Esame Reti 21 Gennaio 2020
Parte B

- Socket iterativo
- Utilizzo UDP
- FTP con esempio
- SNMP - getNextRequest con esempio
- Gerarchia MIB con esempio

Esercizio 1) (16 punti)
Si vuole implementare un'applicazione basata sulle socket per l'ordinazione e consegna a domicilio di pizze. Il server implementa le seguenti funzioni:

- **acquisto: la funzione riceve in ingresso la lista di pizze, l'indirizzo di consegna, l'ora di consegna e restituisce in uscita codice prenotazione di tipo intero;**
- **cancellazione: la funzione riceve in ingresso codice della prenotazione e restituisce in uscita OK, se l'annullamento viene effettuato un'ora prima della consegna, FAIL altrimenti.**

Si richiede di fornire lo pseudocodice del server iterativo che usa socket TCP. Il server riceve in ingresso il nome della funzione da eseguire e i parametri, esegue la funzione e ritorna il risultato al client.
E possibile usare socket UDP per implementare lo stesso servizio? Perché?

N.B. Le funzioni della libreria socket devono essere proposte in modo completo con tutti i parametri specificati. Non verrà accettato uno pseudocodice che utilizza le librerie socket di Java.
N.B. Non e ammesso l'utilizzo di funzioni di traduzione tra stringhe e interi (e viceversa).

```
#include <socket.h>
#include <string.h>

int main(int argc, char* argv) {

      //Init della struttura
      struct sockaddr_in local, client;
      local.sin_family = AF_INET;
      local.sin_port = htons(8080);
      local.sin_address.s_addr = INADDR_ANY

      int sb = socket(AF_INET, SOCK_STREAM, 0);

      bind(sb, (struct sockaddr*) &local, sizeof(local));
      listen(sb, 5);

      while(1) {
            //Accetto iterativamente le connessioni
            int ss = accept(sb, (struct sockaddr*) &client, sizeof(struct
                                                                  sockaddr));

            char fn[10];
            recv(ss, &fn, sizeof(fn), 0);

            if(strcmp(fn, “acquisto”)) {
                  acq(ss);
            }

            else if(strcmp(fn, “cancellazione”)) {
                  canc(ss);
            }

            close(ss);
      }
      exit(0);
```

```
}

void acq(int sock) {
      list_str* pizze;
      char pizza[30];
      char risposta[1];
      char indirizzo[200];
      char oraCons[5];
      char* msg = “Quale pizza desidera?”

      //Lista Pizze
      while(!strcmp(risposta, “N”, 1)) {
            send(sock, &msg, sizeof(msg), 0);
            recv(sock, &pizza, sizeof(pizza), 0);
            list_add(&pizze, &pizza);

            msg = “Aggiungere altre pizze? [y/n]”;
            send(sock, &msg, sizeof(msg), 0);
            recv(sock, &risposta, sizeof(risposta), 0);
      }

      //Indirizzo consegna
      msg = “Inserire indirizzo consegna”;
      send(sock, &msg, sizeof(msg), 0);
      recv(sock, &indirizzo, sizeof(indirizzo), 0);

      //Ora consegna
      msg = “Inserire ora consegna”
      send(sock, &msg, sizeof(msg), 0);
      recv(sock, &oraCons, sizeof(oraCons), 0);

      //Generazione codice ordine
      int codice_ord = htons(acquisto(&pizze, &indirizzo, &oracons));
      msg = “Grazie per l’ordinazione, il suo ordine ha codice ”;
      send(sock, &msg, sizeof(msg), 0);
      send(sock, &codice_ord, sizeof(codice_ord), 0);
}

void canc (int sock) {
      int codice_ord = -1;
      char * res;

      //Richiesta del codice ordine
      msg = “Inserisca il codice dell’ordinazione che desidera cancellare”
      send(sock, &msg, sizeof(msg), 0);
      recv(sock, &codice_ord, sizeof(codice_ord), 0);
      codice_ord = ntohs(codice_ord);

      res = cancellazione(codice_ord);

      //Gestione del messaggio
```

```
	if(strcmp(res, “OK”) {
		msg = “Ordinazione cancellata con successo”;
	}

	else {
		msg = “Impossibile cancellare l’ordinazione a meno di un’ora di
			consegna”;
	}

	send(sock, &msg, sizeof(msg), 0);
}
```

Non è possibile implementare lo stesso servizio tramite socket UDP, in quanto questo servizio necessita lo scambio di più messaggi consecutivi in ordine, cosa garantita da TCP ma non UDP (una client potrebbe inviare un indirizzo di consegna e poi l’orario, ma al server potrebbe arrivare prima l’orario, valorizzando la variabile dell’indirizzo con l’orario).
Ancora, il fatto che TCP si basi su una connessione, permette di avere la sicurezza di scambiare messaggi sempre con lo stesso client, mentre i datagrammi UDP si potrebbero accavallare (una pizza di un cliente potrebbe finire nell’ordinazione di un altro).

