Εργαστήριο Δικτύων Υπολογιστών

4η Διάλεξη:

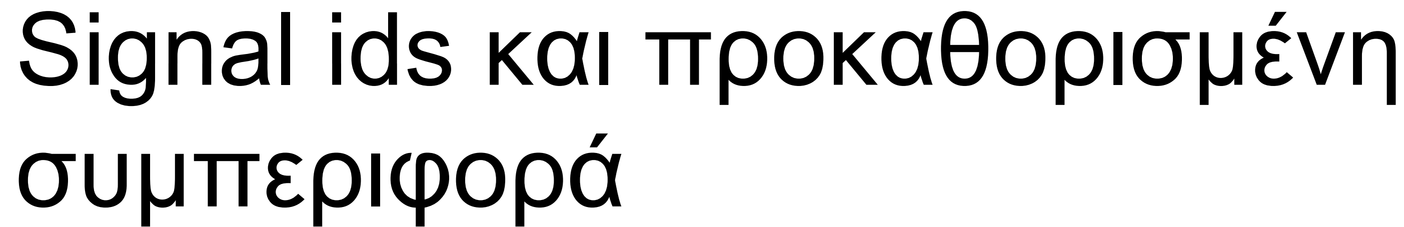
Δικτυακός Προγραμματισμός

- Signals
- UDP Sockets

- Όταν σε ένα TCP server κάνουμε “fork” (γεννάμε διεργασίες-παιδιά servers για να εξυπηρετήσουμε πολλούς clients), υπάρχει η περίπτωση να δημιουργηθούν zombie processes
- Για τον σωστό τερματισμό μιας διεργασίας-παιδί πρέπει ο πατέρας να «ειδοποιηθεί» για τον τερματισμό του
- Μία διεργασία-παιδί η οποία έχει τερματιστεί και της οποίας η διεργασία-πατέρας δεν έχει λάβει ακόμα ειδοποίηση του τερματισμού της ονομάζεται zombie (defunct) process
- Για τον σωστό τερματισμό των παιδιών-servers ένας τρόπος είναι να χρησιμοποιήσουμε signals

Signals

- Μέσω των signals μία διεργασία μπορεί να ειδοποιηθεί για διάφορα γεγονότα
- Τα signals ονομάζονται και “software interrupts”
- Signals μπορούν να σταλούν από μία διεργασία σε μία άλλη διεργασία ή από τον kernel σε μία διεργασία
- Signals μπορούν να σταλούν με την εντολή kill:
  - Από την γραμμή εντολών: kill –sigid –pid
  - Σε ένα C πρόγραμμα: int kill(pid_t pid, int sigid);
- Κάθε signal έχει εξ’ορισμού του μία (προ)καθορισμένη συμπεριφορά

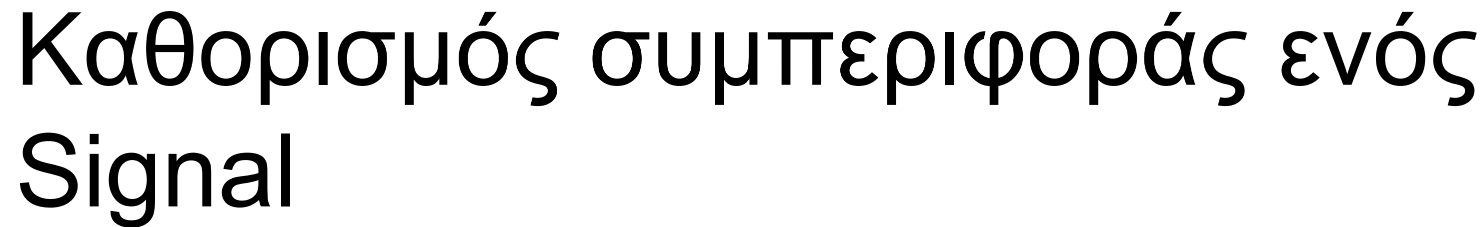


Signal ids και προκαθορισμένη συμπεριφορά

Signal	**id**	**default**
SIGKILL	9	Terminate
SIGALRM	14	Terminate
SIGSYS	12	Core
SIGCHLD	20	Ignore

Όταν μια διεργασία-παιδί τερματίζεται στέλνει ένα σήμα SIGCHLD (id=20) στον πατέρα της

Το σήμα αυτό αγνοείται (προκαθορισμένη συμπεριφορά: ignore)



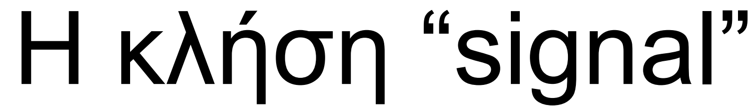
Καθορισμός συμπεριφοράς ενός Signal

- Πρωτογενείς κλήσεις: signal() και sigaction()
- Στο πρόγραμμα μας μπορούμε να καθορίσουμε επ'ακριβώς την συμπεριφορά μίας διεργασίας, ύστερα από την λήψη ενός signal:
  - Εκτέλεση μίας συνάρτησης (signal handler)
  - Καμία ενέργεια (SIG_IGN)
  - Προκαθορισμένη συμπεριφορά (SIG_DFL)
- Τα σήματα SIGKILL και SIGSTOP δεν μπορούν ούτε να «αγνοηθούν» (SIG_IGN) ούτε να «πιαστούν» από μία συνάρτηση (signal handler) -> εκτελείται πάντοτε η default ενέργεια

Οι πρωτογενείς κλήσεις για τα signals

Οι οδηγίες (#include) που χρησιμοποιούνται για αυτές τις κλήσεις είναι οι:

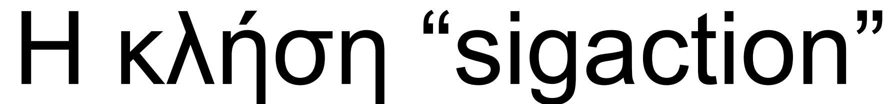
- **`#include <signal.h>`**



Η κλήση “signal”

- Σύνταξη:
  `signal (int sig, void (*disp)(int));`

- Καθορίζει την συμπεριφορά της διεργασίας, μέσα στην οποία καλείται, ύστερα από την λήψη ενός signal **`sig`**
- Το όρισμα ***`disp`** ορίζει αυτή την συμπεριφορά
- Το ***`disp`** μπορεί να είναι η διεύθυνση του signal handler (pointer σε συνάρτηση), SIG_IGN ή SIG_DFL



