Prosjektoppgave

Milepæl 1 [2019-01-10 to.] `oblig`

Dere skal i første milepæl programmere en webtjener med programmeringspråket C. Ta gjerne utgangspunkt i eksemplet hallotjener.

Funksjonskrav

1. Tjeneren skal leverere "as-is"-filer. Det vil si at filene inneholder både http-hode og http-kropp, og sendes "som de er", uten å legge til eller trekke fra noe. Enklere blir det ikke: Tjeneren åpner fila og skriver den til til socket'en som er forbundet med klienten.

   Disse filene skal ende på `.asis` og skal inneholde både *http-hode* og *http-kropp*, som i eksemplet under.

   ./eksempler/index.asis

   ```
   HTTP/1.1 200 OK

   Content-Type: text/plain



   Hallo verden :-)
   ```

2. Dersom en fil som ikke finnes blir forspurt, skal en korrekt *http-feilmeling* retureres til klienten.
3. Logg til STDERR: Åpne en fil (f.eks. `/var/webtjener/error.log`) og koble den til *stderr*, slik at du kan få skrevet ut feil- og avlusnings-info.

Sikkerhets-/robusthets-krav

1. Programmet skal ikke binde seg til porten en gang pr. forespørsel, men kun ved programstart.
2. Hver klient-forsepørsel skal behandles i en egen tråd eller prosess.
3. Tjeneren skal være demonisert slik at den er uavhengig av kontroll-terminal.
4. Tjeneren den lytte på *port 80* uten at den kjører som brukeren *root*.
5. Tjeneren skal kjøre i en *docker-kontainer* med følgende krav:
   - `scratch` skal brukes som grunn-bilde. (Under utvikling kan det være gode grunner til å bruke et annet bilde for lettere å kunne gjøre feilsøking)
   - Katalogen som inneholder filene som skal være tilgjengelige via http *web root*, skal befinne seg på et *docker-volum* montert

på `/var/www` i containeren. Tjener-prosessene skal også (ved hjelp av `chroot(2)`) ha endret sin *rot-katalog* til denne denne webroten..
- root-brukeren i kontaineren er en upriveligert i vertssystemet (ved hjelp av `namespaces(7)`).
- Prosessorbruken til kontaineren skal begrenses med `cgroups(7)`
- Sikkerheten skal økes ved hjelp av (1) `capabilities(7)`.

Tips

Forhindre zombier Se [how-can-i-prevent-zombie-child-processes]

Tips 3: Busybox-static

Legg til busybox-static for debugging. F.eks. slik:

```
FROM scratch

COPY ./webtjener /bin/webtjener

COPY ./busybox   /bin/sh

EXPOSE 80

CMD ["/bin/sh"]
```

Tips 4: Busybox-blide [2018-01-11 to.]

Det offisielle busybox-bildet fra Docker inc. kan også brukes til testing.
Se https://hub.docker.com/_/busybox/

Milepæl 2 `oblig`

I denne fasen skal du utvide http-tjeneren fra milepæl 1:

1. La tjeneren i tillegg til as-is-filer, levere alle filer av typene
   - `text/html`,
   - `text/plain`,
   - `image/png`,
   - `image/svg`,
   - `application/xml`,
   - `application/xslt+xml` og
   - `text/css`
   - `application/javascript`
   - `application/json`

Identifiseringen av type skal gjøres ved hjelp av filendelsen. Eksempel: Hvis fila ender på `.txt` så er det av typen `text/plain`. Sammenheng mellom mime-typer og filendelser finner dere i filen `/etc/mime.types`.

Dersom det kommer en http-forspørsel etter en fil som *ikke finnes*, eller til en fil av en *type som ikke støttes*, så skal korrekt feilmelding returneres til klienten.

2. Tjeneren skal, som respons på `GET / HTTP/1.0` (eller tilsvarende for http 1.1), svare med en HTML-side som:
    - er "stylet" med CSS
    - inneholder minst ett bilde
    - inneholder en *dynamisk katalog-listing av webroten*.

