IPv6 Now!

Jürgen Schönwälder

TU Braunschweig

<schoenw@ibr.cs.tu-bs.de>

IPv6 Now! - Outline

- Why should I care about IPv6?
- Status of IPv6 Development and Deployment
- Porting IPv4 application to IPv6
  - Fundamentals of the socket API
  - IPv6 extensions to the socket API
  - Protocol independent name resolution
  - How to use the theory in practice
  - Considerations for software maintainers
- Further Information

Why should I care about IPv6?

- Enlarged IP Address Space
  - IPv4: 32 Bit (~ 10^9 addresses)
  - IPv6: 128 Bit (~ 10^38 addresses)
- Extensibility
  - Simplified IPv6 header (option header daisy chain)
- Plug and Play
  - Nodes obtain default routes and IPv6 addresses automagically
- Security
  - IPsec (authentication and encryption) is a MUST

But IPv4 can be extended!

- Enlarged IP Address Space
  - Dynamic Address Assignement
  - Network Address Translation
  - Masquerading
- Extensibility
  - Encapsulation, Tunneling, Everything over HTTP
- Plug and Play
  - Dynamic Host Configuration Protocol (DHCP)
- Security
  - IPsec over IPv4

So why should I still care about IPv6?

To answer this question, we first need to understand what Internet Transparency means...

Internet Transparency

- The basic concept of the Internet, established around 1973, is one of transparent transmission of datagrams across an arbitrary network of networks.

- Consequences:
  - logical addresses were unique
  - datagrams were not changed in transit
  - end systems handle error detection, retransmission, security, naming, and binding

- Transparency has gone (for ever)!

>>> Fog on the Internet! <<<

So why should I still care about IPv6?

- Explosion of the Internet will continue:
  - Everyone in the world should have an Internet address
  - It is not a matter of whether, but of when, IPv4 addresses will run out
  - Scarity will lead to rationing, either by arbitrary control or by price
- IPv6 is an architecturally clean solution
- Helps to restore Internet transparency (where possible)
- Overall simplicity results in long term cost reduction

