

Webserver

Peter Andersen, Arne Jørgensen og Lotte Simonsen (gruppe 1)

14. december 2004

Indhold

1. Indledning	5
1.1. Læsevejledning	5
2. Problemformulering	6
2.1. Opgavebeskrivelsen i egne ord	6
2.2. Valg af sprog og programmeringsmetode	6
2.3. Afgrænsning	7
2.4. Ambitionsniveau	7
2.5. Tidsplan	7
3. Problemanalyse	8
3.1. URL	8
3.2. Parallelisme	8
3.3. Fejlhåndtering ved klientforespørgsler	9
3.4. Fejlhåndtering ved serverfejl	10
3.5. MIME-type	10
3.6. HTTP Headers	11
3.7. Struktur	11
3.8. Statistik	12
3.9. CGI-scripts	12
3.10. Leksikalsk analyse	12
3.11. Syntaktisk analyse	13
3.12. Load-balancing	13
4. Design	14
4.1. Brugsmodeldiagram	14
4.2. Sekvensdiagram	15
4.3. Klassediagram	16
4.4. Programmeringsovervejelser	17
5. Valg	17
5.1. URL	17
5.2. Parallelisme	18
5.3. Fejlhåndtering ved klientforespørgsler	18
5.4. Fejlhåndtering ved serverfejl	18
5.5. MIME-type	19
5.6. HTTP Headers	19
5.7. Struktur	19
5.8. Statistik	20
5.9. CGI	20
5.10. Lexer	21
5.11. Parser	22

5.12. Sende og modtage over nettet	22
6. Metode og værktøj	24
7. Forbedringer	24
8. Programdokumentation til webserver	25
9. Test	26
9.1. GET	27
9.2. HEAD	28
9.3. PUT	28
9.4. POST	28
9.5. CGI	28
9.6. Testresultater	28
10. Konklusion	29
A. Tilstandsmaskiner til leksikalsk analyse	30
A.1. Lexning af HTTP-request line	30
A.1.1. Tilstandstabel	31
A.1.2. Aktionstabel	31
A.2. Lexning af absolut path i HTTP-forespørgselslinjen	32
A.2.1. Tilstandstabel	32
A.2.2. Aktionstabel	33
A.3. Lexning af HTTP-headers	33
A.3.1. Tilstandstabel	34
A.3.2. Aktionstabel	34
A.4. Lexning af HTTP-headeren Host	34
A.4.1. Tilstandstabel	35
A.4.2. Aktionstabel	35
B. Testdata	35
B.1. Blandede forespørgsler	35
B.2. GET	36
B.3. HEAD	36
B.4. PUT	37
B.5. POST	37
B.6. CGI	38
C. Koden	38
C.1. Makefile	38
C.2. webd.cpp	40
C.3. server.h	41
C.4. server.cpp	42

C.5. webserv.h	43
C.6. webserv.cpp	44
C.7. connection.h	46
C.8. connection.cpp	48
C.9. request.h	51
C.10. request.cpp	53
C.11. response.h	55
C.12. response.cpp	57
C.13. statistics.h	67
C.14. statistics.cpp	68
C.15. lexer.h	69
C.16. lexer.cpp	71
C.17. requestlineLexer.h	74
C.18. requestlineLexer.cpp	74
C.19. pathLexer.h	76
C.20. pathLexer.cpp	76
C.21. httpheaderLexer.h	78
C.22. httpheaderLexer.cpp	78
C.23. hostheaderLexer.h	79
C.24. hostheaderLexer.cpp	80
C.25. HTTPVersion.h	80
C.26. HTTPVersion.cpp	81
C.27. URL.h	82
C.28. URL.cpp	84
C.29. HTTPHeaders.h	88
C.30. HTTPHeaders.cpp	89
C.31. mimetype.h	90
C.32. mimetype.cpp	91
C.33. statusCode.h	92
C.34. statusCode.cpp	92
C.35. lerror.h	95
C.36. lerror.cpp	95
C.37. mutex.h	96
C.38. mutex.cpp	97
C.39. tools.h	98
C.40. tools.cpp	98
C.41. BrokenConnectionException.h	100
C.42. RequestException.h	101

Litteratur	102
-------------------	------------

1. Indledning

På tredje semester af datamatikeruddannelsen ved Niels Brock undervises i fagene *Datamatiske Systemer* og *Systemprogrammering*. I fagene indgår et større projekt, der dette semester er udviklingen af en fungerende webserver. Dette projekt skal munde ud i en skriftlig rapport, som skal danne grundlag for eksamen.

Vi tager udgangspunkt i at læseren er bekendt med systemudvikling og programmering svarende til mindst tredje semester af datamatikeruddannelsen. Herunder et kendskab til UML og objekt-orienteret programmering.

1.1. Læsevejledning

Indledningsvist lægger vi i afsnit 2 ud med en problemformulering hvor vi beskriver, hvilke krav der stilles til projektet, vores ambitionsniveau, overvejelser omkring programmeringsmetode, afgrænsning og en tidsplan.

I afsnit 3 følger en problemanalyse af en række, af de teknologier og begreber der finder anvendelse i en webserver og i forbindelse med HTTP-protokollen. Analysen har ført til at vi under implementationen har måtte træffe en række valg. Disse valg argumenterer vi for i afsnit 5.

Inden vi begrundet valgene belyser vi, i afsnit 4, vores design af webserveren – herunder hvilke brugsmønstre vi har identificeret, klassediagrammet, programmeringsovervejelser, mm.

Vi beskriver hvilke værktøjer og metoder vi har anvendt og haft nytte af ved løsning af opgaven i afsnit 6, og herpå følger en række bud på forbedringer mm. som kunne implementeres i vores webserver. Disse kan findes i afsnit 7.

Herefter følger i afsnit 8 en kort programdokumentation og dernæst en beskrivelse af vores test i afsnit 9.

I afsnit 10 konkluderer vi på projektet, problemstillingerne og læreprocessen i øvrigt.

Bilag A dokumenterer de tilstandsmaskiner med tilstands- og aktionstabeller vi har haft brug for. Bilag B rummer de data der har dannet grundlag for vores afprøvelse.

Kildekoden til webserveren er optrykt i bilag C.

Henvisninger af typen [8] henviser til litteraturfortegnelsen på side 102.

2. Problemformulering

2.1. Opgavebeskrivelsen i egne ord

Opgaven består i at udvikle en webserver i programmeringssproget C eller C++, i og til et GNU/Linux-miljø og som skal kunne køre på Red Hat 9.0. Webserveren skal være i stand til at håndtere forespørgsler efter HTTP/1.0-protokollen (HEAD og GET). Webserveren skal kunne opføre sig passende på forespørgsler efter tilgængelige og ikke-tilgængelige URL'er. Det går blandt andet ud på at returnere korrekte MIME-typer, og udføre den rette handling hvis en forespørgsel ikke kan opfyldes. Ved forespørgsler på eksisterende mapper skal webserveren kunne returnere indholdet af mapperne i deres korrekte niveau, samt give mulighed for at kunne navigere op og ned i mappestrukturen. Derudover skal webserveren kunne opsamle statistik over forespørgsler og fejl til brug for en administrator.

Opgavens problemområder består blandt andet i at:

- oprette og håndtere netværksforbindelser til klienter
- håndtere flere parallelle forbindelser
- lexe, parse og forstå URL'er og forespørgsler

Udarbejdelse af programmeringsopgaven skal munde ud i et program der kan løse ovenstående opgave, samt en dokumenterende rapport af forløbet.

Vi har fundet nogle områder i den udleverede problembeskrivelse der kunne uddybes. Fx når der skal indsamles statistik kan det være svært at vide, hvad der vil være relevant information at samle på. Også omkring genkendelsen af filtyper og deres MIME-typer kan der være flere valgmuligheder. Derudover kan hele området omkring fejlhåndtering være genstand for flere valgmuligheder. Undervejs i rapporten vil vi se nærmere på disse områder, diskutere nogle af valgmulighederne omkring dem, og træffe nogle valg på baggrund af vores diskussion.

2.2. Valg af sprog og programmeringsmetode

Som udgangspunkt vælger vi at implementere webserveren i C++ som en objektorienteret løsning. Valget er begrundet i, at det er det sprog gruppen har den største erfaring med, og vi vil benytte indkapsling til at skjule tekniske detaljer. Ofte vil det dog være relevant at benytte »klassiske« C-strukturer til fx netværkshåndtering, samt proces- og trådhåndtering.

Vores fremgangsmåde med hensyn til programmering er, at vi vil diskutere hver enkelt classes interface, og derefter vil vi sætte os enkeltvis og udvikle klasserne. Vi vil sætte os i samme lokale og programmere, da det så vil være nemmere at diskutere eventuelle

problemer, der kan opstå undervejs. Endvidere vil vi ved udviklingen af nogle af de større klasser forsøge os med parprogrammering som udviklingsmetode, for der igennem at lære af hinanden og hurtigere identificere fejl.

2.3. Afgrænsning

Vi har valgt generelt at afgrænse implementationen således, at når kernen af et problem er belyst og dets løsninger beskrevet, er det vigtigere for os at komme videre til næste problem end at gentage besvarelsen yderligere. Et eksempel kan være HTTP-headers og MIME-typer hvor vi kun arbejder med få udvalgte.

En anden afgrænsning er at hvis klienten anmoder om en URL der svarer til en mappe på webserveren og denne indeholder en index.html-fil, så præsenterer vi ikke klienten for index.html-filen, men mappeoversigten. Det gør vi ikke selvom det er en de facto-standard der ofte ses implementeret i webservere.

Tidligt i projektforsløbet besluttede vi også kun at koncentrere os om at lexe den del af URL'en der hedder *abs_path*, i stedet for en hel URL. Det gør vi fordi det kun er *abs_path* der følger med i en HTTP-forespørgselslinje.

2.4. Ambitionsniveau

Gruppens ambition er at lave en fyldestgørende besvarelse, der kommer rundt om alle relevante emner i projektet. Det er vigtigt, at vi som minimum overvejer og beskriver de emner, der af tidsmæssige eller andre grunde ikke kan blive implementeret.

Vi bestræber os på at kunne nå at implementere en eller flere udvidelser. De udvidelser vi ikke kan nå vil dog som ovennævnt også blive overvejet og beskrevet.

Vores ambitionsniveau er, som ved de andre studieprojekter, ikke at opnå en bestemt eksamenskarakter, men handler derimod om forståelse af opgaven, problemstillingen og emneområdet. Det vigtigste er at aflevere et produkt, vi selv føler er godt og fyldestgørende.

2.5. Tidsplan

Tidsplanen er en grov skitse af projektforsløbet som løbende vil blive justeret, både i forhold til projektets fremskridt og i forhold til semesterets arbejdsbelastning i øvrigt.

Dato	Opgave
27/09-04	Have skimmet og fået overblik over RFC 1738 (URL), RFC 1945 (HTTP) og RFC 2045 og RFC 2046 (MIME-typer).
19/10-04	Have nærlæst relevante dele af ovennævnte RFC'er.
22/11-04	Start af projekt 3. Analyse og design af webserver er klar.
3/12-04	Programmering færdig.
9/12-04	Projektet færdigt og mangler kun korrekturlæsning og smårettelser.
14/12-04	Projektet afleveres.

3. Problemanalyse

3.1. URL

En URL er en streng der beskriver en ressource på internettet og en HTTP URL er en internet-ressource der kan tilgås ved hjælp af HTTP. En URL er forskelligt beskrevet i RFC 1738 [2] og RFC 1945 [1]. I RFC 1738 har den følgende opbygning:

```
http://<host>:<port>/<path>?<searchpart>
```

Forskellen på denne URL og den i RFC 1945 er, at den i RFC 1945 har valgt at lade parametre være selvstændige symboler, hvor URL'en i RFC 1738 tager dem med som en del af selve path'en. En lexer skal opdele URL'en i en række symboler, en mulighed kunne være at opdele ved »:«, »/« og »?«. Derefter sendes listen af symboler videre til en parser som helt eller delvist er opbygget efter Backus Naur-formen i RFC 1738. En parser ville i princippet kun skulle tjekke om der stod »http« i starten, og derefter om de forskellige symboler indeholdt de, i følge grammatikken, lovlige tegn.

3.2. Parallelisme

Da en webserver ikke er særlig effektiv hvis den kun kan servicere en klient ad gangen, skal man beslutte, hvordan man vil håndtere parallelisme. Der er som udgangspunkt to måder at gribe problemstillingen an på; processer og tråde.

Nye processer skabes med systemkaldet `fork()` og det kreerer en fuldstændig kopi (*barneprocessen*) af den kaldende proces (*forældreprocessen*). Dette kan gøres når serveren har accepteret en forbindelse fra en klient, og herefter kan man lade barneprocessen håndtere klienten mens forældreprocessen lytter efter nye opkald. Da barneprocessen ikke behøver at sende noget tilbage til forældreprocessen, kan man lade processen afslutte. Det signal der kommer ved en afsluttet barneproces, kan eventuelt samles op med en signalhandler eller ved at ignorere signalet.

En anden mulighed er at arbejde med tråde. Tråde er det man kalder letvægtsprocesser fordi de deler processens fælles adresserum med de andre tråde. De vil derfor ikke bruge lige så meget hukommelse som hvis man arbejdede med processer. Tråde krees med kaldet `pthread_create()` og blandt andet en funktion som argument. Denne funktion kan være den der håndterer klienten, imens kan serveren lytte videre efter nye forbindelser og lave en ny tråd til hver forbindelse. Da serveren ikke behøver at vide noget om de tråde den har lavet, kan man vælge at lave trådene »detached«, hvilket betyder at de ikke forsøger at sende noget tilbage til serveren, men bare afslutter når de har serviceret en klient.

En af de større forskelle på de to løsningsmuligheder er, at processer bruger mere hukommelse fordi den laver en fuldstændig kopi af den kaldende proces, mens tråden kun får et id, en program counter, nogle registre og en stak. Resten deler den med serveren og de andre tråde, hvis der er nogle. Det betyder også, at det tager kortere tid at lave en ny tråd end at kopiere hele processen. Hvis man skal gøre det mange gange kan man hurtigt spare tid ved kun at lave tråde, og ikke processer.

3.3. Fejlhåndtering ved klientforespørgsler

I princippet skal klienten have lov til at skrive hvad som helst i URL'en, men hvis det klienten ønsker at få vist eller eksekveret ikke findes, eller serveren ikke har rettigheder til det adspurgte, får klienten en fejlbesked tilbage i HTML. Undtagelsen er dog ved en PUT-forespørgsel, hvor der skrives til en fil der ikke eksisterer i forvejen – serveren vil her oprette filen.

I følge RFC 1945, skal svaret til klienten indeholde en statuskode, begrundelse og en beskrivelse af den opståede fejl. Dette øger brugervenligheden overfor klienten, som bedre vil kunne genkende og forstå fejlbeskederne.

Et sikkerhedsproblem kan være, at klienten forsøger at gå længere op i mappehierarkiet end til roden, fx ved at skrive »../« i URL'en. Man kan vælge at sende en fejlbesked tilbage til klienten, eller man kan også håndtere problemet ved at omdirigere svaret, således at klienten får indholdet af roden tilbage.

Da det er serveren der håndterer filafviklingen, ville der ingen skade ske ved en forespørgsel om at overskrive en fil, serveren ikke har skriverettigheder til. Serveren vil da blive nødt til, at sende en fejlbesked tilbage til klienten, ellers vil klienten ikke få besked om, at handlingen ikke blev gennemført.

Fejlhåndtering under afvikling af CGI-scripts er mere komplekst. Det følgende beskriver primært fejlhåndtering i forbindelse med Non-Parsed-Headers CGI-scripts. Hvis der opstår fejl i forsøget på at starte scriptet kan man trygt sende en fejlmeddelelse til klienten. Der kan imidlertid opstå en række fejl under udførelsen af CGI-scriptet der gør at scriptet afbrydes, fx at scriptet modtager et signal. Da scriptet *har* været under udførelse *kan* det have nået at sende data til klienten, men da man på den anden side

ikke kan garantere det *er* sket, må man overveje om man under alle omstændigheder skal sende en fejlmeddelelse til klienten.

Hvis CGI-scriptet afslutter normalt, men med en returværdi > 0 kan det opfattes som tegn på at CGI-scriptet ikke har kunnet udføre sin opgave (en konvention at programmer returnerer værdier > 0 hvis de ikke har kunnet udføre opgaven). I et sådant tilfælde må man tage stilling til om serveren skal sende en fejlmeddelelse til klienten eller om man vil gå ud fra at CGI-scriptet selv har formået dette.

3.4. Fejlhåndtering ved serverfejl

Når der sker en fejl under håndteringen af en klient eller i serveren generelt, er det vigtigt at administratoren kan finde ud af hvad der gik galt, så han kan forsøge at undgå at det sker igen. Der er flere måder hvor på serveren kan fortælle, at der er opstået en fejl. Den kan skrive ud på skærmen, gemme oplysningerne i en log eller sende en mail til administratoren.

Løsningen med at skrive til skærmen er ikke en god idé da man så ikke har oplysningerne hvis maskinen fryser helt, eller når man genstarter den. Det er også ofte sådan at servere ikke har en tilgængelig skærm eller at selve serveren ikke er let tilgængelig.

Løsningen med at gemme oplysningerne i en log er en bedre idé da man så har oplysningerne liggende på harddisken hvis maskinen skulle fryse. På den måde er det måske kun sidste fejlmeddelelse der ikke kommer med, hvis programmet går ned under skrivning i loggen eller andre steder. Med en log kan man også samle fejl, og andre informationer, over en længere periode og på den måde har man en række oplysninger, man kan udarbejde statistik over.

At sende en mail til administratoren i tilfælde af fatale fejl er også en god idé. Det kræver bare at det er muligt for serveren at sende mails.

3.5. MIME-type

»Content-Type«-headeren beskriver typen på den fil en server sender tilbage til en klient. Værdifeltet i headeren er en MIME-type¹ fordi det giver en fleksibel og åben typebestemmelse. En MIME-type er en typekonvention der er styret af *iana.org* og som står beskrevet i RFC 2045 [4] & RFC 2046 [5]. Hvis en server inkluderer en MIME-type blandt sine headers kan klienten få at vide hvad det er for en filtype, og hvordan den skal vises.

Det er ikke noget krav at man sender en MIME-type, og der er heller ikke noget krav om, at serveren skal kende til alle MIME-typer. Det er dog beskrevet i RFC 1945 at man bør sende en MIME-type, hvis man kender den. Hvis man ikke sender en MIME-type

¹Multipurpose Internet Mail Extensions

retur, vil klienten selv forsøge at finde ud af hvilken MIME-type filen, der er blevet returneret, har. Hvis klienten ikke kan bestemme filtypen, behandles den som om det var den MIME-type der hedder »application/octet-stream«.

For at afgøre hvilken MIME-type filen har, kan man undersøge om der er særlige kendetegn ved filen. Fx kan man se om filen har et bestemt efternavn, eller om filen indeholder et bestemt mønster der kan identificere typen.

3.6. HTTP Headers

En HTTP-header står beskrevet i RFC 1945 som en linje der kan stå i toppen af en HTTP-besked efter statuslinjen. Der findes flere typer af headers og der kan være indtil flere i beskeden. Man kan vælge at undlade dem i HTTP/1.0, men i HTTP/1.1 er der nogle enkelte headers der skal være med. En header har følgende opbygning:

```
felt-navn: (SP) [felt-værdi] CRLF
```

Grunden til at SP står i parentes er, at den ikke er nævnt i Backus Naur-formen af grammatikken, men det fremgår af teksten i RFC 1945, at den skal være med. En lexer vil i dette tilfælde skulle dele op i symboler ved »:« og » « og den ville kunne bruges til, at få skilt feltnavn og feltværdi ad. Det vil være svært at lave en parser til dette, da det er muligt for en person at definere sine egne headers. Det er dog tilladt at ignorere de headers man ikke genkender. Et andet problem, selv med kendte headers er, at værdierne kan være meget forskellige. Det vil derfor være svært at lave en generel parser da det er forskelligt hvad der må stå i værdifeltet afhængigt af, hvad der står som feltnavn. Feltet »Content-Length« må eksempelvis kun indeholde tal, mens »Content-Type« feltet må indeholde alle tegn undtagen specialtegn og kontroltegn. Derfor vil en parser skulle have særtilfælde for næsten alle headers, når der skulle laves en grammatik.

3.7. Struktur

Der er flere måder at opbygge et program på. Man kan vælge udelukkende at lave en masse funktioner, opdele i klasser, eller i en række komponenter. Hvis man kun laver funktioner kan et stort program virke uoverskueligt, og det kan derfor være svært at finde rundt i, når man skal finde fejl eller lignende. På det punkt kan det være lidt lettere med klasser, da man nogle gange har lettere ved at finde den klasse eller de klasser, der giver problemer. Opdeling i klasser gør det også nemmere hvis man senere hen vil skifte enkelte dele ud. Man kan så nøjes med at skifte en enkelt klasse ud, hvis bare man overholder interfacet fra header-filen. Man kan også se på programmet som en samling af komponenter. Dette er en måde, at arbejde med klasser på, hvor de hænger meget tæt sammen og det derfor ikke er umiddelbart muligt at skifte en enkelt klasse ud, uden at det får større indflydelse på de andre klasser der interagerer med den.