Η κλήση “sigaction”

- Σύνταξη:
  `int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);`

- Καθορίζει την συμπεριφορά της διεργασίας, μέσα στην οποία καλείται, ύστερα από την λήψη ενός signal **`sig`**
- Συνήθως χρησιμοποιείται η sigaction αντί της signal
- Το όρισμα **`act`** δείχνει σε μία δομή (**`struct sigaction`**) η οποία ορίζει αυτή την συμπεριφορά
- Το όρισμα **`oact`** δείχνει σε μία δομή (**`struct sigaction`**) όπου αποθηκεύεται η συμπεριφορά που ήταν προηγουμένως συσχετισμένη με το **`sig`**

Η κλήση “sigaction” - sigaction struct

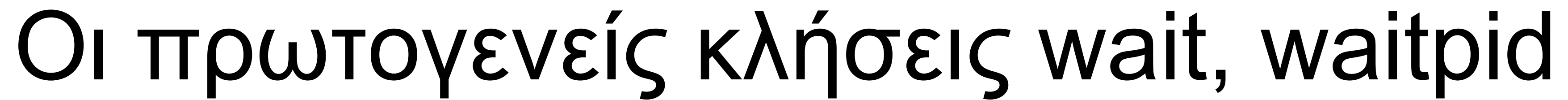
```
struct sigaction {
    void (*sa_handler)();
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t  sa_mask;
    int sa_flags;
    };
```

Zombie processes

- Για τον σωστό τερματισμό μιας διεργασίας-παιδί πρέπει ο πατέρας να «ειδοποιηθεί» (αυτό γίνεται όταν ο πατέρας εκτελέσει την πρωτογενή κλήση wait ή waitpid)
- Μία διεργασία-παιδί η οποία έχει τερματιστεί και της οποίας η διεργασία-πατέρας δεν έχει λάβει ακόμα ειδοποίηση του τερματισμού της ονομάζεται zombie process
- Μία zombie διεργασία υπάρχει μόνο σαν μία καταχώρηση στο process table, όπου διατηρούνται πληροφορίες σχετικά με αυτήν
- Μία zombie διεργασία μπορούμε να την σκοτώσουμε από την γραμμή εντολών με την εντολή kill

Handling zombie processes

- Όταν σε μία διεργασία-πατέρας κάνουμε fork (γεννάμε διεργασίες-παιδιά), θα πρέπει να καλέσουμε για αυτές τις διεργασίες-παιδιά την wait ή την waitpid, έτσι ώστε να τις εμποδίσουμε από το να γίνουν zombie processes
- Για το σκοπό αυτό στην διεργασία-πατέρα εγκαθιστούμε έναν signal handler στο signal SIGCHLD, και μέσα σε αυτόν τον signal handler καλούμε την wait ή την waitpid



Οι πρωτογενείς κλήσεις wait, waitpid

Οι οδηγίες (#include) που χρησιμοποιούνται για αυτές τις κλήσεις είναι οι :

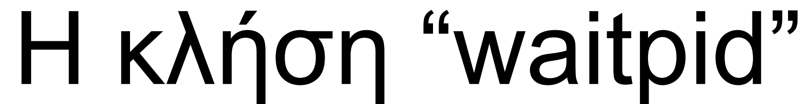
```
#include <sys/wait.h>
```

Η κλήση “wait”

- Σύνταξη:

```
pid_t wait(int *statloc);
```

- Επιστρέφει το process id του τερματιζόμενου παιδιού
- Στον δείκτη `statloc` επιστρέφεται το termination status του παιδιού
- Η wait είναι blocking συνάρτηση



- Σύνταξη:

```
pid t waitpid(pid t pid, int *statloc, int
options);
```

- Επιστρέφει το process id του τερματιζόμενου παιδιού
- Δίνεται η δυνατότητα καθορισμού, μέσω του **pid**, του παιδιού για το οποίο θα εκτελεστεί η συνάρτηση αυτή
- Στον δείκτη **statloc** επιστρέφεται το termination status του παιδιού
- Δίνεται η δυνατότητα καθορισμού επιπλέον επιλογών. Η πιο συνηθισμένη επιλογή είναι η WNOHANG, η οποία εμποδίζει την waitpid να γίνει blocked αν δεν υπάρχουν τερματιζόμενα παιδιά

wait vs waitpid

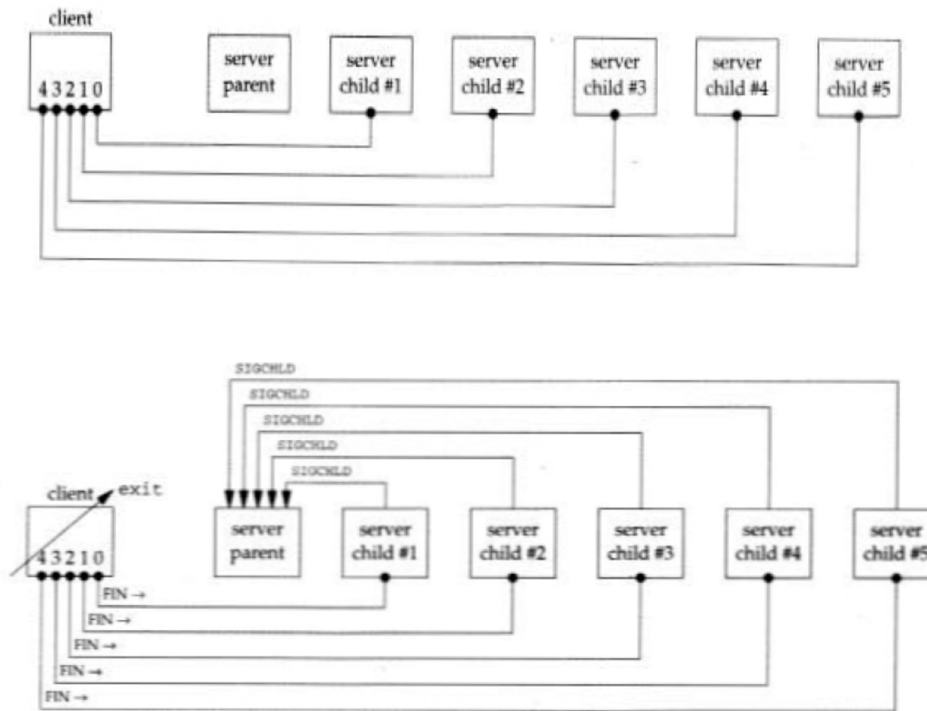
Υποθέτουμε ότι έχουμε καθορίσει σαν signal handler για το signal SIGCHLD την συνάρτηση sig_chld
(e.g. signal(SIGCHLD, sig_chld);)