>>> The fog clears! <<<

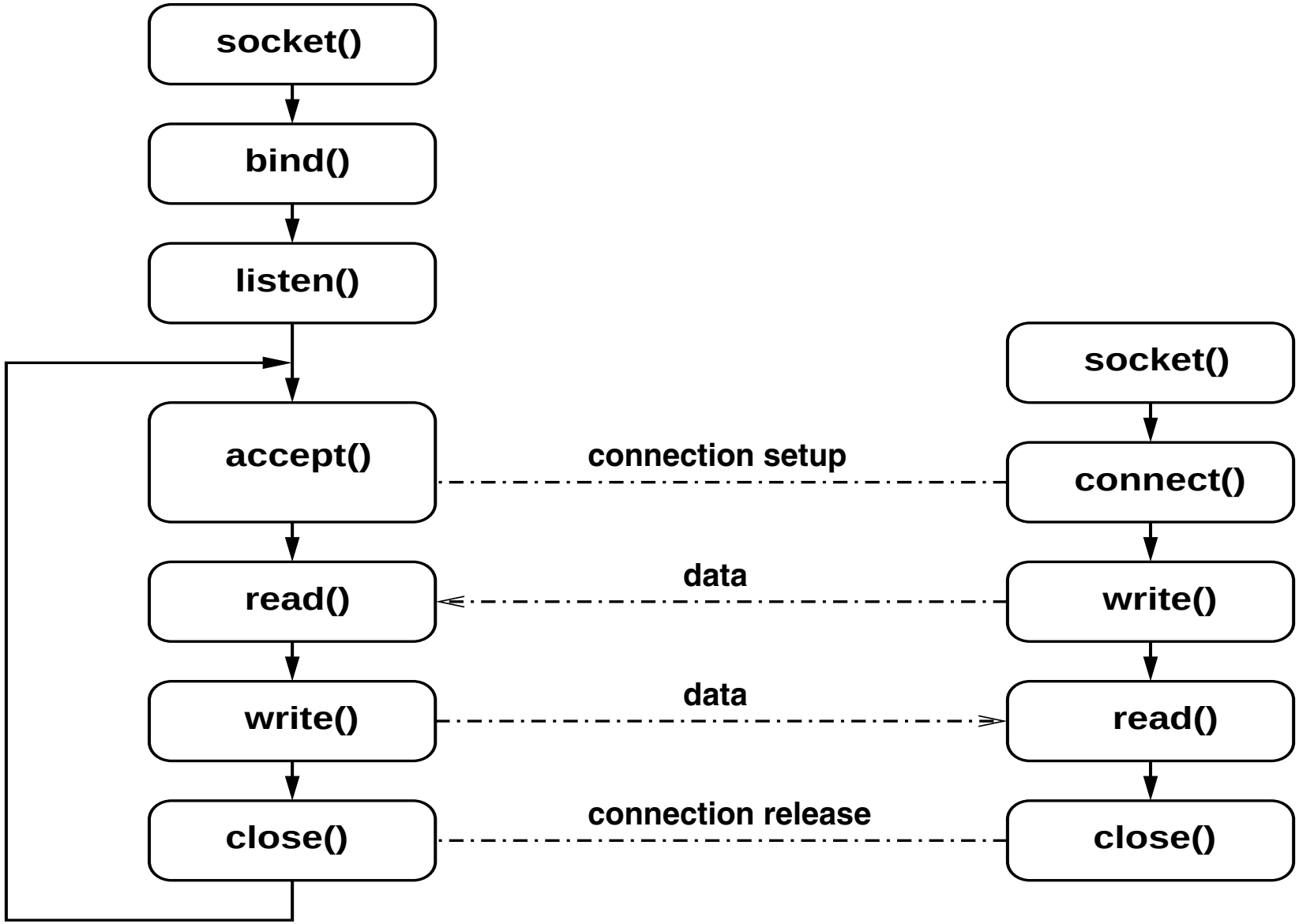
IPv6 Development and Deployment

- History:
  - 1992: Work started on IPng
  - 1993: Fight over the right direction
  - 1994: Agreement on SIPP and finally on IPv6
  - 1995: IPv6 specification (proposed standard)
  - 1996: IPv6 protocol implementations
  - 1997: IPv6 socket API extensions (initial version)
  - 1998: IPv6 specification (draft standard)
  - 1999: IPv6 socket API extensions (revised version)
  - 1999: Delegation of IPv6 address space to regional registries
  - 2000: Availability of IPv6 implementations
- Next Step:
  - Port applications to use IPv6!
  - Build IPv6 networks and let the flower grow!