3.8. Statistik

Når der skal gemmes statistik, skal man først beslutte sig for, hvad der vil være relevant, at gemme. Mange af oplysningerne omkring en forespørgsel, fx hvornår serveren startes op og lukkes ned, samt hvor mange bytes der sendes og modtages kan alle være relevante i forbindelse med en webserver. Det kan være meget godt både at gemme oplysninger om forespørgsler der gik godt, og de forespørgsler der gik dårligt. De kan give en idé om det er serveren der er implementeret forkert, eller om klienten tastede forkert i forespørgslen. Statistikken for en forespørgsel kan eksempelvis indeholde metoden, hvornår forespørgslen blev behandlet, hvor den kom fra, hvilken path den havde og den statuskode, der blev returneret. Man kan også vælge en retning, hvor man kun gemmer nogle enkelte oplysninger, som for eksempel metoden og statuskoden. Det kan dog være svært at bruge de oplysninger til noget, bortset fra at se hvilke forespørgsler, serveren modtager.

3.9. CGI-scripts

Vores diskussion tager udgangspunkt i det tredje udkast til CGI/1.1-specifikationen [3]. Et script er, til forskel fra en statisk HTML-side, et program der kan udføres i real-time og dynamisk give information retur. Hvis man skal implementere CGI, skal man finde ud af hvordan man overfører data til scriptet. Man kan her lave et array af miljøvariabler, der kan sendes med til `execve()`-kaldet som argument. Når man skal sende anden data til scriptet kan det gøres via programmets standard input og standard output. Der er beskrevet tre niveauer af vigtighed i CGI/1.1-specifikationen, `MUST`, `SHOULD` og `MAY`. Det er vigtigt at en server har implementeret de punkter der står som `MUST` i specifikation, mens de andre punkter er nogle man burde få implementeret på sigt.

3.10. Leksikalsk analyse

Den leksikalske analyse handler om at få omdannet en række enkeltstående tegn, eller bytes, til en række af ord. Det kan fx være for at finde de »ord« der indgår i HTTP-forespørgselslinjen:

```
GET /web/side.html HTTP/1.0
```

Det kan også være for at finde de »ord« eller elementer der indgår i en path til ressourcen. I eksemplet ovenfor vil det så være en leksikalsk analyse af det andet ord i HTTP-forespørgselslinjen.

Den teoretiske fremgangsmåde er at bruge en tilstandsmaskine som beskrevet i [7].

Når man i et program har brug for at genkende ord er det en løsningsmulighed, at skrive programmet så det specifikt forstår den anvendelse man kommer ud for. En anden

mulighed er at programmere en fleksibel lexer der kan repræsentere en tilstandsmaskine og dermed benytte teorien om disse. Tredje mulighed kan være at benytte allerede eksisterende værktøjer til at fremstille koden til leksikalsk analyse, fx GNU's flex.

3.11. Syntaktisk analyse

Den syntaktiske analyse går ud på, at forstå en række af ord for at kontrollere at rækkefølgen overholder en grammatik, og kan endvidere gå ud på at opbygge en syntaksgraf.

Syntaksen kan være, og er oftest i forbindelse med RFC'er, angivet i Backus Naur-form.

Fordelene ved at anvende en generel parser er, som ved lexeren, at man ikke skal implementere detaljer og kontrolstrukturer omkring selve analysen, men kan nøjes med at angive grammatikken.

Generelle parsere er dog komplicerede at fremstille. Ofte vil man kunne have fordel af, at anvende et eksisterende værktøj som fx GNU's bison.

3.12. Load-balancing

Indledningsvist skal det konstateres at vi ikke har implementeret nogen form for load-balancing i projektet. Undervejs har vi dog gjort os en række overvejelser herom.

Ved loadbalancing har man flere webservere til at dele belastningen. Et afgørende kriterium er at man får spejlet indholdet af webserverne så de er i stand til, at servicere det samme indhold. I den forbindelse skal man være specielt opmærksom på hvordan dynamisk indhold (fx POST og PUT) håndteres og opdateres.

Der skal udvælges en række algoritmer efter hvilke belastningen fordeles mellem webserverne. En simpel metode kan være et *round robin*-princip hvor indkommende forespørgsler fordeles til webserverne efter tur. En anden metode kan bestå i at måle belastningen på webserverne og vælge den med mindst belastning.

Fordelen ved *round robin*-princippet er, at det ikke kræver kommunikation om belastning mellem webserverne og den enhed der foretager balanceringen. En ulempe kan være hvis forespørgslerne er meget uensartede i belastningshenseende, da kan belastningen ramme skævt på serverne. Fordelen ved at inddrage servernes belastning er et forsøg på at undgå skæv belastning, men det kræver en række overvejelser. En »belastning« kan måles på adskillige måder: nuværende hukommelsesforbrug eller CPU-belastning, en kombination/vægtning af disse, evt. som et gennemsnit over en periode for at udligne små udsving. Såfremt der optræder en vis forsinkelse i belastningsopgørelsen kan man også her komme ud for at belastningen rammer skævt på serverfarmen.

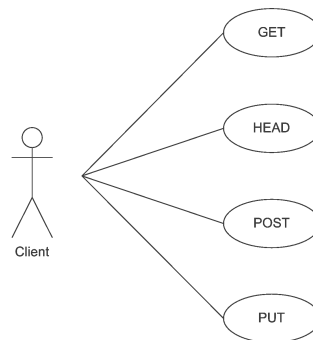
Selve fordelingen mellem serverne ud fra de ovenstående kriterier kan også implementeres på flere måder. Man kan forestille sig at man på TCP-niveau fordeler de indkommende forespørgsler på en NAT-lignende måde eller man kan implementere en proxy-server der leder forespørgslerne videre til den rette webserver. Andre metoder kan være at forbinde indkommende forbindelsers descriptor med en descriptor til den rette webserver vha. `socketpair()` eller blot lade loadbalanceren lede klienterne videre med et »302 Moved Temporarily« HTTP-svar (omend klienterne så vil kunne se forskel på serverne i serverfarmen).

I forbindelse med loadbalancering og dynamisk indhold skal man endvidere tage stilling til hvordan man håndterer eventuelle sessionsforløb hvor tilstanden af sessionen opbevares på serveren. Man kan lade tilstandene tilgå fra alle servere eller sørge for at forespørgsler der er en del af én session altid går til samme server i serverfarmen.

4. Design

Følgende diagrammer er det endelige resultat af systemanalysen og -designet efter flere iterationer.

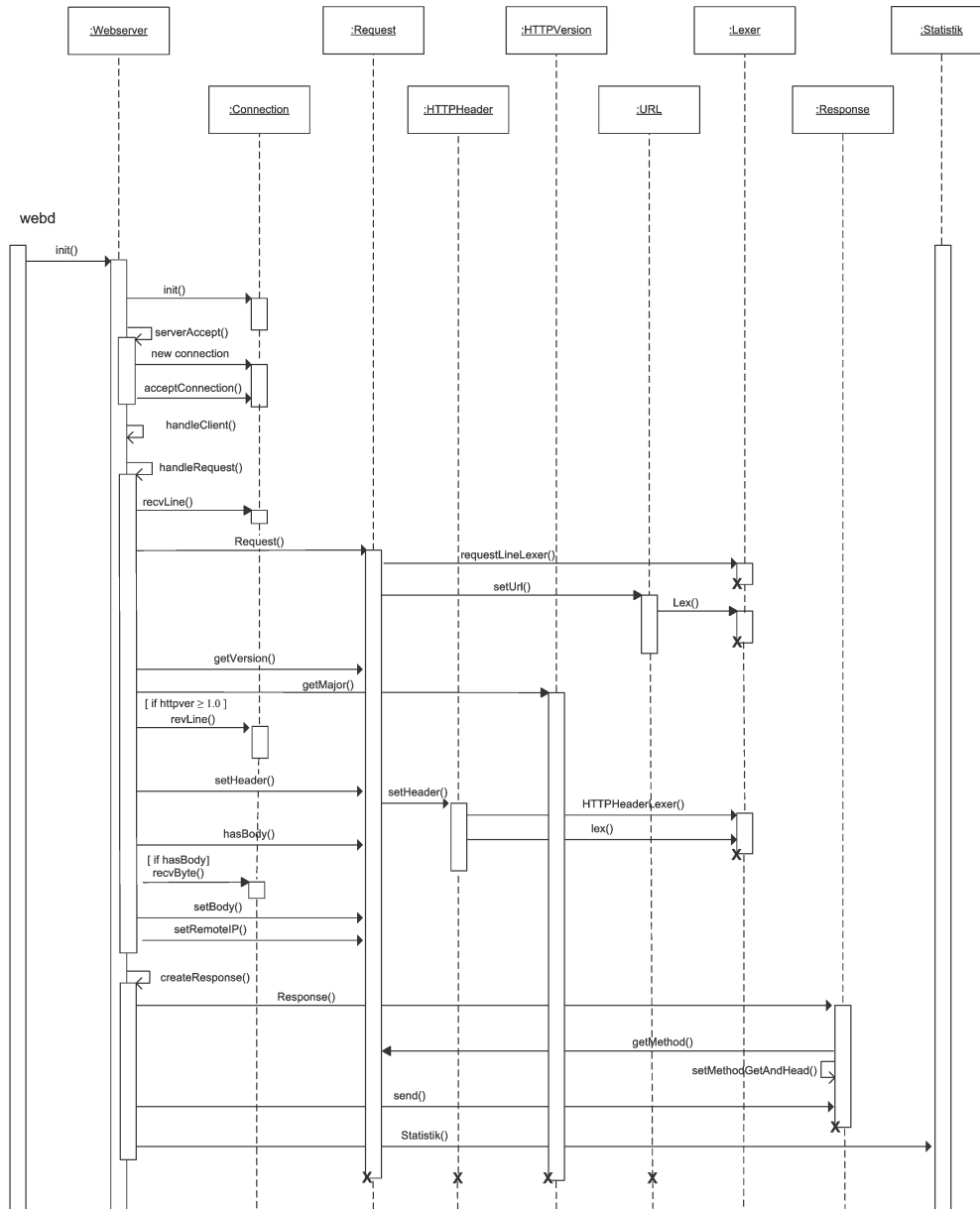
4.1. Brugsmønsterdiagram



Figur 1: Brugsmønsterdiagram

Diagrammet viser aktøren og de brugsmønstre der indgår i webserveren.

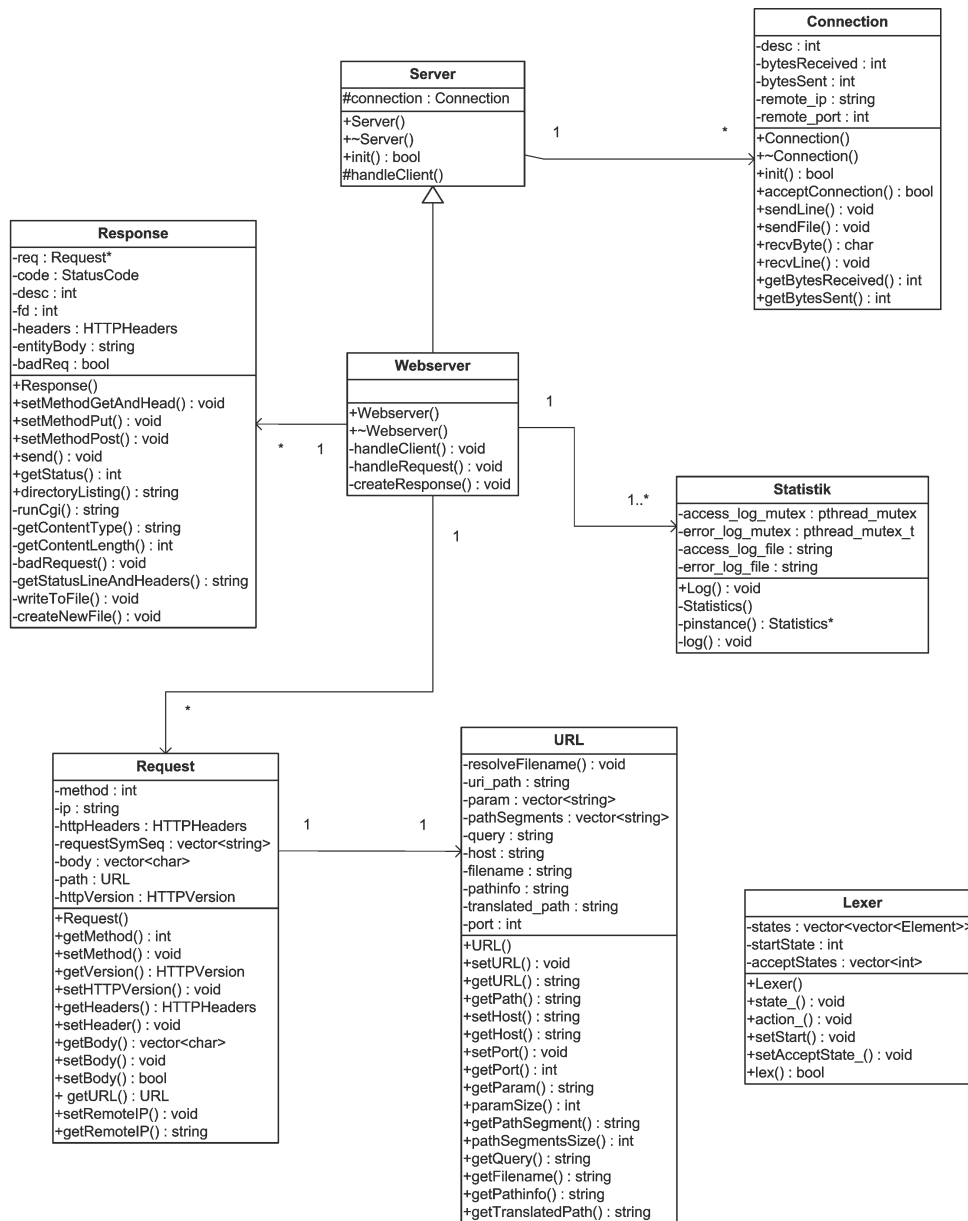
4.2. Sekvensdiagram



Figur 2: Sekvensdiagram

Sekvensdiagrammet viser forløbet for brugsmønstrene GET og HEAD, fra serveren startes, til en klient er serviceret og en statistik er tilføjet loggen. Sekvensdiagrammer for de øvrige brugsmønstre er tilsvarende.

4.3. Klassediagram



Figur 3: Klassediagram

Klassediagrammet viser de vigtigste klasser for webserveren. For overskuelighedens skyld viser vi ikke hjælpeklasserne, men viser i stedet samtlige metoder og attributter som er indeholdt i de viste klasser. Vi har fire forskellige lexere som alle arver fra lexer-klassen. For overskuelighedens skyld har vi dog undladt disse specialiseringer.

4.4. Programmeringsovervejelser

Overvejelserne omkring vores design tager sit udgangspunkt i semesterets »projekt 1.«

Grundlaget er klasserne *server* og *connection*.

Connection-klassen håndterer alt omkring opsætning af sockets og kommunikation over disse, mens *server*-klassen står for initialisering af serveren og *connection*-klassen. Serveren står også for at modtage nye connections/klienter og lade håndteringen af disse foregå i en tråd.

Server- og *connection*-klasserne er så generelle at de vil kunne håndtere en vilkårlig TCP/IP-baseret protokol (fx FTP eller SMTP).

Webserver-klassen er en specialisering af *server*-klassen der implementerer HTTP-protokollen. I forhold til *server*-klassen vil det sige håndteringen af klienter – hvilket nu kan antages at foregå i sin egen tråd.

Serviceringen af en enkelt klient er tænkt at foregå i tre skridt. Første skridt er at indsamle al information om klientens forespørgsel (path, HTTP-version, HTTP-headers, mm.) i en instans af klassen *request*. Dernæst konstrueres et objekt af klassen *response* der indeholder alle de oplysninger klienten efterspørger. Slutteligt sendes indholdet af *response*-objektet og statistikken ajourføres.

5. Valg

5.1. URL

Vi vælger kun at koncentrere os om det, der refereres til som *abs_path* i RFC 1945. Det gør vi fordi det er den del af en URL, der er med i en HTTP-forespørgselslinje. Vi mener at hvis man kan lexe og parse en *abs_path*, så vil det ikke være noget problem at lexe og parse en hel URL. Forskellen på de to er at der i den hele URL står »http://<host>:<port>« foran, mens det er undladt i *abs_path*.

Vi har også valgt at hvis vi modtager en path til en mappe, som ikke ender på en »/«, sender vi en redirection tilbage med den samme path, hvor der er tilføjet en »/« til sidst. Det gør vi for at sikre os at klienten er klar over hvilken mappe, den skal finde de eventuelle links i. Problemet kan være, at klienten tror den står et niveau for højt oppe i mappestrukturen, og det medfører at den kan sammensætte den næste path forkert.

5.2. Parallelisme

Vi har valgt at arbejde med tråde da de er hurtige at kreere og fordi de optager mindst plads i hukommelsen. Vi opretter en ny tråd hver gang vi accepterer en ny forbindelse. Denne tråd er »detached« da vi ikke behøver at fortælle serveren at den er afsluttet. I forbindelse med at vi implementerede CGI flyttede vi også selve serveren over i en tråd. Det gjorde vi fordi når man kalder funktionen `execve()` fra en tråd så er det kun den kaldende tråd, der er aktiv i den nye proces, mens de andre bliver inaktive. Ved også at lægge serveren i en tråd undgår vi at have aktive servere i begge processer. Det ville være et problem hvis serveren i den proces der skulle udskiftes, nåede at acceptere en forbindelse, da denne ville forsvinde lige så snart `execve()` blev kaldt.

5.3. Fejlhåndtering ved klientforespørgsler

Hvis klienten skulle prøve, at gå længere op i mappestrukturen end roden, har vi af sikkerhedsmæssige årsager valgt at sende indholdet af roden tilbage til klienten. Således undgår vi at klienten får adgang til filer som ikke skal være offentligt tilgængelige.

Vi har valgt at oprette statuskoder som følger standarden beskrevet i RFC 1945. Vi har som udgangspunkt sat statuskoden til »500 Internal Server Error«. Opstår der fejl hvor vi med sikkerhed kan placere årsagen, sætter vi så statuskoden til den mest oplagte statuskode og årsag. Dog skal det nævnes, at i fejlbeskrivelsen som returneres til klienten skriver vi kun teksten »bla....bla....bla...«, fordi vi mener, at det vigtige i opgaven på dette punkt må være, at kunne sende et svar til klienten, og desuden at kunne sende den rigtige statuskode. Kan vi dette, vil det ikke være noget problem at indsætte en meningsfyldt tekst på et senere tidspunkt.

Hvis hele forespørgslen fra klienten indeholder lovlige karakterer og er korrekt opbygget, men af andre årsager ikke kan gennemføres sender vi statuskode »400 Bad Request« tilbage som svar. Dette svar vil være oplagt hvis klienten har lavet en stavefejl i URL'en. Hvis derimod metoden der blev brugt ikke eksisterer i vores server, som for eksempel `DELETE` eller lignende, burde vi sende statuskode »501 Not Implemented« tilbage som svar. Vi har dog valgt at undlade at gå ned i denne detaljeringsgrad.

5.4. Fejlhåndtering ved serverfejl

Vi har bestemt at vores server skal skrive til en log hvis der internt går noget galt, den hedder *internal.log*. Loggen kommer i brug hvis vi for eksempel ikke kan lave en socket i starten eller lignende. Vi har valgt at bruge logfilen som løsning, da det gør det nemmere at finde ud af, hvor en fejl opstod. Samtidig har man en form for historik over de fejl der er opstået undervejs i udviklingsprocessen. Vi har gjort det muligt, via Makefilen, at man selv kan bestemme hvad man vil kalde logfilen og hvor den skal ligge. Som udgangspunkt hvis man ikke ændrer noget, kommer den til at ligge

i samme bibliotek som programmet. Herunder ses et par eksempler fra den interne log.

```
Fri, 10 Dec 2004 09:33:16 GMT: Server started.  
Fri, 10 Dec 2004 09:33:16 GMT: Spawning server at port 10010  
Fri, 10 Dec 2004 11:51:45 GMT: Could not execve() CGI-script: ./document_root/uu  
Mon, 13 Dec 2004 11:41:26 GMT: Fejl under accept
```

5.5. MIME-type

Vi har valgt at have et lille udsnit af alle MIME-typer med i vores program. Vi har valgt kun at have et udsnit med, fordi vi mener, at hvis vi kan håndtere nogle enkelte MIME-typer, så vil det ikke være noget problem at tilføje flere senere. Vi har valgt at have en liste med udvalgte efternavne og de tilhørende MIME-typer. Når vi skal sende en fil til klienten, kigger vi i denne liste og hvis filens efternavn findes, sender vi filens MIME-type tilbage. I det tilfælde hvor vi ikke kan finde MIME-typen på filen vælger vi at sende den MIME-type der hedder »application/octet-stream« tilbage. Det gør vi fordi det er den en klient ifølge RFC 1945 bør vælge hvis den ikke kan bestemme filtypen.