    Krav til katalog-listingen:

    - For hver fil skal *filrettigheter*, *UID*, *GID* og *filnavn* vises (slik som i eksempelkoden under).
    - Filnavnene skal være hyperlenket slik at filene vises når bruker klikker på dem.

    Bruk gjerne funksjonen `kataloglisting()` fra eksemplet under, som utgangspunkt.

    ./eksempler/kataloglisting.c

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

#include <sys/stat.h>

#include <dirent.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>


void kataloglisting(char *filsti);


int main (int argc, char **argv) {
```

```
  if (argc > 1)
    kataloglisting( argv[1]       );

  else
    kataloglisting( getenv("PWD") );

  return 0;
}

void kataloglisting(char *filsti){

  struct stat       stat_buffer;
  struct dirent    *ent;
  DIR              *dir;

  if ((dir = opendir (filsti)) == NULL) {
    perror (""); exit(1); }

  chdir(filsti);

  printf ("\nKatalogen %s:\n", filsti);
  printf ("-----------------------------------\n");
  printf ("Rettigheter\tUID\tGID\tNavn\n");
  printf ("-----------------------------------\n");

  while ((ent = readdir (dir)) != NULL) {
```

```
    if (stat (ent->d_name, &stat_buffer) < 0) {
      perror(""); exit(2); }

    printf ("%o\t\t", stat_buffer.st_mode & 0777 );
    printf ("%d\t",   stat_buffer.st_uid);
    printf ("%d\t",   stat_buffer.st_gid);
    printf ("%s\n",   ent->d_name);
  }

  closedir (dir);
}
```

Milepæl 3 [2019-01-21 ma.] `oblig`

1. Lag en kontainer basert på det offisielle docker-bildet httpd:alpine.
2. Sett det opp slik at det støtter CGI
3. Lag et CGI-skript som kjører på denne tjeneren og behandler både POST og GET forspørsler.
4. Lag et eller to html-sider for å teste dette (både POST og GET). Disse skal leveres fra tjeneren dere har jobbet med i milepæl 1 og 2.

Milepæl 4 [2019-01-21 ma.] `oblig`

1. Lag en tredje kontainer som implementerer et REST-API ved hjelp av Express.js mot en en sqlite-database/ med følgende skjema:
   - Bruker (brukerID , passordhash, fornavn, etternavn)
   - Sesjon (sesjonsID , brukerID*)
   - Forfatter (forfatterID, fornavn, etternavn, nasjonalitet)
   - Bok (bokID , tittel, forfatterID*)

   For skjemaet, kan dere ta utgangspunkt i ./eksempler/bokbase.sql.

   Med API'et skal det være mulig å

   - logge inn (sjekke passord mod brukerID, og sette sesjsonsID)
   - logge ut (slette sesjonsID for innlogget bruker)
   - dersom brukeren er logget inn, er det mulig å gjøre følgende i Bok og Forfatter:

        - hente ut en post
        - hente alle poster i en tabell,
        - legge til post,
        - endre en post og
        - slette en post
        - slette alle poster i en tabell
    - dersom brukeren ikke er logget inn skal det *bare* være mulig å gjøre leseoperasjonene i Bok og Forfatter – *ikke* gjøre endringer.

Datautveksling mellom tjener og klient *skal være på XML-format*. Dette gjelder *både HTTP-forespørsel* og *HTTP-respons*.

Tips:

- Siden node-modulen `sqlite3` spytter data ut i json-format, kan dere enkelt konvertere dette med `js2xmlparser.parse()`.
  (Se https://www.npmjs.com/package/js2xmlparser)
    - Eksempel: ./eksempler/json2xml.js

```
#!/usr/bin/node

var js2xmlparser = require("js2xmlparser")

var sqlite3      = require('sqlite3').verbose()

var db           = new
sqlite3.Database('bokbase.db')

var sql          = "SELECT * FROM forfatter"


db.all(

    sql, (err, result) => {

  if (err)    console.log (err)

  if (result) console.log (

      js2xmlparser.parse("forfatter", result)