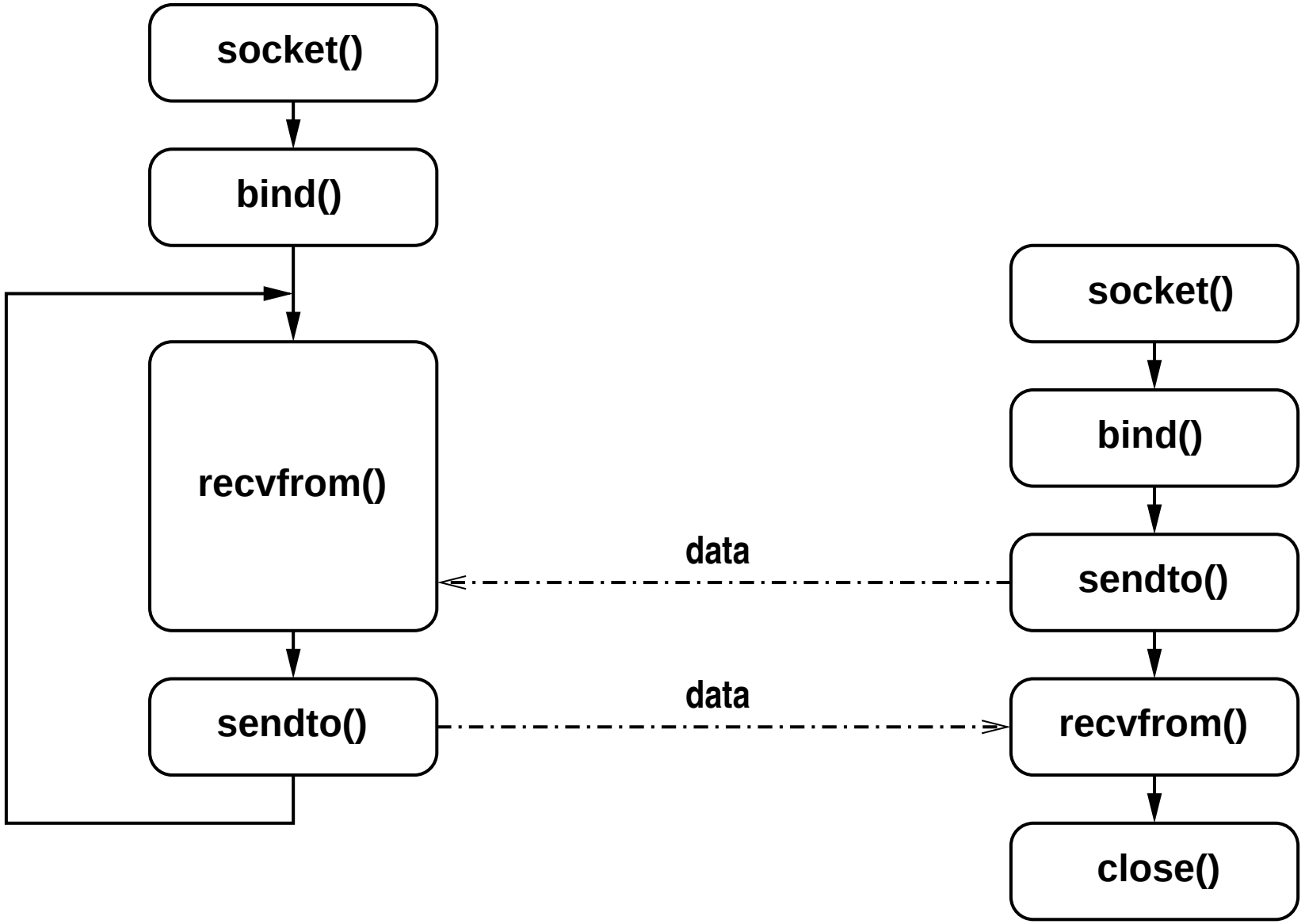
Porting IPv4 Applications to IPv6

- Fundamentals of the socket API
- IPv6 extensions to the socket API
- Protocol independent name resolution
- How to use the theory in practice
- Considerations for software maintainers

TCP Client/Server with Sockets



UDP Client/Server with Sockets



Socket API Summary

```
int socket(int domain, int type, int protocol)
int bind(int socket, struct sockaddr *addr, socklen_t addrlen)
int connect(int socket, struct sockaddr *addr, socklen_t addrlen)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr *addr, socklen_t *addrlen)

ssize_t write(int socket, void *buf, size_t count)
int send(int socket, void *msg, int len, unsigned flags)
int sendto(int socket, void *msg, int len, unsigned flags, struct sockaddr *addr, int addrlen)

ssize_t read(int socket, void *buf, size_t count)
int recv(int socket, void *bug, int len, unsigned flags)
int recvfrom(int socket, void *bug, int len, unsigned flags, struct sockaddr *addr, int addrlen)

int shutdown(int socket, int how)
int close(int socket)

int getsockopt(int socket, int level,
               int optname, void *optval, socklen_t *optlen)
int setsockopt(int socket, int level,
               int optname, void *optval, socklen_t optlen)

int getsockname(int socket, struct sockaddr *addr, socklen_t addrlen)
int getpeername(int socket, struct sockaddr *addr, socklen_t addrlen)
```

>>> The core socket API does not need any changes to support IPv6! <<<

IPv4 and IPv6 Addresses in C

- Structures to represent IPv4 and IPv6 addresses:

```
struct in_addr {
    uint8_t     s_addr[4];   /* IPv4 address */
};

struct in6_addr {
    uint8_t     s6_addr[16]; /* IPv6 address */
};
```

- Functions to convert IPv4/IPv6 addresses to/from numeric format:

```
int inet_pton(int af, const char *src, void *dst);

const char *inet_ntop(int af, const void *src,
                      char *dst, size_t size);
```

