Pranit Tandon

RA1911028010014 (CSE – H2)

**Computer Networks Lab Exercise No:1 Assignment**

**STUDY OF HEADER FILES WITH RESPECT TO SOCKET PROGRAMMING**

# stdio.h:

It has standard input and output library providing simple and efficient buffered stream IO interface. Some of the functionality described on this reference page extends the ISO C standard.

.

The <stdio.h> header shall define the following data types through typedef: FILE A structure containing information about a file.

fpos\_t A non-array type containing all information needed to specify uniquely every position within a file.

off\_t As described in <sys/types.h>. size\_t As described in <stddef.h>. ssize\_t As described in <sys/types.h>. va\_list As described in <stdarg.h>.

The <stdio.h> header shall define the following macros which shall expand to integer constant expressions:

BUFSIZ Size of <stdio.h> buffers. This shall expand to a positive value. L\_ctermid Maximum size of character array to hold ctermid()

output.

L\_tmpnam Maximum size of character array to hold tmpnam() output.

The <stdio.h> header shall define the following macros which shall expand to integer constant expressions with distinct values:

\_IOFBF Input/output fully buffered.

\_IOLBF Input/output line buffered.

\_IONBF Input/output unbuffered.

The <stdio.h> header shall define the following macros which shall expand to integer constant expressions with distinct values:

SEEK\_CUR Seek relative to current position. SEEK\_END Seek relative to end-of-file.

SEEK\_SET Seek relative to start-of-file.

The <stdio.h> header shall define the following macros which shall expand to integer constant expressions denoting implementation limits:

{FILENAME\_MAX}

Maximum size in bytes of the longest pathname that the implementation guarantees can be opened.

{FOPEN\_MAX} Number of streams which the implementation guarantees can be open simultaneously. The value is at least eight.

{TMP\_MAX} Minimum number of unique filenames generated by tmpnam(). Maximum number of times an application

can call tmpnam() reliably. The value of {TMP\_MAX} is at least 25.

On XSI-conformant systems, the value of {TMP\_MAX} is at least 10000.

The <stdio.h> header shall define the following macro which shall expand to an integer constant expression with type int and a negative value:

EOF End-of-file return value.

The <stdio.h> header shall define NULL as described in <stddef.h>.

The <stdio.h> header shall define the following macro which shall expand to a string constant: P\_tmpdir Default directory prefix for tempnam().

The <stdio.h> header shall define the following macros which shall expand to expressions of type ``pointer to FILE'' that point to the FILE objects associated, respectively, with the standard error, input, and output streams:

stderr Standard error output stream. stdin Standard input stream.

stdout Standard output stream.

# unistd.h:

It is a POSIX standard for open system interface. [Portable Operating System Interface]. The <unistd.h> header defines miscellaneous symbolic constants and types, and declares miscellaneous functions.

The <unistd.h> header defines miscellaneous symbolic constants and types, and declares miscellaneous functions. The contents of this header are shown below.

# Version Test Macros

The following symbolic constants are defined:

\_POSIX\_VERSION

Integer value indicating version of the ISO POSIX-1 standard (C language binding).

\_POSIX2\_VERSION

Integer value indicating version of the ISO POSIX-2 standard (Commands).

\_POSIX2\_C\_VERSION

Integer value indicating version of the ISO POSIX-2 standard (C language binding).

\_XOPEN\_VERSION

Integer value indicating version of the X/Open Portability Guide to which the implementation conforms.

\_POSIX\_VERSION is defined in the ISO POSIX-1 standard. It changes with each new version of the ISO POSIX-1 standard.

\_POSIX2\_VERSION is defined to have the value of the ISO POSIX-2 standard's POSIX2\_VERSION limit. It changes with each new version of the ISO POSIX-2 standard.

\_POSIX2\_C\_VERSION is defined in the ISO POSIX-2 standard. It changes with each new version of the ISO POSIX-2 standard. When the C language binding option of the ISO POSIX-2 standard and therefore the X/Open POSIX2 C-language Binding Feature Group is not supported, \_POSIX2\_C\_VERSION will be set to -1.

\_XOPEN\_VERSION is defined as an integer value equal to 500.

\_XOPEN\_XCU\_VERSION is defined as an integer value indicating the version of the XCU specification to which the implementation conforms. If the value is -1, no commands and utilities are provided on the implementation. If the value is greater than or equal to 4, the functionality associated with the following symbols is also supported (see Mandatory Symbolic Constants and Constants for Options and Feature Groups

):

\_POSIX2\_C\_BIND

\_POSIX2\_C\_VERSION

\_POSIX2\_CHAR\_TERM

\_POSIX2\_LOCALEDEF

\_POSIX2\_UPE

\_POSIX2\_VERSION

If this constant is not defined use the sysconf() function to determine which features are supported.

# Mandatory Symbolic Constants

Although all implementations conforming to this specification support all of the FIPS features described below, there may be system-dependent or file-system-dependent configuration procedures that can remove or modify any or all of these features. Such configurations should not be made if strict FIPS compliance is required.

The following symbolic constants are either undefined or defined with a value other than -1. If a constant is undefined, an application should use the sysconf(), pathconf() or fpathconf() functions to determine which features are present on the system at that time or for the particular pathname in question.

\_POSIX\_CHOWN\_RESTRICTED

The use of chown() is restricted to a process with appropriate privileges, and to changing the group ID of a file only to the effective group ID of the process or to one of its supplementary group IDs.

\_POSIX\_NO\_TRUNC

Pathname components longer than {NAME\_MAX} generate an error.

\_POSIX\_VDISABLE

Terminal special characters defined in <termios.h> can be disabled using this character value.

\_POSIX\_SAVED\_IDS

Each process has a saved set-user-ID and a saved set-group-ID.

\_POSIX\_JOB\_CONTROL

Implementation supports job control.

\_POSIX\_CHOWN\_RESTRICTED, \_POSIX\_NO\_TRUNC and \_POSIX\_VDISABLE

will have values other than -1.