tcpcliserv/sigchldwait.c

```
 1 #include    "unp.h"

 2 void
 3 sig_chld(int signo)
 4 {
 5     pid_t   pid;
 6     int     stat;

 7     pid = wait(&stat);
 8     printf("child %d terminated\n", pid);
 9     return;
10 }
```

tcpcliserv/sigchldwait.c

wait vs waitpid



- Τα signals δεν αποθηκεύονται σε ουρά
- Με την wait υπάρχει πρόβλημα όταν δημιουργηθούν ταυτόχρονα σήματα: ο signal handler θα εκτελεστεί μια φορά και τα υπόλοιπα signals θα χαθούν

wait vs waitpid

Υποθέτουμε ότι έχουμε καθορίσει σαν signal handler για το signal SIGCHLD την συνάρτηση sig_chld
(e.g. signal(SIGCHLD, sig_chld);)
waitpid with WHOHANG option: non blocking call

tcpcliserv/sigchldwaitpid.c

```
1 #include    "unp.h"

2 void
3 sig_chld(int signo)
4 {
5     pid_t   pid;
6     int     stat;

7     while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
8         printf("child %d terminated\n", pid);
9     return;
10 }
```

tcpcliserv/sigchldwaitpid.c

Handling interrupted system calls

- Όταν ένα blocking system call (π.χ. accept) διακοπεί από ένα signal, τότε μετά την επιστροφή από τον signal handler το system call θέτει την μεταβλητή errno ίσο με EINTR
- Το system call δεν είναι σίγουρο ότι θα επανακκινηθεί αυτόματα από τον kernel
- Για το λόγο αυτό θα πρέπει ο προγραμματιστής να φροντίζει για την επανεκκίνηση του system call

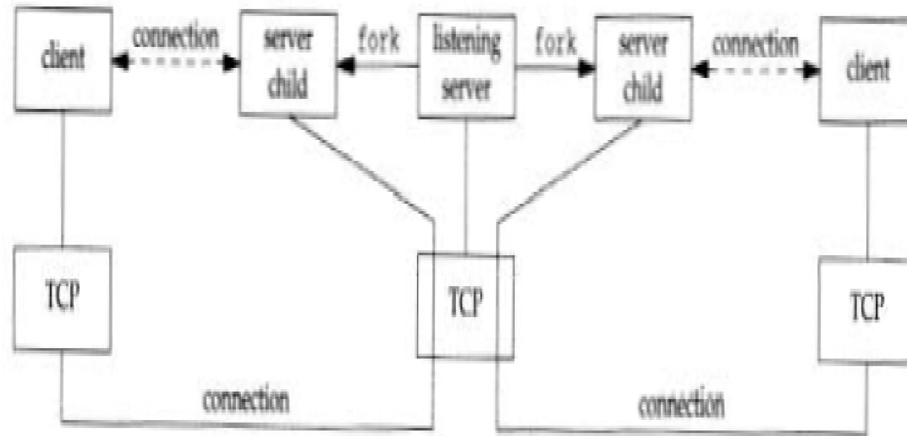
Handling interrupted system calls

```
for ( ; ; ) {
    clilen = sizeof(cliaddr);
    if ( (connfd = accept(listenfd, (SA *) &cliaddr, &clilen)) < 0) {
        if (errno == EINTR)
            continue;       /* back to for() */
        else
            err_sys("accept error");
    }
```

UDP vs TCP

- Το UDP είναι ένα connectionless, μη αξιόπιστο, datagram transport protocol
  - ☒ Δεν μπορεί να χειριστεί διπλά πακέτα ή πακέτα σε λάθος σειρά
  - ☒ Δεν παρέχει flow control και congestion avoidance μηχανισμούς
  - ☑ Έχει ελάχιστο overhead
- Εφαρμογές που χρησιμοποιούν το UDP:
  - DNS, NFS, SNMP, TFTP, real-time multiplayer games, voice conferencing, broadcasting

UDP vs TCP (client/server)