Simple Address Conversions Examples

- IPv4 address conversion:

```
char str[INET_ADDRSTRLEN];
struct in_addr inaddr;
int res;

res = inet_pton(AF_INET, argv[i], &inaddr);
ptr = inet_ntop(AF_INET, &inaddr, str, sizeof(str));
```

- IPv6 address conversion:

```
char str[INET6_ADDRSTRLEN];
struct in6_addr in6addr;
int res;

res = inet_pton(AF_INET6, argv[i], &in6addr);
ptr = inet_ntop(AF_INET6, &in6addr, str, sizeof(str));
```

>>> These functions result in IPv4/IPv6 specific code! <<<

>>> It is therefore not recommended to use them. <<<

Socket Addresses in C (Linux)

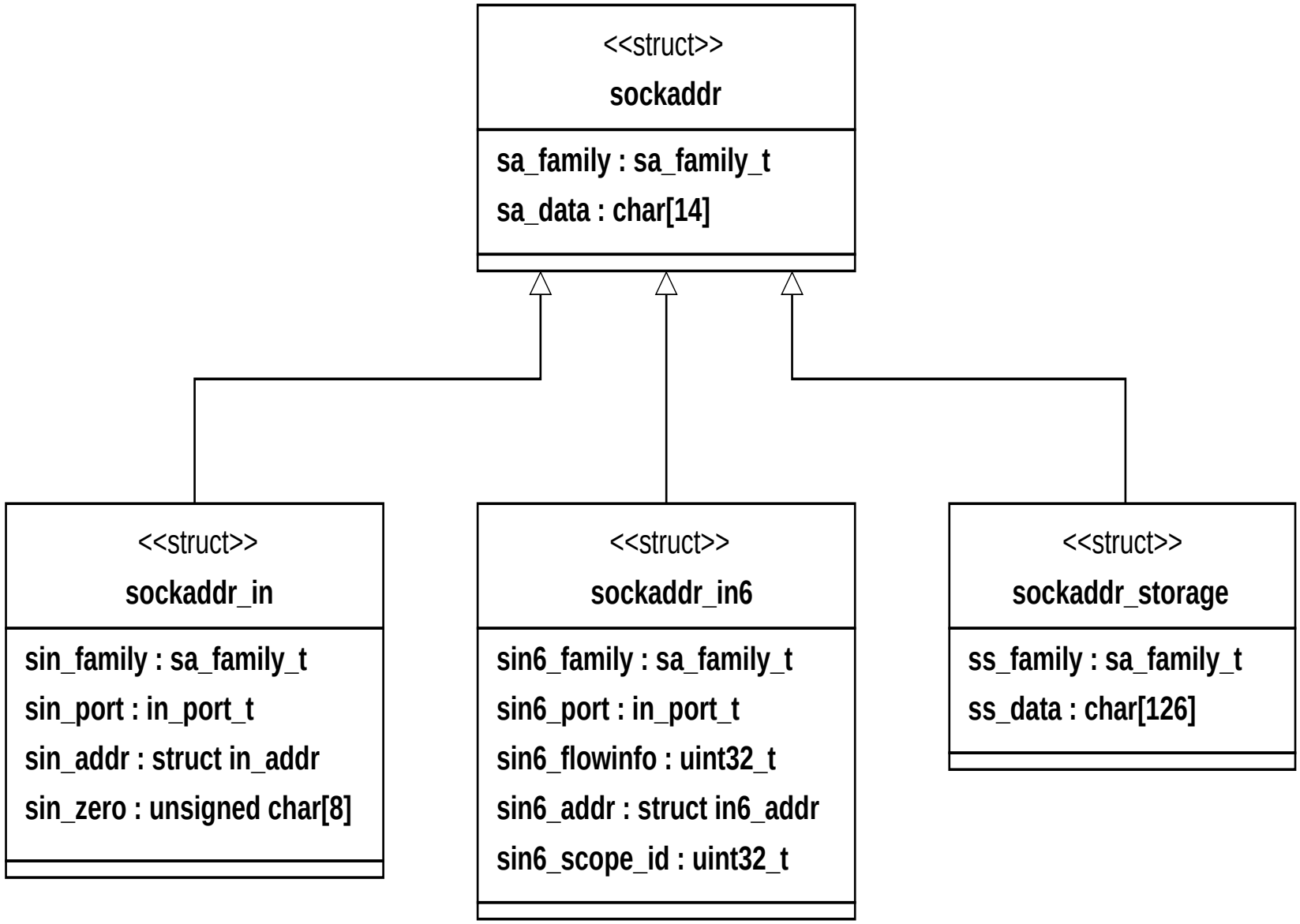
- Classic IPv4 socket address structures:

```
struct sockaddr {
    sa_family_t     sa_family;     /* any address family */
    char            sa_data[14];   /* generic socket address */
};

struct sockaddr_in {
    sa_family_t     sin_family;    /* AF_INET */
    in_port_t       sin_port;      /* transport layer port # */
    struct in_addr  sin_addr;      /* IPv4 address */
    unsigned char   sin_zero[8];   /* padding */
};
```

