Høyskolen Kristiania

MENU APPLICATION

Du skal lage en applikasjon som mottar instrukser fra bruker basert på input fra tastaturet (for eksempel i en form for meny) og be brukeren om data som er nødvendig for å kalle forskjellige funksjoner. Main skal rydde opp alle data før den returnerer (et valg i menyen må være å avslutte).

Hva betyr det?

Høyskolen Kristiania

```
> ./oppgave_a
Skriv inn ønsket kommando [1-5]:
1. Legg til et element i databasen
2. Finn et element i databasen
3. Finn antall elementer i databasen
4. Slett et element fra databasen
5. Avslutte

_
```

Bygge oppgaven “iterativt”

Høyskolen Kristiania

Du skal lage en main metode som styres av en meny.

Å jobbe iterativt betyr i praksis å gjøre en ting om gangen. Utvikleren starter med et tomt skall, kompilerer det og kjører applikasjonen (ofte Hello World), så bygger man på med EN og EN funksjonalitet. Det betyr at hvis noe krasjer så vet man at det er bare noen få kodelinjer som kan være feil! Dette bruker man i det virkelige liv, og det bør dere gjøre på eksamen også.

Lag kode som skriver ut meny valgene.

Les inn data fra en bruker, og lag en switch-case.

Task 1: Create the “project”

Høyskolen Kristiania

1. You need a new folder
2. Create a ./include folder and ./obj folder
3. Copy the makefile (the one we have agreed to use) into the folder
4. Create a .c file in the main folder, and a .h file in the include folder
5. Change the correct files and the desired output filename in the makefile
6. Add a printf(“Hello world”); line to the main() function
7. Compile and run the file, ensure that it works as it should!

Task 2: Add .c file for the “actual” functions

Høyskolen Kristiania

1. Add a new .c file to the makefile and create the file

2. If this is to interact with a database, for instance call it database.c

3. Add two functions called CreateDatabase() and AddEntry(), both to the database.c file and to the header file in the include folder

4. Call one of these functions from main to test it

Task 3: Add INCLUDE GUARDS to the header file

Høyskolen Kristiania

If you don’t already, create a “project” include file; database.h for instance, that holds a function prototype for your exercises function (that is called from main).

Add the #include “database.h” statement to both C files.

```
#ifndef __TK1104_DATABASE_H__
#define __TK1104_DATABASE_H__

void CreateDatabase(int iType);

#endif // __TK1104_DATABASE_H__
```

Task 4: Add text output

Høyskolen Kristiania

1. Add the 6 lines of printf that will print our “menu”

2. Return from main after printing

3. Compile and run the file, ensure that it works as it should!

4. Make special care to have \r\n in each of the printf statements so that it is not all outputted on one line, but each menu choice on its own line

Task 5: Read input from the user

Høyskolen Kristiania

1. After printing the text output, it is time to read text from the user

2. Use any of the read text functions, but getchar() is a common choice

3. Print a new line printf(“Input selected (%i)”, iChar);

4. Compile and run the file, ensure that it works as it should!

https://man7.org/linux/man-pages/man3/getchar.3p.html

Task 6: Implement a switch-case

Høyskolen Kristiania

1. After reading the input add a switch statement

```
iChar = getchar();
switch (iChar){
   case ‘1’:
      printf(“Add element selected”);
      break;

   case ‘2’:
      printf(“Find element selected”);
      break;

   default:
      break;
}
```

Task 7: More input from user

Høyskolen Kristiania

1. Adding an element needs to ask for more input from user

```
case ‘1’:
      printf(“Add element selected”);
      printf(“Enter name: “);
      gets(szBuffer);
      printf(“Will add element ‘%s’”, szBuffer);
      break;
```

2. gets() is not exactly a safe function, which function would be better?

Task 8: Flesh out application

Høyskolen Kristiania

1. Create the rest of the application

NORMAL CLIENT-SERVER APP

Du skal lage 1 server applikasjon og 1 klient applikasjon. Et tips for å teste denne oppgaven under programmeringen er å åpne 2 terminalvinduer, og kjøre hver applikasjon i eget vindu.
Server applikasjonen skal eksekveres med portnummer og en brukerspesifisert ID på kommandolinjen, for eksempel «oppgave_b1 -port 25 -id SmtpTest». Når startet skal applikasjonen åpne oppgitt port for LISTEN, for denne oppgaven holder det å binde til loopback på 127.0.0.1. Det anbefales å bruke TCP.
Klient applikasjonen skal eksekveres med serverens portnummer på kommandolinje, for eksempel «oppgave_b2 -server 25». Når startet skal applikasjonen CONNECTE til oppgitt port på serverapplikasjonen.