5.6. HTTP Headers

Som ved MIME-typer har vi valgt kun at koncentrere os om et lille udsnit af headers, da vi også her mener at hvis vi kan læse, opbevare og håndtere nogle enkelte headers, kan man også have flere. Når vi modtager headers har vi valgt at gemme dem i en liste hvor feltnavn og feltværdi ligger hver for sig. På den måde kan vi søge i listen efter et bestemt feltnavn og hvis det findes, kan vi analysere feltværdien og tage stilling til hvad programmet skal gøre ud fra værdien. Vi har valgt kun at tage stilling til Host- og Content-Length-headeren. Når vi sender headers retur er det Content-Length, Content-Type og Date. Hvis man sætter en værdi i Makefilen sender vi også Cache-Control: no-cache med. Den findes ikke i HTTP/1.0 men derimod i HTTP/1.1. Vi har valgt at tage den med da det i forbindelse med test er rart, hvis klienten ikke gemmer oplysninger lokalt.

5.7. Struktur

Vores struktur er objektorienteret. Vi har valgt at arbejde med klasser da det er med til at give et større overblik over programmet og det er nemmere at finde fejl når man kan isolere problemerne. Med klasser er det også muligt at arbejde med arv, hvilket er hensigtsmæssigt i forbindelse med vores generelle lexer-klasse. Man kan så lave specialiseringer af klassen i stedet at lave mange enkeltstående lexer-klasser. Vi kan også bruge arv i forbindelse med vores server. Vi har valgt at lave en generel server der

blot opretter en forbindelse og laver nye tråde. Således kan man lave en specialisering af den og tilføje de funktioner der skal til for eksempelvis at lave en webserver.

5.8. Statistik

Vi har valgt at have to forskellige logfiler til vores statistik: *accesslog.csv* og *errorlog.csv*. Hvis en forespørgsel er gået godt og klienten har fået et svar med statuskoden 2xx eller 3xx bliver der tilføjet en post i *accesslog.csv*. Hvis forespørgslen ikke gik godt og klienten har fået et svar med statuskoden 4xx eller 5xx bliver der tilføjet en post i *errorlog.csv*. Vi gemmer oplysningerne i kommaseparerede filer så de let kan åbnes i eksempelvis regnearksprogrammer. Herunder kan ses et par eksempler på linjer fra vores to logfiler. Posterne i filen er (i denne rækkefølge): statuskode, URL, referer-URL (hvis en sådan findes), redirect-URL (hvis det er aktuelt), klient-IP og port, antal bytes modtaget, antal bytes sendt, og dato/klokkeslæt.

accesslog.csv

```
200,"http://djengis:10010/favicon.ico","",172.16.242.175:2708,330,106,  
"Tue, 07 Dec 2004 09:31:16 GMT"
```

errorlog.csv:

```
404,"http://djengis:10010/answers.html","",127.0.0.1:32818,33,898,  
"Tue, 07 Dec 2004 09:33:24 GMT"
```

5.9. CGI

Vores implementation af CGI-håndtering har taget udgangspunkt i CGI/1.1-specifikationen. Implementationen lever ikke op til samtlige krav i specifikationen, men vi har udvalgt en delmængde vi har fundet interessant i forbindelse med et studieprojekt.

De dele af CGI-understøttelse vi har fundet interessante i projektet er

1. kaldet af et eksternt program (CGI-scriptet),
2. kommunikationen mellem server og script, og
3. kommunikationen mellem klient og script.

Konkret har det bl.a. mundet ud i, at vi kun understøtter Non-Parsed-Header CGI-scripts – dvs. scripts der selv sender HTTP-headers retur til klienten.

Vi har benyttet systemkaldet `execve()` til at få udført et andet program. Dette kald giver gode muligheder for at overføre oplysninger fra serveren til scriptet (jf. punkt 2) gennem kommandolinjeargumenter til programmet og gennem miljøvariable.

Kaldet af `execve()` resulterer dog i, at det kaldte program overtager processen og udskifter program, programtæller, stak, mm. med sit eget. For at omgå dette så webserveren kan fortsætte med at servicere andre klienter laver vi forinden et kald af `fork()` og kalder `execve()` fra barneprocessen.

Kaldet af `fork()` giver anledning til et par overvejelser angående mulige bivirkninger af kaldet. Første overvejelse gik på om man, når man har dubleret den oprindelige proces har to versioner af serveren kørende og hvilke dele af serveren der er kørende. Jf. [8] er det kun den kaldende tråd der fortsætter som aktiv tråd i barneprocessen. Herved undgår vi at øvrige tråde fortsætter med at servicere klienter i to processer og undgår derved den bivirkning at klienterne risikerer at få (delvist) dobbelt svar tilbage på deres forespørgsler.

Yderligere en potentiel risiko ved kaldet af `fork()` er at de dele der ligger uden for trådene bliver dubleret og fortsætter i både forældreprocessen og barneprocessen. Under implementationen af CGI-håndteringen lå den del af webserveren der lytter efter nye klienter uden for trådene og vi overvejede derfor om vi risikerede at begge processer ville tage mod nye klienter. Eftersom barneprocessen kort efter `fork()`-kaldet ville skifte processen ud med CGI-scriptet, ville eventuelle nye klienter barneprocessen var begyndt at servicere tabe forbindelsen når `execve()` blev kaldt. Vi omgik dette ved at flytte den del af serveren der lytter efter nye klienter ind i sin egen tråd.

Overførsler af oplysninger fra serveren (oplysninger om path, klientens IP-nummer, mm.) til scriptet (punkt 2) har vi implementeret ved at overføre dem gennem miljøvariable og overførslen af data fra klienten til scriptet (punkt 3) sker ved at serveren skriver dem til scriptets *standard in*.

Til dette formål har vi benyttet systemkaldene `pipe()` og `dup()` til at forbinde en file-descriptor i serveren til scriptets *standard in* file-descriptor.

Kommunikationen den anden vej – fra script til klient – har vi valgt at lade foregå direkte og uden om serveren da det gav os erfaring med at koble descriptoren til klienten sammen med scriptets *standard out* file-descriptor ved hjælp af systemkaldet `dup2()`.

Vi har endvidere valgt at implementere en grundig opfølgning på, hvordan udførelsen af CGI-scriptet forløber ved at kontrollere scriptets exit-status mm.

5.10. Lexer

I dette projekt har vi valgt at implementere en lexer der er en generel tilstandsmaskine. I en webserver optræder der mange rækker af tegn, der på forskellig vis skal inddeles i symboler – eksempelvis HTTP-forespørgselslinjer, URL'er, HTTP-headers, indhold af Date-HTTP-header, indhold af Host-HTTP-header, etc.

Implementeringen af en generel lexer har betydet at vi kun har skulle implementere den basale del af symbolgenkendelsen én gang, mens vi til selve anvendelsen har kunnet nøjes med at angive tilstandsovergange, aktioner, samt start- og accepttilstande.

Denne fremgangsmåde har gjort det nemt og hurtigt at lave en lexer-til-lejligheden når vi er stødt på en anvendelsesmulighed heraf.

5.11. Parser

De grammatikker vi har mødt i forbindelse med implementeringen af webserveren, har alle været forholdsvis simple og har bestået af få elementer der følger hinanden i en fastsat rækkefølge.

De simpleste (som fx indholdet af en Host-HTTP-header eller en generel HTTP-header) består af fx tre ord hvor grammatikken specificerer at det midterste skal være et »:«.

På grund af de simple grammatikker og kompleksiteten i at programmere en (generel) parser selv, har vi valgt en »høkerløsning.« Vores lexere tager lidt forskud på parsningen og accepterer i de fleste tilfælde kun rækker af tegn der er gyldige i grammatikkerne. Eneste undtagelse er lexningen af en HTTP-forespørgselslinje, hvor det overlades til den syntaktiske analyse at konstatere om første ord i symbolsekvensen er en gyldig HTTP-metode (GET, HEAD, m.fl.) og ved fulde (ikke-HTTP/0.9) HTTP-forespørgselslinjer om tredje ord er »HTTP«.

Efter megen overvejelse og indledende design valgte vi den simple løsning. I betragtning af de simple grammatiker finder vi ikke, at de omkostninger der er forbundet med udviklingen af en generel parser står mål med fordelene ved dette.

5.12. Sende og modtage over nettet

I forbindelse med kommunikation over nettet benytter vi os af Linux' socket-koncept. Vi har fundet god inspiration og forklaring til konceptet i hhv. [6] og [8].

Brugen har afstedkommet en række valg mellem metoder at benytte sockets på.

Opsætningen af en lyttende port ved hjælp af systemkaldene `listen()`, `bind()` og `accept()` har ikke givet anledning til større problemer.

Når serveren ønsker at læse de data en klient sender kan man benytte systemkaldet `recv()`. Dette kald er dog blokerende i den forstand, at hvis der ikke er ankommet nye data fra klienten (og klienten ikke har lukket forbindelsen) vil kaldet af `recv()` vente på at der kommer data før end det returnerer. Med systemkaldet `select()` har man mulighed, for at vælge blandt de descriptorer der har data klar til læsning. Man kan således nøjes med at bruge `recv()` på descriptorer hvor man ved der er data til rådighed. Eftersom vi har valgt en trådet løsning hvor hver klient serviceres i sin egen

tråd er det dog ikke noget problem at `recv()` blokerer, vi har derfor valgt ikke at benytte `select()`.

Til at sende data bruger vi systemkaldet `send()`. Vi forsøger at sende data i pakker af 1024 bytes der er den maksimale størrelse af TCP-pakker. Det kræver et check af `send()`'s returværdi for at se hvor mange bytes det reelt lykkedes at sende og om nødvendigt forsøge at gensende dataene.

I begyndelsen af vores implementation brugte vi en pakkestørrelse på én byte da man således ikke skal implementere større kontrol med om data skal gensesendes. I den forbindelse fik vi en del fejl hvor forbindelsen til klienten bliver lukket før samtlige data er sendt til klienten på trods af, at alle data er sendt fra serveren. Fejlen kan vi genskabe med pakkestørrelser op til ca. 70 bytes på skolens Red Hat-systemer, men har ikke kunnet genskabe den på Debian Unstable eller Mac OS X.

Ved modtagelsen af data fra klienten anvender vi `recv()`. Implementationen læser kun én byte ad gangen. Teknikker der minder om, at sende mere end én byte af gangen kan finde tilsvarende anvendelse her, så vi anser det ikke for nogen større opgave at ændre implementationen.

Vi har også måtte træffe et valg angående de situationer hvor forbindelsen til klienten lukker i »utide.« Serverprocessen modtager et `SIGPIPE`-signal når forbindelsen lukkes. Da det ikke er muligt at fange dette signal i den rette tråd har vi installeret en signalhandler der ignorerer signalet og lader det være op til de tråde der sender og modtager data, at udvise den nødvendige forsigtighed.

Systemkaldene `send()` og `recv()` returnerer begge `-1` eller `0` hvis forbindelsen ikke længere eksisterer og vi vælger derfor at kontrollere for dette og kaste en exception i så fald.

I en webserver får man brug for at sende indholdet af filer fra serverens filsystem til klienterne, typisk statiske HTML-sider og billeder. Vi har valgt at benytte os af Linux-kernens `sendfile()`-systemkald. Kaldet kan skrive fra en descriptor til en anden. Descriptoren man læser fra skal være en file descriptor, mens den anden fx kan være en socket.

Fordelene ved kaldet er, udover at det i forbindelse med implementation er nemt at sende indholdet af filer til klienterne, at hele operationen foregår i *kernel space*. Derfor skal der ikke kopieres data frem og tilbage mellem *user space* og *kernel space*.

Systemkaldet er en speciel implementation der kun findes i Linux-kernen. Vi har derfor skrevet en simpel og naiv implementation af funktionen (hvor data flyttes via *user space*) for at kunne teste webserveren på Mac OS X.

6. Metode og værktøj

Vi har gennem hele udviklingsforløbet mødtes hver dag på skolen i et grupperum, og arbejdet på projektet. Hver dag er vi startet med et statusmøde og er også afsluttet med et. Dette har bevirket at alle i gruppen har haft kendskab til om projektet er skredet frem efter planen, og alle har været en del af udviklingsforløbet.

Programkoden har vi arbejdet på både individuelt og som parprogrammering. De dele vi hver især har programmeret har vi løbende testet og diskuteret med hinanden, for at sikre at koden virker optimalt i en større sammenhæng.

Som udviklingsværktøj har vi brugt Emacs og UltraEdit, samt GNU's C++ compiler g++, hvilket har fungeret upåklageligt. Vi har oversat programmet både på Red Hat 9.0, Debian Unstable og Mac OS X. Vi oplevede dog ved oversættelse på Mac OS X, at vi måtte lægge specielle Mac-definitioner ind i toppen af programkoden for at denne kunne afvikles. Dette er dog forløbet uden større problemer.

Til test af forespørgsler og svar, har vi primært brugt telnet og Internet Explorer 6.0. Firefox 1.0 er også brugt til test. Telnet og Firefox 1.0 har virket uden problemer, hvorimod Internet Explorer 6.0 på nogle områder har været mere tvivlsom (se afsnit 9 på side 26).

Til samlingssted for vores kode har vi brugt versionsstyringssystemet Subversion. Således har alle i gruppen hele tiden haft adgang til ny-uploadede filer, tidligere versioner og logfiler. Dette udviklingsværktøj har vi brugt på tidligere projekter og har erfaringer med at det fungerer fint, hvis man konsekvent opdaterer inden man uploader sine redigerede filer.

Til rapportskrivning har vi brugt L^AT_EX. Fordelen ved dette system, i modsætning til traditionel tekstbehandling, har været, at et dokument nemt kan deles ud i flere filer, samt at disse filer integrerer godt med Subversion. Systemet holder desuden nemt referencer, henvisninger, indholdsfortegnelser m.m. opdateret.

7. Forbedringer

I en version 2 af webserveren kunne vi godt tænke os at udvide og tilføje nogle forbedringer til vores program.

Webserveren burde udbygges til at følge standarden for HTTP/1.0, med hensyn til at medtage flere HTTP-headers og MIME-typer, hvor vi nu har valgt kun at tage nogle få udvalgte med.

Desuden kunne vi også ønske os en mere udbygget fejlhåndtering med brug af exceptions.

Klassen *response* burde splittes op i to klasser, således at alt der har med filhåndtering at gøre, bliver lagt i sin egen klasse.

Vi vil gerne have implementeret at serveren kan lytte på mere end én port, og kunne håndtere flere hosts.

Eftersom vi kun implementerer Non-Parsed Headers CGI-scripts vil vi gerne implementere Parsed Headers CGI-scripts. De nødvendige teknikker (kommunikation mellem processer via pipes og lexning/parsning af headers) har vi på nuværende tidspunkt erfaring med.

8. Programdokumentation til webserver

Dette afsnit kan læses som et selvstændigt dokument der giver et overordnet indblik i webserverens opbygning.

Webserveren er udviklet til at skulle køre på en GNU/Linux-server som kører med Red Hat 9.0. Programmet er objektorienteret, og udviklet i programmeringssproget C++. Webserveren håndterer HTTP-forespørgslerne GET, HEAD, POST og PUT efter standarden beskrevet i RFC 1945, og håndterer en URL som beskrevet i RFC 1738.

For at starte webserveren skal programmet startes med et portnummer som argument:

```
./webd 80
```

Eksisterer filen `webd` ikke, skal programmet vha. `make` oversættes, fx med GNU's C++ compiler `g++`.

Kaldet af `make` danner programfilen `webd`. Det er derfor vigtigt at opdatere `Makefile` hvis der sker tilføjelser af nye filer til webserveren, eller hvis filerne flyttes til andre mapper. Hvis det er nødvendigt at ændre i programfilerne, skal `make` udføres så `webd` opdateres.

Vigtigste klasser i programmet er *server.h*, *webserver.h*, *connection.h*, *request.h* og *response.h*, hvor funktionerne til at oprette forbindelse til serveren og funktionerne til at servicere klienterne ligger.

Server

init En port er initialiseret og klar til at servicere klienter via et kald til init i connection.

Webserver

handleClient Håndterer en forespørgsel fra klienten og sender et svar retur.

Connection

init Skaber forbindelsen til socket, binder socket til en port og lytter på porten.

Request

Request Checker om forespørgslen fra klienten syntaksmæssigt kan godkendes.

Response

Response Udvælger efter GET, HEAD, POST og PUT hvordan svaret til klienten skal udformes.

send Sender svar tilbage til klienten.

Webserveren skriver til 3 logfiler, som alle ligger i mappen log_files.

accesslog.csv Opbevarer informationer om hvorvidt en skrivning til eller læsning fra en fil er gået godt eller ej. Statuskoderne 1xx, 2xx og 3xx i RFC 1945.

errorlog.csv Opbevarer informationer om hvorvidt en fejl er opstået i forbindelse med forespørgsler eller serverfejl. Statuskoder 4xx og 5xx i RFC 1945.

internal.log Opbevarer informationer om interne serveroplysninger.

9. Test

Når programmet er nået til et punkt hvor det begynder at se færdigt ud skal vi til at teste det i sin helhed. Vi har flere forskellige værktøjer vi kan bruge til at teste med.

Vi har vores browsere Firefox og Internet Explorer, men udover dem vil vi også bruge telnet da det ikke umiddelbart er muligt at teste metoderne PUT og HEAD i browserne. Telnet er også et godt værktøj til at teste de andre metoder med, da vi har fuld kontrol over hvad vi sender til serveren. Det var også telnet vi brugte i starten når vi skulle sikre os at der var forbindelse til vores server.

Vores ambition er at få testet alle de metoder og udvidelser vi har implementeret, samt at forsøge at stress-teste serveren. Med stress-test mener vi, at vi vil forsøge at sende mange forespørgsler parallelt for at se om serveren kan håndtere dette.

Vi vil herunder gå nærmere i detaljer omkring test af de forskellige metoder og udvidelser. Vi vil også opstille skemaer hvor vi viser, hvad vi vil teste og hvad vi forventer at resultaterne bliver (se bilag B). Resultaterne kan beskrives i afsnit 9.6.

Hvor det er muligt at teste metoderne med browsere vil begge browsere blive brugt. Derudover vil alle test blive kørt via telnet. Udover denne række test har vi undervejs i udviklingen løbende testet større og mindre stykker af koden.

Udover test af de specifikke metoder vil vi også teste et par ting der er generelle for serveren. Vi vil teste hvad der sker hvis man sender en tom forespørgsel afsted, altså hvis man bare sender »CRLF«. Vi vil også teste hvad der sker hvis man sender en forespørgsel med en metode vi ikke har implementeret, samt andre forespørgsler som er forkerte i forhold til RFC 1945.

9.1. GET

For at teste vores GET-metode vil vi forsøge at hente forskellige filer og mapper på vores server. Her vil der være nogle yderområder der skal testes. Vi skal se om vi svarer rigtigt tilbage hvis vi beder om ting der ikke findes, eller om vi får en mappestruktur tilbage hvis vi beder om en mappe. Vi skal også teste om det er muligt at gå længere op i mappestrukturen end roden af vores server ved at skrive »../« i path'en, således:

```
GET ../ HTTP/1.0
```

Det skulle gerne vise sig at vi har sikret os imod at det vil kunne ske. Desuden skal vi også teste at vi sender et simpelt svar, hvis vi modtager en simpel forespørgsel. En stress-test på GET ville være, at man lavede en masse parallelle forespørgsler på filer og mapper. I de test hvor vi bruger telnet vil vi bruge følgende headers: Accept-Language og Host.

9.2. HEAD

HEAD skal testes på samme måde som GET, dog med den undtagelse at vi ikke skal have nogen entitybody retur som svar, men kun statuslinjen og de eventuelle headers der måtte være. En stress-test af HEAD vil foregå på en lignende måde som GET. I de test hvor vi bruger telnet vil vi bruge følgende headers: Accept-Language og Host.

9.3. PUT

En test af PUT vil foregå ved at vi forsøger at uploade filer der ikke findes, filer der findes, og filer der findes men som serveren ikke har skriverettigheder til. For at teste det sidste vil vi lægge en fil på serveren der hedder »privat.txt«, og gøre den skrivebeskyttet.

Her vil en stress-test betyde at vi gør det samme, bare parallelt. I de test hvor vi bruger telnet vil vi bruge følgende headers: Content-Type, Content-Length og Host.

9.4. POST

Når vi skal teste POST gør vi det ved at forsøge at sende nogle oplysninger til et script i entitybody'en på forespørgslen. Vi vil også prøve at sende en forespørgsel med oplysninger til en mappe, eller til en fil der ikke er eksekverbar.

Her vil en stress-test også betyde at vi gør det samme, parallelt. I de test hvor vi bruger telnet vil vi bruge følgende headers: Content-Type, Content-Length og Host.

