen, så den besked med den højeste prioritet vil blive læst først.

Denne funktion skriver en besked med typen, type, til køen i variabelen queueID. Ved succesfuld skrivning til køen skal funktionen returnere 0 og -1 ved fejl.

readQueue(queueID, messagebuf, type)

Funktionen læser fra beskedkøen specificeret i variabelen queueID. Hvis der ikke er nogen besked i denne kø skal funktionen stå og vente indtil der kommer en besked, placere den i messagebuf og returnere 0.

closeQueue(queueID)

Denne funktion lukker køen refereret til i queueID. Hvis køen ikke findes eller ikke kan lukkes returneres -1 ellers skal funktionen returnere 0.

6.5.2 gpio.c

Dette modul står for at håndtere inputs fra GPIO kernemodulet.

openGPIOs()

Denne funktion skal åbne event interfacet til GPIO kernemodulet. Ved succes returneres interfacet og ved fejl returneres -1

readGPIOs(fd, queueNetOut, queueDisp)

Funktionen lytter til event interfacet specificeret i fd. Når en event indtræffer, vil den blive konverteret til den tilsvarende kommando og sendt til netværkskøen queueNetOut. Efter kommandoen er sendt til køen skal funktionen starte forfra og igen lytte efter et nyt event.

Funktionen skal ikke returnere med mindre der opstår en fejl.

closeGPIOs(fd)

Denne funktion lukker event interfacet til GPIO kernemodulet. Skal returnere 0 ved succes og -1 ved fejl.

6.5.3 network.c

Dette modul står for kommunikationen på netværksinterfacet. Modulet kan benytte TCP/IP eller USB protokolstacken.

initSocket()

Denne funktion opretter et netværksinterface og skal returnere interfacet ved succes og -1 ved fejl.

sendSocket(sock, message)

sendSocket funktionen sender data gemt i message til netværksinterfacet sock. Skal returnere 0 ved succes og -1 ved fejl.

recvSocket(sock, message)

Funktionen modtager data fra netværksinterfacet sock og placerer det i variablen message. Funktionen skal kun vente indtil der er indgående data og derefter returnere antallet af modtagne bytes. Ved fejl skal funktionen returnere -1.

closeSocket(sock)

Denne funktion lukker netværksinterfacet specificeret i sock. Hvis interfacet ikke findes eller ikke kan lukkes, returneres -1. Ved succesfuld nedlukning returneres 0.

6.5.4 disp.c

Dette modul står for kommunikationen til displayet. Kommunikationen sendes over SPI protokollen der benyttes igennem SPI kernemodulet.

openDisp()

Denne funktion åbner interfacet til SPI kernemodulet. Ved succes skal det returnere interfacet ellers returneres -1.

sendDisp(fd, tx)

Funktionen sendDisp skal sende den data specificeret i tx til SPI interfacet fd. Ved succes returneres 0 ellers skal funktionen returnere -1.

closeDisp(fd)

Skal lukke for SPI interfacet specificeret i fd. Returnerer 0 ved succes og -1 hvis interfacet ikke findes eller ikke kan lukkes.

6.5.5 main.c

Dette modul starter de fire processer beskrevet i det følgende når programmet startes.

main()

Denne funktion er det første der bliver kørt når programmet på klienten startes. Den skal oprette de fire processer, beskrevet i de nedenstående afsnit.

Funktionen returnerer kun hvis programmet lukkes.

gpioFunction()

Denne funktion er GPIO-håndteringsprocessen og benytter createQueue funktionen fra queue.c modulet til at oprette de to køer, dispQueue og netQueue. Efterfølgende skal den oprette interfacet til gpio kerne-modulet og begynde indlæsningen fra gpio modulet.

Funktionen returnerer aldrig.

netOutFunction()

Funktionen er den udgående netværskommunikationsproces. Denne funktion benytter readQueue til at modtage data fra de andre processer og kalder sendSocket til, at sende den modtagne data til netværks-kernemodulet. Funktionen er en process og returnerer aldrig.

netInFunction()

Denne funktion er den indgående netværskommunikationsproces. Funktionen læser fra netværkskerne-modulet med funktionen recvSocket og sender den modtagne data videre til displaykøen.

Funktionen er en process og vil aldrig returnere.

dispFunction()

Denne funktion åbner interfacet til SPI displayet, venter på data fra displaykøen og sender det ud på displayet.

Funktionen returnerer aldrig.

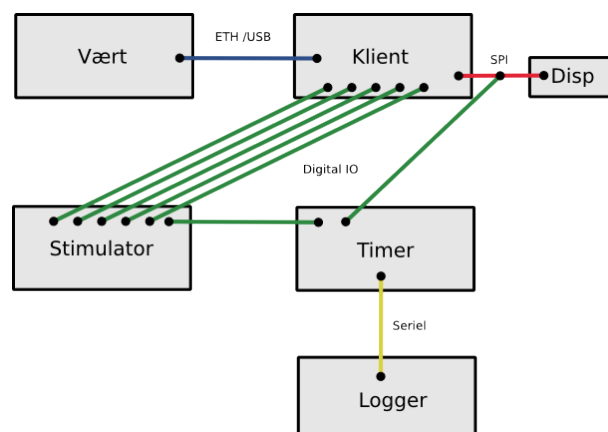
6.6 Implementation

Dokumentationen for implementationen af designet kan ses i Appendix A.1.1 på side 49. Værktøjet Doxygen er brugt til at lave dokumentationen både for værten og klienten.

Dette kapitel indeholder stress-tests af systemet beskrevet i design kapitlet. Testene består af en række stress-tests på klienten, værten og forbindelsen imellem de to enheder.

7.1 Testopstilling

For at kunne teste systemet er der konstrueret en testopstilling som genererer inputs og efterfølgende måler SRT for disse inputs. Denne opsætning giver mulighed for at kunne lave ensartede målinger ved alle tests, så resultaterne bliver sammenlignelige. På figur 7.1 ses testopstillingen bestående af værten, klienten og de tre ekstra enheder, der benyttes til at foretage tests.



Figur 7.1 Testopstilling.

7.1.0.1 Stimulator

7. Test

Denne enhed står for at generere ensartede inputs til systemet. Enheden består af et ICnova AP7000 Base board, der kører FreeRTOS med en task som skiftevis toggler de fem digitale outputs koblet til fem digitale inputs på klienten. Den sjette digitale pin er koblet til Timerenheden og bliver toglet hver gang en af de fem andre pins bliver toglet.

På denne enhed kan der indstilles hvilke pins der toggles og med hvilket interval.

7.1.0.2 Timer

Denne enhed består af et Arduino Diecimila board, og benytter en Atmel Atmega168 microcontroller. Den ene af timerenhedens interrupt pins er koblet til den sjette pin på Stimulatorenheden.

Den anden interrupt pin er koblet til clock pin'en på SPI forbindelsen til displayet. Klienten clocker 32 gange per respons, men Timer enheden registrerer kun et interrupt da SPI kommunikationen foregår ved høj frekvens (5mhz).

Timer enheden registrer derved hver gang stimulator enheden giver et input til klienten og timer hvor lang tid der går før klienten sender et respons til displayet. SRT bliver sendt over seriel forbindelse til Loggerenheden.

7.1.0.3 Logger

Denne enhed består af en pc med et python script, der læser fra den serielle forbindelse til Arduino Diecimila boardet. Dataen bliver gemt til en fil, der senere kan analyseres.