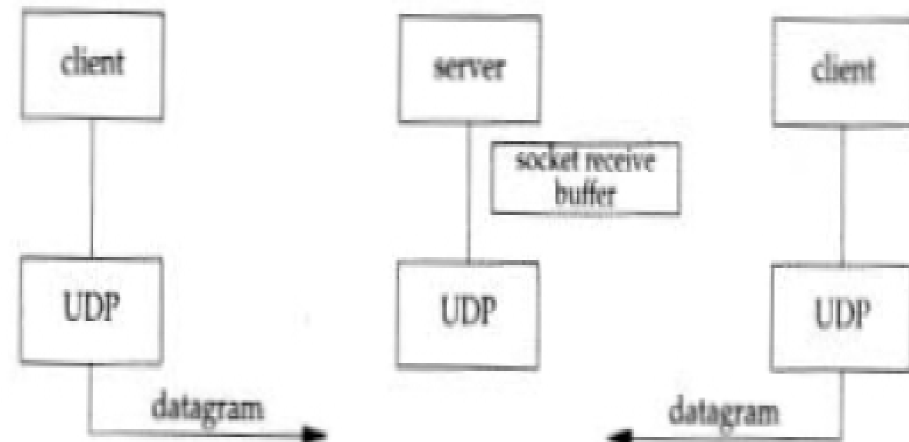


TCP

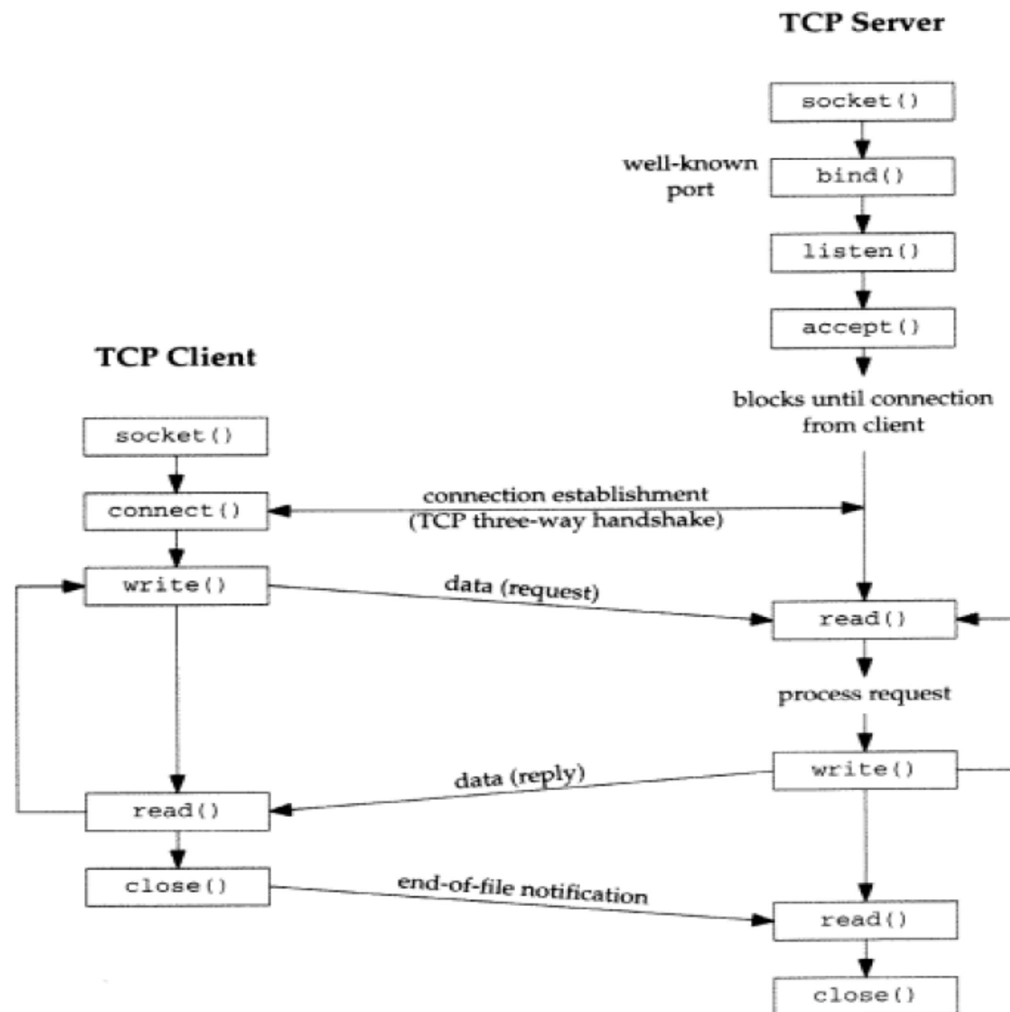
Ο client επικοινωνεί με έναν «αφοσιωμένο» αντίγραφο του server, ενώ ο «αρχικός» server μπορεί να δέχεται νέες κλήσεις σύνδεσης

UDP

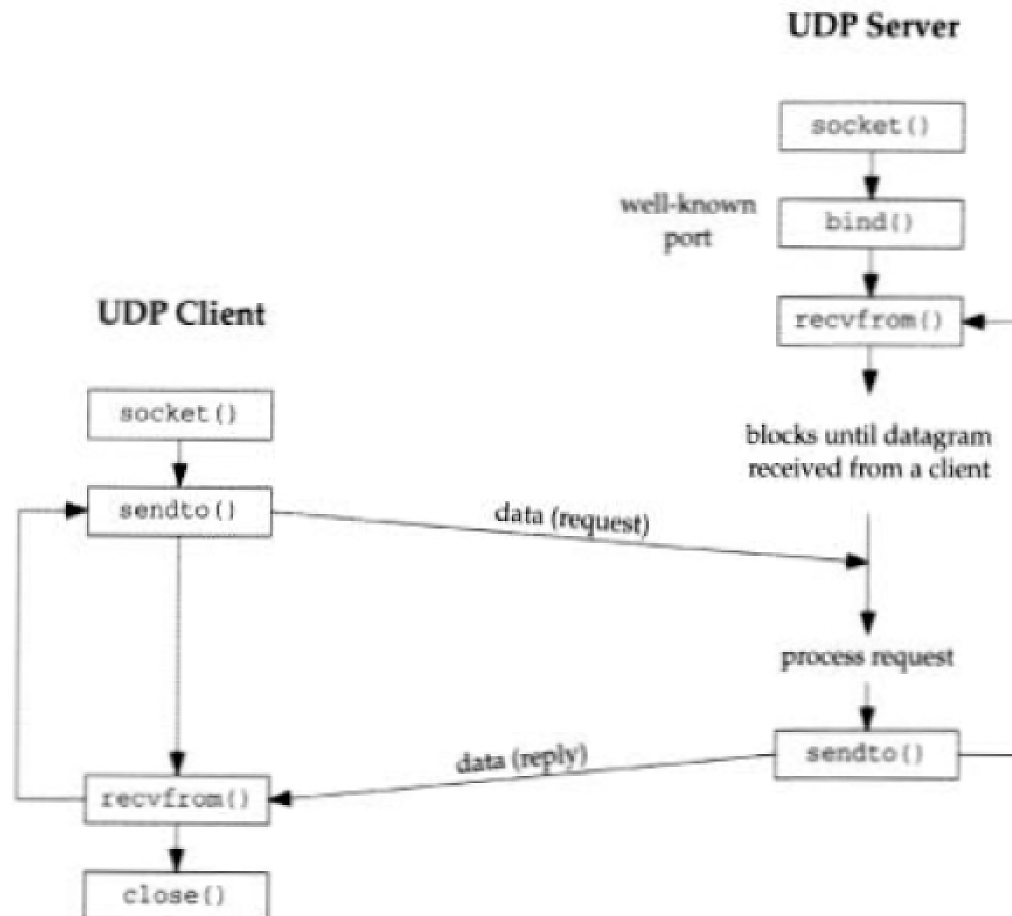
Ο client στέλνει πακέτα (datagram) στον server.
Τα πακέτα αποθηκεύονται σε μια ουρά και επεξεργάζονται από τον server