- New IPv6 socket address structures:

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;    /* AF_INET6 */
    in_port_t       sin6_port;      /* transport layer port # */
    uint32_t        sin6_flowinfo;  /* IPv6 traffic class & flow info */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id;  /* set of interfaces for a scope */
};

struct sockaddr_storage {
    sa_family_t     ss_family;      /* address family */
    char            ss_data[126];   /* padding */
};
```

Socket Addresses (UML Diagramm)



Protocol Independent Name Resolution

```
struct addrinfo {
  int     ai_flags;     /* AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST */
  int     ai_family;    /* PF_xxx */
  int     ai_socktype;  /* SOCK_xxx */
  int     ai_protocol;  /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
  size_t  ai_addrlen;   /* length of ai_addr */
  char   *ai_canonname; /* canonical name for nodename */
  struct sockaddr  *ai_addr; /* binary address */
  struct addrinfo  *ai_next; /* next structure in linked list */
};

int getaddrinfo(const char *nodename, const char *servname,
                const struct addrinfo *hints,
                struct addrinfo **res);

void freeaddrinfo(struct addrinfo *ai);
char *gai_strerror(int ecode);
```

- getaddrinfo() returns a list of addrinfo structures that represent addresses and port numbers matching the nodename and servname parameters.
- The third argument provides hints for the lookup operation.

Protocol Independent Name Resolution

```
int getnameinfo(const struct sockaddr *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen,
                int flags);
```

- getnameinfo() returns node names and service names for a given socket address.
- The flags argument can be used to control the conversion.
- getaddrinfo() is defined in POSIX 1003.1g and in RFC 2553.
  getnameinfo() is defined in RFC 2553 and does not yet exist in POSIX land.
- Both, getaddrinfo() and getnameinfo() are thread-safe.
- getnameinfo() lacks a gni_strerror() function.

IPv4/IPv6 Daytime Client

```
/* IPv4/IPv6 daytime client.  <schoenw@ibr.cs.tu-bs.de> */

#include <stdio.h>
#include <unistd.h>

#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>

int
main(int argc, char **argv)
{
    char buffer[128];
    struct addrinfo hints, *ai_list, *ai;
    int sockfd, n;

    struct sockaddr_storage ss;
    socklen_t ss_len = sizeof(ss);
    char host[NI_MAXHOST];
    char serv[NI_MAXSERV];

    if (argc != 3) {
        fprintf(stderr, "usage: daytime host port\n");
        exit(1);
    }
```

IPv4/IPv6 Daytime Client (cont.)

```
/* Get the list of addresses (and service number). */
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;

n = getaddrinfo(argv[1], argv[2], &hints, &ai_list);
if (n) {
    fprintf(stderr, "getaddrinfo failed: %s\n", gai_strerror(n));
    exit(2);
}

/* Loop through the address list and try to connect. */
for (ai = ai_list; ai; ai = ai->ai_next) {
    sockfd = socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
    if (sockfd < 0) continue;
    if (connect(sockfd, ai->ai_addr, ai->ai_addrlen) == 0) break;
    close(sockfd);
}

/* Release the address list and check whether we were successful. */
freeaddrinfo(ai_list);

if (ai == NULL) {
    fprintf(stderr, "connect failed for %s port %s\n", argv[1], argv[2]);
    exit(3);
}
```

IPv4/IPv6 Daytime Client (cont.)