7.2 Stress-test

For at undersøge hvordan systemet opfører sig under stress, laves der en række tests, som vil belaste udvalgte dele af systemet. Disse tests deles op i fire hovedkategorier:

1. Klient
2. Vært
3. Forbindelse
4. Bruger interaktion

7.2.1 Klient

I dette afsnit bliver testen af klienten beskrevet og resultaterne diskuteres efterfølgende.

Test 1

På klienten foretages en test hvor programmet Cpulimit benyttes. Dette program kan begrænse CPU forbruget for en proces og dets tilhørende underprocessor. Ved at begrænse en applikation der normalt vil bruge 100% CPU forbrug f.eks en while løkke med assembler instruktion NOP (No Operation), er det muligt at styre CPU belastningen på klienten. Denne test er en ren CPU test og genererer ingen trafik på filsystemet eller trafik på forbindelsen.

Denne test består af 11 undertests, hvor der laves 60 registreringer af SRT for hver af de 5 kommandoer:

toggle: Denne kommando starter eller pauser afspilningen på værten. Kommandoen gemmer det sted den nuværende sangs er kommet til i nummeret, henter en tom musik fil fra filsystemet og venter på nyt input. Hvis afspilleren allerede er på pause hentes den gamle sang ind, den gemte fremdrift af nummeret genoprettes og afspilningen fortsættes .

getVerses: Henter værdien fra databasen med antal vers i den valgte sang.

getSongs: Henter værdien fra databasen med antallet af samlede sange i systemet.

getTitle: Henter fra databasen titlen på det udvalgte nummer.

play: Henter den udvalgte fil fra filesystemet og starter afspilningen af nummeret.

Første undertest er med 0% CPU forbrug fra NOP løkken, anden undertest er med 10% CPU forbrug og så videre.

Fejlkilder

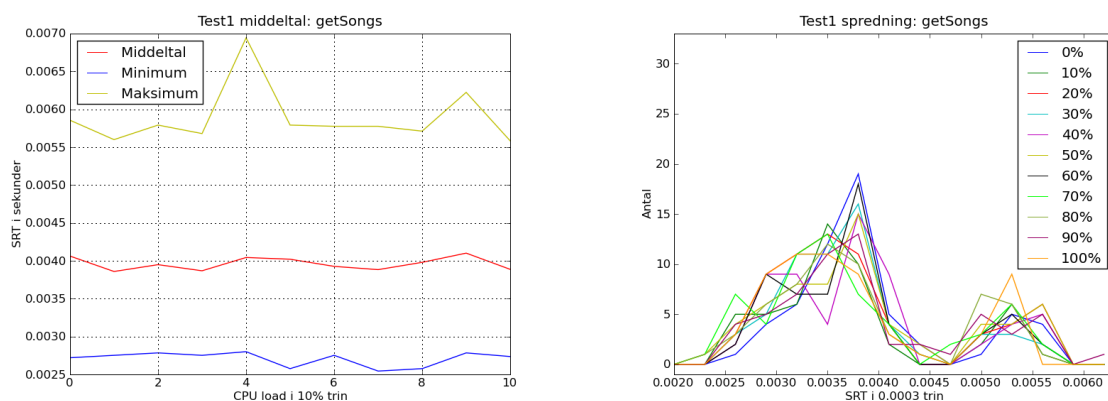
NOP instruktionen kan være optimeret ud af koden af AVR32 kompilatoren. Dette er dog usandsynligt da CPU forbruget ved hver test blev kontrolleret med Busybox programmet Top.

Timeslotslængden i programmet Cpulimit var kun på 100 millisekunder i denne test da programmet Cpulimit ellers ville bruge størstedelen af klientens CPU resourcer og forhindre testens udførelse.

Resultater

Kommandoerne getVerses, getSongs og getTitle er næsten implementeret ens og forventes også at opføre sig ens i testen. Derfor er det kun kommandoen getSongs der vil blive diskuteret i dette afsnit. Graferne for de to andre kommandoer kan ses i appendix A.1 på side 49. Måledataen for alle resultaterne i denne test er fra det samme målesæt.

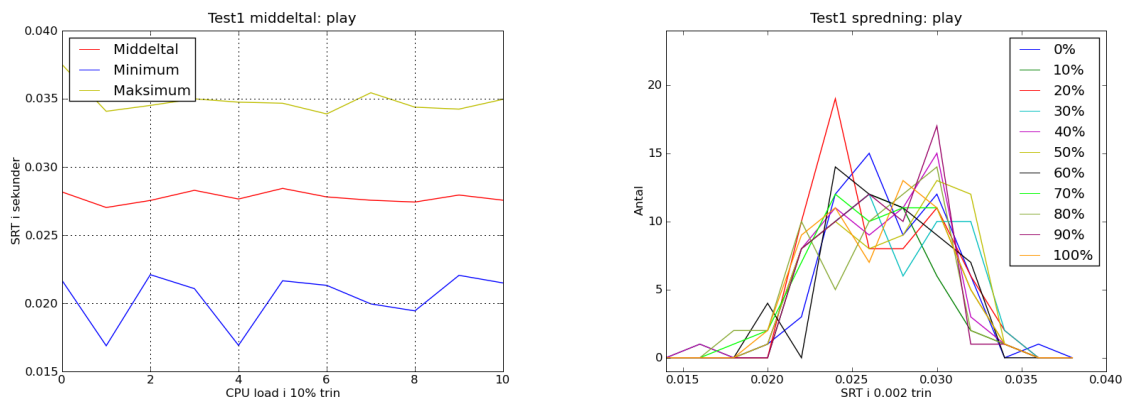
På figur 7.2 ses resultaterne for kommandoen getSongs. På figuren til venstre ses gennemsnittet af målingerne for de 11 undertests. Den laveste gennemsnitlige SRT på 3,86 millisekunder ses ved 10% CPU belastning og den højeste på 4,10 millisekunder ses ved 90% belastning. SRTerne for alle målingerne overholder kravspecifikationen.



Figur 7.2 Middeltal og spredning for kommandoen getSongs i test 1

Figureerne for kommandoen play kan ses på figur 7.3 på næste side. På den venstre figur kan det ses, at alle graferne nogenlunde holder den samme værdi ved alle undertests. Spredningen, som er på figuren til højre, viser at størstedelen af værdierne for alle undertests er placeret i det samme område og hyppigheden aftager i randområderne.

7. Test



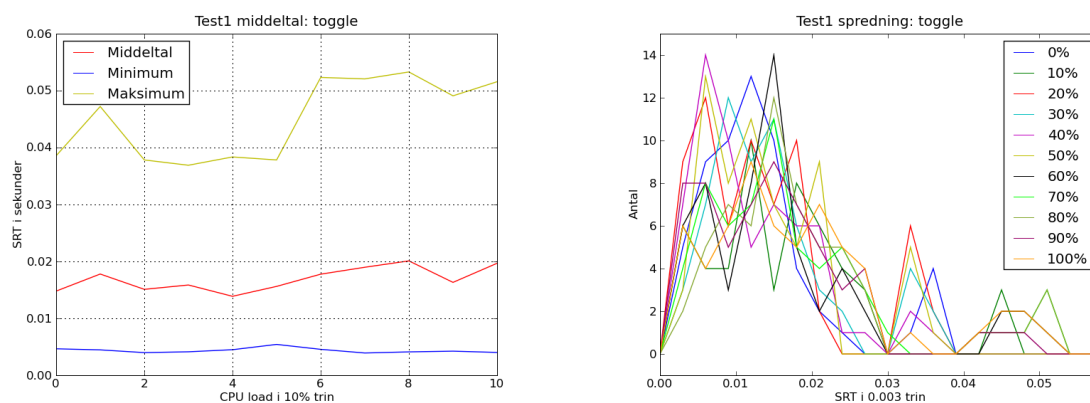
Figur 7.3 Middeltal og spredning for kommandoen play i test 1

Resultaterne for kommandoen toggle kan ses på figurene 7.4. På den venstre figur ses, at middeltallet er stiger når CPU belastningen på klienten kommer over 60%.