The following symbolic constants are always defined to unspecified values to indicate that this functionality from the POSIX Threads Extension is always present on XSI- conformant systems:

\_POSIX\_THREADS

The implementation supports the threads option.

\_POSIX\_THREAD\_ATTR\_STACKADDR

The implementation supports the thread stack address attribute option.

\_POSIX\_THREAD\_ATTR\_STACKSIZE

The implementation supports the thread stack size attribute option.

\_POSIX\_THREAD\_PROCESS\_SHARED

The implementation supports the process-shared synchronisation option.

\_POSIX\_THREAD\_SAFE\_FUNCTIONS

The implementation supports the thread-safe functions option. Constants for Options and Feature Groups

The following symbolic constants are defined to have the value -1 if the implementation will never provide the feature, and to have a value other than -1 if the implementation always provides the feature. If these are undefined, the sysconf() function can be used to determine whether the feature is provided for a particular invocation of the application.

\_POSIX2\_C\_BIND

Implementation supports the C Language Binding option. This will always have a value other than -1.

\_POSIX2\_C\_DEV

Implementation supports the C Language Development Utilities option.

\_POSIX2\_CHAR\_TERM

Implementation supports at least one terminal type.

\_POSIX2\_FORT\_DEV

Implementation supports the FORTRAN Development Utilities option.

\_POSIX2\_FORT\_RUN

Implementation supports the FORTRAN Run-time Utilities option.

\_POSIX2\_LOCALEDEF

Implementation supports the creation of locales by the localedef utility.

\_POSIX2\_SW\_DEV

Implementation supports the Software Development Utilities option.

\_POSIX2\_UPE

The implementation supports the User Portability Utilities option.

\_XOPEN\_CRYPT

The implementation supports the X/Open Encryption Feature Group.

\_XOPEN\_ENH\_I18N

The implementation supports the Issue 4, Version 2 Enhanced Internationalisation Feature Group. This is always set to a value other than -1.

\_XOPEN\_LEGACY

The implementation supports the Legacy Feature Group.

\_XOPEN\_REALTIME

The implementation supports the X/Open Realtime Feature Group.

\_XOPEN\_REALTIME\_THREADS

The implementation supports the X/Open Realtime Threads Feature Group.

\_XOPEN\_SHM

The implementation supports the Issue 4, Version 2 Shared Memory Feature Group. This is always set to a value other than -1.

\_XBS5\_ILP32\_OFF32

Implementation provides a C-language compilation environment with 32-bit int, long, pointer and off\_t types.

\_XBS5\_ILP32\_OFFBIG

Implementation provides a C-language compilation environment with 32-bit int, long and pointer types and an off\_t type using at least 64 bits.

\_XBS5\_LP64\_OFF64

Implementation provides a C-language compilation environment with 32-bit int and 64- bit long, pointer and off\_t types.

\_XBS5\_LPBIG\_OFFBIG

Implementation provides a C-language compilation environment with an int type using at least 32 bits and long, pointer and off\_t types using at least 64 bits.

If \_XOPEN\_REALTIME is defined to have a value other than -1, then the following symbolic constants will be defined to an unspecified value to indicate that the features are supported.

\_POSIX\_ASYNCHRONOUS\_IO

Implementation supports the Asynchronous Input and Output option.

\_POSIX\_MEMLOCK

Implementation supports the Process Memory Locking option.

\_POSIX\_MEMLOCK\_RANGE

Implementation supports the Range Memory Locking option.

\_POSIX\_MESSAGE\_PASSING

Implementation supports the Message Passing option.

\_POSIX\_PRIORITY\_SCHEDULING

Implementation supports the Process Scheduling option.

\_POSIX\_REALTIME\_SIGNALS

Implementation supports the Realtime Signals Extension option.

\_POSIX\_SEMAPHORES

Implementation supports the Semaphores option.

\_POSIX\_SHARED\_MEMORY\_OBJECTS

Implementation supports the Shared Memory Objects option.

\_POSIX\_SYNCHRONIZED\_IO

Implementation supports the Synchronised Input and Output option.

\_POSIX\_TIMERS

Implementation supports the Timers option.

The following symbolic constants are always defined to unspecified values to indicate that the functionality is always present on XSI-conformant systems.

\_POSIX\_FSYNC

Implementation supports the File Synchronisation option.

\_POSIX\_MAPPED\_FILES

Implementation supports the Memory Mapped Files option.

\_POSIX\_MEMORY\_PROTECTION

Implementation supports the Memory Protection option.

The following symbolic constant will be defined if the option is supported; otherwise, it will be undefined:

\_POSIX\_PRIORITIZED\_IO

Implementation supports the Prioritized Input and Output option.

If \_XOPEN\_REALTIME\_THREADS is defined to have a value other than -1, then the following symbolic constants will be defined to an unspecified value to indicate that the features are supported:

\_POSIX\_THREAD\_PRIORITY\_SCHEDULING

The implementation supports the thread execution scheduling option.

\_POSIX\_THREAD\_PRIO\_INHERIT

The implementation supports the priority inheritance option.

\_POSIX\_THREAD\_PRIO\_PROTECT

The implementation supports the priority protection option. Execution-time Symbolic Constants

If any of the following constants are not defined in the header <unistd.h>, the value varies depending on the file to which it is applied.

If any of the following constants are defined to have value -1 in the header <unistd.h>, the implementation will not provide the option on any file; if any are defined to have a value other than -1 in the header <unistd.h>, the implementation will provide the option on all applicable files.

All of the following constants, whether defined in <unistd.h> or not, may be queried with respect to a specific file using the pathconf() or fpathconf() functions.

\_POSIX\_ASYNC\_IO

Asynchronous input or output operations may be performed for the associated file.

\_POSIX\_PRIO\_IO

Prioritized input or output operations may be performed for the associated file.

\_POSIX\_SYNC\_IO

Synchronised input or output operations may be performed for the associated file.