NORMAL CLIENT-SERVER APP #2

Du skal simulere SMTP trafikk mellom klient og tjener, ingen feilhåndtering nødvendig, kun hardkodet nettverksflyt.
Server: 220 127.0.0.1 SMTP SmtpTest 27 okt 2020, 12:42:42
Klient: HELO bengt.127.0.0.1
Server: 250 127.0.0.1 Hello bengt
Klient: MAIL FROM: <bengt@test.com>
Server: 250 Sender address ok
Klient: RCPT TO: <student@kristiania.no>
Server: 250 Recipent address ok
Klient: DATA
Server: 354 Ready for message
Klient: Test\r\n\r\n.\r\n
Server: 250 Message accepted
Klient: QUIT
Server: 221 127.0.0.1 closing connection

Server.c

Høyskolen Kristiania

```
#include <errno.h>

struct sockaddr_in saAddr = {0};//Bind
struct sockaddr_in saConClient = {0};//Accept

int sockFd, sockNewFd = 0, iPort = atoi("80"), readValue, addrLen = sizeof(saAddr);
char buffer[256];

ockFd = socket(AF_INET, SOCK_STREAM, 0);
if(sockFd < 0){
   printf("socket failed with %i\n", errno);
}

saAddr.sin_family = AF_INET;
saAddr.sin_port = htons(iPort);
saAddr.sin_addr.s_addr = INADDR_ANY;
```

Server.c

Høyskolen Kristiania

```
if(bind(sockFd, (struct sockaddr *) &saAddr, sizeof(saAddr)) < 0){
   printf("bind failed with %i\n", errno);
}

listen(sockFd, 5);

sockNewFd = accept(sockFd, (struct sockaddr *) &saConClient, (socklen_t*) &addrLen);
if(sockNewFd < 0){
   printf(“accept failed with %i\n", errno);
}

//Set buffer to 0 and use it to read the value from client
memset(buffer, 0, 256);
readValue = read(sockNewFd, buffer, 256 -1);
if(readValue < 0){
   printf(“read failed with %i\n", errno);
}

close(sockNewFd); sockFd = -1;
close(sockFd); sockFd = -1;
```

client.c

Høyskolen Kristiania

```
struct sockaddr_in saAddr = {0};//Connect
int sockFd, iPort = atoi("80");
char *msg = "This is a message from the client";

sockFd = socket(AF_INET, SOCK_STREAM, 0);
if(sockFd < 0){
   printf("socket failed with %i\n", errno);
}

saAddr.sin_family = AF_INET;
saAddr.sin_port = htons(iPort);
saAddr.sin_addr.s_addr = htonl(0x7F000001);//Home

if(connect(sockFd, (struct sockaddr *)&saAddr, sizeof(saAddr)) < 0){
   printf("connect failed with %i\n", errno);
}

send(sockFd, msg, strlen(msg), 0);

close(sockFd); sockFd = -1;
```

HINT

Høyskolen Kristiania

If you add arguments like –port 25 –name MyName, then EACH of these should be without spaces, the argv[] argument to main separates at spaces ☺

If you write

> ./myapp –port 25 –name My Name

(with a space in name)

Argv[1] = “-port”

Argv[2] = “25”

Argv[3] = “-name”

Argv[4] = “My”

Argv[5] = “Name”

Can be solved by using quotes, but to keep it simple this is true…

Task 1: Create the “project”

Høyskolen Kristiania

1. You need 2 new folders (one for the client and one for the server)
2. Create a ./include folder and ./obj folder
3. Copy the makefile (the one we have agreed to use) into the folder
4. Create a .c file in the main folder, and a .h file in the include folder
5. Change the correct files and the desired output filename in the makefile
6. Copy the source for client and server respectivly
7. Compile and run the file, ensure that it works as it should!

Task 2: Create the “project”

Høyskolen Kristiania

1. Add ONE by ONE the send/recv statements from the “dialog”

2. Test that it compiles and runs as expected after EACH change

MULTITHREADED SERVER

Bruk klient-server koden du lærte i leksjon 11, videreutvikle koden slik at serveren blir multithreadet og kan ha flere samtidige klienter.

(Dette er mer avansert enn hva vi forventer av mestring på eksamen, selv for en A – så kun for de som ønsker flere utfordringer i faget :-)

Add multithreaded code to server

Høyskolen Kristiania

1. Add a while loop around the accept() call

2. After an accept create a new thread

3. Move the read/recv function to the new thread function, also call close on the new socket inside this thread before returning

This is not easy, and if you do it wrong it is hard to debug. However it is also outside of what I can expect from you after this course, even the A students ☺