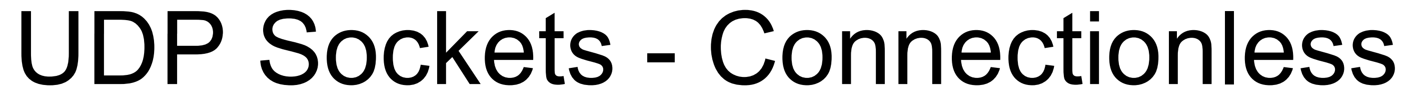


TCP Client/Server interaction (Stream Communication)



UDP Client/Server interaction (Datagram Communication)





UDP Sockets - Connectionless

- **Server**
  - create endpoint (socket())
  - bind address (bind())
  - transfer data (sendto() recvform())
- **Client**
  - create endpoint (socket())
  - transfer data (sendto() recvform())

Πρωτογενείς κλήσεις για τα sockets

Οι οδηγίες (#include) που χρησιμοποιούνται για αυτές τις κλήσεις είναι οι :

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

Η κλήση “recvfrom”

- Σύνταξη:
  `ssize_t recvfrom (int sockfd, void *buff, size_t nbytes, int flags, struct sockaddr *from, socklen_t *addrlen);`
- Περιμένει μέχρι να φτάσουν δεδομένα (blocking)
- Επιστρέφει το size των δεδομένων (datagram) που έλαβε
- Στο **`void *buff`** αποθηκεύει τα δεδομένα (datagram) με μέγιστο μέγεθος **`nbytes`**
- Στην δομή **`sockaddr *from`** επιστρέφει την διεύθυνση του αποστολέα
- Δεν υπάρχει καμία σύνδεση (connection) με τον αποστολέα

Η κλήση “sendto”

- Σύνταξη:
  ```
  ssize_t sendto (int sockfd, void *buff, size_t nbytes, int flags, const struct sockaddr *to, socklen_t *addrlen);
  ```
- Στέλνει τα δεδομένα (datagram) που βρίσκονται στη παράμετρο **void *buff** με μέγεθος ***nbytes*** στον προορισμό (δηλώνεται μέσω της δομής **const struct sockaddr *to**)
- Δεν υπάρχει καμία σύνδεση (connection) με τον παραλήπτη

Ένας απλός UDP Server

udpcliserv/udpserv01.c

```
 1 #include    "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sockfd;
 6     struct sockaddr_in servaddr, cliaddr;

 7     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

 8     bzero(&servaddr, sizeof(servaddr));
 9     servaddr.sin_family = AF_INET;
10     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
11     servaddr.sin_port = htons(SERV_PORT);

12     Bind(sockfd, (SA *) &servaddr, sizeof(servaddr));

13     dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr));
14 }
```

udpcliserv/udpserv01.c

… Ένας απλός UDP Server

lib/dg_echo.c

```
 1 #include    "unp.h"

 2 void
 3 dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
 4 {
 5     int       n;
 6     socklen_t len;
 7     char      mesg[MAXLINE];

 8     for ( ; ; ) {
 9         len = clilen;
10         n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

11         Sendto(sockfd, mesg, n, 0, pcliaddr, len);
12     }
13 }
```

lib/dg_echo.c

Ένας απλός UDP Client

udpcliserv/udpcli01.c

```
1 #include    "unp.h"

2 int
3 main(int argc, char **argv)
4 {
5     int     sockfd;
6     struct sockaddr_in servaddr;

7     if (argc != 2)
8         err_quit("usage: udpcli <IPaddress>");

9     bzero(&servaddr, sizeof(servaddr));
10    servaddr.sin_family = AF_INET;
11    servaddr.sin_port = htons(SERV_PORT);
12    Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

13    sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

14    dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));

15    exit(0);
16 }
```

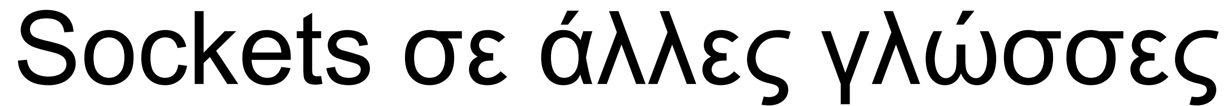
udpcliserv/udpcli01.c

… Ένας απλός UDP Client

lib/dg_cli.c

```
 1 #include    "unp.h"

 2 void
 3 dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
 4 {
 5     int     n;
 6     char    sendline[MAXLINE], recvline[MAXLINE + 1];

 7     while (Fgets(sendline, MAXLINE, fp) != NULL) {

 8         Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

 9         n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

10         recvline[n] = 0;        /* null terminate */
11         Fputs(recvline, stdout);
12     }
13 }
```

lib/dg_cli.c



Sockets σε άλλες γλώσσες

- Java, PhP, .NET, Perl ….
- Ακολουθούν πιστά το μοντέλο των Berkeley sockets της C
- Συνήθως ευκολότερο error handling (exceptions)
- Object-oriented
- Όμως παρέχουν και higher-level μηχανισμούς για network communication (RMI, Web services, CORBA, Protocol-specific libraries)
  - Εφόσον χρειάζεται να προγραμματίσετε με sockets, σημαίνει ότι ήδη θα έχετε συνήθως επιλέξει C\C++ !

Java Sockets

- java.net.InetAddress
  - Μια IP διεύθυνση
- java.net.Socket
  - Client socket
- java.net.ServerSocket
  - Server socket

- IPv4 και IPv6 compatible! (από 1.4 και πάνω)

InetAddress class

java.net.InetAddress

- static InetAddress getByName(String *name*)
- static InetAddress[] getAllByName(String *name*)
- static InetAddress getLocalHost()
- static InetAddress getByAddress(byte[] *addr*)

Client socket

- java.net.Socket
  - Socket(InetAddress *addr*, int *port*);
  - InputStream getInputStream();
  - OutputStream getOutputStream();
  - close();

- Ενέργειες:
  1. Open socket
  2. Open input stream και output stream στο socket
  3. Read και write στο stream σύμφωνα με το πρωτόκολλο
  4. Close streams
  5. Close socket

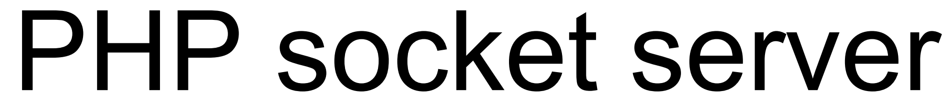
Server socket

- java.net.ServerSocket
  - ServerSocket(int *port*);
  - Socket accept();
- Multiple connections: Java threads

```
while (true) {
    serverSocket.accept();
    // create a Java thread to deal with the client
}
```