# string.h:

This header file is used to perform string manipulation operations on NULL terminated strings. The <string.h> header shall define NULL and **size\_t** as described in **<stddef.h>**.

The <string.h> header shall define the **locale\_t** type as described in **<locale.h>.**

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided for use with ISO C standard compilers.

void \*memccpy(void \*restrict, const void \*restrict, int, size\_t); void \*memchr(const void \*, int, size\_t);

int memcmp(const void \*, const void \*, size\_t);

void \*memcpy(void \*restrict, const void \*restrict, size\_t); void \*memmove(void \*, const void \*, size\_t);

void \*memset(void \*, int, size\_t);

char \*stpcpy(char \*restrict, const char \*restrict);

char \*stpncpy(char \*restrict, const char \*restrict, size\_t); char \*strcat(char \*restrict, const char \*restrict);

char \*strchr(const char \*, int);

int strcmp(const char \*, const char \*); int strcoll(const char \*, const char \*);

int strcoll\_l(const char \*, const char \*, locale\_t); char \*strcpy(char \*restrict, const char \*restrict); size\_t strcspn(const char \*, const char \*);

char \*strdup(const char \*); char \*strerror(int);

char \*strerror\_l(int, locale\_t);

int strerror\_r(int, char \*, size\_t); size\_t strlen(const char \*);

char \*strncat(char \*restrict, const char \*restrict, size\_t); int strncmp(const char \*, const char \*, size\_t);

char \*strncpy(char \*restrict, const char \*restrict, size\_t); char \*strndup(const char \*, size\_t);

size\_t strnlen(const char \*, size\_t);

char \*strpbrk(const char \*, const char \*); char \*strrchr(const char \*, int);

char \*strsignal(int);

size\_t strspn(const char \*, const char \*); char \*strstr(const char \*, const char \*);

char \*strtok(char \*restrict, const char \*restrict);

char \*strtok\_r(char \*restrict, const char \*restrict, char \*\*restrict); size\_t strxfrm(char \*restrict, const char \*restrict, size\_t);

size\_t strxfrm\_l(char \*restrict, const char \*restrict, size\_t, locale\_t);

Inclusion of the <string.h> header may also make visible all symbols from <stddef.h>.

# stdlib.h:

stdlib.h — standard library definitions. This header file contains the utility functions such as string conversion routines, memory allocation routines, random number generator, etc.

The <stdlib.h> header shall define the following macros which shall expand to integer constant expressions:

EXIT\_FAILURE

Unsuccessful termination for exit(); evaluates to non-zero value.

EXIT\_SUCCESS

Successful termination for exit(); evaluates to 0.

{RAND\_MAX}

Maximum value returned by rand(); at least 32767.

The <stdlib.h> header shall define the following macro which shall expand to a positive integer expression with type size\_t:

{MB\_CUR\_MAX}

Maximum number of bytes in a character specified by the current locale (category LC\_CTYPE).

In the POSIX locale the value of {MB\_CUR\_MAX} shall be 1.

The <stdlib.h> header shall define NULL as described in <stddef.h>.

The <stdlib.h> header shall define the following data types through typedef: div\_t Structure type returned by the div() function.

ldiv\_t Structure type returned by the ldiv() function. lldiv\_t Structure type returned by the lldiv() function. size\_t As described in <stddef.h>.

wchar\_t As described in <stddef.h>.

Inclusion of the <stdlib.h> header may also make visible all symbols from <stddef.h>,

<limits.h>, <math.h>, and <sys/wait.h>.

# sys/types.h:

Defines the data type of socket address structure in unsigned long. The <sys/types.h> header contains a number of basic derived types that should be used whenever appropriate. In particular, the following are of special interest:

clock\_t

The type clock\_t represents the system times in clock ticks.

dev\_t

The type dev\_t is used for device numbers.

off\_t

The type off\_t is used for file sizes and offsets.

ptrdiff\_t

The type ptrdiff\_t is the signed integral type for the result of subtracting two pointers.

size\_t

The type size\_t is for the size, in bytes, of objects in memory.

ssize\_t

The signed size type ssize\_t is used by functions that return a count of bytes or an error indication.

time\_t

The type time\_t is used for time in seconds.

# sys/socket.h:

The socket functions can be defined as taking pointers to the generic socket address structure called sockaddr. <sys/socket.h> makes available a type, socklen\_t, which is an unsigned opaque integral type of length of at least 32 bits. To forestall portability problems, it is recommended that applications should not use values larger than 232 - 1.

The <sys/socket.h> header defines the unsigned integral type sa\_family\_t.

The <sys/socket.h> header defines the sockaddr structure that includes at least the following members:

sa\_family\_t sa\_family address family

char sa\_data[] socket address (variable-length data)

The <sys/socket.h> header defines the msghdr structure that includes at least the following members:

void \*msg\_name optional address socklen\_t msg\_namelen size of address struct iovec \*msg\_iov scatter/gather array

int msg\_iovlen members in msg\_iov

void \*msg\_control ancillary data, see below socklen\_t msg\_controllen ancillary data buffer len int msg\_flags flags on received message

The <sys/socket.h> header defines the cmsghdr structure that includes at least the following members:

socklen\_t cmsg\_len data byte count, including the cmsghdr int cmsg\_level originating protocol

int cmsg\_type protocol-specific type

Ancillary data consists of a sequence of pairs, each consisting of a cmsghdr structure followed by a data array. The data array contains the ancillary data message, and the cmsghdr structure contains descriptive information that allows an application to correctly parse the data.

The values for cmsg\_level will be legal values for the level argument to the getsockopt() and setsockopt() functions. The system documentation should specify the cmsg\_type definitions for the supported protocols.

Ancillary data is also possible at the socket level. The <sys/socket.h> header defines the following macro for use as the cmsg\_type value when cmsg\_level is SOL\_SOCKET:

SCM\_RIGHTS: Indicates that the data array contains the access rights to be sent or received.