Implementationen af toggle kommandoen på værten minder meget om play kommandoen. Den største forskel er at toggle kommandoen har to forskellige hændelsesforløb afhængig af om en sang pauses eller fremdriften af en sang genoprettes. Dette ses på den større forskel mellem minimumsværdierne og maksimumsværdierne for toggle kommandoen.

Minimumgrafens værdier er i det samme område mens maksimumgrafens værdier stiger ved højere belastningsgrader.

På sprednings figuren ses, at spredningen er langt større sammenlignet med grafen for kommandoen play hvilket skyldes de to forskellige hændelsesforløb på værten.



Figur 7.4 Middeltal og spredning for kommandoen toggle i test 1

7.2.2 Vært

I dette afsnit beskrives testen udført på værten og efterfølgende præsenteres og diskuteres resultaterne.

Test 2

På værten udføres den samme test, som der blev lavet på klienten i test 1. I test 2 er det igen de samme kommandoer der bliver udført og alle resultater er fra det samme målesæt.

I denne test bruges programmet Cpulimit sammen med en while løkke til at generere CPU belastning. Værten har en Intel Atom Z530 CPU der har to tråde implementeret med HyperThreading. Disse to tråde opfattes af styresystemet som to separate CPU kerner og i testen startes en instans af while løkken til hver tråd. Disse begrænses fra 10%, på hver tråd, til 100% på hver tråd.

Styresystemet bootes ind i et simpelt kommandolinje interface, så antallet af services kørende på værten er minimeret. Powermanagement er også slået fra, så CPU'en kører ved fuld hastighed ved alle tests.

Fejlkilder

I alle graferne for test 2 er der en markant stigning af SRT ved deltesten med 60% CPU belastning. Denne markante stigning virker ulogisk og kan skyldes, at programmet Cpulimit ikke har været opsat korrekt under denne deltest. Ved begrænsning af flere processor opførte programmet Cpulimit sig ustabil og gik ned flere gange under testene på værten så det er en plausibel fejlkilde.

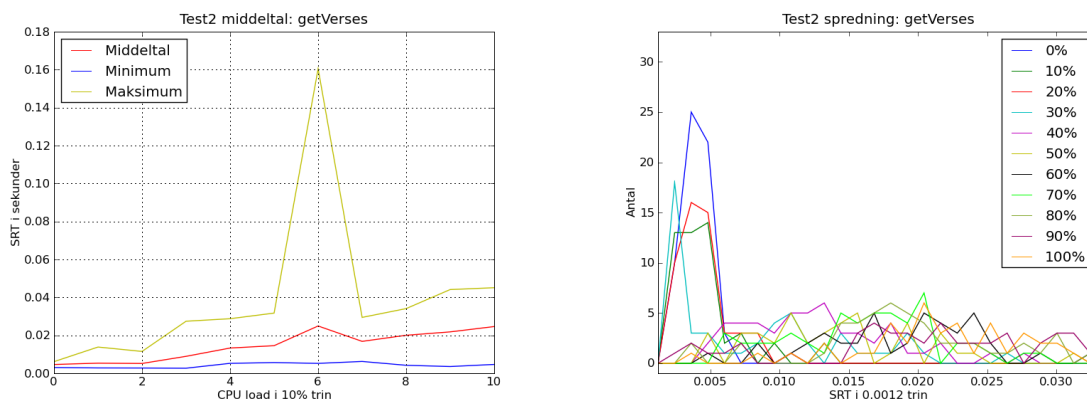
I de første fire deltests fra 0 til 30% belastning var dataopbygningen i datafilerne anderledes end de sidste 6 fra 40 til 100%. Da disse fire første datafiler efterfølgende, er blevet konverteret til samme format som de andre deltests, kan der være sket en fejl under den manuelle konvertering, hvilket kan forklare den markante maksimumsværdi for disse CPU begrænsninger i resultaterne.

Cpulimit havde i denne test en timeslotlængde på 0,0001 sekund hvilket burde være passende til denne test.

Resultater

I denne test vil getTitle og getSongs ikke blive diskuteret da de er implementeret på samme måde som getVerses og resultaterne minder om hinanden. Graferne for getTitle og getSongs kan findes i appendix A.1 på side 49.

På figurerne 7.5 ses resultaterne for test 2 for kommandoen getVerses. Hvis der ses bort fra målingerne ved 60% ses på den venstre figur, at middeltallet stiger tilnærmelsesvist lineært. Den gennemsnitlige SRT er ved 0% CPU belastning på 0,006 sekunder og stiger op til 0,045 sekunder ved 100% belastning. På figuren til højre, med spredningen, kan det ses at værdierne for deltestene 0 til 30% ligger placeret markant anderledes end de andre deltests, der ligger mere bredt spredt.



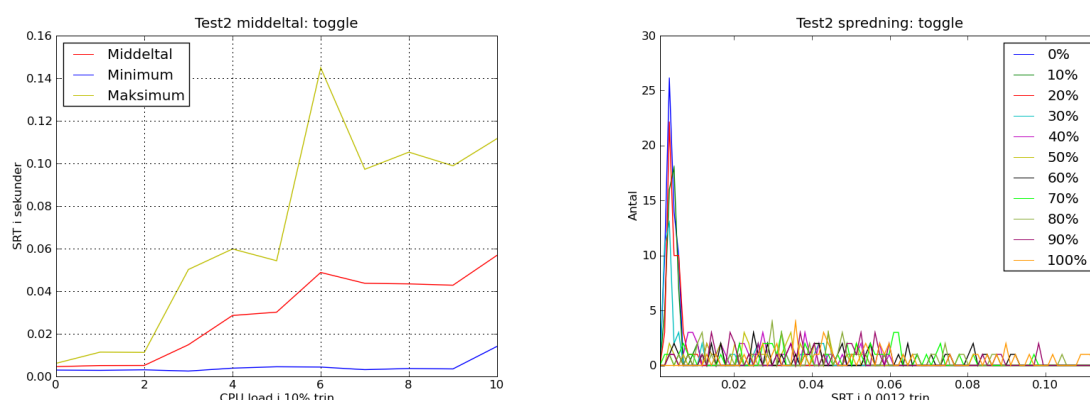
Figur 7.5 Middeltal og spredning for kommandoen getVerses i test 2

7. Test

På figurerne 7.6 ses resultaterne for kommandoen toggle. På den venstre figur ses maksimumsgrafen hvor målingen for 60% ligger markant højere end alle de andre målinger. Hvis der ses bort fra denne måling, kan det ses at maksimumsværdierne og gennemsnitsværdierne for play kommandoen, stiger tilnærmelsesvist lineært.