9.5. CGI

Vi kan her teste om vi får det samme resultat uanset om vi sender en forespørgsel til et CGI-script som en GET med oplysningerne i query-feltet, eller om man sender forespørgslen som en POST med oplysningerne i entitybody'en. I de test hvor vi bruger telnet vil vi bruge de samme headers som vi brugte i henholdsvis GET og POST.

9.6. Testresultater

Vores test forløb godt og i næsten alle tilfælde fik vi det forventede svar retur. De betyder at de simple forespørgsler returnerer entitybody'en, mens de fulde forespørgsler returnerer statuslinjen, headers og entitybody'en.

Metoderne GET, HEAD, og POST forløb uden problemer og de forespørgsler til metoden GET der kunne testes i Explorer og Firefox, virkede også som de skulle. Metoden PUT

gav nogle fejl da vi forsøgte at skrive til en skrivebeskyttet fil. Vi fandt og rettede hurtigt fejlen og de efterfølgende test virkede som forventet.

CGI-scripts gik også godt. Det forløb som forventet med telnet og Firefox, både med GET og POST som metode. Internet Explorer virkede fint da vi testede med metoden GET, men den ville ikke vise resultaterne af vores scripts når metoden var POST. Vi er ikke klar over problemet ligger i de scripts vi bruger, eller om det er Internet Explorer der opfører sig underligt. Vi hælder mest det sidste.

Vi har stress-testet vores server med metoderne GET og HEAD, det forløb uden de store vanskeligheder og serveren håndterede forespørgslerne korrekt. Vi forsøgte også at stress-teste med PUT, men det gjorde at djengis-serveren gik ned. Da det tog omkring 7-8 timer før den var oppe at køre igen, har vi af hensyn til de andre grupper valgt ikke at forsøge os med den eller POST igen.

10. Konklusion

Udviklingen på tredje semester har været en lærerig proces, som har givet os et godt indblik i hvordan en webserver fungerer, både omkring netværk, omkring programmering, og om at bruge de standarder som er foreskrevet.

Vi har nået de mål vi satte os ved projektets start, altså løst den konkrete opgave, samt implementeret to udvidelser, beskrevet den sidste og vi har en kørende webserver. Projektet har fulgt den tidsplan der blev lagt i september og vi har ikke været i tidsmæssige problemer.

Vi mener også at have opnået de færdigheder der kræves af os, indenfor emnerne Datamatiske Systemer og Systemprogrammering. Vi er i stand til at håndtere kommunikation via sockets, arbejde med shared memory, signalhåndtering, tråde, udføre syntaktisk- og semantisk analyse mm.

Vi mener, at vi i det store hele har truffet de rigtige valg, med hensyn til designet og programmeringen af webserveren.

Gruppesarbejdet har fungeret godt. Vi har haft mange gode og frugtbare diskussioner, og alles mening i gruppen er blevet hørt og vurderet. Vi har også haft stor glæde af at kunne udveksle erfaringer med de andre grupper fra klassen. Vi har været i stand til at få uddannelse og privatliv til at gå op i en højere enhed, som vi fra tidligere projekter ved kan betyde meget for at opnå konstant fremdrift og et godt resultat.

A. Tilstandsmaskiner til leksikalsk analyse

Efterfølgende diagrammer er tilstandsmaskiner til lexeren, med dertil hørende tilstandstabeller og aktionstabeller.

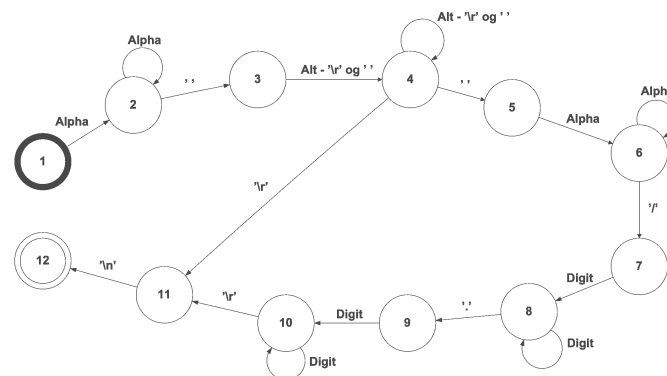
Ens for alle gælder at fejltilstanden er tilstand 0 og starttilstanden er tilstand 1.

Fork.	Aktion	I kildekoden
A	Det læste tegn tilføjes bufferen	ADD_TO_BUFFER
B	Indholdet af bufferen tilføjes symbolsekvensen	BUFFER_TO_SEQ
E	Indholdet af bufferen slettes	EMPTY_BUFFER

Tabel 1: Aktioner

I aktionstabellerne anvender vi kombinationer af aktionerne i tabel 1. For hvert symbol kan der udføres nul eller flere af aktionerne. Aktionerne udføres i den rækkefølge de står angivet i de følgende tabeller.

A.1. Lexning af HTTP-request line



Figur 4: Tilstandsmaskine til en HTTP-forespørgselslinje.

HTTP-forespørgselslinje-lexeren opdeler fx nedenstående tegn i de understregede symboler

GET /tekst.txt;params?a=3&b=5.7 HTTP / 1 . 0 \r \n

A.1.1. Tilstandstabel

Symbol/Tilstand	ALPHA	SP	CR	'/'	DIGIT	LF	'.'	Andet
1	2	0	0	0	0	0	0	0
2	2	3	0	0	0	0	0	0
3	4	0	0	4	4	4	4	4
4	4	5	11	4	4	4	4	4
5	6	0	0	0	0	0	0	0
6	6	0	0	7	0	0	0	0
7	0	0	0	0	8	0	0	0
8	0	0	0	0	8	0	9	0
9	0	0	0	0	10	0	0	0
10	0	0	11	0	10	0	0	0
11	0	0	0	0	0	12	0	0

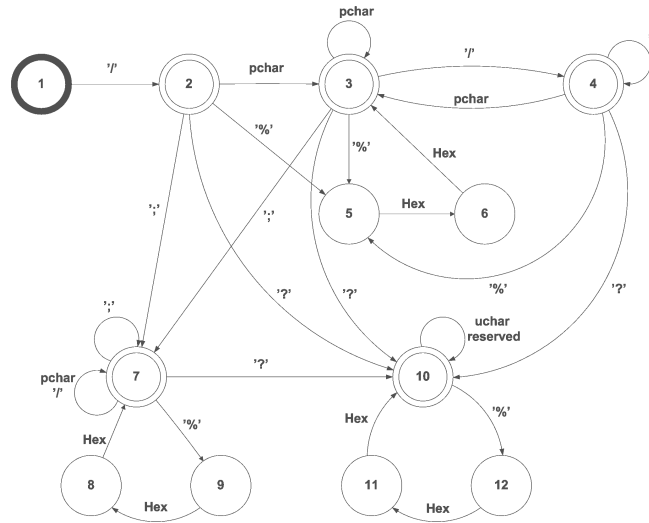
Tabel 2: Tilstandstabel for HTTP-forespørgselslinje-lexeren. Accepttilstanden er tilstand 12.

A.1.2. Aktionstabel

Symbol/ Tilstand	ALPHA	SP	CR	'/'	DIGIT	LF	'.'	Andet
1	A	-	-	-	-	-	-	-
2	A	BEABE	-	-	-	-	-	-
3	A	E	E	A	A	A	A	A
4	A	BEABE	BEA	A	A	A	A	A
5	A	-	-	-	-	-	-	-
6	A	-	-	BEABE	-	-	-	-
7	-	-	-	-	A	-	-	-
8	-	-	-	-	A	-	BEABE	-
9	-	-	-	-	A	-	-	-
10	-	-	BEA	-	A	-	-	-
11	-	-	-	-	-	ABE	-	-

Tabel 3: Aktionstabel for HTTP-forespørgselslinje-lexeren.

A.2. Lexning af absolut path i http-forespørgselslinjen



Figur 5: Tilstandsmaskine til absolut path.

Absolut path-lexeren opdeler fx nedenstående tegn i de understregede symboler

/ bibliotek / tekst.txt ; params ? a=3&b=5.7

A.2.1. Tilstandstabel

Symbol/Tilstand	'/'	not_pchar	'%'	';'	'?'	HEX	reserved_	Andet
1	2	0	0	0	0	0	0	0
2	3	0	5	7	10	3	3	3
3	4	0	5	7	10	3	3	3
4	4	0	5	3	10	3	3	3
5	0	0	0	0	0	6	0	0
6	0	0	0	0	0	3	0	0
7	7	0	9	7	10	7	7	7
8	0	0	0	0	0	7	0	0
9	0	0	0	0	0	8	0	0
10	10	0	12	10	10	10	10	10
11	0	0	0	0	0	10	0	0
12	0	0	0	0	0	11	0	0

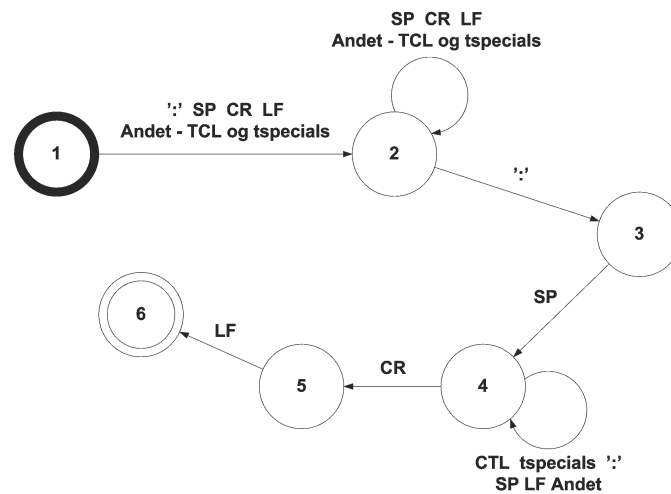
Tabel 4: Tilstandstabel for absolut path-lexeren. Accepttilstande er tilstandene 2, 3, 4, 7 og 10.

A.2.2. Aktionstabel

Symbol/ Tilstand	'/'	not_pchar	'%'	','	'?'	HEX	reserved_	Andet
1	ABE	-	-	-	-	-	-	-
2	A		BEABE	BEABE	BEABE	A	A	A
3	BEABE	A	A	BEABE	BEABE	A	A	A
4	BEABE		A	A	BEABE	A	A	A
5	-	-	-	-	-	A	-	-
6	-	-	-	-	-	A	-	-
7	A		A	A	BEABE	A	A	A
8	-	-	-	-	-	A	-	-
9	-	-	-	-	-	A	-	-
10	A	A	A	A	A	A	A	A
11	-	-	-	-	-	A	-	-
12	-	-	-	-	-	A	-	-

Tabel 5: Aktionstabel for absolut path-lexeren.

A.3. Lexning af HTTP-headers



Figur 6: Tilstandsmaskine til HTTP-headers

HTTP-header-lexeren opdeler fx nedenstående tegn i de understregede symboler

Content-Length : 4711 \r\n

A.3.1. Tilstandstabel

Symbol/Tilstand	CTL	tspecials	'.'	SP	CR	LF	Andet
1	0	0	2	2	2	2	2
2	0	0	3	2	2	2	2
3	0	0	0	4	0	0	0
4	4	4	4	4	5	4	4
5	0	0	0	0	0	6	0

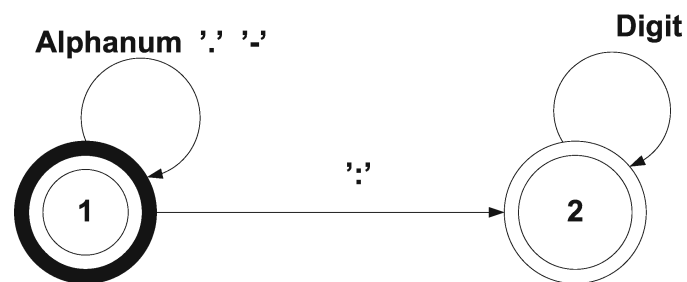
Tabel 6: Tilstandstabel for HTTP-header-lexer. Accepttilstand er tilstand 6.

A.3.2. Aktionstabel

Symbol/Tilstand	CTL	tspecials	'.'	SP	CR	LF	Andet
1	A	A	A	A	A	A	A
2	A	A	BEA	A	A	A	A
3	-	-	-	ABE	-	-	-
4	A	A	A	A	BEA	A	A
5	-	-	-	-	-	ABE	-

Tabel 7: Aktionstabel for HTTP-header-lexer.

A.4. Lexning af HTTP-headeren Host



Figur 7: Tilstandsmaskine til HTTP-host-header.

HTTP-Host-header-lexer opdeler fx nedenstående tegn i de understregede symboler

djengis : 10000

A.4.1. Tilstandstabel

Symbol/Tilstand	ALPHA	'.'	'/'	':'	DIGIT
1	1	1	1	2	1
2	0	0	0	0	2

Tabel 8: Tilstandstabel for HTTP-host-header-lexeren. Accepttilstande er tilstandene 1 og 2.

A.4.2. Aktionstabel

Symbol/ Tilstand	ALPHA	'.'	'/'	':'	DIGIT
1	A	A	A	BEABE	A
2	-	-	-	-	A

Tabel 9: Aktionstabel for HTTP-host-header-lexeren.

B. Testdata

I tabellerne herunder står der udover metode, path og version også den forventede statuskode og hvad den betyder. Da en simpel forespørgsel kun virker for GET, er det kun der, den kommer med i alle testforsøg. For at teste hvordan HEAD, POST og PUT opfører sig, hvis vi forsøger at lave en simpel forespørgsel med dem, har vi lavet et tekstsempel med hver under *generelle forespørgsler*.

B.1. Blandede forespørgsler

I tabellen herunder står der en række forespørgsler der ikke følger RFC 1945, idet de ikke har en HTTP-version i deres forespørgsel. Det drejer sig om HEAD, POST og PUT. Derudover er der en helt tom forespørgsel, og en forespørgsel med en metode vi ikke har implementeret. Til sidst er der en GET forespørgsel der ikke har nogen path. Det forventer vi giver en fejl, da RFC 1945 foreskriver at *abs_path* skal starte med en »/«.

Metode	Path	Version	Statuskode	Beskrivelse
»Tom forespørgsel«			400	Bad request
GET	""		400	Bad request
HEAD	/		400	Bad request
PUT	/text.txt		400	Bad request
POST	/cgitest/test.sh		400	Bad request
DELETE	/text.txt	HTTP/1.0	400	Bad request

Tabel 10: Test af blandede forespørgsler

B.2. GET

I forbindelse med test af GET har vi valgt de to headers *Accept-Language* og *Host* med værdierne *da_DK* og *djengis:10010*, respektivt. I de test hvor versionen er mindre er 1.0 er der ikke headers med, da disse ikke er med i en simpel forespørgsel. Da vi ved simple forespørgsler ikke sender nogle headers eller statuslinje retur, anser vi testen som være gået godt, hvis vi får den forventede fejlkode retur i entitybody'en, eller hvis vi får den forventede side retur. Filen *findes_ikke* er en fil der ikke eksisterer på serveren. Forespørgslen bruges til at teste om vi returnerer statuskoden 404, hvis en ressource ikke findes.

Metode	Path	Version	Statuskode	Beskrivelse
GET	/		(200)	
GET	/	HTTP/1.0	200	OK
GET	/../		(200)	
GET	/../	HTTP/1.0	200	OK
GET	/index.html		(200)	
GET	/index.html	HTTP/1.0	200	OK
GET	/findes_ikke		404	Not found
GET	/findes_ikke	HTTP/1.0	404	Not found
GET	/cgitest		301	Moved permanently
GET	/cgitest	HTTP/1.0	301	Moved permanently
GET	/cgitest/		(200)	
GET	/cgitest/	HTTP/1.0	200	OK

Tabel 11: Test af GET

B.3. HEAD

I HEAD har vi valgt de samme to headers *Accept-Language* og *Host*, også med værdierne *da_DK* og *djengis:10010*.

Metode	Path	Version	Statuskode	Beskrivelse
HEAD	/	HTTP/1.0	200	OK
HEAD	/index.html	HTTP/1.0	200	OK
HEAD	/findes_ikke	HTTP/1.0	404	Not found
HEAD	/cgitest	HTTP/1.0	301	Moved permanently
HEAD	/cgitest/	HTTP/1.0	200	OK

Tabel 12: Test af HEAD

B.4. PUT

Headerne i testen af PUT hedder: *Content-Type*, *Content-Length* og *Host* med værdierne *test/plain*, *16* og *djengis:10010* respektivt. Som entitybody har vi valgt sætningen »Dette er en test«. Grunden til at en af testlinjerne står der to gange er, at vi vil teste om statuskoden bliver anderledes hvis man prøver at lægge en fil på serveren, som ligger der i forvejen. Filen *privat.txt* er skrivebeskyttet.

Metode	Path	Version	Statuskode	Beskrivelse
PUT	/	HTTP/1.0	400	Bad request
PUT	/text.txt	HTTP/1.0	201	Created
PUT	/text.txt	HTTP/1.0	202	Accepted
PUT	/cgitest	HTTP/1.0	400	Bad request
PUT	/cgitest/	HTTP/1.0	400	Bad request
PUT	/privat.txt	HTTP/1.0	500	Internal server error

Tabel 13: Test af PUT

B.5. POST

Til POST har vi følgende headers: *Content-Type*, *Content-Length* og *Host* med de respektive værdier *text/plain*, *32* og *djengis:10010*. Til entitybody'en har vi valgt teksten: *name=Hans+Hansen&mail=hh@nnet.dk*. Filen *text.txt* er en almindelig fil, der ikke er eksekverbar.

Metode	Path	Version	Statuskode	Beskrivelse
POST	/	HTTP/1.0	403	Forbidden
POST	/cgitest/test.sh	HTTP/1.0	200	OK
POST	/text.txt	HTTP/1.0	403	Forbidden

Tabel 14: Test af POST

B.6. CGI

Her vil vi lave to udgaver af en hjemmeside der kalder CGI-scripts. En hvor der i formen står `method="POST"`, og en hvor der står `method="GET"`. Formene ligger i de hjemmesider der hedder `test1.html` og `test2.html`, som ligger i mappen `cgitest` i vores server. De programmer der bliver kaldt skal gerne sende et svar tilbage til klienten med statuskode 200. Herunder kan man se eksempler på, hvordan de to stykker kode vil kunne se ud.