The <sys/socket.h> header defines the following macros to gain access to the data arrays in the ancillary data associated with a message header:

CMSG\_DATA(cmsg): If the argument is a pointer to a cmsghdr structure, this macro returns an unsigned character pointer to the data array associated with the cmsghdr structure.

CMSG\_NXTHDR(mhdr,cmsg): If the first argument is a pointer to a msghdr structure and the second argument is a pointer to a cmsghdr structure in the ancillary data, pointed to by the msg\_control field of that msghdr structure, this macro returns a pointer to the next cmsghdr structure, or a null pointer if this structure is the last cmsghdr in the ancillary data. CMSG\_FIRSTHDR(mhdr)

If the argument is a pointer to a msghdr structure, this macro returns a pointer to the first cmsghdr structure in the ancillary data associated with this msghdr structure, or a null pointer if there is no ancillary data associated with the msghdr structure.

The <sys/socket.h> header defines the linger structure that includes at least the following members:

|  |  |  |
| --- | --- | --- |
| int | l\_onoff | indicates whether linger option is enabled |
| int | l\_linger | linger time, in seconds |

The <sys/socket.h> header defines the following macros, with distinct integral values: SOCK\_DGRAM: Datagram socket

SOCK\_STREAM: Byte-stream socket SOCK\_SEQPACKET: Sequenced-packet socket

The <sys/socket.h> header defines the following macro for use as the level argument of setsockopt() and getsockopt().

SOL\_SOCKET: Options to be accessed at socket level, not protocol level.

The <sys/socket.h> header defines the following macros, with distinct integral values, for use as the option\_name argument in getsockopt() or setsockopt() calls:

SO\_ACCEPTCONN: Socket is accepting connections. SO\_BROADCAST: Transmission of broadcast messages is supported. SO\_DEBUG: Debugging information is being recorded.

SO\_DONTROUTE: bypass normal routing SO\_ERROR: Socket error status.

SO\_KEEPALIVE: Connections are kept alive with periodic messages. SO\_LINGER: Socket lingers on close.

SO\_OOBINLINE: Out-of-band data is transmitted in line. SO\_RCVBUF: Receive buffer size.

SO\_RCVLOWAT: receive "low water mark" SO\_RCVTIMEO: receive timeout

SO\_REUSEADDR: Reuse of local addresses is supported. SO\_SNDBUF: Send buffer size.

SO\_SNDLOWAT: send "low water mark" SO\_SNDTIMEO: send timeout SO\_TYPE: Socket type

The <sys/socket.h> header defines the following macros, with distinct integral values, for use as the valid values for the msg\_flags field in the msghdr structure, or the flags parameter in recvfrom(), recvmsg(), sendto() or sendmsg() calls:

MSG\_CTRUNC: Control data truncated. MSG\_DONTROUTE: Send without using routing tables.

MSG\_EOR: Terminates a record (if supported by the protocol). MSG\_OOB: Out-of-band data.

MSG\_PEEK: Leave received data in queue. MSG\_TRUNC: Normal data truncated.

MSG\_WAITALL: Wait for complete message.

The <sys/socket.h> header defines the following macros, with distinct integral values: AF\_UNIX: UNIX domain sockets

AF\_UNSPEC: Unspecified

AF\_INET: Internet domain sockets

The <sys/socket.h> header defines the following macros, with distinct integral values: SHUT\_RD: Disables further receive operations.

SHUT\_WR: Disables further send operations. SHUT\_RDWR: Disables further send and receive operations.

# netinet/in.h:

Defines the IPv4 socket address structure commonly called Internet socket address structure called sockaddr\_in. When header file <netinet/in.h> is included, the following types are defined through typedef.

in\_port\_t : An unsigned integral type of exactly 16 bits. in\_addr\_t : An unsigned integral type of exactly 32 bits.

The <netinet/in.h> header defines the in\_addr structure that includes at least the following member:

in\_addr\_t s\_addr

The <netinet/in.h> header defines the sockaddr\_in structure that includes at least the following member:

sa\_family\_t sin\_family in\_port\_t sin\_port struct in\_addr sin\_addr unsigned char sin\_zero[8]

The sockaddr\_in structure is used to store addresses for the Internet protocol family. Values of this type must be cast to struct sockaddr for use with the socket interfaces defined in this document.

The <netinet/in.h> header defines the type sa\_family\_t as described in <sys/socket.h>.

The <netinet/in.h> header defines the following macros for use as values of the level argument of getsockopt() and setsockopt():

IPPROTO\_IP: Dummy for IP. IPPROTO\_ICMP: Control message protocol. IPPROTO\_TCP: TCP.

IPPROTO\_UDP: User datagram protocol.

The <netinet/in.h> header defines the following macros for use as destination addresses for connect(), sendmsg() and sendto():

INADDR\_ANY: Local host address. INADDR\_BROADCAST: Broadcast address.

# netdb.h:

Defines the structure hostent for using the system call gethostbyname to get the network host entry. The <netdb.h> header may define the in\_port\_t type and the in\_addr\_t type as described in <netinet/in.h>.

The <netdb.h> header shall define the hostent structure that includes at least the following members:

char \*h\_name Official name of the host.

char \*\*h\_aliases A pointer to an array of pointers to alternative host names, terminated by a null pointer.

int h\_addrtype Address type.

int h\_length The length, in bytes, of the address.

char \*\*h\_addr\_list A pointer to an array of pointers to network addresses (in network byte order) for the host, terminated by a null pointer.

The <netdb.h> header shall define the netent structure that includes at least the following members:

char \*n\_name Official, fully-qualified (including the domain) name of the host. char \*\*n\_aliases A pointer to an array of pointers to alternative network names, terminated by a null pointer.

int n\_addrtype The address type of the network. uint32\_t n\_net - The network number, in host byte order.

The uint32\_t type shall be defined as described in <inttypes.h>.