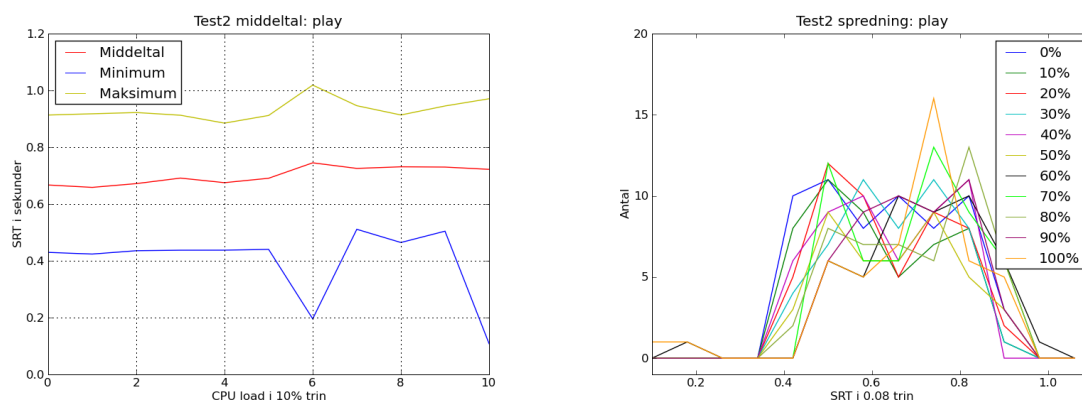
Gennemsnitsværdierne starter på 0,005 sekunder og stiger til 0,112. Ved 70% CPU belastning stiger kommandoens maksimums SRT op til 0.1 sekunder hvilket er kravet fra kravspecifikationen.

Spredningen, for kommandoen toggle, minder meget om spredningen ved kommandoen getVerses. Værdierne for deltestene 0 til 30% ligger markant lavere placeret end de andre deltests som er spredt mere ud.



Figur 7.6 Middeltal og spredning for kommandoen toggle i test 2

For kommandoen play kan der på figurerne 7.7 ses resultaterne af test 2. På grafen over middeltallet ses, at SRT for denne kommando ligger forholdsvis stabilt under alle deltests. Alle middeltalsværdierne ligger over de 0,1 sekund og overholder kravspecifikationen. Dog er maksimumsværdierne over 0,72 sekund hvilket ikke overholder kravspecifikationen.



Figur 7.7 Middeltal og spredning for kommandoen play i test 2

7.2.3 Forbindelse

Forbindelsen er bindeleddet mellem de to enheder og i dette afsnit testes hvordan systemet opfører sig når dette led bliver stresset.

Test 3

Forbindelsen testes med en Zyxel ZyWall 5 switch sat ind mellem netforbindelsen på værten og klienten. Switchen kan begrænse båndbredden mellem to porte og det er denne funktionalitet der benyttes i denne test.

Båndbredden begrænses i deltestene fra 6kb/s ned til 1kb/s hvilket er den største begrænsning switchen tillader. I hver test registreres SRT for kommandoerne.

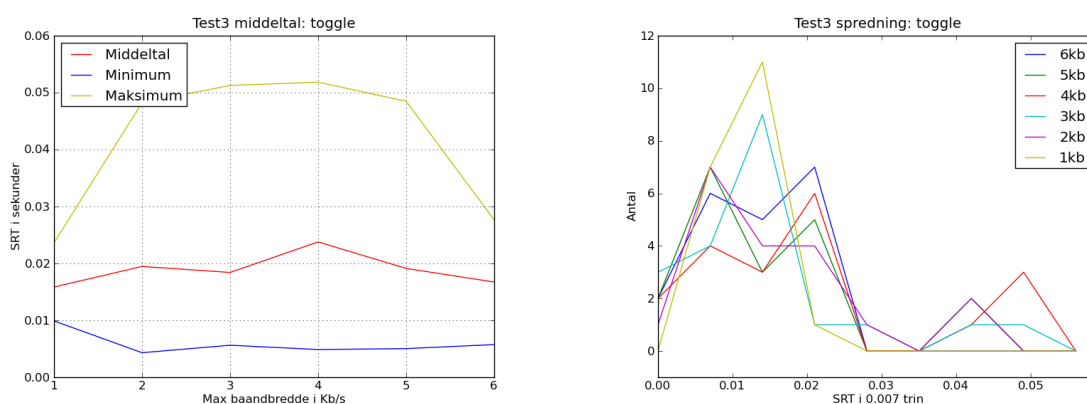
Kommandoerne bliver sendt over netværket i grupper pulser i samme rækkefølge, men med et længere mellemrum mellem hver gruppe. Rækkefølgen er tidligere beskrevet i afsnit 7.2.1 og rækkefølgen er ens i alle tests og granulariteten af begrænsningen på switchen ikke vides, kan det være, at den første kommando, der bliver sendt igennem switchen, øjeblikkeligt, mens næste kommando skal vente indtil næste timeslot. Ved forskellig båndbreddebegrænsning flytter punktet, hvor switchen er mættet, sig og kommandoerne kan opleve stor forskel i SRT mellem to, på følgende, deltests.

Fejlkilder

Zyxel Zywall 5 switchen brugt i denne test har ingen information omkring granulariteten af begrænsningen af båndbredden. I værste fald kan hvert timeslot være på 1 sekund og switchen tillader fuld trafik indtil begrænsningen er nået. Hvis det er tilfældet vil forskellen på maksimumsværdierne og minimumsværdierne i resultaterne være stor da nogle af kommandoerne kommer igennem switchen øjeblikkeligt, mens andre må vente til næste timeslot med at komme frem til destinationen.

Resultater

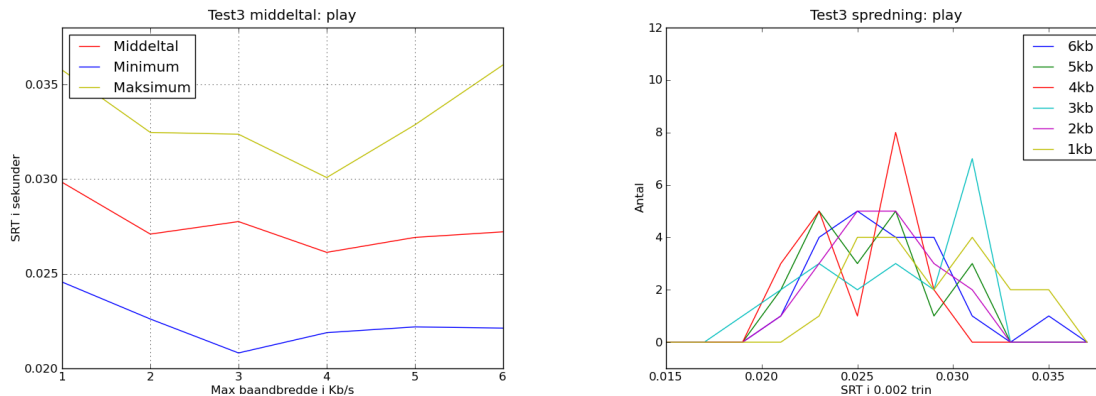
Resultaterne for kommandoen toggle kan ses på figurerne 7.8 hvor den venstre figur beskriver middeltal, maksimumsværdi samt minimumsværdi og den høje figur viser spredningen for den registrerede data. Middeltallet for kommandoen holder sig i det samme område igennem alle deltests. Hvilket kan skyldes at toggle er den første kommando der bliver sendt afsted i hver gruppe af kommandoer da den ikke bliver påvirket af begrænsningen.