`test1.html`

```
<form action="collect.cgi" method="GET">
  <p>Please type your input (max. 80 chars)</p>
  <input name="data" size="60" MAXLENGTH="80"><br>
  <input type="SUBMIT" name="submit" value="Send">
</form>
```

`test2.html`

```
<form action="collect.cgi" method="POST">
  <p>Please type your input (max. 80 chars)</p>
  <input name="data" size="60" MAXLENGTH="80"><br>
  <input type="SUBMIT" name="submit" value="Send">
</form>
```

C. Koden

C.1. Makefile

```
1 CPPFLAGS=-D_REENTRANT

# This forces webd to allways send a "Cach-Control: no-cache" HTTP header
# Good for testing purposes
NOCACHE=-DNOCACHE

6
# Inter Explorer doesn't show error peages with less than 512 kb
# define this to generate bigger error pages.
# Good for testing purposes
IE_ERROR=-DIE_ERROR

11
# This defines the document root (must be properly escaped)
DOCUMENTROOT=-DDOCUMENTROOT="\./document_root"
```

```

# This defines the log files (must be properly escaped)
16 ACCESSLOG=-DACCESSLOG=\"./log_files/accesslog.csv\"
ERRORLOG=-DERRORLOG=\"./log_files/errorlog.csv\"
INTERNALLOG=-DINTERNALLOG=\"./log_files/internal.log\"

.PHONY: all clean realclean

21 all: webd

connection.o: connection.cpp connection.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

26 server.o: server.cpp server.h connection.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

webserver.o: webserver.cpp webserver.h server.h request.h connection.h statistics.h
31 $(CXX) $(CPPFLAGS) -c -o $@ $<

webd: webd.cpp webserver.o requestlineLexer.o URL.o request.o lexer.o tools.o\
httpheaderLexer.o pathLexer.o HTTPVersion.o server.o connection.o\
HTTPHeaders.o response.o statusCode.o statistics.o lerror.o mutex.o\
36 mimetype.o hostheaderLexer.o
$(CXX) $(CPPFLAGS) -lpthread -o $@ $^

client: client.cc
$(CXX) $(CPPFLAGS) -o $@ $<

41 tools.o: tools.cpp tools.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

lexer.o: lexer.cpp lexer.h
46 $(CXX) $(CPPFLAGS) -c -o $@ $<

mimetype.o: mimetype.cpp mimetype.h lexer.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

51 requestlineLexer.o: requestlineLexer.cpp requestlineLexer.h lexer.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

httpheaderLexer.o: httpheaderLexer.cpp httpheaderLexer.h lexer.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

56 hostheaderLexer.o: hostheaderLexer.cpp hostheaderLexer.h lexer.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

postBodyLexer.o: postBodyLexer.cpp postBodyLexer.h lexer.h
61 $(CXX) $(CPPFLAGS) -c -o $@ $<

request.o: request.cpp request.h HTTPVersion.h URL.h requestlineLexer.h tools.h
HTTPHeaders.h lerror.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

66 response.o: response.cpp response.h HTTPVersion.h request.h statusCode.h connection.h
HTTPHeaders.h mimetype.h URL.h

```

```

$(CXX) $(CPPFLAGS) $(NOCACHE) -c -o $@ $<

URL.o: URL.cpp URL.h tools.h pathLexer.h hostheaderLexer.h
$(CXX) $(CPPFLAGS) $(DOCUMENTROOT) -c -o $@ $<
71
HTTPVersion.o: HTTPVersion.cpp HTTPVersion.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

pathLexer.o: pathLexer.cpp pathLexer.h lexer.h
76 $(CXX) $(CPPFLAGS) -c -o $@ $<

lextest: lextest.cpp requestlineLexer.o httpheaderLexer.o pathLexer.o postBodyLexer.o
lexer.o
$(CXX) $(CPPFLAGS) -o $@ $^

81 statusCode.o: statusCode.cpp statusCode.h
$(CXX) $(CPPFLAGS) $(IE_ERROR) -c -o $@ $<

HTTPHeaders.o: HTTPHeaders.cpp HTTPHeaders.h httpheaderLexer.h
$(CXX) $(CPPFLAGS) -c -o $@ $<
86
statistics.o: statistics.cpp statistics.h response.h request.h connection.h URL.h
$(CXX) $(CPPFLAGS) $(ACCESSLOG) $(ERRORLOG) -c -o $@ $<

lerror.o: lerror.cpp lerror.h tools.h
91 $(CXX) $(CPPFLAGS) $(INTERNALLOG) -c -o $@ $<

datotest: datotest.cpp tools.o
$(CXX) $(CPPFLAGS) -o $@ $^

96 mutex.o: mutex.cpp mutex.h
$(CXX) $(CPPFLAGS) -c -o $@ $<

clean:
$(RM) *.o *~ *.log
101
realclean: clean
$(RM) webd datotest lextest client

```

C.2. webd.cpp

```

#include "webserver.h"
2 #include <signal.h>
#include <string>
#include "mutex.h"
#include "lerror.h"
#include <pthread.h>
7 #include "tools.h"

void exitHandler(int n);
void* serverSpawner(void* port_);

12 Webserver* server;

```

```

int main(int argc, char *argv[])
{
    if(argc!= 2)
17     return 1;

    ErrorLog::Log("Server started."); // we need this because ErrorLog is not thread safe
    Mutex::Instance();
    // Ignore signals from broken pipes – broken pipes are caught by an exception
22     signal(SIGPIPE, SIG_IGN);
    signal(SIGINT, exitHandler);

    pthread_t tid;

27     int* port = new int(std::atoi(argv[1]));
    if ( pthread_create(&tid, NULL, serverSpawner, (void*) port) != 0 )
        ErrorLog::Log("Fejl under traadning af server.");

    pthread_join(tid, NULL);
32     return 0;
}

void exitHandler(int n)
{
37     if(server)
        delete server;
    exit(0);
}

42 void* serverSpawner(void* port_)
{
    int port = *(int*)port_;
    server = new Webserver();
    ErrorLog::Log("Spawning server at port " + itos(port));
47     server->init(port);
    delete server;
}

```

C.3. server.h

```

1  #ifndef serverH
   #define serverH

   #include "connection.h"
   #include <pthread.h>
6  #include <unistd.h>
   #include <sys/types.h>

   class Server
   {
11     public:
        Server();
        //Pre: –

```

```

//Post: misc. settings are initialized for the server

16     virtual ~Server();
        //Pre: –
        //Post: the server is closed nicely

        virtual bool init(int port_);
21     //Pre: –
        //Post: a connection is established and ready to serve clients

        friend void* requestWrapper(void* args_);

26     protected:
        virtual void handleClient(Connection* client_) = 0;
        //Pre: client_ is a connection with a client
        //Post: the communication with the client is handled and finished
        Connection connection;

31 };

struct argWrapper
{
    Connection* client;
36     Server* server;
};

#endif

```

C.4. server.cpp

```

1  #include "server.h"
   #include "lerror.h"
   #include "tools.h"

Server::Server()
6  {
    // intentionally left blank
}

Server::~Server()
11 {
    // intentionally left blank
}

bool Server::init(int port_)
16 {
    if ( !connection.init(port_) )
    {
        ErrorLog::Log("Could not initialize server port " + itos(port_) + ". Is the server
            running already?");
        return false;
21 }

    pthread_t tid;

```

```

pthread_attr_t attr_detached;

26  if ( pthread_attr_init(&attr_detached) != 0 )
    ErrorLog::Log("Fejl under thread detach init");
    if ( pthread_attr_setdetachstate(&attr_detached, PTHREAD_CREATE_DETACHED) != 0 )
        ErrorLog::Log("Fejl under thread setdetachstate");

31  Connection* client;

    //Variant: none (handled by signals).
    //Invariant: all accepted connections will be handled.
    while(1)
36  {
        client = new Connection;
        struct argWrapper* args = new struct argWrapper;
        args->client = client;
        args->server = this;

41        if ( ! client->acceptConnection(connection) )
            {
                delete client;
                delete args;
46        ErrorLog::Log("Fejl under accept");
                continue;
            }
            if ( pthread_create(&tid, &attr_detached, requestWrapper, (void*) args) != 0 )
51        {
                delete client;
                delete args;
                ErrorLog::Log("Fejl under thread create");
                continue;
            }

56        }
        ErrorLog::Log("Fejl under while(1)...");
        return false; // this should not happen;
    }

61 void* requestWrapper(void* args_)
    {
        struct argWrapper* args = (struct argWrapper*) args_;
        args->server->handleClient(args->client);
        //clean up and exit thread
66        delete args;
        pthread_exit(0);
    }

```

C.5. webserver.h

```

#ifndef webserverH
2  #define webserverH

#include "server.h"
#include "request.h"

```

```

#include "connection.h"
7
class Webserver : public Server
{
    public:
        Webserver();
12        // Pre: —
        // Post: misc. settings are initialized for the server

        virtual ~Webserver();
17        // Pre: —
        // Post: the server is shut down nicely

    private:
        virtual void handleClient(Connection* client_);
22        // Pre: client_ is a connection from a client
        // Post: the communication with the client is handled

        virtual void handleRequest(Connection* client_, Request* &request_);
27        // Pre: a client has made a request.
        // Post: values has been given to the request object.

        virtual void createResponse(Connection* client_, Request* &request_);
        // Pre: client_ is the one that has made a request_.
        // Post: a response to the client is sent and statistics is written to the logfil
32 };

#endif

```

C.6. webserver.cpp

```

1 #include "webserver.h"
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
6 #include <sys/shm.h>

#include "response.h"
#include "tools.h"
#include "statistics.h"
11 #include "lerror.h"

Webserver::Webserver() : Server()
{
    // blank
16 }

Webserver::~Webserver()
{
    // intentionally left blank
21 }

```

```

void Webserver::handleClient(Connection* client_)
{
    //Handle request
    26 Request* request = NULL;
    try
    {
        handleRequest(client_, request);
        // create response
    31 createResponse(client_, request);
    }
    catch(BrokenConnectionException)
    {
        if(request)
    36     delete request;
        delete client_;
        return;
    }
    catch(RequestException)
    41 {
        request = new Request("BAD /request\r\n");
        createResponse(client_, request);
    }
    catch(...)
    46 {
        if(request)
            delete request;
        delete client_;
        return;
    51 }

    // close connection to client etc
    delete client_;
    56 delete request;
}

void Webserver::handleRequest(Connection* client_, Request* &request_)
{
    61 std::string s;

    // Get the requestline
    client_->recvLine(s);
    request_ = new Request(s);
    66

    HTTPVersion version = request_->getVersion();

    // if version > 0.9 - full request else simple request
    if (version.getMajor() > 0)
    71 {
        std::string s1, s2;

        client_->recvLine(s1);
        if (s1 != "\r\n")

```

```

76     {
        do
        {
            client_ ->recvLine(s2);

81         if (s2[0] == ' ' || s2[0] == '\t')
            {
                s1 = s1.erase(s1.length()-3, 2);
                s1 = s1 + ' ';

86         int index = 0;

                while(s2[index] == ' ' || s2[index] == '\t')
                {
                    index++;
91                }
                s1 = s1 + s2.erase(0, index);
            }
            else
            {
96                request_ ->setHeader(s1);
                s1 = s2;
            }
        }while(s1 != "\r\n");
    }
101 int length;
    if(request_ ->hasBody(length))
    {
        std::vector<char> bodyBytes;
        for(int i = 0; i < length; i++)
106         bodyBytes.push_back(client_ ->recvByte());
        request_ ->setBody(bodyBytes);
    }
    // set remote ip in dotted decimal notation
    request_ ->setRemoteIP(client_ ->getRemoteIp());
111 }
}

void Webserver::createResponse(Connection* client_, Request* &request_)
{
116 Response response(request_, client_ ->getDesc());
    if ( response.getStatus() != 100 ) // 100 pseudo status that indicates CGI
        response.send(client_);
    Statistics::Log(response, request_, client_);
}

```

C.7. connection.h

```

#ifndef connectionH // -*- c++ -*-
#define connectionH

#include <sys/types.h>
5 #include <sys/socket.h>

```

```

#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <cstdlib>
10 #include <cstdio>
#include <string>
#include "BrokenConnectionException.h"

class Connection
15 {
    public:
        Connection();
        // Pre: –
        // Post: a connection object is created

20
        ~Connection();
        // Pre: –
        // Post: the connection is sjut down nicely

25
        bool init(int port);
        // Pre: port is the port you want to listen on
        // Post: the connection is intilized, returns false on error

        bool acceptConnection(Connection& connection);
30
        // Pre: the connection must be initialized
        // Post: returns true when a client is connected, and false otherwise

        void sendLine(std::string s_);
        // Pre: –
        // Post: the string s_ is sent to the client as a line
35

        void sendFile(int fd_, int length_);
        // Pre: –
        // Post: The file asked for is sent to the client. Default as binary file.
40

        char recvByte();
        // Pre: –
        // Post: waits for a byte from the client and returns it

45
        void recvLine(std::string& s_);
        // Pre: –
        // Post: waits for a line of text from the client and returns it in s_

        int getBytesReceived();
50
        // Pre: –
        // Post: returns the number of bytes received through the connection

        int getBytesSent();
        // Pre: –
55
        // Post: returns the number of bytes sent through the connection

        int getDesc();
        // Pre: –
        // Post: returns the number of the filedescriptor

```

```

60         std::string getRemoteIp();
           // Pre: –
           // Post: returns remote Ip-address

65         int getRemotePort();
           // Pre: –
           // Post: returns remote port

           private:
70             int desc;
               int bytesReceived;
               int bytesSent;
               std::string remote_ip;
               int remote_port;
75 #ifndef __APPLE__
           void sendfile(int desc_, int fd_, void* unused_, int size_);
       #endif
       };
       #endif

```

C.8. connection.cpp

```

1  #include "connection.h"

   // sendfile() is a linux only thing
   #ifndef __linux__
   #include <sys/sendfile.h>
6  #endif

   #include "lerror.h"
   #include <netinet/in.h>
   #include <arpa/inet.h>
11  Connection::Connection()
   {
       bytesReceived = 0;
       bytesSent = 0;
16  remote_port = 0;
       remote_ip = "";
   }

   Connection::~~Connection()
21  {
       close(desc);
   }

   bool Connection::init(int port)
26  {
       struct sockaddr_in s_a;

       if((desc = socket(AF_INET, SOCK_STREAM, 0)) < 0)
           {

```

```

31     ErrorLog::Log("Fejl under socket");
        return false;
    }

    // Make the port immediately reuseable
36     int on = 1;
    setsockopt(desc, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

    s_a.sin_family = AF_INET;
    s_a.sin_addr.s_addr = htonl(INADDR_ANY);
41     s_a.sin_port = htons(port);

    if(bind(desc,(struct sockaddr*) &s_a, sizeof(s_a)) < 0)
    {
        ErrorLog::Log("Fejl under bind");
46         return false;
    }

    if(listen(desc, 10) < 0)
    {
51         ErrorLog::Log("Fejl under listen");
        return false;
    }

    return true;
56 }

bool Connection::acceptConnection(Connection& connection)
{
    struct sockaddr_in addr;
61     socklen_t sin_size = sizeof(struct sockaddr_in);
    if((desc = accept(connection.desc, (struct sockaddr *)&addr, &sin_size)) < 0)
        return false;

66     remote_port = ntohs(addr.sin_port);
    remote_ip = inet_ntoa(addr.sin_addr);

    return true;
}
71

void Connection::sendLine(std::string s_)
{
    int send_length = 1024; //TCP packets maxlength 1024
76     int test;
    for(int i = 0; i < s_.length(); i++)
    {
        // Send will signal SIGPIPE if trying to send on broken pipe.
        // This is ignored with SIGNAL(SIGPIPE,SIG_IGN) in main.cpp
81     // The exception is caught in webservers.cpp
        if ( i+send_length > s_.length() )
            send_length = s_.length()-i;
        test = send(desc, s_.substr(i,i+send_length+1).c_str(), send_length, 0);
    }
}

```

```

    if(test <= 0)
86     throw BrokenConnectionException();
    i += test-1;
}

bytesSent += s_.length();
91 }

void Connection::sendFile(int fd_, int length_)
{
    sendfile(desc, fd_, NULL, length_);
96     close(fd_);
}

char Connection::recvByte()
{
101     int exception;
    char buf;
    exception = recv(desc, &buf, 1, 0);
    if(!exception)
        throw BrokenConnectionException();
106     return buf;
}

void Connection::recvLine(std::string& s_)
{
111     char buf;
    s_ = "";
    //Variant: when the client sends a '\r' followed by a '\n'.
    //Invariant: s_ is the sum of received bytes [s_ = s_ + buf].
    while(recv(desc, &buf, 1, 0))
116     {
        bytesReceived++;
        if (buf == '\r')
            {
                if(recv(desc, &buf, 1, 0))
121                {
                    bytesReceived++;
                    if (buf == '\n')
                        {
                            s_ += "\r\n";
126                            return;
                        }
                    else
                        s_ = s_ + '\r' + buf;
                }
            }
131     else
        break;
    }
    else
        s_ = s_ + buf;
136 }
    throw BrokenConnectionException();
}

```

```

141 int Connection::getBytesSent()
    {
        return bytesSent;
    }

146 int Connection::getBytesReceived()
    {
        return bytesReceived;
    }

151 std::string Connection::getRemoteIp()
    {
        return remote_ip;
    }

156 int Connection::getRemotePort()
    {
        return remote_port;
    }

161 int Connection::getDesc()
    {
        return desc;
    }

166 #ifndef __linux__
    // This is a simpel, rough and unefficient implementation of what
    // linux' sendfile() does and taking the same number of arguments.
    void Connection::sendfile(int desc_, int fd_, void* unused_, int size_)
    {
171     std::string s="";
        char buf[1];
        for (int i=0; i < size_; i++)
            {
176         read(fd_, buf, 1);
            s += buf[0];
            }
        sendLine(s);
    }
#endif

```

C.9. request.h

```

#ifndef requestH // -- c++ --
#define requestH

#include <vector>
#include <string>
5 #include "HTTPVersion.h"
#include "URL.h"
#include "requestlineLexer.h"

```

```

#include "HTTPHeaders.h"
10 #include "RequestException.h"

enum method{GET, HEAD, POST, PUT, BAD};

class Request
15 {
  public:
    Request(std::string requestLine);
    // Pre: a request from the client is recieved
    // Post: if lexer and parser is succeeded, Method, HTTPVersion and URL is
20 // set otherwise return an error

    int getMethod();
    // Pre: –
    // Post: returns the method
25
    void setMethod();
    // Pre: –
    // Post: type of method is set

30 HTTPVersion getVersion();
    // Pre: –
    // Post: returns an HTTPVersion object

    void setHTTPVersion();
35 // Pre: –
    // Post: type of HTTPVersion is set

    HTTPHeaders getHeaders();
    // Pre: –
40 // Post: httpHeaders is returned

    void setHeader(const std::string header_);
    // Pre: header_ contains the value of the header that is going to be set
    // Post: headers is set
45
    std::vector<char> getBody();
    // Pre: –
    // Post: a vector is returned. Contains the body text as char's

50 void setBody(std::vector<char>& body_);
    // Pre: –
    // Post: body has been set

    bool hasBody(int& length_);
55 // Pre: –
    // Post: returns true if bodylength exists otherwise false

    URL getURL();
    // Pre: –
60 // Post: path is returned

    void setRemoteIP(std::string ip_);

```

```

        // Pre: ip_ is the clients ip-address
        // Post: ip is set
65     std::string getRemoteIP();
        // Pre: –
        // Post: the clients ip adress is returned

70     private:
        int method;
        std::string ip;
        HTTPHeaders httpHeaders;
        std::vector<std::string> requestSymSeq;
75     std::vector<char> body;
        URL path;
        HTTPVersion httpVersion;
    };

80     struct headerContent
    {
        std::string content;
        std::string data;
    };
85     #endif

```

C.10. request.cpp

```

#include "request.h"
#include "tools.h"
#include "lerror.h"
4     Request::Request(std::string requestLine)
    {
        RequestLineLexer lexer;

9         if (requestLine == "\r\n")
            throw RequestException();

        if(!lexer.lex(requestLine, requestSymSeq))
14            throw RequestException();

        if (!(requestSymSeq.size() == 4 || requestSymSeq.size() == 10))
            throw RequestException();

19        if (requestSymSeq.size() == 10 && requestSymSeq[4] != "HTTP")
            throw RequestException();

        setMethod();
        setHTTPVersion();
24        path.setURL(requestSymSeq[2]);
    }

```

```

int Request::getMethod()
29 {
    return method;
}

void Request::setMethod()
34 {
    if (requestSymSeq.size() != 4)
    {
        if(requestSymSeq[0] == "GET")
            method = GET;
39     else if(requestSymSeq[0] == "HEAD")
            method = HEAD;
        else if(requestSymSeq[0] == "POST")
            method = POST;
        else if(requestSymSeq[0] == "PUT")
44     method = PUT;
        else
        {
            method = BAD;
        }
49     }
    else
    {
        if(requestSymSeq[0] == "GET")
            method = GET;
54     else
        {
            method = BAD;
        }
    }
59 }

void Request::setHTTPVersion()
{
    if (requestSymSeq.size() != 4)
64     {
        httpVersion.setMajor((atoi(requestSymSeq[6].c_str())));
        httpVersion.setMinor((atoi(requestSymSeq[8].c_str())));
    }
    else
69     {
        httpVersion.setMajor(0);
        httpVersion.setMinor(9);
    }
}

74 HTTPVersion Request::getVersion()
{
    return httpVersion;
}
79

```

```

    HTTPHeaders Request::getHeaders()
    {
84     return httpHeaders;
    }

    void Request::setHeader(const std::string header_)
    {
        headerPair header = httpHeaders.setHeader(header_);
89     if (header.name == "Host")
        path.setHost(header.content);
    }

94     void Request::setBody(std::vector<char>& body_)
    {
        body = body_;
    }

99     std::vector<char> Request::getBody()
    {
        return body;
    }

104    bool Request::hasBody(int& length_)
    {
        length_ = -1;
        std::string contentLengthStr;
        if (httpHeaders.getHeader("Content-Length", contentLengthStr))
109         length_ = atoi(contentLengthStr.c_str());
        if (length_ >= 0)
            return true;
        return false;
    }

114    URL Request::getURL()
    {
        return path;
    }

119    void Request::setRemoteIP(std::string ip_)
    {
        ip = ip_;
    }

124    std::string Request::getRemoteIP()
    {
        return ip;
    }

```

C.11. response.h

// This is the class that handles the response to the client.