The <netdb.h> header shall define the protoent structure that includes at least the following members:

char \*p\_name Official name of the protocol.

char \*\*p\_aliases A pointer to an array of pointers to alternative protocol names, terminated by a null pointer.

int p\_proto The protocol number.

The <netdb.h> header shall define the servent structure that includes at least the following members:

char \*s\_name Official name of the service.

char \*\*s\_aliases A pointer to an array of pointers to alternative service names, terminated by a null pointer.

int s\_port The port number at which the service resides, in network byte order. char \*s\_proto The name of the protocol to use when contacting the service.

The <netdb.h> header shall define the IPPORT\_RESERVED macro with the value of the highest reserved Internet port number.

When the <netdb.h> header is included, h\_errno shall be available as a modifiable lvalue of type int. It is unspecified whether h\_errno is a macro or an identifier declared with external linkage.

The <netdb.h> header shall define the following macros for use as error values for gethostbyaddr() and gethostbyname():

HOST\_NOT\_FOUND NO\_DATA NO\_RECOVERY TRY\_AGAIN

Address Information Structure

The <netdb.h> header shall define the addrinfo structure that includes at least the following members:

int ai\_flags Input flags.

int ai\_family Address family of socket. int ai\_socktype Socket type.

int ai\_protocol Protocol of socket.

socklen\_t ai\_addrlen Length of socket address. struct sockaddr \*ai\_addr Socket address of socket.

char \*ai\_canonname Canonical name of service location. struct addrinfo \*ai\_next Pointer to next in list.

The <netdb.h> header shall define the following macros that evaluate to bitwise- distinct integer constants for use in the flags field of the addrinfo structure:

AI\_PASSIVE: Socket address is intended for bind(). AI\_CANONNAME: Request for canonical name.

AI\_NUMERICHOST: Return numeric host address as name. AI\_NUMERICSERV: Inhibit service name resolution.

AI\_V4MAPPED: If no IPv6 addresses are found, query for IPv4 addresses and return them to the caller as IPv4-mapped IPv6 addresses.

AI\_ALL: Query for both IPv4 and IPv6 addresses.

AI\_ADDRCONFIG: Query for IPv4 addresses only when an IPv4 address is configured; query for IPv6 addresses only when an IPv6 address is configured.

The <netdb.h> header shall define the following macros that evaluate to bitwise- distinct integer constants for use in the flags argument to getnameinfo():

NI\_NOFQDN: Only the nodename portion of the FQDN is returned for local hosts. NI\_NUMERICHOST: The numeric form of the node's address is returned instead of its name.

NI\_NAMEREQD: Return an error if the node's name cannot be located in the database. NI\_NUMERICSERV: The numeric form of the service address is returned instead of its name.

NI\_NUMERICSCOPE: For IPv6 addresses, the numeric form of the scope identifier is returned instead of its name.

NI\_DGRAM: Indicates that the service is a datagram service (SOCK\_DGRAM).

Address Information Errors

The <netdb.h> header shall define the following macros for use as error values for getaddrinfo() and getnameinfo():

EAI\_AGAIN: The name could not be resolved at this time. Future attempts may succeed.

EAI\_BADFLAGS: The flags had an invalid value. EAI\_FAIL: A non-recoverable error occurred.

EAI\_FAMILY: The address family was not recognized or the address length was invalid for the specified family.

EAI\_MEMORY: There was a memory allocation failure.

EAI\_NONAME: The name does not resolve for the supplied parameters. NI\_NAMEREQD is set and the host's name cannot be located, or both nodename and servname were null.

EAI\_SERVICE: The service passed was not recognized for the specified socket type. EAI\_SOCKTYPE: The intended socket type was not recognized.

EAI\_SYSTEM: A system error occurred. The error code can be found in errno. EAI\_OVERFLOW: An argument buffer overflowed.

Inclusion of the <netdb.h> header may also make visible all symbols from

<netinet/in.h>, <sys/socket.h>, and <inttypes.h>.

# time.h:

Has structures and functions to get the system date and time and to perform time manipulation functions. We use the function ctime(), that is defined in this header file , to calculate the current date and time.

The <time.h> header declares the structure tm, which includes at least the following members:

int tm\_sec seconds [0,61] int tm\_min minutes [0,59] int tm\_hour hour [0,23]

int tm\_mday day of month [1,31] int tm\_mon month of year [0,11] int tm\_year years since 1900

int tm\_wday day of week [0,6] (Sunday = 0) int tm\_yday day of year [0,365]

int tm\_isdst daylight savings flag

The value of tm\_isdst is positive if Daylight Saving Time is in effect, 0 if Daylight Saving Time is not in effect, and negative if the information is not available.

This header defines the following symbolic names:

NULL

Null pointer constant. CLK\_TCK

Number of clock ticks per second returned by the times() function (LEGACY). CLOCKS\_PER\_SEC

A number used to convert the value returned by the clock() function into seconds. The <time.h> header declares the structure timespec, which has at least the following members:

time\_t tv\_sec seconds

long tv\_nsec nanoseconds

This header also declares the itimerspec structure, which has at least the following members:

struct timespec it\_interval timer period struct timespec it\_value timer expiration

The following manifest constants are defined: CLOCK\_REALTIME

The identifier of the systemwide realtime clock. TIMER\_ABSTIME

Flag indicating time is absolute with respect to the clock associated with a timer.

The clock\_t, clockid\_t, size\_t, time\_t and timer\_t types are defined as described in

<sys/types.h>.

Although the value of CLOCKS\_PER\_SEC is required to be 1 million on all XSI- conformant systems, it may be variable on other systems and it should not be assumed that CLOCKS\_PER\_SEC is a compile-time constant.

The value of CLK\_TCK is currently the same as the value of sysconf(\_SC\_CLK\_TCK); however, new applications should call sysconf() because the CLK\_TCK macro may be withdrawn in a future issue.

The <time.h> header provides a declaration for getdate\_err.