PHP sockets

- Λογικά θα τρέξουμε την PHP από command line (σαν ένα απλό interpreted executable)
- IPv4 και IPv6 compatible! (5.0.0 και πάνω)
- getservbyname()
- gethostbyname()



PHP socket server

- socket_create($domain,$type,$protocol)
- socket_bind($socket,$address,$port)
- socket_listen($socket)
- socket_accept($socket)
- socket_write($socket) / socket_read($socket)
- socket_close($socket)

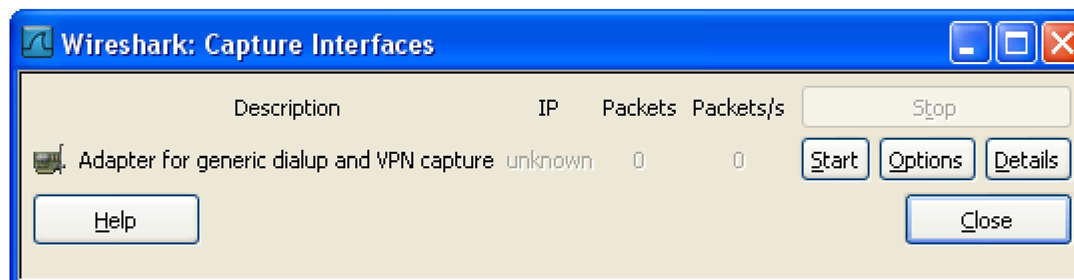
PHP socket client

- socket_create($domain,$type,$protocol)
- socket_connect($socket,$address,$port)
- socket_write($socket) / socket_read($socket)
- socket_close($socket)

Wireshark example

- Κάνουμε capture μία απλή HTTP σύνοδο
- Το HTTP χρησιμοποιεί TCP
- Κάνουμε filter IP πακέτα (κρύβουμε τυχόν L2 frames)

1. Επιλογή interface:



Αρχικό capture

netlab_http_capture.pcap - Wireshark

File Edit View Go Capture Analyze Statistics Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.73.133.27	150.140.141.181	TCP	3182 > http [SYN] Seq=0 Len=0 MSS=1420
2	0.050072	150.140.141.181	85.73.133.27	TCP	http > 3182 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
3	0.050072	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.050072	85.73.133.27	150.140.141.181	HTTP	GET /~stamos/netlab.html HTTP/1.1
5	0.110158	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=1 Ack=224 Win=6432 Len=0
6	0.150216	150.140.141.181	85.73.133.27	HTTP	HTTP/1.1 200 OK (text/html)
7	0.160230	150.140.141.181	85.73.133.27	TCP	http > 3182 [FIN, ACK] Seq=301 Ack=224 Win=6432 Len=0
8	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=224 Ack=302 Win=65235 Len=0
9	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [FIN, ACK] Seq=224 Ack=302 Win=65235 Len=0
10	0.220317	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=302 Ack=225 Win=6432 Len=0
11	0.270389	48:0f:20:52:41:53	Locate-Directory-Serv	LLC	U, func=UI; DSAP LLC Sub-Layer Management Group, SSAP LLC Sub-Layer