2

```

#ifndef responseH // *- c++ *-
#define responseH
#include <string>
#include "tools.h"
7 #include "request.h"
#include "statusCode.h"
#include "HTTPVersion.h"
#include "HTTPHeaders.h"
#include "connection.h"

12
class Response
{
public:
    Response();
17    // Pre: -
    // Post: A response object is created.

    Response(Request *request_, int desc_);
    // Pre: A request from the client is received.
22    // Post: The method is set otherwise an error is returned.

    void setMethodGetAndHead();
    // Pre: -
    // Post: Headers, entityBody and statusCode is set.

27    void setMethodPut();
    // Pre: -
    // Post: Headers, entityBody and statusCode is set.
    // A file is overwritten or created, from the
32    // clients request, otherwise an error is returned.

    void setMethodPost();
    // Pre: -
    // Post: A cgi-script has been executet.

37    void send(Connection *client_);
    // Pre: A request from the client is received.
    // Post: A response to the client is sent.

42    int getStatus();
    // Pre: -
    // Post: Returns the statusCode.

    std::string directoryListing(const std::string docroot_, const std::string path_);
47    // Pre: Path_ is the full path to a directory
    // Post: A HTML listing is returned

private:
    Request* req;
52    StatusCode code;
    int desc;
    int fd;
    HTTPHeaders headers;
    std::string entityBody;

```

```

57  bool badReq;

    std::string runCgi();
    // Pre: –
62  // Post: All meta variabels has been set.
    // If not the cgi stript is succesfully executed an error message string is returned
    // otherwise an empty string is returned.

    std::string getContentType(std::string path_);
67  // Pre: A request from the client is received.
    // Post: Returns the mineType.

    int getLength(std::string path_);
72  // Pre: A request from the client is received.
    // Post: Returns the total contentLength of the path.

    void badRequest();
    // Pre: –
    // Post: statusCode for badRequest is set, and badReq is set to true.

77  std::string getStatusLineAndHeaders();
    // Pre: –
    // Post: Returns the statusline and headers

82  void writeToFile(std::string path_);
    // Pre: A request from the client is received.
    // Post: The file the client asked for, has been overwritten if it went well.
    // A statusCode and headers has been set.

87  void createNewFile(std::string path_);
    // Pre: A request from the client is received.
    // Post: A new file is created, statusCode and headers are set.

92  // Denne funktion er endnu ikke lavet? Eller måske er den flyttet?
    std::string getDirStructure(std::string path_);
    // Pre:
    // Post:

97  };
#endif

```

C.12. response.cpp

```

#include <stdio.h>
2  #include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
7  #include <dirent.h>
#include <netdb.h> // gethostbyname()

```

```

#include "mimetype.h"
#include "response.h"
#include "tools.h"
12 #include "mutex.h"
#include "URL.h"
#include "lerror.h"

Response::Response()
17 {
    // Left blank
}

Response::Response(Request *request_, int desc_)
22 {
    desc = desc_;
    req = request_;
    entityBody = "";
    fd = -1;
27 badReq = false;
    switch(req->getMethod())
    {
        case GET:
        case HEAD:
32     if((req->getVersion()).getMajor() < 1 && req->getMethod() != GET)
        badRequest();
        else
            setMethodGetAndHead();
        break;
37     case POST:
        if((req->getVersion()).getMajor() < 1)
            badRequest();
        else
            setMethodPost();
42     break;
        case PUT:
        if((req->getVersion()).getMajor() < 1)
            badRequest();
        else
47     setMethodPut();
        break;
        default:
            badRequest();
            break;
52     }
}

void Response::setMethodGetAndHead()
57 {
    std::string path = (req->getURL().getFilename());
    if(isFile(path))
    {
        if(isExecutable(path))
62     {

```

```

        runCgi();
        return;
    }
    else if(isReadable(path))
67     {
        code.setStatus(200);
        headers.setHeader("Content-Type: " + getContentType(path) + "\r\n");
        fd = open(path.c_str(), O_RDONLY, NULL);
    }
72     else
    {
        code.setStatus(403);
        entityBody = code.getEntityBody();
        headers.setHeader("Content-Type: text/html\r\n");
77     }
        headers.setHeader("Content-Length: " + itos(getContentLength(path)) + "\r\n");
#ifdef NOCACHE
        headers.setHeader("Cache-Control: no-cache\r\n");
#endif // NOCACHE
82     headers.setHeader("Date: " + rfcdate() + "\r\n");
        return;
    }
    else if(isDir(path))
    {
87     if(isReadable(path))
        {
            if (req->getURL().getURL()[req->getURL().getURL().size()-1] != '/')
            {
                code.setStatus(301);
92                entityBody = code.getEntityBody();
                headers.setHeader("Location: " + (req->getURL().getURL() + "/\r\n");
            }
            else
            {
97                code.setStatus(200);
                entityBody = directoryListing(req->getURL().getFilename(), req->getURL().getPath
                    ());
            }
        }
        else
102    {
        code.setStatus(403);
        entityBody = code.getEntityBody();
    }
}
107 else
    {
        code.setStatus(404);
        entityBody = code.getEntityBody();
    }
112 headers.setHeader("Content-Length: " + itos(entityBody.length()) + "\r\n");
headers.setHeader("Content-Type: text/html\r\n");
#ifdef NOCACHE
headers.setHeader("Cache-Control: no-cache\r\n");

```

```

#endif // NOCACHE
117     headers.setHeader("Date: " + rfcdate() + "\r\n");
    }

    void Response::setMethodPut()
    {
122         std::string path = (req->getURL().getFilename());

        if(isDir(path))
        {
            code.setStatus(400);
            entityBody = code.getEntityBody();
127             headers.setHeader("Content-Length: " + itos(entityBody.length()) + "\r\n");
            headers.setHeader("Content-Type: text/html\r\n");
#ifndef NOCACHE
            headers.setHeader("Cache-Control: no-cache\r\n");
132 #endif // NOCACHE
            headers.setHeader("Date: " + rfcdate() + "\r\n");
        }
        else if(isFile(path))
            writeToFile(path);
137     else
        createNewFile(path);
    }

    void Response::setMethodPost()
142 {
    runCgi();
}

    void Response::send(Connection *client_)
147 {
    if(badReq)
        client_->sendLine(entityBody);
    else if((req->getVersion()).getMajor() < 1)
    {
152     if(fd == -1)
        client_->sendLine(entityBody);
        else
            client_->sendFile(fd, getContentLength(req->getURL().getFilename()));
    }
157 else //statusline
    {
        client_->sendLine(getStatusLineAndHeaders());
        if(req->getMethod() != HEAD)
        {
162     if(fd == -1)
            client_->sendLine(entityBody);
            else
                client_->sendFile(fd, getContentLength(req->getURL().getFilename()));
        }
    }
167 }
}

```

```

    int Response::getStatus()
    {
172     return code.getStatus();
    }

std::string Response::directoryListing(const std::string filename_, const std::string path_)
{
177     std::string content = "";

    content += "<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict //EN\"\n";
    content += " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">\n";
    content += "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n";
182     content += "<head>\n";
    content += "<title>Index of ";
    content += path_;
    content += "</title>\n";
    content += "</head>\n";\
187     content += "<body>\n";
    content += "<h1>Index of ";
    content += path_;
    content += "</h1>\n";
    content += "<ul>\n";

192     DIR* dp = opendir(filename_.c_str());
    struct dirent *de;

    while( de = readdir(dp) )
197     {
        if (!strncmp(de->d_name, ".", strlen(de->d_name)))
            continue;
        if (!strncmp(de->d_name, "..", strlen(de->d_name)) && path_ == "/")
            continue;
202     content += "<li><a href=\"";
    content += de->d_name;
    if (de->d_type == DT_DIR)
        content += "/";
    content += "\">";
207     content += de->d_name;
    if (de->d_type == DT_DIR)
        content += "/";
    content += "</a>";
    content += "</li>\n";
212     };

    closedir(dp);

    content += "</ul>\n";
217     content += "</body>\n";
    content += "</html>\n\0";

    return content;
}
222
std::string Response::runCgi()

```

```

{
  std::vector<std::string> meta;
  // ** MUST FOR CGI/1.1 ** //
227
  /*
    These are the requirements for server handling of message–bodies directed to CGI/1.1 resources:

    1. The message–body the server provides to the CGI script MUST have any transfer encodings removed.
    232 2. The server MUST derive and provide a value for the CONTENT_LENGTH metavariable that reflects
    the length of the message–body after any transfer decoding.
    3. The server MUST leave intact any content–encodings of the message–body.
    OBS OBS OBS – item 1 however does not apply to us because transfer–encoding is not implemented in
    HTTP/1.0
  */
237
  // CONTENT_LENGTH
  int length;
  if (req->hasBody(length))
    meta.push_back(("CONTENT_LENGTH=" + itos(length)));
242  else
    meta.push_back("CONTENT_LENGTH=");

  // CONTENT_TYPE
  std::string content = "";
247 (req->getHeaders()).getHeader("Content-Type",content);
  meta.push_back(("CONTENT_TYPE=" + content));

  // GATEWAY_INTERFACE
  meta.push_back("GATEWAY_INTERFACE=CGI/1.1");
252

  // PATH_INFO
  meta.push_back("PATH_INFO=");

  // QUERY_STRING
257 meta.push_back("QUERY_STRING=" + ((req->getURL()).getQuery()));

  // REMOTE_ADDR
  meta.push_back("REMOTE_ADDR=" + req->getRemoteIP());

262 // REQUEST_GETMETHOD
  switch(req->getMethod())
  {
    case GET:
      meta.push_back("REQUEST_METHOD=GET");
267     break;
    case POST:
      meta.push_back("REQUEST_METHOD=POST");
      break;
    default:
272     break;
  }

  // SCRIPT_NAME
  meta.push_back("SCRIPT_NAME=" + (req->getURL()).getPath());

```

```

277 // SERVER_NAME
meta.push_back("SERVER_NAME=" + (req->getURL()).getHost());

// SERVER_PORT
meta.push_back("SERVER_PORT=" + itos((req->getURL()).getPort()));
282

// SERVER_PROTOCOL
meta.push_back("SERVER_PROTOCOL=HTTP/1.0");

// SERVER_SOFTWARE
287 meta.push_back("SERVER_SOFTWARE=NB_SEM3_GRP1_SERVER/1.0.33");

// ** SHOULD FOR CGI/1.1 ** //

// AUTH_TYPE
292 std::string authData = "";
if((req->getHeaders()).getHeader("Authorization",authData)
    {
        int sp = authData.find(" ");
        // substr returns the substring that begins at position o, and contains an "empty" length.
297 authData = authData.substr(o, sp);
    }
meta.push_back(("AUTH_TYPE=" + authData));

302 // REMOTE_HOST
struct hostent* host = gethostbyname((req->getRemoteIP()).c_str());
std::string name = (host->h_name);
meta.push_back("REMOTE_HOST=" + name);

307 // ** MAY FOR CGI/1.1 ** //
// PATH_TRANSLATED
if((req->getURL()).getPathinfo() != "")
    meta.push_back("PATH_TRANSLATED=");
312

// ** MAY FOR CGI/1.1 - not implemented** //
// REMOTE_IDENT
// REMOTE_USER

317 // Place the metavariables in an environment array. It has room for all must and all should
// variables and is NULL terminated.
char* env[16];
for (int i = 0; i < meta.size(); i++)
322 env[i] = (char*)meta[i].c_str();
env[meta.size()] = NULL;

// time to fork()

327 code.setStatus(100); // pseudo status to indicate CGI-script

pid_t pid;

```

```

332  int pipes[2];
    pipe(pipes);

    pid = fork();
    if ( pid == 0 ) // child
    {
337      close(fileno(stdin));
          dup(pipes[0]);
          close(fileno(stdout));
          dup2(desc, fileno(stdout));

342      if (execve((req->getURL()).getFilename().c_str(), NULL, env) < 0)
          {
              ErrorLog::Log("Could not execve() CGI-script: " + (req->getURL()).getFilename());
              exit(8); // ENOEXEC - Exec format error
          }
    }
347  }
    else if ( pid < 0 ) // fork fejl
    {
        ErrorLog::Log("Could not fork in runCgi");
        code.setStatus(500);
352    }

    std::string body = "";
    for (int i=0; i < req->getBody().size(); i++)
    {
357        body += req->getBody()[i];
    }

    if ( body.length() > 0)
        write(pipes[1], body.c_str(), body.length());
362

    close(pipes[0]);
    close(pipes[1]);

    //Suspend execution until children has terminated - also children that has stopped - WUNTRACED
367  int status;
    waitpid(pid, &status, WUNTRACED);

    // WIFEXITED(status) is false if the CGI-script terminated abnormally (i.e. received a signal)
    // WIFEXITSTATUS(status) is the exit status of the CGI-script
372  if ( !WIFEXITED(status) || WEXITSTATUS(status) != 0 )
    {
        code.setStatus(500);

        if ( WEXITSTATUS(status) == 8 )
377          code.setStatus(403);
        else if ( WEXITSTATUS(status) != 0 )
            ErrorLog::Log("CGI-script (" + (req->getURL()).getFilename() + ") exited with exit
                status " + itos(WEXITSTATUS(status)) + ".");
        // WIFSIGNALED is true if the CGI-script received a signal
        // WTERMSIG is the signal number it received
382  if ( WIFSIGNALED(status) )
            ErrorLog::Log("CGI-script (" + (req->getURL()).getFilename() + ") received the signal:

```

```

        " + itos(WTERMSIG(status)) + ".");
// WIFSTOPPED is true if the CGI-script did not terminate, but just stopped (see WUNTRACED above)
if ( WIFSTOPPED(status) )
{
387     ErrorLog::Log("CGI-script (" + req->getURL().getFilename() + ") did not terminate,
        but has stopped and can be restarted.");
        kill(pid, SIGKILL);
    }
}

392     entityBody = code.getEntityBody();
headers.setHeader("Content-Length: " + itos(entityBody.length()) + "\r\n");
headers.setHeader("Content-Type: text/html\r\n");
#ifdef NOCACHE
headers.setHeader("Cache-Control: no-cache\r\n");
397 #endif // NOCACHE
headers.setHeader("Date: " + rfcdate() + "\r\n");
return "";
close(desc);
return "";
402 }

std::string Response::getContentType(std::string path_)
{
    MimeType type;
407     return type.getMimeType(path_);
}

int Response::getContentLength(std::string path_)
{
412     int length;
    struct stat buffer;
    stat(path_.c_str(), &buffer);
    length = buffer.st_size;
    return length;
417 }

void Response::badRequest()
{
    code.setStatus(400);
422     entityBody = code.getEntityBody();
    badReq = true;
}

std::string Response::getStatusLineAndHeaders()
427 {
    std::string str = "";
    str = "HTTP/1.0 " + code.getReasonPhrase() + "\r\n";
    int num = headers.getNumberOfHeaders();

432     struct headerPair header;
    for(int i = 0; i < num; i++)
    {
        header = headers[i];
    }
}

```

```

        str += header.name + ": " + header.content + "\r\n";
437     }

    str += "\r\n";
    return str;
}

442 void Response::writeToFile(std::string path_)
{
    Mutex::Lock(PUT_MUTEX);
    FILE* fp = fopen(path_.c_str(), "w");
447     if(fp != NULL)
    {
        for(int i = 0; i < (req->getBody()).size(); i++)
        {
452             fprintf(fp, "%c", (req->getBody())[i]);
        }

        fflush(fp);
        fclose(fp);
457     Mutex::Unlock(PUT_MUTEX);
        code.setStatus(202);
    }
    else
    {
462         // fopen failed
        if(isWriteable(path_))
            code.setStatus(500);
        else
            code.setStatus(403);
467     }

    entityBody = code.getEntityBody();
    headers.setHeader("Content-Length: " + itos(entityBody.length()) + "\r\n");
    headers.setHeader("Content-Type: text/html\r\n");
472 #ifdef NOCACHE
        headers.setHeader("Cache-Control: no-cache\r\n");
    #endif // NOCACHE
        headers.setHeader("Date: " + rfcdate() + "\r\n");
    }

477 void Response::createNewFile(std::string path_)
{
    Mutex::Lock(PUT_MUTEX);
    FILE* fp = fopen(path_.c_str(), "w");
482     if(fp != NULL)
    {
        for(int i = 0; i < (req->getBody()).size(); i++)
        {
487             fprintf(fp, "%c", (req->getBody())[i]);
        }
    }
}

```

```

        fflush(fp);
        fclose(fp);
492     Mutex::Unlock(PUT_MUTEX);
        code.setStatus(201);
    }
    else
        code.setStatus(500);
497
    entityBody = code.getEntityBody();
    headers.setHeader("Content-Length: " + itos(entityBody.length()) + "\r\n");
    headers.setHeader("Content-Type: text/html\r\n");
#ifdef NOCACHE
502     headers.setHeader("Cache-Control: no-cache\r\n");
#endif // NOCACHE
    headers.setHeader("Date: " + rfcdate() + "\r\n");
}

```

C.13. statistics.h

```

#ifdef statisticsH // -*- c++ -*-
#define statisticsH

#include <pthread.h>
5 #include <string>
#include "response.h"
#include "request.h"
#include "connection.h"

10 class Statistics
{
    public:
        static void Log(Response& response, Request* request, Connection *client);
        // Pre: response, request og client are the objects that holds the informations,
15 // that is writtten to the logfile.
        // Post: one statistic is written to the logfile.

    private:
        Statistics();
        // Pre: -
        // Post: the statistics logfile is initialized otherwise return an error to errorlog file

        static Statistics *pinstance;

25 void log(Response& response, Request* request, Connection *client);
        // Pre: response, request og client are the objects that holds the informations,
        // that is writtten to the logfile.
        // Post: statistic has been written to the logfile

30 pthread_mutex_t access_log_mutex;
    pthread_mutex_t error_log_mutex;
    std::string access_log_file; // status code 1xx, 2xx, 3xx
    std::string error_log_file; // status code 4xx, 5xx
};

```

35

```
#endif // statisticsH
```

C.14. statistics.cpp

```
#include "statistics.h"
#include "tools.h"
#include <time.h>
4 #include <stdio.h>
#include "mutex.h"
#include "lerror.h"
#include "URL.h"

9 Statistics* Statistics::pinstance = NULL;

void Statistics::Log(Response& response, Request* request, Connection* client)
{
    if ( !pinstance )
14     {
        Mutex::Lock(STAT_MUTEX);
        if ( !pinstance )
            {
                pinstance = new Statistics;
19            }
        Mutex::Unlock(STAT_MUTEX);
    }
    pinstance->log(response, request, client);
}

24 Statistics::Statistics()
{
    #ifdef ACCESSLOG
        access_log_file = ACCESSLOG;
29 #else
        access_log_file = "accesslog.csv";
    #endif

    #ifdef ERRORLOG
34     error_log_file = ERRORLOG;
    #else
        error_log_file = "errorlog.csv";
    #endif

39     if (pthread_mutex_init(&access_log_mutex, NULL) != 0)
        ErrorLog::Log("Unable to initialize log file: " + access_log_file + ".");
    if (pthread_mutex_init(&error_log_mutex, NULL) != 0)
        ErrorLog::Log("Unable to initialize log file: " + error_log_file + ".");
}

44 void Statistics::log(Response& response, Request* request, Connection* client)
{
    std::string log_file;
    std::string entry;
```

```

49  int status = response.getStatus();
    std::string referer;
    (request->getHeaders()).getHeader("Referer", referer);
    std::string location;
    (request->getHeaders()).getHeader("Location", location);

54  entry = itos(status); // status
    entry += ", \"\" + (request->getURL()).getURL() + "\"\"; // request URL
    entry += ", \"\" + referer + "\"\"; // Referer
    entry += ", \"\" + location + "\"\"; // Redirect location
59  entry += ", " + client->getRemoteIp(); // remote ip
    entry += ": " + itos(client->getRemotePort()); // remote port
    entry += ", " + itos(client->getBytesReceived()); // bytes received
    entry += ", " + itos(client->getBytesSent()); // bytes sent
    entry += ", \"\" + rfcdate() + "\"\"; //date

64  if (status < 400 || error_log_file == access_log_file) // 1xx, 2xx, 3xx
    {
        log_file = access_log_file;
        pthread_mutex_lock(&access_log_mutex);
69  }
    else // 4xx, 5xx
    {
        log_file = error_log_file;
        pthread_mutex_lock(&error_log_mutex);
74  }

    FILE *fp = fopen(log_file.c_str(), "a");

    std::string logText = entry;

79  for(int i = 0; i < logText.size(); i++)
    {
        fprintf(fp, "%c", logText.c_str()[i]);
    }

84  fprintf(fp, "\n");
    fflush(fp);
    fclose(fp);

89  if (status < 400 || error_log_file == access_log_file)
    pthread_mutex_unlock(&access_log_mutex);
    else
    pthread_mutex_unlock(&error_log_mutex);
}

```

C.15. lexer.h

```

// -*- c++ -*-
2
#ifdef lexerH
#define lexerH
#include <vector>

```

```

#include <string>
7 #include <stdarg.h>

// actions
#define NOOP 0
#define ADD_TO_BUFFER 1
12 #define BUFFER_TO_SEQ 2
#define EMPTY_BUFFER 3
#define REMOVE_ACTIONS 10 // pseudo action

// symbols
17 #define CTL 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
    29, 30, 31
#define DIGIT '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'
#define ALPHAL 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',
    's', 't', 'u', 'v', 'w', 'x', 'y', 'z'
#define ALPHAU 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
    'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'
#define ALPHA ALPHAL, ALPHAU
22 #define ALPHANUM ALPHA, DIGIT
#define HEX DIGIT, 'A', 'B', 'C', 'D', 'E', 'F', 'a', 'b', 'c', 'd', 'e', 'f'
#define CR '\r'
#define LF '\n'
#define SP ' '
27 #define HT '\t'

// macrofunctions
#define state(...) state_(__VA_ARGS__, -1)
#define action(...) action_(__VA_ARGS__, -1)
32 #define setAcceptState(...) setAcceptState_(__VA_ARGS__, -1)

// special states
#define errorState 0

37 class Element
{
public:
    Element();
    // Pre: -
42    // Post: a element object is initialized

    int newState;
    std::vector<int> actions;
};

47 class Lexer
{
public:
    Lexer();
52    // Pre: -
    // Post: a lexer object is initialized

    void state_(int current, int next, ...);
    // Pre: current and next are state numbers, ... are char's