The following are declared as functions and may also be defined as macros. Function prototypes must be provided for use with an ISO C compiler.

char \*asctime(const struct tm \*);

char \*asctime\_r(const struct tm \*, char \*); clock\_t clock(void);

int clock\_getres(clockid\_t, struct timespec \*); int clock\_gettime(clockid\_t, struct timespec \*);

int clock\_settime(clockid\_t, const struct timespec \*); char \*ctime(const time\_t \*);

char \*ctime\_r(const time\_t \*, char \*); double difftime(time\_t, time\_t);

struct tm \*getdate(const char \*); struct tm \*gmtime(const time\_t \*);

struct tm \*gmtime\_r(const time\_t \*, struct tm \*); struct tm \*localtime(const time\_t \*);

struct tm \*localtime\_r(const time\_t \*, struct tm \*); time\_t mktime(struct tm \*);

int nanosleep(const struct timespec \*, struct timespec \*); size\_t strftime(char \*, size\_t, const char \*, const struct tm \*); char \*strptime(const char \*, const char \*, struct tm \*); time\_t time(time\_t \*);

int timer\_create(clockid\_t, struct sigevent \*, timer\_t \*); int timer\_delete(timer\_t);

int timer\_gettime(timer\_t, struct itimerspec \*); int timer\_getoverrun(timer\_t);

int timer\_settime(timer\_t, int, const struct itimerspec \*, struct itimerspec \*);

void tzset(void);

The following are declared as variables:

extern int daylight; extern long int timezone; extern char \*tzname[];

Has structures and functions to get the system date and time and to perform time manipulation functions. We use the function ctime(), that is defined in this header file , to calculate the current date and time.

# sys/stat.h:

Contains the structure stat to test a descriptor to see if it is of a specified type. Also it is used to display file or file system status.stat() updates any time related fields.when copying from 1 file to another.

The <sys/stat.h> header shall define the structure of the data returned by the functions

fstat(), lstat(), and stat().

The stat structure shall contain at least the following members:

dev\_t st\_dev Device ID of device containing file. ino\_t st\_ino File serial number.

mode\_t st\_mode Mode of file (see below). nlink\_t st\_nlink Number of hard links to the file. uid\_t st\_uid User ID of file.

gid\_t st\_gid Group ID of file.

dev\_t st\_rdev Device ID (if file is character or block special). off\_t st\_size For regular files, the file size in bytes.

For symbolic links, the length in bytes of the pathname contained in the symbolic link.

For a shared memory object, the length in bytes. For a typed memory object, the length in bytes.

For other file types, the use of this field is unspecified.

time\_t st\_atime Time of last access.

time\_t st\_mtime Time of last data modification. time\_t st\_ctime Time of last status change.

blksize\_t st\_blksize A file system-specific preferred I/O block size for this object. In some file system types, this may vary from file to file.

blkcnt\_t st\_blocks Number of blocks allocated for this object.

The st\_ino and st\_dev fields taken together uniquely identify the file within the system.

The blkcnt\_t, blksize\_t, dev\_t, ino\_t, mode\_t, nlink\_t, uid\_t, gid\_t, off\_t, and time\_t types shall be defined as described in <sys/types.h>. Times shall be given in seconds since the Epoch.

Unless otherwise specified, the structure members st\_mode, st\_ino, st\_dev, st\_uid, st\_gid, st\_atime, st\_ctime, and st\_mtime shall have meaningful values for all file types defined in IEEE Std 1003.1-2001.

For symbolic links, the st\_mode member shall contain meaningful information, which can be used with the file type macros described below, that take a mode argument. The st\_size member shall contain the length, in bytes, of the pathname contained in the symbolic link.

File mode bits and the contents of the remaining members of the stat structure are unspecified. The value returned in the st\_size field shall be the length of the content of the symbolic link, and shall not count a trailing null if one is present.

The following symbolic names for the values of type mode\_t shall also be defined.

File type:

S\_IFMT Type of file. S\_IFBLK Block special. S\_IFCHR Character special. S\_IFIFO FIFO special.

S\_IFREG Regular.

S\_IFDIR Directory. S\_IFLNK Symbolic link. S\_IFSOCK Socket.

File mode bits:

S\_IRWXU Read, write, execute/search by owner. S\_IRUSR Read permission, owner.

S\_IWUSR Write permission, owner.

S\_IXUSR Execute/search permission, owner. S\_IRWXG Read, write, execute/search by group. S\_IRGRP Read permission, group.

S\_IWGRP Write permission, group.

S\_IXGRP Execute/search permission, group. S\_IRWXO Read, write, execute/search by others. S\_IROTH Read permission, others.

S\_IWOTH Write permission, others.

S\_IXOTH Execute/search permission, others. S\_ISUID Set-user-ID on execution.

S\_ISGID Set-group-ID on execution.

S\_ISVTX On directories, restricted deletion flag.

The bits defined by S\_IRUSR, S\_IWUSR, S\_IXUSR, S\_IRGRP, S\_IWGRP, S\_IXGRP, S\_IROTH,

S\_IWOTH, S\_IXOTH, S\_ISUID, S\_ISGID, and S\_ISVTX shall be unique. S\_IRWXU is the bitwise-inclusive OR of S\_IRUSR, S\_IWUSR, and S\_IXUSR. S\_IRWXG is the bitwise-inclusive OR of S\_IRGRP, S\_IWGRP, and S\_IXGRP. S\_IRWXO is the bitwise-inclusive OR of S\_IROTH, S\_IWOTH, and S\_IXOTH.

Implementations may OR other implementation-defined bits into S\_IRWXU, S\_IRWXG, and

S\_IRWXO, but they shall not overlap any of the other bits defined in this volume of IEEE Std 1003.1-2001. The file permission bits are defined to be those corresponding to the bitwise-inclusive OR of S\_IRWXU, S\_IRWXG, and S\_IRWXO.

The following macros shall be provided to test whether a file is of the specified type.