Esercizio 2) (9 punti)
Dopo aver descritto le principali caratteristiche del protocollo FTP, si discutano nel dettaglio tutte le richieste e risposte di una comunicazione FTP in modalità attiva dove il client nome studente:
(1) si autentica al server
(2) richiede la lista di file nella sua cartella \nome_studente\mydocuments\;
(3) scarica il file SIFA.txt.
È possibile pianificare una porzione del server con tutti i le accessibili senza previa autenticazione? Se sì come?

File Transfer Protocol si basa su TCP e consente sia il trasferimento dei file che l’accesso interattivo, ossia fornisce una user interface usufruibile da un utente finale (il dialogo avverrà tramite l’utilizzo di comandi e la ricezione di risposte sotto forma di codici di stato).
FTP prevede l’apertura e la chiusura di una connessione (essendo basato su TCP).
FTP lavora dietro autenticazione (nome utente e password), ed è basato su un’architettura client-server.
Il server disporrà di un servizio in ascolto delle richieste FTP sulla relativa porta nota 21.
FTP è costituito da due processi paralleli: il processo di controllo che comunica con il processo di controllo remoto (basato su NVT) e il processo di trasferimento dati, che trasferisce il file indicato.
Il client provvederà a comunicare al server una porta locale in grado di ricevere connessioni, sulla quale poi il server fornirà il risultato del comando, in modo da evitare problemi con il firewall.

Esempio di comunicazione FTP
(1)
Collegamento al server FTP
→ `ftp ftp.unimi.it`
Risposta dal server per segnalare che il server è pronto a ricevere la connessione di un utente
← `220 bruno FTP server (SunOS 4.1) ready.`
Specificazione dello username
→ `USER nome_studente`
Risposta dal server per segnalare che il nome utente è corretto ed è richiesta la password
← `331 Guest login ok, send ident as password.`

Invio della password al server
→ PASS password
Risposta dal server per segnalare che la login è avvenuta correttamente e che l'utente è connesso
← 230 Benvenuto, nome_studente

(2)
*Inizializzazione della connessione dati attiva, tramite la quale il client segnala al server la porta su cui vuole venga effettuata la connessione peri l trasferimento dati (specificati secondo le regole di FTP, in questo caso l'ip sarà 192.168.0.5 e la porta 2594 (10*256+34))*
→ PORT 192,168,0,5,10,34
Risposta del server ad indicare la richiesta è avvenuta con successo e si collegherà su quella porta
← 200 PORT command successful
Richiesta al server di mostrare il contenuto della directory
→ NLST \nome_studente\mydocuments\
Risposta dal server ad identificare che la connessione sulla porta precedentemente specificata è stata aperta e predisposta in ASCII mode
← 150 Opening ASCII mode data connection for file list.
Risposta dal server con il contenuto directory
← * Contenuto directory *
Segnalazione dal server della fine del trasferimento dati
← 226 Transfer Complete

(3)
Richiesta del file di scaricare il file SIFA.txt
→ RETR SIFA.txt
Come sopra sulla porta 2592
→ PORT 192,168,0,5,10,32
Come sopra
← 200 PORT command successful
Apertura della connessione nella porta designata per effettuare il trasferimento del file
← 150 ASCII data connection for SIFA.txt (*ip*,*port*) (*num* bytes)
Segnalazione della fine del trasferimento dati + statistiche varie
← ASCII Transfer complete
← *num* byte transmitted in *x* seconds

È possibile pianificare una porzione di server con tutti i file accessibili senza previa autenticazione. Per questa zona bisogna abilitare il cosiddetto "accesso anonimo", che permette di utilizzare delle credenziali fittizie per accedere appunto ad una zona riservata (presumibilmente in sola lettura) del file system.