```

```

57 // Post: a state change from current to next is configured on ... as input

    void action_(int current, int action, ...);
    // Pre: current is a state, action an action as specified above, ...
    // is char's
62 // Post: the action is configured on ... as input in state current

    void setStartState(int startState_);
    // Pre: startState_ is a state
    // Post: startState_ is set a the start state
67

    void setAcceptState_(int acceptState_, ...);
    // Pre: takes a va_list of states as arguments
    // Post: the states are configured as accept states

72 bool lex(std::string expression, std::vector<std::string>& symbolSequence);
    // Pre: expression is a string to be lexed
    // Post: return true/false if lexing succeeded/failed. The
    // symbolSequence is placed in symbolSequence in both cases

77 private:
    std::vector< std::vector<Element> > states;
    int startState;
    std::vector<int> acceptStates;
};
82 #endif // lexerH

```

C.16. lexer.cpp

```

#include "lexer.h"
2
Element::Element()
{
    newState = errorState;
    actions = std::vector<int>(0);
7 }

Lexer::Lexer()
{
    startState = errorState;
12 acceptStates = std::vector<int>(0);
}

void Lexer::state_(int current_, int next_, ...)
{
17 if ( states.size() <= current_ )
    for (int i=states.size(); i <= current_; i++)
        states.push_back( std::vector<Element>(256));

    if ( states.size() <= next_ )
22 for (int i=states.size(); i <= next_; i++)
        states.push_back( std::vector<Element>(256));
}

```

```

// Variable arguments list
va_list ap;
27 va_start(ap, next_);
    int symbol = 0;

    symbol = va_arg(ap, int);

32 if (symbol == -1) // if no symbol arguments, set state for all symbols
    {
        for (int i=0; i<256; i++)
            states[current_][i].newState = next_;
    }

37 while (symbol != -1)
    {
        states[current_][symbol].newState = next_;
        symbol = va_arg(ap, int);
42 }
    va_end(ap);
}

void Lexer::action_(int current_, int action_, ...)
47 {
    if ( states.size() <= current_ )
        for (int i=states.size(); i <= current_; i++)
            states.push_back( std::vector<Element>(256));

52 // Variable arguments list
    va_list ap;
    va_start(ap, action_);
    int symbol = 0;

57 symbol = va_arg(ap, int);

    if (symbol == -1) // if no symbol arguments, set action for all symbols
        {
            for (int i=0; i<256; i++)
162         if (action_ == REMOVE_ACTIONS)
                states[current_][i].actions.resize(0);
            else
                states[current_][i].actions.push_back(action_);
        }

67 while (symbol != -1)
    {
        if (action_ == REMOVE_ACTIONS)
            states[current_][symbol].actions.resize(0);
72         else
            states[current_][symbol].actions.push_back(action_);
            symbol = va_arg(ap, int);
    }
    va_end(ap);
77 }

```

```

void Lexer::setStartState(int startState_)
{
    startState = startState_;
82 }

void Lexer::setAcceptState_(int acceptState_, ...)
{
    if (acceptState_ == -1)
87     return;

    acceptStates.push_back(acceptState_);

    // Variable arguments list
92     va_list ap;
    va_start(ap, acceptState_);
    int next = 0;

    next = va_arg(ap, int);
97

    while (next != -1)
    {
        acceptStates.push_back(next);
        next = va_arg(ap, int);
102    }
    va_end(ap);
}

bool Lexer::lex(std::string expression, std::vector<std::string>& symbolSequence)
107 {
    int currentState = startState;
    int index = 0;
    char c;
    std::string buffer = "";
112    symbolSequence.resize(0);

    while ( index < expression.length() )
    {
        c = expression[index++];
117

        for (int i=0; i < states[currentState][c].actions.size(); i++)
            switch (states[currentState][c].actions[i])
            {
                case ADD_TO_BUFFER:
122                 buffer += c;
                 break;
                case BUFFER_TO_SEQ:
                 if (buffer != "")
                     symbolSequence.push_back(buffer);
127                 break;
                case EMPTY_BUFFER:
                 buffer = "";
                 break;
                default:

```

```

132         break;
        }

        currentState = states[currentState][c].newState;
    }
137     if (buffer != "")
        symbolSequence.push_back(buffer);

    for (int i=0; i < acceptStates.size(); i++)
142         if (currentState == acceptStates[i])
            return true;
    return false;
}

```

C.17. requestlineLexer.h

```

#ifndef requestlineLexerH // -*- c++ -*-
#define requestlineLexerH

#include "lexer.h"
5
class RequestLineLexer: public Lexer
{
    public:
    RequestLineLexer();
10    // Pre: –
    // Post: the object has been initialized
};

#endif // requestlineLexerH

```

C.18. requestlineLexer.cpp

```

1 #include "requestlineLexer.h"

RequestLineLexer::RequestLineLexer()
{
    state( 1, 2, ALPHA);
6    state( 2, 2, ALPHA);
    state( 2, 3, SP);
    state( 3, 4);
    state( 3, errorState, SP);
    state( 3, errorState, CR);
11    state( 4, 4);
    state( 4, 5, SP);
    state( 4, 11, CR);
    state( 5, 6, ALPHA);
    state( 6, 6, ALPHA);
16    state( 6, 7, '/' );
    state( 7, 8, DIGIT);
    state( 8, 8, DIGIT);
    state( 8, 9, '.' );
}

```

```

state( 9, 10, DIGIT);
21 state(10, 10, DIGIT);
state(10, 11, CR);
state(11, 12, LF);

setStartState(1);
26 setAcceptState(12);

action( 1, ADD_TO_BUFFER, ALPHA);

action( 2, ADD_TO_BUFFER, ALPHA);
31 action( 2, BUFFER_TO_SEQ, SP);
action( 2, EMPTY_BUFFER, SP);
action( 2, ADD_TO_BUFFER, SP);
action( 2, BUFFER_TO_SEQ, SP);
action( 2, EMPTY_BUFFER, SP);

36 action( 3, ADD_TO_BUFFER);
action( 3, REMOVE_ACTIONS, SP, CR);
action( 3, EMPTY_BUFFER, SP, CR);

41 action( 4, ADD_TO_BUFFER);
action( 4, REMOVE_ACTIONS, SP, CR);
action( 4, BUFFER_TO_SEQ, SP, CR);
action( 4, EMPTY_BUFFER, SP, CR);
action( 4, ADD_TO_BUFFER, SP, CR);
46 action( 4, BUFFER_TO_SEQ, SP);
action( 4, EMPTY_BUFFER, SP);

action( 5, ADD_TO_BUFFER, ALPHA);

51 action( 6, ADD_TO_BUFFER, ALPHA);
action( 6, BUFFER_TO_SEQ, '/' );
action( 6, EMPTY_BUFFER, '/' );
action( 6, ADD_TO_BUFFER, '/' );
action( 6, BUFFER_TO_SEQ, '/' );
56 action( 6, EMPTY_BUFFER, '/' );

action( 7, ADD_TO_BUFFER, DIGIT);

action( 8, ADD_TO_BUFFER, DIGIT);
61 action( 8, BUFFER_TO_SEQ, ' .' );
action( 8, EMPTY_BUFFER, ' .' );
action( 8, ADD_TO_BUFFER, ' .' );
action( 8, BUFFER_TO_SEQ, ' .' );
action( 8, EMPTY_BUFFER, ' .' );

66 action( 9, ADD_TO_BUFFER, DIGIT);

action(10, ADD_TO_BUFFER, DIGIT);
action(10, BUFFER_TO_SEQ, CR);
71 action(10, EMPTY_BUFFER, CR);
action(10, ADD_TO_BUFFER, CR);

```

```

    action(11, ADD_TO_BUFFER, LF);
    action(11, BUFFER_TO_SEQ, LF);
76  action(11, EMPTY_BUFFER, LF);
}

```

C.19. pathLexer.h

```

#ifndef pathLexerH // -- c++ --
#define pathLexerH
3
#include "lexer.h"

#define unsafe CTL, SP, ' ', '#', '%', '<', '>'
#define reserved_ ' ; , ' / , ' ? , ' : , ' @ , ' & , ' = , ' + '
8 #define not_uchar reserved_ , unsafe
#define not_pchar unsafe, ' ; , ' / , ' ? '

class PathLexer: public Lexer
{
13 public:
    PathLexer();
    // Pre: -
    // Post: A PathLexer object i initialized.
};
18
#endif // pathLexerH

```

C.20. pathLexer.cpp

```

1 #include "pathLexer.h"

PathLexer::PathLexer()
{
    state( 1, 2, ' / '); // Read as: Change from state 1 to state 2 if the character is a /.
6
    state( 2, 3);
    state( 2, errorState, not_pchar);
    state( 2, 5, '%');
    state( 2, 7, ' ; ');
11 state( 2, 10, ' ? ');

    state( 3, 3);
    state( 3, errorState, not_pchar);
    state( 3, 4, ' / ');
16 state( 3, 5, '%');
    state( 3, 7, ' ; ');
    state( 3, 10, ' ? ');

    state( 4, 3);
21 state( 4, errorState, not_pchar);
    state( 4, 4, ' / ');
    state( 4, 5, '%');
    state( 4, 10, ' ? ');

```

```

26 state( 5, 6, HEX);

state( 6, 3, HEX);

state( 7, 7);
31 state( 7, errorState, not_pchar);
state( 7, 7, '/' , ' ');
state( 7, 9, '%');
state( 7, 10, '?');

36 state( 8, 7, HEX);

state( 9, 8, HEX);

state(10, 10);
41 state(10, errorState, not_uchar);
state(10, 10, reserved_);
state(10, 12, '%');

state(11, 10, HEX);
46 state(12, 11, HEX);

setStartState(1);

51 setAcceptState(2, 3, 4, 7, 10);

action( 1, ADD_TO_BUFFER, '/' ); // Read as: On state 1, ADD_TO_BUFFER,
action( 1, BUFFER_TO_SEQ, '/' ); // BUFFER_TO_SEQ,
action( 1, EMPTY_BUFFER, '/' ); // EMPTY_BUFFER, if the character is a /.

56 action( 2, ADD_TO_BUFFER);
action( 2, REMOVE_ACTIONS, not_pchar, '%', ' '; ' ', '?');
action( 2, BUFFER_TO_SEQ, '%', ' '; ' ', '?');
action( 2, EMPTY_BUFFER, '%', ' '; ' ', '?');
61 action( 2, ADD_TO_BUFFER, '%', ' '; ' ', '?');
action( 2, BUFFER_TO_SEQ, '%', ' '; ' ', '?');
action( 2, EMPTY_BUFFER, '%', ' '; ' ', '?');

action( 3, ADD_TO_BUFFER);
66 action( 3, REMOVE_ACTIONS, '/' , ' '; ' ', '?');
action( 3, BUFFER_TO_SEQ, '/' , ' '; ' ', '?');
action( 3, EMPTY_BUFFER, '/' , ' '; ' ', '?');
action( 3, ADD_TO_BUFFER, '/' , ' '; ' ', '?');
action( 3, BUFFER_TO_SEQ, '/' , ' '; ' ', '?');
71 action( 3, EMPTY_BUFFER, '/' , ' '; ' ', '?');

action( 4, ADD_TO_BUFFER);
action( 4, REMOVE_ACTIONS, not_pchar, '?', '/' );
action( 4, ADD_TO_BUFFER, '%');
76 action( 4, BUFFER_TO_SEQ, '?', '/' );
action( 4, EMPTY_BUFFER, '?', '/' );
action( 4, ADD_TO_BUFFER, '?', '/' );

```

```

    action( 4, BUFFER_TO_SEQ, '??', '/' );
    action( 4, EMPTY_BUFFER, '??', '/' );
81    action( 5, ADD_TO_BUFFER, HEX);

    action( 6, ADD_TO_BUFFER, HEX);

86    action( 7, ADD_TO_BUFFER);
    action( 7, REMOVE_ACTIONS, not_pchar, '/', ';', '?', '%');
    action( 7, ADD_TO_BUFFER, '/', ';', '%');
    action( 7, BUFFER_TO_SEQ, '??');
    action( 7, EMPTY_BUFFER, '??');
91    action( 7, ADD_TO_BUFFER, '??');
    action( 7, BUFFER_TO_SEQ, '??');
    action( 7, EMPTY_BUFFER, '??');

    action( 8, ADD_TO_BUFFER, HEX);

96    action( 9, ADD_TO_BUFFER, HEX);

    action(10, ADD_TO_BUFFER);

101    action(11, ADD_TO_BUFFER, HEX);

    action(12, ADD_TO_BUFFER, HEX);
}

```

C.21. httpheaderLexer.h

```

1  #ifndef httpheaderLexerH // -*- c++ -*-
    #define httpheaderLexerH

    #include "lexer.h"

6  class HTTPHeaderLexer: public Lexer
    {
    public:
        HTTPHeaderLexer();
        // Pre: –
11    // Post: A HTTPHeaderLexer object is initialized.
    };

    #endif // httpheaderLexerH

```

C.22. httpheaderLexer.cpp

```

1  #include "httpheaderLexer.h"

    #define specials '(, )', '<', '>', '@', ',', ';', ':', '\\', '"', '/', '[, ]', '?', '=', '{, }', SP,
        HT

    HTTPHeaderLexer::HTTPHeaderLexer()
6  {

```

```

state(1, 2); // Read as: Change from state 1 to state 2.
state(1, errorState, CTL, tspecials); // Read as: Change from state 1 to errorstate on CTL and tspecials
    characters.

state(2, 2);
11 state(2, errorState, CTL, tspecials);
state(2, 3, ' ');

state(3, 4, SP);

16 state(4, 4);
state(4, 5, CR);

state(5, 6, LF);

21 setStartState(1);
setAcceptState(6);

action(1, ADD_TO_BUFFER);

26 action(2, ADD_TO_BUFFER); // Read as: On state 2, ADD_TO_BUFFER.
action(2, REMOVE_ACTIONS, ' '); // Read as: On state 2, REMOVE_ACTIONS,
action(2, BUFFER_TO_SEQ, ' '); // BUFFER_TO_SEQ,
action(2, EMPTY_BUFFER, ' '); // EMPTY_BUFFER,
action(2, ADD_TO_BUFFER, ' '); // ADD_TO_BUFFER, if the character is a .

31 action(3, ADD_TO_BUFFER, SP);
action(3, BUFFER_TO_SEQ, SP);
action(3, EMPTY_BUFFER, SP);

36 action(4, ADD_TO_BUFFER);
action(4, REMOVE_ACTIONS, CR);
action(4, BUFFER_TO_SEQ, CR);
action(4, EMPTY_BUFFER, CR);
action(4, ADD_TO_BUFFER, CR);

41 action(5, ADD_TO_BUFFER, LF);
action(5, BUFFER_TO_SEQ, LF);
action(5, EMPTY_BUFFER, LF);
}

```

C.23. hostheaderLexer.h

```

#ifndef hostheaderLexerH // -*- c++ -*-
#define hostheaderLexerH

#include "lexer.h"

5 class HostheaderLexer: public Lexer
{
    public:
        HostheaderLexer();
10 // Pre: -

```

```

    // Post: a HostheaderLexer object has been initialized
};

#endif // hostheaderLexerH

```

C.24. hostheaderLexer.cpp

```

1 #include "hostheaderLexer.h"

HostheaderLexer::HostheaderLexer()
{
    state(1, 1, ALPHANUM, ' ', ' ');
6    state(1, 2, ' : ');
    state(2, 2, DIGIT);

    setStartState(1);
    setAcceptState(1, 2);

11    action(1, ADD_TO_BUFFER, ALPHANUM, ' ', ' ');
    action(1, BUFFER_TO_SEQ, ' : ');
    action(1, EMPTY_BUFFER, ' ');
    action(1, ADD_TO_BUFFER, ' ');
16    action(1, BUFFER_TO_SEQ, ' : ');
    action(1, EMPTY_BUFFER, ' ');
    action(2, ADD_TO_BUFFER, DIGIT);
}

```

C.25. HTTPVersion.h

```

1 #ifndef HTTPVersionH
#define HTTPVersionH

class HTTPVersion
{
6 public:

    HTTPVersion();
    HTTPVersion(const HTTPVersion& v);

11 int getMajor();
int getMinor();
    // pre. og post. for both get-functions
    // Pre: -
    // Post: the private variable 'major' or 'minor' is set

16 void setMajor(int major_);
void setMinor(int minor_);
    // pre. og post. for both set-functions
    // Pre: -
21 // Post: the private variable 'major' or 'minor' is returned

    // Overloaded functions
friend bool operator <(const HTTPVersion& v1, const HTTPVersion& v2);

```

```

// Pre: two objects has been created and their respective variables
26 // has been set
// Post: true is returned if v1 is smaller than v2, and false if otherwise

friend bool operator >(const HTTPVersion& v1, const HTTPVersion& v2);
// Pre: two objects has been created and their respective variables
31 // has been set
// Post: true is returned if v1 is larger than v2, and false if otherwise

friend bool operator ==(const HTTPVersion& v1, const HTTPVersion& v2);
// Pre: two objects has been created and their respective variables
36 // has been set
// Post: true is returned if v1 is equal to v2, and false if otherwise

private:

41 int major;
int minor;

};

46 #endif

```

C.26. HTTPVersion.cpp

```

#include "HTTPVersion.h"

4 HTTPVersion::HTTPVersion()
{
    //empty
}

9 HTTPVersion::HTTPVersion(const HTTPVersion& v)
{
    major = v.major;
    minor = v.minor;
}

14 /*
HTTPVersion::HTTPVersion(int major_, int minor_)
{
19     setMajor(major_);
    setMinor(minor_);
}
*/

int HTTPVersion::getMajor()
24 {
    return major;
}

int HTTPVersion::getMinor()

```

```

29 {
    return minor;
}

void HTTPVersion::setMajor(int major_)
34 {
    major = major_;
}

void HTTPVersion::setMinor(int minor_)
39 {
    minor = minor_;
}

bool operator <(const HTTPVersion& v1, const HTTPVersion& v2)
44 {
    if(v1.major < v2.major)
        return true;
    else if(v1.major == v2.major && v1.minor < v2.minor)
        return true;
49     return false;
}

bool operator >(const HTTPVersion& v1, const HTTPVersion& v2)
{
54     if(v1.major > v2.major)
        return true;
    else if(v1.major == v2.major && v1.minor > v2.minor)
        return true;
59     return false;
}

bool operator ==(const HTTPVersion& v1, const HTTPVersion& v2)
{
64     return (v1.major == v2.major && v1.minor == v2.minor);
}

```

C.27. URL.h

```

1  #ifndef URLH // -*- c++ -*-
    #define URLH

    #include <string>
    #include <vector>

6   class URL
    {
    public:
        URL();
11     // URL(std::string url_);

        void setURL(std::string url_);
        // Pre: -

```

```

16 // Post: an url has been placed in the variable url
std::string getURL();
// Pre: –
// Post: url is returned

21 std::string getPath();
// Pre: the path has been initialized
// Post: path is returned

void setHost(std::string host_);
26 // Pre: –
// Post: the host part of the url is set (so is the port if it is
// part of host_)

std::string getHost();
31 // Pre: –
// Post: the host part of the url is returned

void setPort(int port_);
// Pre: –
36 // Post: the port part of the url is set

int getPort();
// Pre: –
// Post: the port part of the url is returned
41

std::string getParam(const int i);
// Pre: –
// Post: the i'th param of the url

46 int paramSize();
// Pre: –
// Post: return the number of params in the url

std::string getPathSegment(const int i);
51 // Pre: –
// Post: the i'th segment of the path

int pathSegmentsSize();
// Pre: –
56 // Post: return the number of segments in the path

std::string getQuery();
// Pre: –
// Post: the query part of the url is returned. url is empty ("")
61 // if no query is present

std::string getFilename();
// Pre: –
// Post: filename is returned

66 std::string getPathinfo();
// Pre: –

```

```

// Post: pathinfo is returned

71  std::string getTranslatedPath();
    // Pre: –
    // Post: translated_path is returned

private:
76  void resolveFilename();
    // Pre: –
    // Post: the filename is resolved

    std::string uri_path;
81  std::vector<std::string> param;
    std::vector<std::string> pathSegments;
    std::string query;
    std::string host;
    std::string filename;
86  std::string pathinfo;
    std::string translated_path; // CGI/1.1
    int port;
};

91 #endif

```

C.28. URL.cpp

```

#include "URL.h"
#include <string>
#include <vector>
4  #include "tools.h"
#include <unistd.h>
#include <sys/param.h>
#include "pathLexer.h"
#include "hostheaderLexer.h"
9  #include "RequestException.h"

URL::URL()
{
    char c[255];
14  host = gethostname(c, 255);
    query = "";

    port = 80; // bad default, but best information avail at the moment
}

19 void URL::setURL(std::string url_)
{
    PathLexer pathLexer;
    std::vector<std::string> symseq;

24  uri_path = url_;

    if ( !pathLexer.lex(url_, symseq) )