The value m supplied to the macros is the value of st\_mode from a stat structure. The macro shall evaluate to a non-zero value if the test is true; 0 if the test is false.

S\_ISBLK(m)

Test for a block special file.

S\_ISCHR(m)

Test for a character special file.

S\_ISDIR(m)

Test for a directory.

S\_ISFIFO(m)

Test for a pipe or FIFO special file.

S\_ISREG(m)

Test for a regular file.

S\_ISLNK(m)

Test for a symbolic link.

S\_ISSOCK(m)

Test for a socket.

The implementation may implement message queues, semaphores, or shared memory objects as

distinct file types. The following macros shall be provided to test whether a file is

of

the specified type. The value of the buf argument supplied to the macros is a

pointer to a

stat structure. The macro shall evaluate to a non-zero value if the specified object

is stat

implemented as a distinct file type and the specified file type is contained in the structure referenced by buf. Otherwise, the macro shall evaluate to zero.

S\_TYPEISMQ(buf)

Test for a message queue.

S\_TYPEISSEM(buf)

Test for a semaphore.

S\_TYPEISSHM(buf)

Test for a shared memory object.

The implementation may implement typed memory objects as distinct file types, and the following macro shall test whether a file is of the specified type. The value of the buf argument supplied to the macros is a pointer to a stat structure. The macro shall evaluate to a non-zero value if the specified object is implemented as a distinct file type and the specified file type is contained in the stat structure referenced by buf.

Otherwise, the macro shall evaluate to zero.

S\_TYPEISTMO(buf): Test macro for a typed memory object.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

int chmod(const char \*, mode\_t); int fchmod(int, mode\_t);

int fstat(int, struct stat \*);

int lstat(const char \*restrict, struct stat \*restrict); int mkdir(const char \*, mode\_t);

int mkfifo(const char \*, mode\_t);

int mknod(const char \*, mode\_t, dev\_t);

int stat(const char \*restrict, struct stat \*restrict); mode\_t umask(mode\_t);

# sys/ioctl.h:

Macros and defines used in specifying an ioctl request are located in this header file. We use the function ioctl() that is defined in this header file. ioctl() function is used to perform ARP cache operations.

The ioctl() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., terminals) may be controlled with ioctl() requests. The argument fd must be an open file descriptor.

The second argument is a device-dependent request code. The third argument is an untyped pointer to memory. It's traditionally char \*argp (from the days before void \* was valid C), and will be so named for this discussion.

An ioctl() request has encoded in it whether the argument is an in parameter or out parameter, and the size of the argument argp in bytes. Macros and defines used in specifying an ioctl() request are located in the file <sys/ioctl.h>.

Usually, on success zero is returned. A few ioctl() requests use the return value as an output parameter and return a nonnegative value on success. On error, -1 is returned, and errno is set to indicate the error.

Ioctl command values are 32-bit constants. In principle these constants are completely arbitrary, but people have tried to build some structure into them.

The old Linux situation was that of mostly 16-bit constants, where the last byte is a serial number, and the preceding byte(s) give a type indicating the driver. Sometimes the major number was used: 0x03 for the HDIO\_\* ioctls, 0x06 for the LP\* ioctls.

And sometimes one or more ASCII letters were used. For example, TCGETS has value 0x00005401, with 0x54 = 'T' indicating the terminal driver, and CYGETTIMEOUT has value 0x00435906, with 0x43 0x59 = 'C' 'Y' indicating the cyclades driver.

Later (0.98p5) some more information was built into the number. One has 2 direction bits (00: none, 01: write, 10: read, 11: read/write) followed by 14 size bits (giving the size of the argument), followed by an 8-bit type (collecting the ioctls in groups for a common purpose or a common driver), and an 8-bit serial number.

The macros describing this structure live in <asm/ioctl.h> and are \_IO(type,nr) and

{\_IOR,\_IOW,\_IOWR}(type,nr,size). They use sizeof(size) so that size is a misnomer here: this third argument is a data type.

Note that the size bits are very unreliable: in lots of cases they are wrong, either because of buggy macros using sizeof(sizeof(struct)), or because of legacy values.

Thus, it seems that the new structure only gave disadvantages: it does not help in checking, but it causes varying values for the various architectures.

# pcap.h:

Has function definitions that are required for packet capturing. Some of the functions are pcap\_lookupdev(),pcap\_open\_live() and pcap\_loop(). pcap\_lookupdev() is used to initialize the network device.The device to be sniffed is opened using the pcap\_open\_live(). Pcap\_loop() determines the number of packets to be sniffed.

pcap\_init() initializes the library. It takes an argument giving options; currently, the options are:

PCAP\_CHAR\_ENC\_LOCAL

Treat all strings supplied as arguments, and return all strings to the caller, as being in the local character encoding.

PCAP\_CHAR\_ENC\_UTF\_8

Treat all strings supplied as arguments, and return all strings to the caller, as being in UTF- 8.

On UNIX-like systems, the local character encoding is assumed to be UTF-8, so no character encoding transformations are done.On Windows, the local character encoding is the local ANSI code page.

If pcap\_init() is called, the deprecated pcap\_lookupdev() routine always fails, so it should not be used, and, on Windows, pcap\_create() does not attempt to handle UTF-16LE strings.

If pcap\_init() is not called, strings are treated as being in the local ANSI code page on Windows, pcap\_lookupdev() will succeed if there is a device on which to capture, and pcap\_create() makes an attempt to check whether the string passed as an argument is a UTF-16LE string - note that this attempt is unsafe, as it may run past the end of the string

- to handle pcap\_lookupdev() returning a UTF-16LE string. Programs that don't call pcap\_init() should, on Windows, call pcap\_wsockinit() to initialize Winsock; this is not necessary if pcap\_init() is called, as pcap\_init() will initialize Winsock itself on Windows.