Figur 7.8 Middeltal og spredning for kommandoen toggle i test 3

7. Test

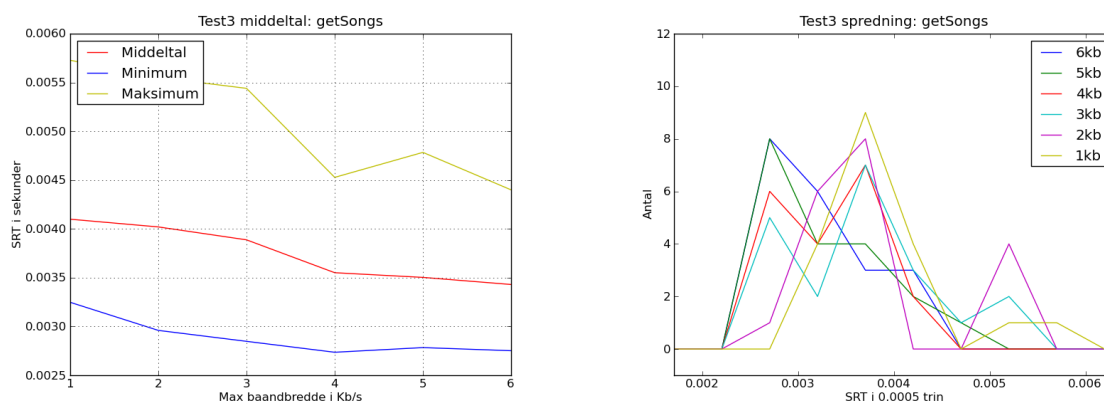
Figureerne for kommandoen play kan findes på 7.9. På den venstre figur ses det at middeltalsværdierne falder når begrænsningen falder. Ved en begrænsning på 4Kb/s falder middeltallet til et minimum og stiger lidt ved begrænsningen på 5 Kb/s hvor den forsætter på omkring 0,027 sekunder. Graferne over spredningen ligger fint placeret uden høje værdier i randområderne.



Figur 7.9 Middeltal og spredning for kommandoen play i test 3

Som i test 1 og test 2 bliver kun en af de tre kommandoer getSongs, getTitle og getVerses da de er implementeret ens i systemet og resultaterne for disse tre kommandoer minder også meget om hinanden.

Figureerne for getSongs kan ses på 7.10 hvor graferne for middeltal, minimumsværdi og maksimumsværdi, er på den venstre figur og graferne for spredningen, på figuren til højre. Middeltallets højeste værdi er ved begrænsningen på 1 Kb/s og falder indtil den er lavest ved begrænsningen på 6 Kb/s. Spredningen kan ses på figuren til højre og det ses at værdierne for deltestene med 4, 5 og 6 Kb/s begrænsning ligger før deltestene med 1, 2 og 3 Kb/s begrænsning.



Figur 7.10 Middeltal og spredning for kommandoen getSongs i test 3

7.2.4 Brugerinteraktion

I systemet bladrer brugeren igennem sangkataloget med en pulsgiver. For at undersøge om systemet kan håndtere, at en bruger drejer ved maksimum hastighed udføres følgende test.

Test 4

Først laves en test med pulsgiveren. Her findes det korteste mellemrum mellem to pulser ved operation af en bruger. Til at finde denne tid bruges et simpelt test setup med pulsgiveren koblet til et oscilloskop, mens en bruger drejer pulsgiveren i højeste tempo.

Ved hurtig operation af pulsgiveren er det kommandoen `getTitle` der vil blive sendt til værten. Stimulatoren bliver indstillet til at genererer interrupts til systemet med højt interval og intervallet nedtrappes indtil systemet kan håndtere hvert af disse interrupts, før det næste bliver startet. Tiden mellem hvert interrupt genereret af Stimulatorenheden og SRT'en i systemet aflæses på et oscilloskop.

Resultat

Korteste mellemrum mellem to pulser fra pulsgiveren ved operation fra en bruger er fundet til ca. 6,16 millisekunder. Dette resultat er den hurtigste registrering i en serie af tests.

Den korteste mellemrum mellem to interrupts hvor systemet fungerer normalt, er fundet til 4 millisekunder.

Ved et mellemrum på 3 millisekunder fungerer systemet stadig, men SRT'erne begynder at overskride mellemrummet, så der efter længerevarende maksimum belastning kan forekomme længere SRT'er.

Ved interval på 2 millisekunder stopper systemet med at virke, da kommandoerne bliver sendt over netværket med så kort mellemrum, at værten ikke opfatter kommandoer som separate strenge, men som en samlet streng. Dette problem skyldes implementeringen af applikationslagsprotokollen og kan løses ved at ændre separatoren i protokollen fra en ny linie til en tab.

7.3 Delkonklusion

I test 1 i afsnit 7.2.1 hvor systemet blev testet mens CPU'en på klienten blev belastet, blev det vist at SRT'en for kommandoerne ikke ændrede sig ved høj CPU belastning. Da størstedelen af koden i programmet på klienten kører i user-space og derfor har lavere prioritet i scheduleren, kan det konkluderes at programmet på klienten ikke er CPU intensivt.

Dette betyder at der er CPU tid til, at kunne udbygge klientsiden af systemet med mere software eller skifte til hardware med mindre ressourcer, og lavere indkøbspris.

Test 2 i afsnit 7.2.2 viste at CPU belastning på klienten havde en stor indflydelse på SRT'en i systemet. SRT'en steg tilnærmelsesvist lineært for kommanden `getVerses` og endte over 22 gange højere end ved deltesten uden CPU belastning.

Kommandoen `play` blev ikke nævneværdigt berørt af CPU belastningen, hvilket viser at det er andre systemressourcer der sætter grænser for kommandoen. Toggle kommandoen, der i høj grad bruger mindre filsystemetlæsning end `play` kommandoen, blev mere berørt af CPU belastningen, hvilket indikerer at det kan være filsystemetlæsningen fra sd-kortet, der sætter grænsen for både `play` og `toggle` kommandoen. En test af filsystemet vil dog ikke blive udført i dette projekt af tidsmæssige årsager, så det er ikke muligt at komme med en bekræftelse på denne indikation.

Netværkstesten i afsnit 7.2.3 viste at dataudvekslingen mellem klienten og værten ikke blev nævneværdigt påvirket af restriktionerne af båndbredde, så det kan udledes at det er muligt at kunne benytte systemet i et eksisterende netværk, hvor der er tilkoblet andre netværksenheder.

Ved et lavt interruptinterval vil kernel-space CPU forbruget på klienten stige og det kan føre til længere behandlingstid på klienten.

Det blev vist i test 4 i afsnit 7.2.4 at en bruger ikke kan dreje pulsgiveren hurtigere end systemet kan nå

7. Test

at give respons på.

I test 4 ses det dog også at implementationen af applikationlagsprotokollen vil fejle ved et lavt interrupt-interval. Da der ikke er nogen form for fejlhåndtering af dette problem på værten er det vigtigt at det bliver rettet, så et enkeltstående tilfælde af et lavt interval mellem to interrupts, kan lægge systemet ned.

I test 1 og test 2 er der stor forskel på SRT for kommandoen play. Denne forskel virker uforklarlig og denne rapport vil ikke kaste sig ud i gætterier omkring årsagen, men nøjes med at konstatere at forskellen er til stede.

Konklusion

8.1 Konklusion

Målet for dette projekt har været at konstruere, implementere og teste et modul baseret jukebokssystem målrettet musikere og dj's.