```

```

        throw RequestException();
29
    int i=0;

    while (i < symseq.size() && symseq[i] != ";" && symseq[i] != "?")
        pathSegments.push_back(symseq[i++]);
34
    while (i < symseq.size() && symseq[i] != "?")
    {
        if (symseq[i] != ";")
            param.push_back(symseq[i]);
39        i++;
    }

    if (i+1 < symseq.size())
        query = symseq[i+1];
44
    resolveFilename();
}

std::string URL::getPath()
49 {
    std::string path = "";

    if (pathSegments.size() == 0)
        path = "/";
54    for (int i=0; i < pathSegments.size(); i++)
        path += pathSegments[i];

    return unescapeHex(path);
}
59
std::string URL::getURL()
{
    std::string s = "http://";
    if ( port != 80 )
64    {
        s += getHost();
        s += ":" + itos(getPort());
    }
    s += uri_path;
69    for (int i=0; i < param.size(); i++)
        s += ";" + param[i];
    if (query != "")
        s += "?" + query;

74    return s;
}

void URL::setHost(std::string host_)
{
79    HostheaderLexer hostLexer;
    std::vector<std::string> symseq;

```

```

    hostLexer.lex(host_, symseq);
84  if (symseq.size() == 1)
    host = symseq[0];
    else if (symseq.size() == 3)
    {
        host = symseq[0];
89    port = stoi(symseq[2]);
    }
    else
    host = host_;
}
94 std::string URL::getHost()
{
    return host;
}
99 void URL::setPort(int port_)
{
    port = port_;
}
104 int URL::getPort()
{
    return port;
}
109 std::string URL::getParam(const int i)
{
    if (i < param.size())
        return unescapeHex(param[i]);
114    else
        return "";
}

int URL::paramSize()
119 {
    return param.size();
}

std::string URL::getPathSegment(const int i)
124 {
    if (i < pathSegments.size())
        return unescapeHex(pathSegments[i]);
    else
        return "";
129 }

int URL::pathSegmentsSize()
{
    return pathSegments.size();
134 }

```

```

std::string URL::getQuery()
{
    return unescapeHex(query);
139 }

std::string URL::getFilename()
{
    return filename;
144 }

std::string URL::getPathinfo()
{
    return pathinfo;
149 }

std::string URL::getTranslatedPath()
{
    return translated_path;
154 }

void URL::resolveFilename()
{
    // find path parts for file access and CGI-stuff
159 std::string docroot;

#ifdef DOCUMENTROOT
    docroot = DOCUMENTROOT ;
#else
164 char c[MAXPATHLEN];
    getcwd(c, MAXPATHLEN);
    docroot = c;
#endif // DOCUMENTROOT
    filename = docroot;
169 pathinfo = "";

    int j=0;

    std::string elem;
174 while ( ! isFile(filename) && j < pathSegments.size() )
    {
        elem = unescapeHex(pathSegments[j]);
        // if the path segment is a '.' ignore it
        if ( elem == "." )
179 ;
        // if the path segment is a '/' and current filename
        // ends in a '/' ignore it
        else if ( elem == "/" && filename[filename.size()-1] == '/' )
            ;
184 // if we have reached this far and we have a '..' we are
        // moving out of our document root
        else if ( elem == ".." )
            { j++; break;}
        // otherwise add it unless the next next path segment is '..'
189 else if ( j+2 <= pathSegments.size()-1 && unescapeHex(pathSegments[j+2]) == ".." )

```

```

        j += 2;
        // everything seems fine – just add it!
        else
            filename += elem;
194     j++;
    }

    for ( ; j < pathSegments.size(); j++)
        pathinfo += pathSegments[j];
199     translated_path = docroot + pathinfo;

    if ( ! isFile(filename) && ! isDir(filename) )
    {
204     filename += pathinfo;
        pathinfo = "";
    }
}

```

C.29. HTTPHeaders.h

```

#ifndef HTTPHeadersH // –*- c++ –*-
#define HTTPHeadersH
3
#include <vector>
#include <string>

struct headerPair
8 {
    std::string name;
    std::string content;
};

13 class HTTPHeaders
{
    public:
        HTTPHeaders();
        // Pre: –
18     // Post: An HTTPHeaders object is initialized.

        headerPair setHeader(std::string headerLine);
        // Pre: headerLine is received from the client.
        // Post: A headerPair is set – name and content have got values.
23
        bool hasHeader(const std::string headerName_);
        // Pre: headerName_ is the headerPair name that may exist.
        // Post: Return true if headerName_ is a name in a headerPair, otherwise false.

28     bool getHeader(const std::string headerName_, std::string& headerContent_);
        // Pre: headerName_ is the headerPair name that may exist.
        // Post: If headerName_ exists, true was returned and headerContent_ has got a value,
        // otherwise false is returned.

```

```

33  int getNumberOfHeaders();
    // Pre: –
    // Post: The amount of headers is returned.

    headerPair operator[](int n);
38  // Pre: n is the n'th header in a vector you want to receive.
    // Post: The n'th header is returned as an object.

    private:
        std::vector<headerPair> headers;
43 };

#endif // HTTPHeadersH

```

C.30. HTTPHeaders.cpp

```

#include "HTTPHeaders.h"
#include "httpheaderLexer.h"
#include <string>

5  HTTPHeaders::HTTPHeaders()
    {
        // intentionally left blank
    }

10 headerPair HTTPHeaders::setHeader(std::string headerLine_)
    {
        std::vector<std::string> symseq;
        HTTPHeaderLexer lexer;

15  if (!lexer.lex(headerLine_, symseq))
        {
            struct headerPair header = { "", "" }; //name and content is empty.
            return header;
        }

20  struct headerPair header = { symseq[0], symseq[2] };

        headers.push_back(header);

25  return header;
    }

bool HTTPHeaders::hasHeader(const std::string headerName_)
    {
30  for (int i=0; i < headers.size(); i++)
        if (headers[i].name == headerName_)
            return true;
        return false;
    }

35 bool HTTPHeaders::getHeader(const std::string headerName_, std::string& headerContent_)
    {

```

```

    headerContent_ = "";
    for (int i=0; i < headers.size(); i++)
40     if (headers[i].name == headerName_)
        {
            headerContent_ = headers[i].content;
            return true;
        }
45     return false;
}

int HTTPHeaders::getNumberOfHeaders()
{
50     return headers.size();
}

headerPair HTTPHeaders::operator[](int n)
{
55     return headers[n];
}

```

C.31. mimetype.h

```

#ifndef mimetypeH
#define mimetypeH

4  #include <string>
   #include "lexer.h"

class MimeType
{
9   public:
    MimeType();
    // Pre: –
    // Post: a MimeType object has been initialized

14    std::string getMimeType(std::string path_);
    // Pre: the request from the client has a path_
    // Post: mimeType is returned

    void addMimeType(std::string ext_, std::string type_);
19    // Pre: ext_ is the extensions type and type_ is the type ex. ("txt" , "text/plain")
    // Post: the mimeType is added to the vector of types

   private:

24    struct extensiontype
        {
            std::string extension;
            std::string type;
29    };

    std::vector<struct extensiontype> types;

```

```

    Lexer mimeLexer;

34 };

#endif

C.32. mimetype.cpp

#include "mimetype.h"

3 MimeType::MimeType()
{
    // definer lexer
    mimeLexer.state(1,1);
    mimeLexer.state(1,2,' ');

8
    mimeLexer.state(2,2);
    mimeLexer.state(2,1,'/ ');

    mimeLexer.setStartState(1);
13 mimeLexer.setAcceptState(2);

    mimeLexer.action(1, ADD_TO_BUFFER, ' ');

    mimeLexer.action(2, ADD_TO_BUFFER);
18 mimeLexer.action(2, REMOVE_ACTIONS, ' ', '/ ');
    mimeLexer.action(2, EMPTY_BUFFER, ' ', '/ ');
    mimeLexer.action(2, ADD_TO_BUFFER, ' ');

23 // add mimetypes
    addMimeType(".pdf", "application/pdf");
    addMimeType(".txt", "text/plain");
    addMimeType(".cpp", "text/plain");
    addMimeType(".h", "text/plain");
28 addMimeType(".htm", "text/html");
    addMimeType(".html", "text/html");
    addMimeType(".vsd", "application/vnd.visio");
    addMimeType(".gif", "image/gif");
    addMimeType(".jpg", "image/jpeg");
33 }

std::string MimeType::getMimeType(std::string path_)
{
    std::vector<std::string> symSeq;
38 std::string file;
    bool acceptState;
    acceptState = mimeLexer.lex(path_, symSeq);
    if(acceptState) // file has an extension
    {
43     file = symSeq[symSeq.size()-1];
        for(int i = 0; i < types.size(); i++)
        {

```

```

        if(types[i].extension == file)
            return types[i].type;
48     }
    }
    // file with no extension or extension unknown
    return "application/octet-stream";
}

53 void MimeType::addMimeType(std::string ext_, std::string type_)
{
    struct extensiontype t;
    t.extension = ext_;
58   t.type = type_;
    types.push_back(t);
}

```

C.33. statusCode.h

```

#ifndef statusCodeH
#define statusCodeH

4  #include <string>

class StatusCode
{
public:
9   StatusCode();
    // Pre: –
    // Post: An object is created.

    std::string getReasonPhrase();
14  // Pre: An error has occurred.
    // Post: A Status–Code is returned as text.

    std::string getEntityBody();
    // Pre: An error has occurred.
19  // Post: An entityBody with Status–Code description is returned as text/HTML.

    void setStatus(int status_);
    // Pre: status_ is the statuscode that will be set.
    // Post: status has been given value

24  int getStatus();
    // Pre: –
    // Post: status is returned

29 private:
    int status;
};
#endif

```

C.34. statusCode.cpp

```

#include "statusCode.h"

3  StatusCode::StatusCode()
   {
     status = 500;
   }

8  std::string StatusCode::getReasonPhrase()
   {
     std::string statusText;

     switch(status)
     {
13      case 100: // internal pseudo return code
           statusText = "100 This was a CGI-script";
           break;

           case 200:
18             statusText = "200 OK";
             break;

           case 201:
             statusText = "201 Created";
             break;

23          case 202:
             statusText = "202 Accepted";
             break;

           case 204:
             statusText = "204 No Content";
             break;

28          case 301:
             statusText = "301 Moved Permanently";
             break;

           case 302:
33             statusText = "302 Moved Temporarily";
             break;

           case 304:
             statusText = "304 Not Modified";
             break;

38          case 400:
             statusText = "400 Bad Request";
             break;

           case 401:
             statusText = "401 Unauthorized";
             break;

43          case 403:
             statusText = "403 Forbidden";
             break;

           case 404:
48             statusText = "404 Not Found";
             break;

           case 501:
             statusText = "501 Not Implemented";
             break;

53          case 502:
             statusText = "502 Bad Gateway";

```

```

        break;
    case 503:
        statusText = "503 Service Unavailable";
        break;
58     default:
        statusText = "500 Internal Server Error";
        break;
    }
63     return statusText;
}

std::string StatusCode::getEntityBody()
{
68     std::string bodyText;

    bodyText += "<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict //EN\n";
    bodyText += " \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\">\n";
    bodyText += "<html xmlns=\"http://www.w3.org/1999/xhtml\">\n";
73     bodyText += "<head>\n";
    bodyText += "<title>";
    bodyText += getReasonPhrase();
    bodyText += "</title>\n";
    bodyText += "</head>\n"; \
78     bodyText += "<body>\n";
    bodyText += "<h1>";
    bodyText += getReasonPhrase();
    bodyText += "</h1>\n";
    bodyText += "<p>\n";
83     bodyText += "bla...bla...bla...\n";
    bodyText += "</p>\n";

#ifdef IE_ERROR
    // MS Internet Explorer show its own error page if this page
88     // is less than 512 bytes, so we write a lot of bytes here to
    // see our own page
    bodyText += "<!-- \n";
    for (int i=0; i < 512; i++)
        bodyText += "X";
93     bodyText += "\n-->\n";
#endif // IE_ERROR

    bodyText += "</body>\n";
    bodyText += "</html>\n";
98

    return bodyText;
}

void StatusCode::setStatus(int status_)
103 {
    status = status_;
}

int StatusCode::getStatus()
108 {

```

```
    return status;
}
```

C.35. lerror.h

```
#ifndef lerrorH
#define lerrorH

#include <string>
5 #include <pthread.h>

/* Please note that this class is designed using the singleton pattern
   and this class is not thread safe during creation of the first (and
   only) instance of the class.
10
   So remember to call this function for the first time before
   threading occurs.
*/

15 class ErrorLog
{
    public:
        static void Log(std::string msg);
        // Pre: msg is the message that is to be written in the logfile.
20        // Post: An error message has been written to the logfile – "internal.log".

    private:
        ErrorLog();
        // Pre: –
25        // Post: An ErrorLog object is initialized.

        void log(std::string msg);
        // Pre: msg is the message that is to be written in the logfile.
        // Post: An error message has been written to the logfile – "internal.log".
30
        static ErrorLog *pinstance;
        pthread_mutex_t lerror_mutex;
};

35 #endif // lerrorH
```

C.36. lerror.cpp

```
#include "lerror.h"
#include <stdio.h>
#include "tools.h"

5 ErrorLog* ErrorLog::pinstance = NULL;

void ErrorLog::Log(std::string msg)
{
    if (!pinstance)
10     pinstance = new ErrorLog;
```

```

    pinstance->log(msg);
}

ErrorLog::ErrorLog()
15 {
    //pthread_mutex_init always returns 0.
    pthread_mutex_init(&lerror_mutex, NULL);
}

20 void ErrorLog::log(std::string msg)
{
    pthread_mutex_lock(&lerror_mutex);
#ifndef INTERNALLOG
    FILE *fp = fopen(INTERNALLOG, "a");
25 #else
    FILE *fp = fopen("internal.log", "a");
#endif

    std::string logText = (rfcdate() + ": " + msg + "\n");
30
    for(int i = 0; i < logText.size(); i++)
    {
        fprintf(fp, "%c", logText.c_str()[i]);
    }
35
    fflush(fp);
    fclose(fp);
    pthread_mutex_unlock(&lerror_mutex);
}

```

C.37. mutex.h

```

1 #ifndef mutexH // -*- c++ -*-
#define mutexH

#include <pthread.h>

6 enum mutex_id {
    STAT_MUTEX, PUT_MUTEX,
    INTERNAL_MUTEX // this must be the last listed mutex
};

11 class Mutex
{
    public:
    static Mutex* Instance();
    // Pre: -
16 // Post: Returns pinstance.

    static bool Lock(mutex_id id);
    // Pre: -
    // Post: Returns true if lock succeeded otherwise false.
21

```

```

    static bool Unlock(mutex_id id);
    // Pre: –
    // Post: Returns true if unlock succeeded otherwise false.

26 private:
    Mutex();
    // Pre: –
    // Post: Initialize mutex. Write to error log if initialize mutex failed.

31 static Mutex* pinstance;
    pthread_mutex_t mutex_list[2];
};

#endif // mutexH

```

C.38. mutex.cpp

```

#include "mutex.h"
#include "lerror.h"
#include "tools.h"

5 Mutex* Mutex::pinstance = NULL;

Mutex::Mutex()
{
    pthread_mutex_t m;
10 for (int i=0; i < INTERNAL_MUTEX; i++)
    {
        if (pthread_mutex_init(&m, NULL) != 0)
            ErrorLog::Log("Unable to initialize Mutex(" + itos(i) + ")");
        mutex_list[i] = m;
15 }
}

Mutex* Mutex::Instance()
{
20 if (!pinstance)
    pinstance = new Mutex;
    return pinstance;
}

25 bool Mutex::Lock(mutex_id mutex)
{
    if (pinstance)
        if (pthread_mutex_lock(&(pinstance->mutex_list[mutex])) == 0)
            return true;
30 return false;
}

bool Mutex::Unlock(mutex_id mutex)
{
35 if (pinstance)
    if (pthread_mutex_unlock(&(pinstance->mutex_list[mutex])) == 0)

```

```
    return true;
    return false;
}
```

C.39. tools.h

```
1  #ifndef toolsH
   #define toolsH

   #include <string>
   #include <cstdlib>

6
   std::string itos(int i);
   // Pre: –
   // Post: returns a string with textual representation of i

11  int stoi(std::string s);
   // Pre: s is a string consisting of digits
   // Post: the decimal version of the digits are returned

   int htoi(std::string s);
16  // Pre: s is a string consisting of hexadecimal digits
   // Post: returns the decimal version of the number and –1 on error

   std::string rfcdate();

21  std::string unescapeHex(std::string s1);

   bool isFile(std::string path_);
   // Pre: Path_ is the full path.
   // Post: Return true if it was a regular file otherwise false.

26  bool isDir(std::string path_);
   // Pre: Path_ is the full path.
   // Post: Return true if it was a directory file otherwise false.

31  bool isReadable(std::string path_);
   // Pre: Path_ is the full path.
   // Post: Returns true if the file asked for is readable otherwise false.

   bool isWritable(std::string path_);
36  // Pre: Path_ is the full path.
   // Post: Returns true if the file asked for is writable otherwise false.

   bool isExecutable(std::string path_);
   // Pre: Path_ is the full path.
41  // Post: Returns true if the file asked for is executable otherwise false.

   #endif
```

C.40. tools.cpp

```

1  #include "tools.h"
   #include <sys/types.h>
   #include <sys/stat.h>
   #include <fcntl.h>
   #include <unistd.h>
6  #include <stdlib.h>
   #include <limits.h>
   #include <time.h>

std::string itos(int i)
11 {
    std::string s;
    int j = abs(i);
    int isize = 2;
    for (; j >= 1; isize++)
16     j /= 10;
    char *c = (char*)malloc( isize );
    snprintf(c, isize, "%d", i);
    s = std::string(c);
    free(c);
21    return s;
}

int stoi(std::string s)
{
26    return strtol(s.c_str(), NULL, 10);
}

int htoi(std::string s)
{
31    return strtol(s.c_str(), NULL, 16);
}

std::string rfcdate()
{
36    char c[50];
    time_t t = time(NULL);
    struct tm *mytm = gmtime(&t);

    strftime(c, 50, "%a, %d %b %Y %T %Z", mytm);
41
    return std::string(c);
}

std::string unescapeHex(std::string s1)
46 {
    std::string s2 = "";
    for (int i=0; i < s1.length(); i++)
        {
            if (s1[i] == '%' && s1.length() >= i+2)
51                {
                    s2 += htoi( s1.substr(++i, 2) );
                    i++;
                }
        }
}

```

```

        else
56     s2 += s1[i];
    }
    return s2;
}

61 bool isFile(std::string path_)
{
    struct stat buffer;
    stat(path_.c_str(), &buffer);
    return(S_ISREG(buffer.st_mode));
66 }

bool isDir(std::string path_)
{
    struct stat buffer;
71     stat(path_.c_str(), &buffer);
    return(S_ISDIR(buffer.st_mode));
}

bool isReadable(std::string path_)
76 {
    struct stat buffer;
    stat(path_.c_str(), &buffer);
    return(buffer.st_mode & S_IRUSR);
}

81 bool isWritable(std::string path_)
{
    struct stat buffer;
    stat(path_.c_str(), &buffer);
86     return(buffer.st_mode & S_IWUSR);
}

bool isExecutable(std::string path_)
{
91     struct stat buffer;
    stat(path_.c_str(), &buffer);
    return(buffer.st_mode & S_IXUSR);
}

```

C.41. BrokenConnectionException.h

```

#ifndef brokenconnectionexceptionH
#define brokenconnectionexceptionH

class BrokenConnectionException
5 {
public:
    BrokenConnectionException() {}
};

10 #endif

```

C.42. RequestException.h

```
#ifndef brokenconnectionexceptionH
#define brokenconnectionexceptionH

class BrokenConnectionException
5 {
  public:
    BrokenConnectionException() {}
};

10 #endif
```

Litteratur

- [1] Tim Berners-Lee, Roy T. Fielding og Henrik Frystyk Nielsen. *Hypertext Transfer Protocol – HTTP/1.0*, maj 1996. RFC 1945. URL <http://www.ietf.org/rfc/rfc1945.txt>.
- [2] Tim Berners-Lee, Larry Masinter og Mark McCahill. *Uniform Resource Locators (URL)*, december 1994. RFC 1738. URL <http://www.ietf.org/rfc/rfc1738.txt>.
- [3] Ken A. L. Coar og David R. T. Robinson. *The WWW Common Gateway Interface – Version 1.1*, juni 1999. Draft 3. URL <http://cgi-spec.golux.com/draft-coar-cgi-v11-03.txt>.
- [4] Ned Freed og Nathaniel S. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, november 1996. RFC 2045. URL <http://www.ietf.org/rfc/rfc2045.txt>.
- [5] Ned Freed og Nathaniel S. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, november 1996. RFC 2046. URL <http://www.ietf.org/rfc/rfc2046.txt>.
- [6] Brian Hall. *Beej's Guide to Network Programming*, 2001. URL <http://www.ecst.csuchico.edu/~beej/guide/net/>.
- [7] Uffe Kofod. *Language Theory*, januar 2003. Publiceret på Brock Online.
- [8] Bruce Molay. *Understanding Unix/Linux Programming*. Prentice Hall, 2003. ISBN 0-13-008396-8.