  );

    }

)f
```

Responsen skal referere til et dokument som definerer hva som er gyldig. (DTD eller XML schema). Dette dokumentet skal

- reflektere database-skjemaet.
- leveres av web-tjeneren (fra milepæl 2) som en separat fil med korrekt mime-type.

Lag også et shellscript som tester API-et. Tips:

- Bruk curl(1) for å sende http-forespørslser til tjeneren.

2. Lag et cgi-skript, som kjører på httpd:alpine-tjeneren fra milepæl 3 og som lar en bruker
   - logge ut
   - logge inn
   - opprette nye, endre og slette bøker og forfattter (dersom brukeren er innlogget)
   - kunne lese data fra databasen om brukeren ikke er innlogget.

   Ta gjerne utgangspunt i shell-skriptet du brukte for teste api-et.

   Tips:

   - xmlstarlet(1) kan brukes til å søke i xml-dokumenter (se eksempel: xmlstarlet).

Referanse: https://www.npmjs.com/package/js2xmlparser (Lenker til en ekstern side.)Lenker til en ekstern side.

Milepæl 5 [2019-01-21 ma.] [2019-03-08 fr.] `oblig`

Dere skal nå lage en Ajax-klient som

- gir brukeren samme funksjonalitet som CGI-skriptet fra milepæl 4
- leveres av tjeneren fra milepæl 1 og 2 **eller fra "express-tjeneren" fra milepæl 4**. Det sistnevnte alternativet er enklest å implementere, fordi klient og tjener da får samme opphav (same origin).

Tips: Hvis dere møter på problemer med "same origin policy", kan lønne seg å ta en titt på denne:

- https://stackoverflow.com/questions/19673450/xmlhttprequest-same-origin-policy/19766649#19766649

Milepæl 6 [2019-02-11 ma.][2019-03-01 fr.][2019-03-05 ti.] [2019-03-08 fr.] `oblig`

Lag en *webapplikasjon* for android som

- har et *web-basert brukergrensensnitt* basert på milepæl 5
- viser brukerens fulle navn mens brukeren er logget inn.

~~Filene som definerer brukergrensesnittet (*.html, *.js og *.css) skal følge med i *apk*-fila. (For å få til dette må dere omgå sikkerhets-policyen *same-origin*.)~~

Det skal være oppstart-startside som følger med i *apk*-fila. Resten av filene som definerer brukergrensesnittet (*.html, *.js og *.css) skal ligge på tjeneren fra milepæl 1 og 2 **eller på "express-tjeneren" fra milepæl 4**.

Brukerens fulle navn skal *hentes fra androidenhetens kontaktliste* basert på `Bruker.brukerID`. Anta at bruker-id på tjeneren er lik e-postadressen (eller "nick") i kontakt-databasen på android-enheten.

Husk å deklarere hvilke spesialrettigheter appen må ha i *AndroidManifest.xml*. Denne appen må kunne lese fra kontakt-databasen og hente data fra internett.

Tips: Et sammensatt eksempel

Struktur på innlevering

For at vi skal kunne holde tidsskjemaet ved gjennomføring av eksamen vil jeg at dere alle organiserer innleveringen på følgende måte:

- Lag *én mappe pr. kontainer*, og *én for android*. Kall mappene: *c*, *apache*, *express* og *android*.
- Mappestrukturen pakkes i én fil (-.zip, -.tar.gz, -.rar).

Innhold i mappene bør i hovedsak være som følger:

c

- Dockerfile
- C-kode
- kommando for å starte kontaineren (f.eks. i et shell-skript eller makefile)
- de statiske filene i web-rot-katalogen

apache

- Dockerfile
- httpd.conf
- cgi-scripts

express

- Dockerfile
- node-kode
- ajax-kode (her eller i c-kontaineren)

android

- java-fil(er)
- AndroidManifest.xml
- innholdet i assets-katalogen