Use-casen tog udgangspunkt i virksomheden Event-Danmarks ønske om et jukebokssystem, der kan give virksomhedens kunder mulighed for, at afspille en række færdigindspillede akkompagnementer til brug ved fester og andre lejligheder.

I foranalysen blev det undersøgt hvilke krav brugere har til SRT i et sådant system, om systemet skulle baseres på en model med en tynd klient og hvilken datakommunikationen systemet ville kræve.

Kravspecifikationen formaliserede kravene fra foranalysen og designkapitlet forsøgte kravene opfyldt i et system.

Testafsnittet viste at var muligt at konstruere et modulært jukeboxsystem som kunne opfylde kravet om at kommandoerne `getVerses`, `getTitle` og `getSongs` ikke må have en SRT på mere end 100 millisekunder. For kommandoerne `toggle` og `play` kunne systemet overholde kravet på en SRT på under 720 millisekunder, i test 1 og 3 men ikke når CPU'en på værten blev belastet i test 2.

Testafsnittet viste også at systemet kunne opføre sig stabilt under ekstreme situationer.

Til spørgsmålet i problemformuleringen - om det er muligt at konstruere et system med to enheder der kan give systemet en SRT på 100 millisekunder ved simple kommandoer og en SRT på 720 millisekunder ved mere komplicerede kommandoer - må svaret være ja, men ikke i alle situationer.

Projektforslaget indeholdt en mulighed for en implementation af USB protokolstacken, men det har ikke været muligt, at nå at implementere og teste det inden for tidsrammen af projektet. Dog er designet af programmerne, på klienten og værten, veldefineret, så der er mulighed for, uden de store modificeringer, efterfølgende at implementere systemet med USB protokolstacken.

Perspektivering

Dette projekt er ment som en prototype til demonstration men er samtidig et godt fundament til et færdigudviklet jukebokssystem.

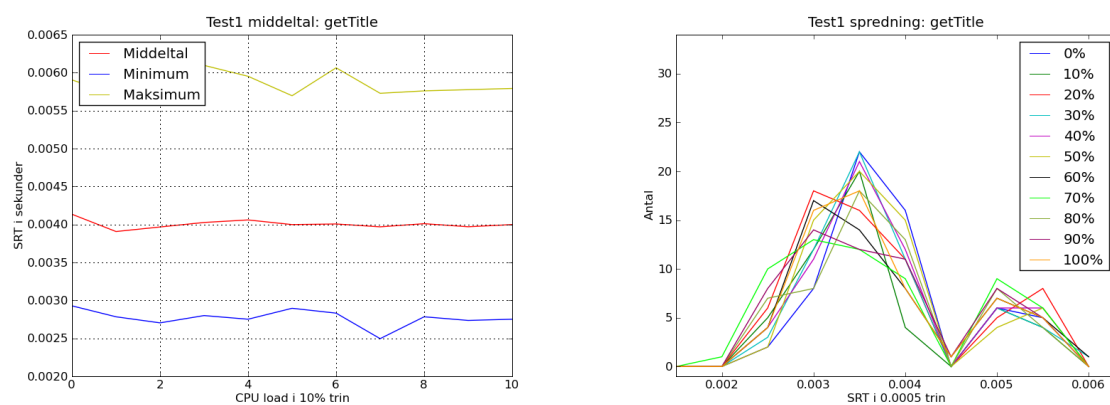
Hvis systemet dog skal bruges kommercielt af en virksomhed er der en række aspekter der først skal adresseres. Det kan f. eks. være sikkerhed så for at undgå kopiering af akkompagnementerne gemt i filsystemet kan der med fordel implementeres et krypteret filesystem til at opbevare musiknumrene i. Dette kan give et performance tab men kan medvirke til at sikre en producent af produktet imod at musikfilerne bliver kopieret af langfingrede brugere.

Prototypen udviklet og dokumenteret i dette system kan også benyttes i andre sammenhænge end musikafspilning. Det kan være til fjernkontrol af enheder i f.eks. et hus hvor der allerede er lagt en netværksinfrastruktur.

Implementationen af systemet giver også mulighed for flere klienter kan været koblet til den samme vært og kan udvides til at benytte en trådløs forbindelse med f.eks. trådløst netværk.

A.1 Resultater

A.1.1 Tests



Figur A.1 Middeltal og spredning for kommandoen getTitle i Test1

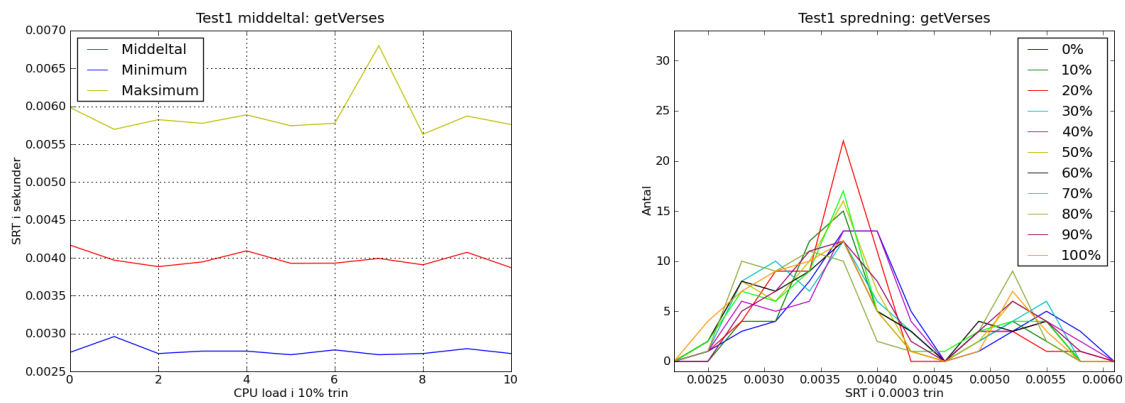
A.2 Vært

A.2.1 main.Player Class Reference

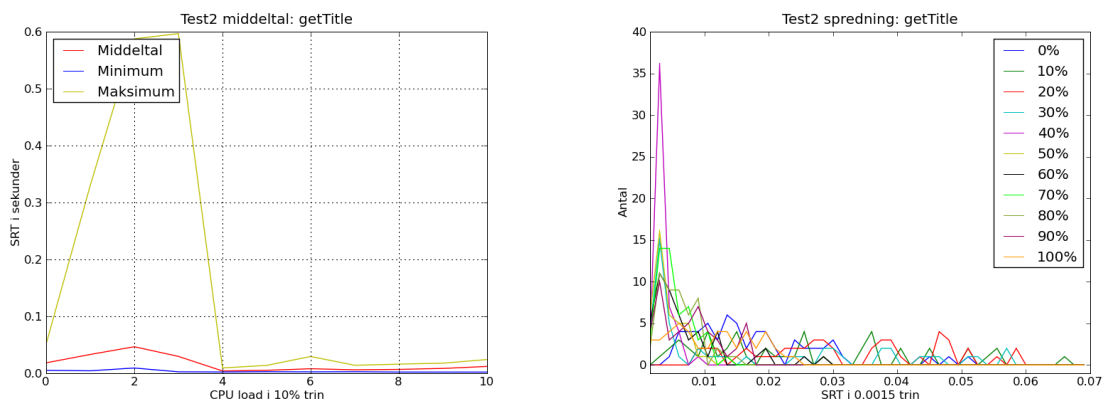
Denne klasse står for afspilningen af musiknumre. Den er en tråd og kommunikerer med Application klassen over køer.