Domanda 1) (5 punti)
Nell'ambito del protocollo SNMP, si discuta il ruolo della funzione get-next-request presentando un esempio. In questo contesto, si presenti nel dettaglio la gerarchia dei nomi MIB.

La funzione get-next-request ha lo scopo di ottenere il valore del prossimo nodo nella mib, partendo dal nodo corrente. Questo permette di scorrere le entries della MIB in maniera iterativa, rendendo molto più semplice la navigazione all'interno della MIB (diversamente, bisognerebbe sapere un OID a priori per ogni comando "get-request").

Il manager potrà inviare ai vari agent dei messaggi SNMP (sotto forma di datagrammi UDP).
Ad esempio
Invio di una richiesta all'agent per sapere quanti datagrammi gli sono stati consegnati

Manager → get-request(1.3.6.1.2.1.7.1) → Agent

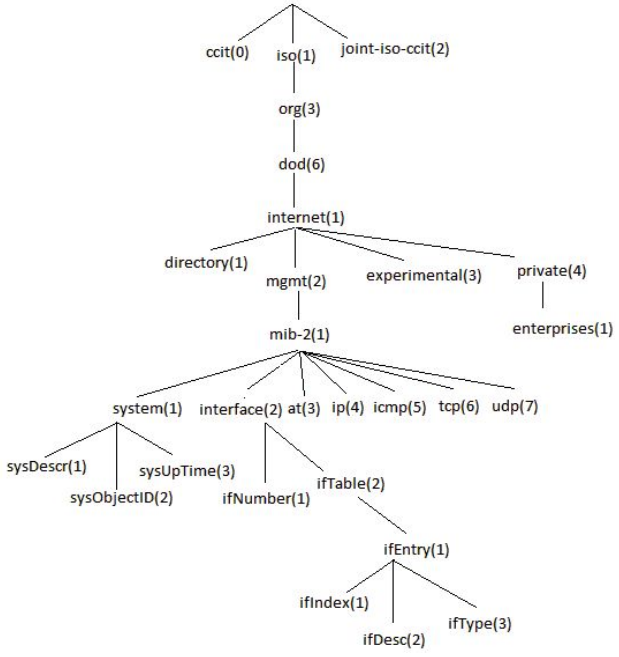
Risposta dall'agent
Agent → response(1.3.6.1.2.1.7.1) => 30 (numero di datagrammi consegnati al nodo) → Manager

Invio di una richiesta all'agent per sapere quanti datagrammi non sono stati consegnati per mancanza di applicazioni in ascolto sulla porta di destnazione
Manager → get-request(1.3.6.1.2.1.7.2) → Agent
oppure
Manager → get-next-request(1.3.6.1.2.1.7.1) → Agent

Risposta dall'agent
Agent → response((1.3.6.1.2.1.7.2) => 10 (datagrammi non consegnabili per mancanza di applicazioni in ascolto sulla porta di destin.) → Manager

Come si può vedere, la get-next-request permette di riutilizzare l'OID tornato da una response per ottenere il valore immediatamente successivo nell'ordine lessicografico della MIB, e così scorrere iterativamente la MIB

Il MIB è un db in formato testuale che riunisce tutte le info disponibili su tutti i dispositivi installati sulla rete. Esso è strutturato in modo gerarchico (ad albero). La gerarchia dei nomi MIB permette il raggiungimento di una informazione da leggere o scrivere tramite la specifica progressiva del percorso dei nodi da percorrere per arrivare all'informazione. Ogni nodo è espresso tramite un numero (che specifica quale figlio è del rispettivo nodo padre.
Una gerarchia MIB è formata come segue:



Ogni numero all'interno della gerarchia MIB rappresenta un modulo (sottoalbero della gerarchia).
Un esempio di un OID notevole può essere
1.3.6.1.2.1.7.1
Sequenzialmente questi numeri indicano:
1 → ISO: identifica l'abero di generazione degli OID di ISO
3 → org: identifica che a seguire sarà specificata una organizzazione
6 → Dipartimento della difesa degli stati uniti (che ha definito l'ISO)
1 → Riferimento al sottoalbero degli oggetti inerenti alle interreti
2 → Riferimento al sottoalbero degli oggetti di gestione

1 → Utilizzo del sottoinsieme di oggetti che fanno parte del core di MIB2
7 → Sottoinsieme degli oggetti che riguardano UDP
1 → Identificatore dell'oggetto specifico, in questo caso il numero dei datagrammi UDP consegnati al nodo