PROTOCOALE DE COMUNICATIE: Tema #1
Let's play a game

Termen de predare: 27 MARTIE 2015

Titulari curs: *Valentin CRISTEA, Florin POP, Gavril GODZA*

Responsabili Tema: **Cristian Chilipirea**

Vi se da un executabil ce ia rolul de client. Acest client incearca sa se conecteze la un server pe care voi va trebui sa il scrieti. Dupa conexiune acesta va trimite o comanda serverului. In textul acestei comenzi aveti instructiuni despre pasul urmator pe care trebuie sa il luati. De exemplu vi se va cere sa trimite-ti un mesaj cu un anumit text.

Stiti ca acest executabil poate functiona in 4 moduri in functie de parametri. Astfel serverul vostru trebuie sa permita rulare in 4 moduri diferite, in functie de parametrul cu care este pornit:

- modul simplu – trebuie sa faceti exact ceea ce va cere clientul si nimic mai mult. De exemplu:

```
Client              Server
     -----spune salut---------->
     <----salut-----------------
     ---iti trimit 2 mesaje---->
     -----un mesaj gol--------->
     -----trimite exit--------->
     <----exit------------------
```

- protocolul *STOP AND WAIT* – fiecare pachet este urmat de un pachet ACK, fie ca este un pachet de la client, fie ca este de la server. Folosind acest protocol comunicatia de mai sus ar deveni ar deveni.

```
Client              Server
     -----spune salut---------->
     <-----ACK------------------
     <----salut-----------------
     -----ACK------------------>
     ---iti trimit 2 mesaje---->
     <-----ACK------------------
     -----un mesaj gol--------->
     <-----ACK------------------
     -----trimite exit--------->
     <-----ACK------------------
     <----exit------------------
     -----ACK------------------>
```

- modul de *verificare paritate si retransmitere* – pastreaza protocolul STOP AND WAIT dar primul byte din fiecare mesaj (fie ca este de la client sau de la server) este folosit ca bit de paritate. Pachetul poate avea acum dimensiune maxima 1399. Daca este detectat un pachet eronat se cere retransmiterea lui prin inlocuirea ACK cu NACK

```
+-----------------------------------+
| Parity byte|Byte 0|Byte 1|....|Byte N|
+-----------------------------------+
```

  Din parity byte doar bitul cel mai din dreapta este folosit.
- modul de *transmitere cu metoda hamming si corectie* – pastreaza protocolul STOP AND WAIT dar

fiecare byte din pachetul trimis va trebui "codat" folosind metoda hamming prezentata in laboratorul 3, fiecare byte va ocupa astfel 2 bytes (mai exact 12 biti) , se vor putea astfel transmite pachete de maxim 700 de bytes de informatie. La primire fiecare grup de 2 bytes trebuie corectat urmat de decodarea acestora in byte-ul initial.
Un byte in forma initiala (in biti):

```
+-----------------------+
|0 |1 |2 |3 |4 |5 |6 |7 |
+-----------------------+
|D1|D2|D3|D4|D5|D6|D7|D8|
+-----------------------+
```

Dupa codare va deveni (Vom numi cei doi bytes HiHi, unde i reprezinta pozitia din buffer-ul initial):

```
+-----------------------------------------------+
|0 |1 |2 |3 |4 |5 |6 |7 |0 |1 |2 |3 |4 |5 |6 |7 |
+-----------------------------------------------+
|  |  |  |  |P1|P2|D1|P4|D2|D3|D4|P8|D5|D6|D7|D8|
+-----------------------------------------------+
```

Aici P1,P2,P4,P8 sunt biti de paritate, iar D1-8 sunt biti din byte-ul initial.
Mesajul transmis va avea urmatoarea forma (in bytes):

```
+-------------------------------------------------+
|0 |1 |2 |3 |4 |5 |6 |7 |8 |.............|N*2|N*2+1|
+-------------------------------------------------+
|H0|H0|H1|H1|H2|H2|.....................|HN |HN   |
+-------------------------------------------------+
```

Aici H0H0 are forma de mai sus si reprezinta codarea primului byte de date in 2 bytes.
Testarea temei se va face ruland serverul scris de voi alaturi de clientul OFERIT. Din acest motiv trebuie sa respectati cu strictete formatele prezentate in enunt.

Mentiuni

- nu aveti voie sa modificati documentul lib.h.
- nu este necesar decat scrierea serverului dar va recomandam sa implementati si un client al vostru pentru o mai buna intelegere a algormilor si eficientizare a debuging-ului.
- functiile de send si recv sunt blocante, aveti grija ca serverul sa fie o oglinda a clientului, sa respectati perfect protocolul.
- si pachetele ACK/NACK pot fi corupte, pentru a diferentia intre cele 2 verificati marimea pachetului, aceasta NU poate fi corupta.
- alaturi de tema aveti executabilul clientului si fisierul .o al acestuia pentru a-l putea compila pe orice sistem. pentru siteme 64 biti golositi client64.o
- clientul si serverul trebuie sa primeasca parametrii ack, parity, hamming sau nimic.
- sunteti indemnati sa folositi doar libraria oferita si limbajul C, temele vor fi testate automat.

Hints

Va recomandam sa folositi:

- sprintf
- strncmp
- strlen
- sizeof
- memcpy
- exit

Nu uitati de operatiile binare

- setarea unui bit
  `number |= 1 << x;`
- stergerea unui bit
  `number &= ~(1 << x);`
- schimbarea unui bit
  `number ^= 1 << x;`
- extragerea valorii unui bit
  `bit = (number >> x) & 1;`

Unde number e numarul cu care lucrati, iar x este pozitia bitului de interes.

Aveti extrem de mare grija la:

- stringuri vs date binare (majoritatea functiilor ce incep cu s intorc rezultat relativ la \n)
- numarul de bytes a unei valori si numarul de biti. E foarte usor sa faceti buffer overflow.

Upload

Uploadul se va face pe vmchecker si pe cs.curs.

Tema trebuie sa contina:

- server.c – sa contina tot codul pentru server.
- Readme – sa continua si explicatii detaliate a ce face clientul, cum ati descoperit si care este functionalitatea client/server.
- ATENTIE: A NU se adauga Makefile sau alte fisiere, compilarea se face DOAR cu Makefile-ul oferit.

VmChecker serveste drept platforma de upload si faciliteaza testarea temelor de catre echipa de asistenti. Testarea temelor de catre studenti trebuie facuta local. Orice plangeri despre "supra-aglomerarea" VmCheckerului vor fi ignorate.

In cazul in care exista probleme de neconcordanta intre testele locale si vmchecker folositi masina virtuala de aici.