Functioner

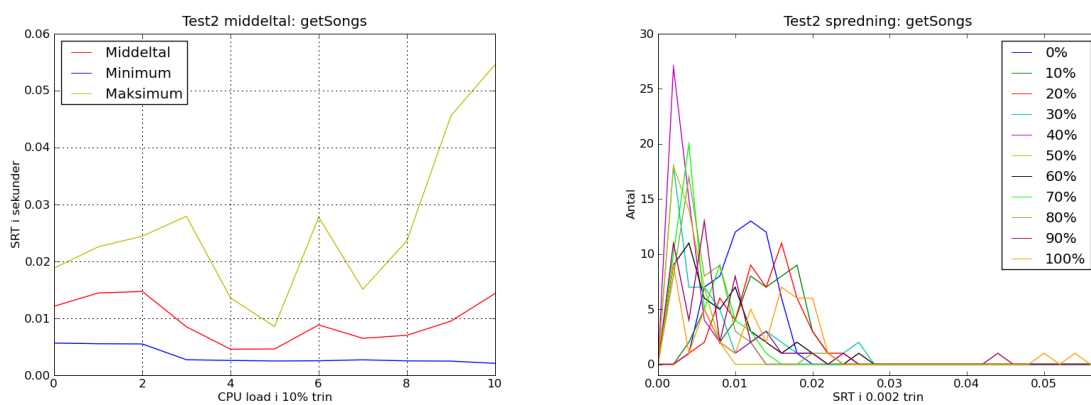
A. Appendiks



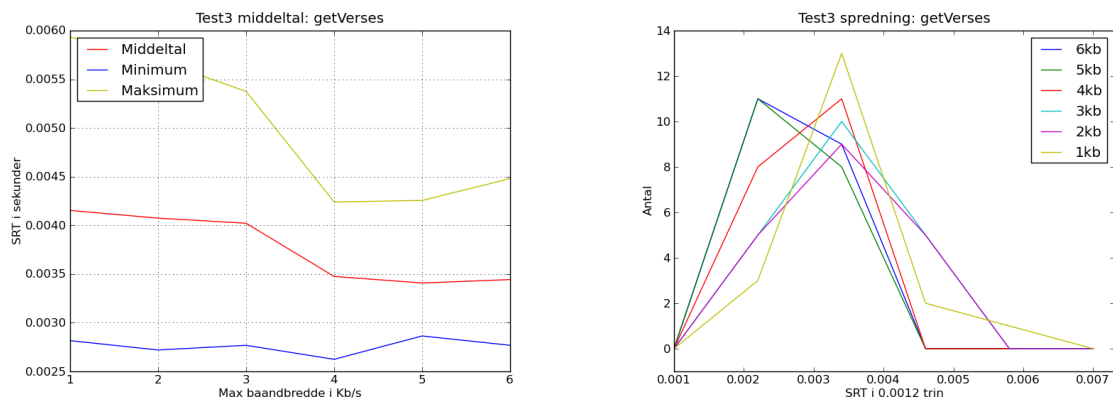
Figur A.2 Middeltal og spredning for kommandoen getVerses i Test1



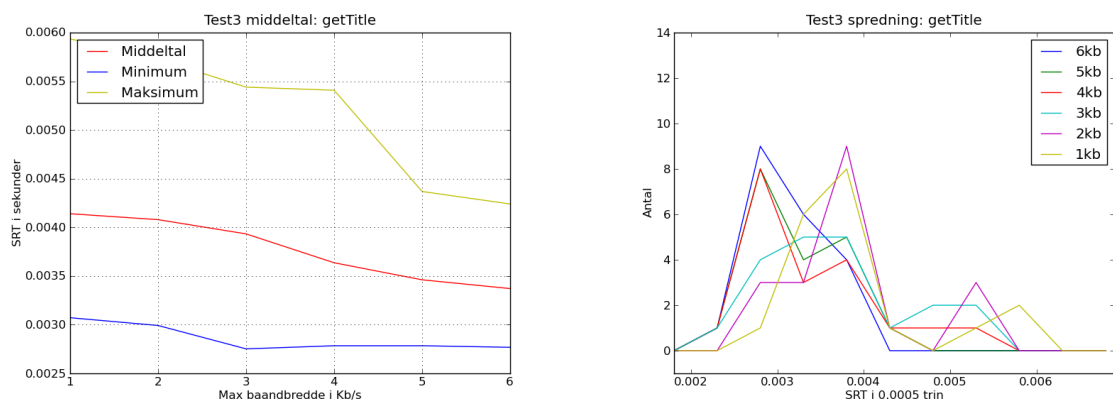
Figur A.3 Middeltal og spredning for kommandoen getTitle i test 2



Figur A.4 Middeltal og spredning for kommandoen getSongs i test 2



Figur A.5 Middeltal og spredning for kommandoen getVerses i test 3



Figur A.6 Middeltal og spredning for kommandoen getTitle i test 3

- `def main.Player.__init__( self, queuePlayer, queueApplication)`
Initieringen af klassen. Her bliver de to køer der bruges til at kommunikere med Application klassen sat.
- `def main.Player.run ( self)`
Den evige løkke hvor play funktionen bliver kaldt igen og igen.
- `def main.Player.play ( self)`
Afspiller funktion der afspiller filen i variablen file.
- `def main.Player.checkQueue ( self)`
Funktion der bliver brugt at play funktionen, under afspilning, til at kontrollere om der er kommet ny data i en afspiller køn.

A.2.2 main.JukeProtocol Class Reference

Denne klasse definerer regler for hvordan værten skal opføre sig ved forskellige forbindelses hændelser. Klassen nedarver fra twisted's generelle protocol klasse.

Nedarver fra twisted::internet::protocol::Protocol.

Public Member Functions

- def main.JukeProtocol.connectionMade (self)
Når en forbindelse oprettes skrives der forbindelse oprettet i loggen samt et tidsstempel.
- def main.JukeProtocol.connectionLost (self)
Når en forbindelse oprettes skrives der forbindelse oprettet i loggen samt et tidsstempel.
- def main.JukeProtocol.dataReceived (self, data)
Denne funktion modtager data fra netværket og sender det videre til factory klassen som processerer det. Processeringen foregår asynkront og efter endt processing bliver status sendt tilbage til klienten.

A.2.3 main.JukeFactory Class Reference

Denne klasse er en wrapper klasse som sørger for at processeringen af netværks data bliver gjort asynkront.

Public Member Functions

- def main.JukeFactory.processData (self, data)
Funktionen sender processeringsdata videre.

A.2.4 main.Application Class Reference

Denne klasse står for selve processeringen af netværks hændelser.

Public Member Functions

- def main.Application.processCommand (self, data)
Denne funktion splitter dataen op i to dele. Den ene del er kommandoen og den anden del er et medfølgende argument. Disse kommandoer bliver udført og resultatet af udførelsen bliver returnet.
- def main.Application.play (self, file_id)
Funktionen starter sender signal til Player tråden om at starte afspilningen af filen specificeret i file_id.
- def main.Application.toggle (self)
Toggle sender signal til Player tråden om at toggle den nuværende sang.
- def main.Application.stop (self)
Denne funktion sender signal til Player tråden om at stoppe den nuværende sang.
- def main.Application.getSongs (self)
Retunerer antallet af sange tilgængelige.