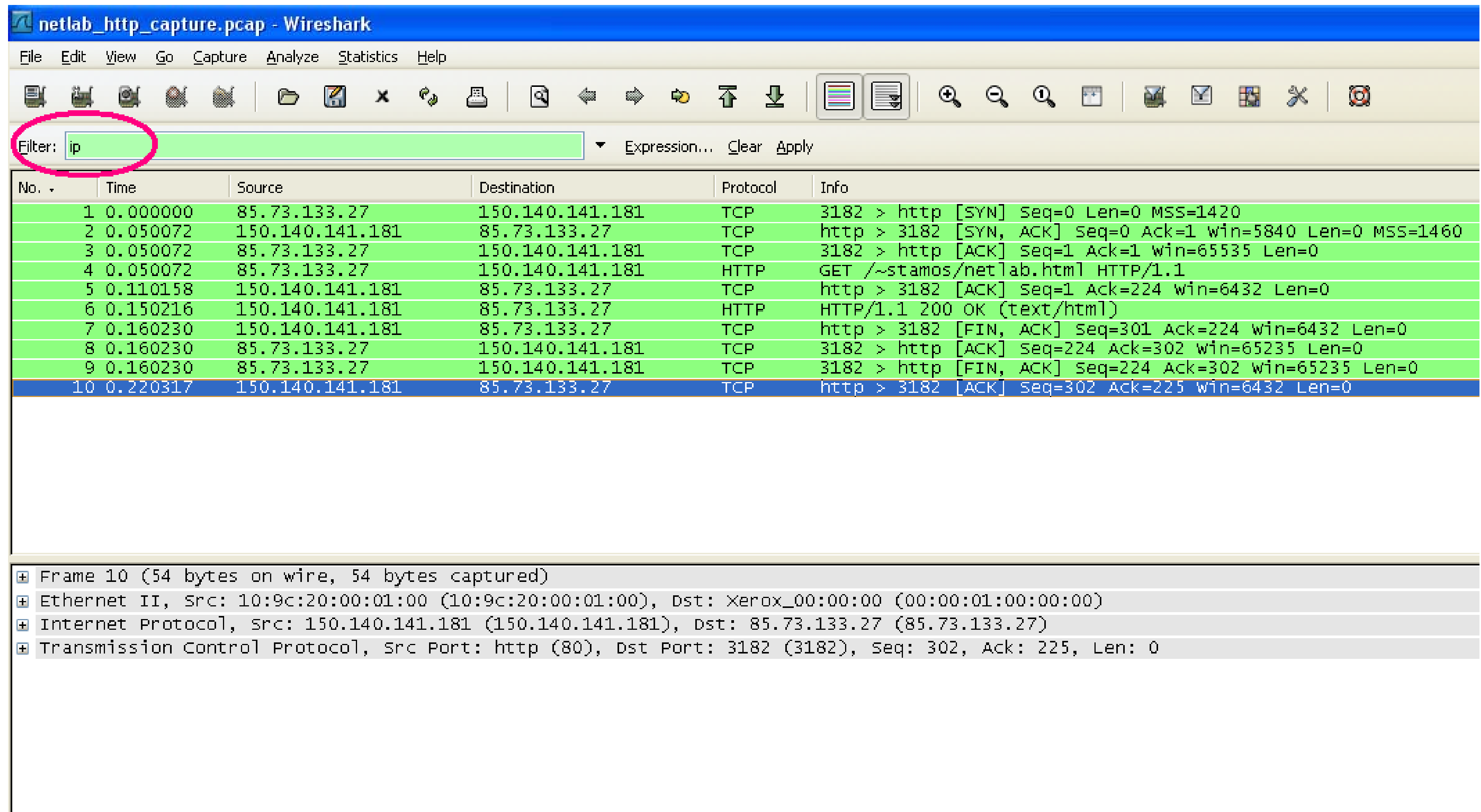
```
Frame 11 (197 bytes on wire, 197 bytes captured)
IEEE 802.3 Ethernet
Logical-Link Control
  DSAP: LLC Sub-Layer Management (0x02)
  IG Bit: Group
  SSAP: LLC Sub-Layer Management (0x02)
  CR Bit: Command
  Control field: U, func=UI (0x03)
    000. 00.. = Command: Unnumbered Information (0x00)
    .... ..11 = Frame type: Unnumbered frame (0x03)
Data (177 bytes)
```

```
0000  03 00 00 00 00 02 48 0f  20 52 41 53 00 b4 03 02   ......H.  RAS....
0010  03 52 54 53 53 03 00 00  00 00 00 a8 00 01 00 00   .RTSS... ........
0020  00 a2 6f 08 00 53 4c 41  59 45 52 00 00 00 00 00   ..o..SLA YER.....
0030  00 00 00 00 00 6b 6f 73  74 61 73 00 00 00 00 00   .....kos tas.....
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```

Ethernet (eth), 14 bytes

P: 11 D: 11 M: 0

Εφαρμογή φίλτρου



TCP 3-way handshake

netlab_http_capture.pcap - Wireshark

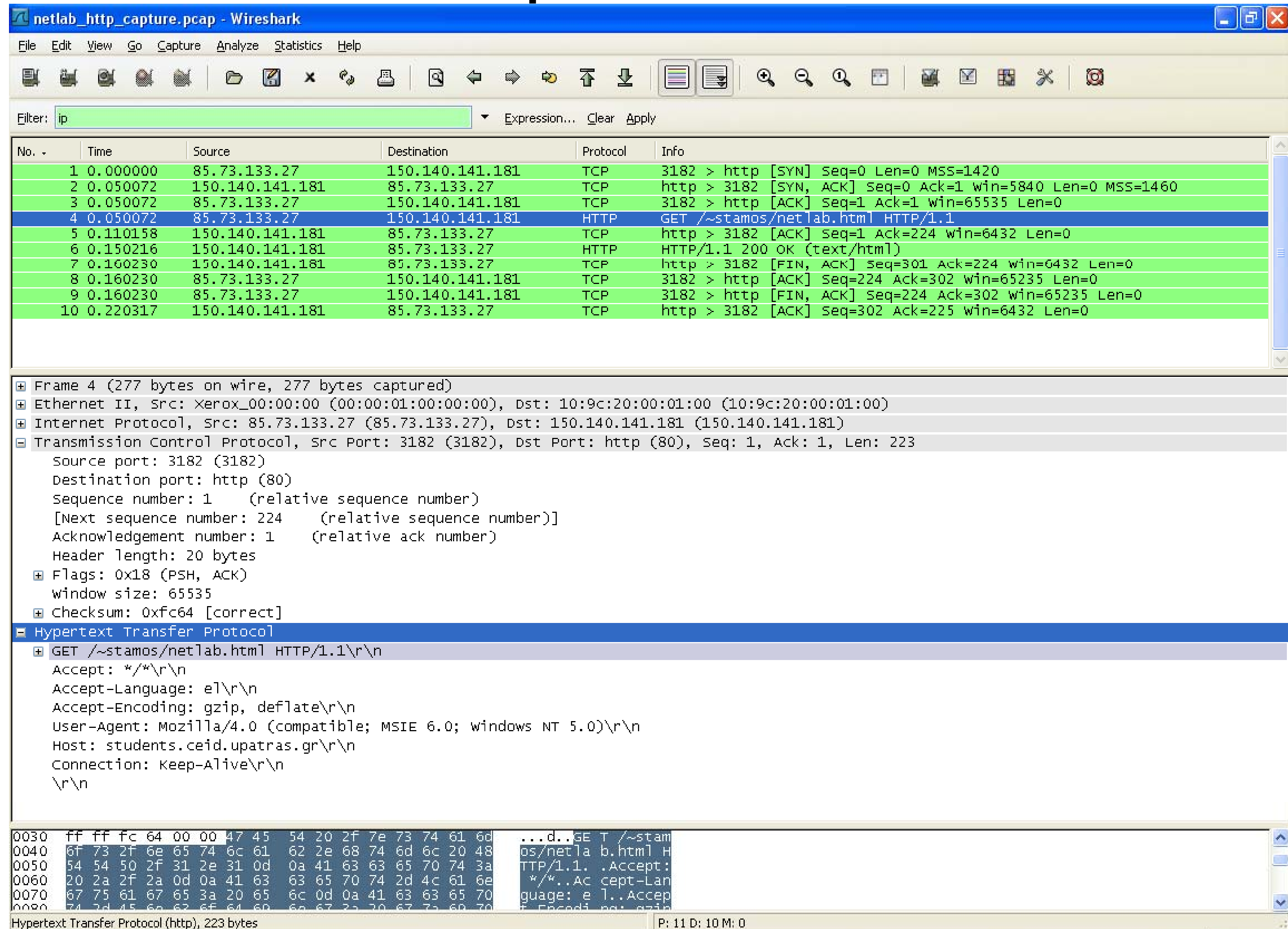
File Edit View Go Capture Analyze Statistics Help

Filter: ip Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.73.133.27	150.140.141.181	TCP	3182 > http [SYN] Seq=0 Len=0 MSS=1420
2	0.050072	150.140.141.181	85.73.133.27	TCP	http > 3182 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
3	0.050072	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.050072	85.73.133.27	150.140.141.181	HTTP	GET /~stamos/netlab.html HTTP/1.1
5	0.110158	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=1 Ack=224 Win=6432 Len=0
6	0.150216	150.140.141.181	85.73.133.27	HTTP	HTTP/1.1 200 OK (text/html)
7	0.160230	150.140.141.181	85.73.133.27	TCP	http > 3182 [FIN, ACK] Seq=301 Ack=224 Win=6432 Len=0
8	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=224 Ack=302 Win=65235 Len=0
9	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [FIN, ACK] Seq=224 Ack=302 Win=65235 Len=0
10	0.220317	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=302 Ack=225 Win=6432 Len=0