# net/if\_arp.h:

Contains the definitions for Address Resolution Protocol. We use this to manipulate the ARP request structure and its data members arp\_pa,arp\_dev and arp\_ha. The arp\_ha structure’s data member sa\_data[ ] has the hardware address.

The STREAMS device /dev/arp is not a Transport Level Interface ("TLI) " transport provider and may not be used with the TLI interface.

To facilitate communications with systems that do not use ARP, ioctl() requests are provided to enter and delete entries in the IP-to-Ethernet tables.

#include <sys/sockio.h> #include <sys/socket.h>

#include <net/if.h> #include <net/if\_arp.h> struct arpreq arpreq;

ioctl(s, SIOCSARP, (caddr\_t)&arpreq); ioctl(s, SIOCGARP, (caddr\_t)&arpreq); ioctl(s, SIOCDARP, (caddr\_t)&arpreq);

Each ioctl() request takes the same structure as an argument. SIOCSARP sets an ARP entry, SIOCGARP gets an ARP entry, and SIOCDARP deletes an ARP entry. These ioctl() requests may be applied to any Internet family socket descriptor s , or to a descriptor for the ARP device, but only by the privileged user.

The arpreq structure contains:

/\*

\* ARP ioctl request

\*/

struct arpreq {

struct sockaddr arp\_pa; /\* protocol address \*/ struct sockaddr arp\_ha; /\* hardware address \*/ int arp\_flags; /\* flags \*/

};

/\* arp\_flags field values \*/

#define ATF\_COM 0x2 /\* completed entry (arp\_ha valid) \*/ #define ATF\_PERM 0x4 /\* permanent entry \*/

#define ATF\_PUBL 0x8 /\* publish (respond for other host) \*/ #define ATF\_USETRAILERS 0x10 /\* send trailer packets to host \*/

The address family for the arp\_pa sockaddr must be AF\_INET ; for the arp\_ha sockaddr , it must be AF\_UNSPEC . The only flag bits that may be written are ATF\_PUBL and ATF\_USETRAILERS. ATF\_PERM makes the entry permanent if the ioctl() request succeeds. The peculiar nature of the ARP tables may cause the ioctl() request to fail if too many permanent IP addresses hash to the same slot. ATF\_PUBL specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an "ARP server", which may be useful in convincing an ARP - only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations. Trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable- sized headers. Hosts that wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; trailer encapsulations are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either host or both may request trailers. The ATF\_USETRAILERS flag records the receipt of such a reply and enables the transmission of trailer packets to that host.

# errno.h:

It sets an error number when an error and that error can be displayed using perror function. It has symbolic error names. The error number is never set to zero by any library function

The <errno.h> header file defines the integer variable errno, which is set by system calls and some library functions in the event of an error to indicate what went wrong.

errno

The value in errno is significant only when the return value of the call indicated an error (i.e., -1 from most system calls; -1 or NULL from most library functions); a function that succeeds is allowed to change errno. The value of errno is never set to zero by any system call or library function.

For some system calls and library functions (e.g., getpriority(2)), -1 is a valid return on success. In such cases, a successful return can be distinguished from an error return by setting errno to zero before the call, and then, if the call returns a status that indicates that an error may have occurred, checking to see if errno has a nonzero value.

errno is defined by the ISO C standard to be a modifiable lvalue of type int, and must not be explicitly declared; errno may be a macro. errno is thread-local; setting it in one thread does not affect its value in any other thread.

Error numbers and names

Valid error numbers are all positive numbers. The <errno.h>header file defines symbolic names for each of the possible error numbers that may appear in errno.

All the error names specified by POSIX.1 must have distinct values, with the exception of EAGAIN and EWOULDBLOCK, which may be the same. On Linux, these two have the same value on all architectures.

The error numbers that correspond to each symbolic name vary across UNIX systems, and even across different architectures on Linux. Therefore, numeric values are not included as part of the list of error names below. The perror(3) and strerror(3) functions can be used to convert these names to corresponding textual error messages.

On any particular Linux system, one can obtain a list of all symbolic error names and the corresponding error numbers using the errno(1) command (part of the moreutils package):

$ errno -l

EPERM 1 Operation not permitted ENOENT 2 No such file or directory ESRCH 3 No such process

EINTR 4 Interrupted system call EIO 5 Input/output error

The errno(1) command can also be used to look up individual error numbers and names, and to search for errors using strings from the error description, as in the following examples:

$ errno 2

ENOENT 2 No such file or directory

$ errno ESRCH

ESRCH 3 No such process

$ errno -s permission

EACCES 13 Permission denied

It sets an error number when an error and that error can be displayed using error function. It has symbolic error names. The error number is never set to zero by any library function.

# arpa/inet.h:

This is used to convert internet addresses between ASCII strings and network byte ordered binary values (values that are stored in socket address structures). It is used for inet\_aton, inet\_addr, inet\_ntoa functions.

The <arpa/inet.h> header makes available the type in\_port\_t and the type in\_addr\_t as defined in the description of <netinet/in.h>.

The <arpa/inet.h> header makes available the in\_addr structure, as defined in the description of <netinet/in.h>.

The following may be declared as functions, or defined as macros, or both: uint32\_t htonl(uint32\_t hostlong);

uint16\_t htons(uint16\_t hostshort);

uint32\_t ntohl(uint32\_t netlong); uint16\_t ntohs(uint16\_t netshort);

The uint32\_t and uint16\_t types are made available by inclusion of <inttypes.h> (see referenced document XSH).

The following are declared as functions, and may also be defined as macros: in\_addr\_t inet\_addr(const char \*cp);

in\_addr\_t inet\_lnaof(struct in\_addr in);

struct in\_addr inet\_makeaddr(in\_addr\_t net, in\_addr\_t lna); in\_addr\_t inet\_netof(struct in\_addr in);

in\_addr\_t inet\_network(const char \*cp); char \*inet\_ntoa(struct in\_addr in);

Inclusion of the <arpa/inet.h> header may also make visible all symbols from <netinet/in.h> and <inttypes.h>.