- `def main.Application.getTitle ( self, index)`
Funktionen returnerer fil stien på sangen tilhørende fil id'et index.
- `def main.Application.getVerses ( self)`
Retunerer antallet af vers i sangen med fil id'et index.
- `def main.Application.ping ( self)`
Retunerer "pong" til klienten.
- `def main.Application.error ( self)`
Hvis der opstår en fejl på klienten vil fejlkoden blive sendt til værten og skrevet til loggen.

A.3 Klient

A.3.1 main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include "network.h"
#include "disp.h"
#include "queue.h"
#include "gpio.h"
```

- `namespace main::c`
Dette er klient programmet. Dette program står for at registrere inputs fra knapper og videresende dem som kommandoer til værten. Respons fra værten bliver vist på displayet der er koblet til klienten.

Functions

- `void * gpioFunction (void *arguments)`
Denne funktion er input tråden. Tråden kommunikerer med display tråden og netværk ud tråden ved hjælp af to besked køer. Denne funktion åbner forbindelsen til de to køer samt forbindelsen til input knappernes kerne interface. Efter det ventes der på input fra knapperne.
- `void * netOutFunction (void *arguments)`
Funktionen er netværk ud tråden. Denne tråd åbner de to køer til displayet og netværket, lytter til netværks køen og videresender alt data fra køen over netværket til værten.
- `void * netInFunction (void *arguments)`
Lytter til netværket og videresender alt data til display køen.
- `void * dispFunction (void *arguments)`

Tråden lytter til display køen og sender data sendt til køen ud til displayet.

- `int main (void)`

Hoved programmet der initialiserer de fire tråde med netværk ind, netværk ud, display og input fra knapperne.

-

A.3.2 Msgbuf Struct

```
#include <queue.h>
```

Data Fields

- `long mtype`
- `char message [QUEUE_STRING_SIZE]`

A.3.3 queue.c

```
#include <stdlib.h>
```

```
#include <sys/msg.h>
```

```
#include "queue.h"
```

```
#include <stdio.h>
```

- `namespace queue::c`

Modulet opretter, skriver, læser og lukker besked køer.

Defines

- `#define ALL_RW 0666`

Functions

- `int createQueue (int number)`

Opretter en kø svarende til den key i argumented number.

← *number er en key der svarer til en bestemt kø.*

Retunerer kø file descriptor ved succes og -1 ved fejl. Hvis køen allerede findes i forvejen returneres file descriptor til den eksisterende kø.

- `int writeQueue (int mQueueId, msgbuf *msg)`

Funktionen tilføjer en besked til en kø.

← *mQueueID er file descriptor på en besked kø.*

← *msg er en pointer der peger på en msgbuf.*

Retunerer 0 ved succes og -1 ved fejl.

- `int readQueue (int mQueueId, msgbuf *msg, int type)`

Denne funktion læser og fjerner den først tilkommende besked i en kø.

← *mQueueId er en file descriptor til en besked kø.*

→ *msg er en pointer til hvor den læste msgbuf skal placeres.*

← type er hvilken gruppe beskeden der skal læses tilhører.
Retunerer 0 ved succesfuld læsning og -1 ved fejl.

- int closeQueue (int mQueueId)

Lukker for en kø.

← mQueueId er en file descriptor på den kø der skal lukkes ned.
Retunere 0 ved succesfuld nedlukning af kø og -1 ved fejl.

A.3.4 gpio.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <linux/input.h>
#include <fcntl.h>
#include "queue.h"
#include "gpio.h"
```

- namespace gpio::c

Dette modul står for at oversætte knappernes nummer til kommandoer og videresende dem til netværks køen.

Defines

- #define EVENT_INTERFACE "/dev/input/event0"
- #define BUTTON_1 KEY_1
- #define BUTTON_2 KEY_2
- #define BUTTON_3 KEY_3
- #define BUTTON_4 KEY_4
- #define BUTTON_5 KEY_5
- #define BUTTON_6 KEY_6

Functions

- int openGPIOs (void)

Funktionen åbner event interfacet til kernemodulet der registrerer interrupts fra knapperne.

Denne funktion retunerer en file descriptor til det åbne event interface eller -1 ved fejl.

- int readGPIOs (int fd, int queueNetOut, int queueDisp)

Denne funktion læser fra event interfaceet og oversætter knappens værdi til en kommando der sendes videre ved hjælp af netværks køen. Ved sendes en fejlbesked til display køen.

- ← *fd er en file descriptor til et event interface.*
- ← *queueNetOut er en file descriptor til netværks besked køen.*
- ← *queueDisp er en file descriptor til display besked køen.*
Funktionen er en evig løkke der venter på input og returnerer aldrig.

- **int closeGPIOs (int fd)**

Lukker event interfacet til file descriptoren i argumentet.

- ← *fd er en file descriptor til event interfacet*
Retunerer 0 ved succes og -1 ved fejl.

A.3.5 disp.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>
#include <fcntl.h>
#include "disp.h"
```

- **namespace disp::c**

Dette modul står for at åbne SPI forbindelsen til displayet, sende de maximum 4 HEX cifre og lukke forbindelsen.

Defines

- **#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))**
- **#define ERROR -1**

Functions

- **int sendDisp (int fd, uint8_t *tx)**

Denne funktion sender data ved hjælp af SPI til displayet.

- ← *fd er en file descriptor til SPI interfacet.*
- ← *tx er en integer pointer til den data der skal sendes.*
Retunerer 1 ved succes og ERROR ved fejl.

- **int openDisp (void)**

Åbner SPI forbindelse til displayet.
Retunerer en file descriptor til SPI displayet.

- **int closeDisp (int fd)**

- Lukker SPI forbindelsen til displayet.*
- ← *fd er en file descriptor til SPI interfacet.*
Retunerer 1 ved succes og -1 ved fejl.

A.3.6 network.c

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "network.h"
```

- namespace network::c

Dette module står for kommunikationen over netværket. Det åbner en socket til værten, sender data, modtager data og lukker forbindelsen til værten.

Functions

- int initSocket (void)

Denne funktion opretter en tcp/ip netværks forbindelse til værten defineret som SERVER_IP på porten PORT.

Retunerer file descriptor til en socket eller ERROR ved fejl.

- int sendSocket (int sock, char *message)

Sender en besked over tcp/ip til værten.

← sock er en file descriptor til socket'en.

← message er en char pointer til beskeden der skal sendes.

Retunerer længden på den afsendte besked eller ERROR ved fejl.

- int recvSocket (int sock, char *message)

Modtager indgående data fra værten.

← sock er en file descriptor til socket'en.

→ message er en char pointer hvor den modtagne besked placeres.

Returnerer antal modtagne bytes ved succe eller ERROR ved fejl.

- int closeSocket (int sock)

Lukker for forbindelsen til værten.

← sock er en file descriptor til den åbne socket.

Retunerer 0 ved succes og -1 ved fejl.

Litteratur

Alps: 2000, *Datasheet - 11mm Size Metal Shaft Encoders*

Atmel: 2007, *Datasheet - AT32AP7000*

Densitron: 2008, *Datasheet - DD-25664BE-4A*

Goodman, T. & Spence, R.: 1978, *The Effect of System Response Time on Interactive Computer Aided Problem Solving*

In-circuit: 2009, *Schematic - ICnova AP7000 base*

Kontron: 2009, *Datasheet - pItx-SP-preliminary*

Miller, R. B.: 1968, *Response time in man-computer conversational transactions*

Shneiderman B. & Plaisant C.: 2005, *Designing the user interface*