```
⊞ Frame 2 (62 bytes on wire, 62 bytes captured)
⊞ Ethernet II, Src: 10:9c:20:00:01:00 (10:9c:20:00:01:00), Dst: Xerox_00:00:00 (00:00:01:00:00:00)
⊞ Internet Protocol, Src: 150.140.141.181 (150.140.141.181), Dst: 85.73.133.27 (85.73.133.27)
⊟ Transmission Control Protocol, Src Port: http (80), Dst Port: 3182 (3182), Seq: 0, Ack: 1, Len: 0
    Source port: http (80)
    Destination port: 3182 (3182)
    Sequence number: 0    (relative sequence number)
    Acknowledgement number: 1    (relative ack number)
    Header length: 28 bytes
  ⊟ Flags: 0x12 (SYN, ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgment: Set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
      .... ..1. = Syn: Set
      .... ...0 = Fin: Not set
    Window size: 5840
  ⊞ Checksum: 0x5e7d [correct]
  ⊞ Options: (8 bytes)
```

HTTP GET request



HTTP response

netlab_http_capture.pcap - Wireshark

File Edit View Go Capture Analyze Statistics Help

Filter: ip Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.73.133.27	150.140.141.181	TCP	3182 > http [SYN] Seq=0 Len=0 MSS=1420
2	0.050072	150.140.141.181	85.73.133.27	TCP	http > 3182 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
3	0.050072	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.050072	85.73.133.27	150.140.141.181	HTTP	GET /~stamos/netlab.html HTTP/1.1
5	0.110158	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=1 Ack=224 Win=6432 Len=0
6	0.150216	150.140.141.181	85.73.133.27	HTTP	HTTP/1.1 200 OK (text/html)
7	0.160230	150.140.141.181	85.73.133.27	TCP	http > 3182 [FIN, ACK] Seq=301 Ack=224 Win=6432 Len=0
8	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=224 Ack=302 Win=65235 Len=0
9	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [FIN, ACK] Seq=224 Ack=302 Win=65235 Len=0
10	0.220317	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=302 Ack=225 Win=6432 Len=0

```
Frame 6 (354 bytes on wire, 354 bytes captured)
Ethernet II, Src: 10:9c:20:00:01:00 (10:9c:20:00:01:00), Dst: Xerox_00:00:00 (00:00:01:00:00:00)
Internet Protocol, Src: 150.140.141.181 (150.140.141.181), Dst: 85.73.133.27 (85.73.133.27)
Transmission Control Protocol, Src Port: http (80), Dst Port: 3182 (3182), Seq: 1, Ack: 224, Len: 300
    Source port: http (80)
    Destination port: 3182 (3182)
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 301    (relative sequence number)]
    Acknowledgement number: 224    (relative ack number)
    Header length: 20 bytes
  Flags: 0x18 (PSH, ACK)
    Window size: 6432
  Checksum: 0x02c6 [correct]
Hypertext Transfer Protocol
  HTTP/1.1 200 OK\r\n
    Date: Wed, 14 Nov 2007 19:15:47 GMT\r\n
    Server: Apache/2.2.3 (Red Hat)\r\n
    Last-Modified: Wed, 14 Nov 2007 19:14:20 GMT\r\n
    ETag: "618289-2f-43ee85f5b5b00"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 47
    Connection: close\r\n
    Content-Type: text/html\r\n
    \r\n
Line-based text data: text/html
```

```
0130  0a 0d 0a 54 65 73 74 20  70 61 67 65 20 66 6f 72   ...Test  page for
0140  20 4e 65 74 6c 61 62 20  75 73 61 67 65 2e 0a 4b    Netlab  usage..K
0150  6f 73 74 61 73 20 53 74  61 6d 6f 73 20 32 30 30   ostas St amos 200
0160  37 0a                                              7.
```

Line-based text data (data-text-lines), 47 bytes | P: 11 D: 10 M: 0

TCP connection termination

Filter: ip   Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
1	0.000000	85.73.133.27	150.140.141.181	TCP	3182 > http [SYN] Seq=0 Len=0 MSS=1420
2	0.050072	150.140.141.181	85.73.133.27	TCP	http > 3182 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
3	0.050072	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.050072	85.73.133.27	150.140.141.181	HTTP	GET /~stamos/netlab.html HTTP/1.1
5	0.110158	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=1 Ack=224 Win=6432 Len=0
6	0.150216	150.140.141.181	85.73.133.27	HTTP	HTTP/1.1 200 OK (text/html)
7	0.160230	150.140.141.181	85.73.133.27	TCP	http > 3182 [FIN, ACK] Seq=301 Ack=224 Win=6432 Len=0
8	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [ACK] Seq=224 Ack=302 Win=65235 Len=0
9	0.160230	85.73.133.27	150.140.141.181	TCP	3182 > http [FIN, ACK] Seq=224 Ack=302 Win=65235 Len=0
10	0.220317	150.140.141.181	85.73.133.27	TCP	http > 3182 [ACK] Seq=302 Ack=225 Win=6432 Len=0

```
⊞ Frame 9 (54 bytes on wire, 54 bytes captured)
⊞ Ethernet II, Src: Xerox_00:00:00 (00:00:01:00:00:00), Dst: 10:9c:20:00:01:00 (10:9c:20:00:01:00)
⊞ Internet Protocol, Src: 85.73.133.27 (85.73.133.27), Dst: 150.140.141.181 (150.140.141.181)
⊟ Transmission Control Protocol, Src Port: 3182 (3182), Dst Port: http (80), Seq: 224, Ack: 302, Len: 0
    Source port: 3182 (3182)
    Destination port: http (80)
    Sequence number: 224    (relative sequence number)
    Acknowledgement number: 302    (relative ack number)
    Header length: 20 bytes
  ⊟ Flags: 0x11 (FIN, ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgment: Set
      .... 0... = Push: Not set
      .... .0.. = Reset: Not set
      .... ..0. = Syn: Not set
      .... ...1 = Fin: Set
    Window size: 65235
  ⊞ Checksum: 0xa130 [correct]
```

Capture file