```
    /* Get the socket address of the remote end. */
    n = getpeername(sockfd, (struct sockaddr *) &ss, &ss_len);
    if (n) {
        perror("getpeername");
        exit(4);
    }

    /* Convert the address back to a string (numeric format). */
    n = getnameinfo((struct sockaddr *) &ss, ss_len,
                    host, sizeof(host), serv, sizeof(serv), NI_NUMERICHOST);
    if (n) {
        fprintf(stderr, "getnameinfo failed\n"); /* ??? */
        exit(5);
    }
    printf("[host %s port %s]\n", host, serv);

    /* Do the daytime business. */
    while ((n = read(sockfd, buffer, sizeof(buffer))) > 0) {
        buffer[n] = 0;
        puts(buffer);
    }
    close(sockfd);

    return 0;
} /* main */
```

IPv4/IPv6 Daytime Server

```
/* IPv4/IPv6 daytime server.  <schoenw@ibr.cs.tu-bs.de> */

#include <stdio.h>
#include <unistd.h>
#include <time.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>

int
main(int argc, char **argv)
{
    int n, listenfd, connectfd;
    struct addrinfo hints, *ai_list, *ai;
    const int on = 1;
    time_t ticks;
    struct sockaddr_storage ss;
    socklen_t ss_len = sizeof(ss);
    char host[NI_MAXHOST], serv[NI_MAXSERV], *s;

    if (argc != 2) {
        fprintf(stderr, "usage: daytimed port\n");
        exit(1);
    }
```

IPv4/IPv6 Daytime Server (cont.)

```
/* Get the addresses we are listening on.*/
memset(&hints, 0, sizeof(hints));
hints.ai_flags = AI_PASSIVE;
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;

n = getaddrinfo(NULL, argv[1], &hints, &ai_list);
if (n) {
    fprintf(stderr, "getaddrinfo failed: %s\n", gai_strerror(n));
    exit(2);
}

for (ai = ai_list; ai; ai = ai->ai_next) {
    listenfd = socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
    if (listenfd < 0) continue;
    setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on));
    if (bind(listenfd, ai->ai_addr, ai->ai_addrlen) == 0) break;
    close(listenfd);
}

/* Release the address list and check whether we were successful. */
freeaddrinfo(ai_list);
if (ai == NULL) {
    fprintf(stderr, "listen failed for port %s\n", argv[1]);
    exit(3);
}
```

IPv4/IPv6 Daytime Server (cont.)

```
    /* Start listening for clients... */
    if (listen(listenfd, 42) < 0) {
        perror("listen");
        exit(4);
    }
    /* Accept and process each client in turn... */
    while (1) {
        connectfd = accept(listenfd, (struct sockaddr *) &ss, &ss_len);
        if (connectfd == -1) {
            perror("accept");
            continue;
        }
        /* Convert the address back to a string (numeric format). */
        n = getnameinfo((struct sockaddr *) &ss, ss_len,
                        host, sizeof(host), serv, sizeof(serv), NI_NUMERICHOST);
        if (n) {
            fprintf(stderr, "getnameinfo failed\n"); /* ??? */
            exit(5);
        }
        printf("accepted connection from %s port %s\n", host, serv);

        ticks = time(NULL); s = ctime(&ticks);
        write(connectfd, s, strlen(s)); write(connectfd, "\r\n", 2);
        close(connectfd);
    }

    return 0;
} /* main */
```

Portability Considerations

- Not all platforms support IPv6 today.
- Always write code based on the new socket API.
- Do not clutter your source with #ifdef conditionals.
- Support legacy IPv4 platforms by distributing replacement library functions.
- Use autoconf to detect what is supported or missing.
- Look at other packages and borrow good ideas.

Further Information

- Books:
  - T. Braun: IPnG: Neue Internet-Dienste und virtuelle Netze, dpunkt Verlag, 1999
  - W. Richard Stevens: UNIX Network Programming Volume 1, 2nd edition, Prentice Hall, 1998
- Request for Comments (RFCs):
  - Internet Protocol, Version 6 (IPv6) Specification (RFC 2460)
  - DNS Extensions to support IP version 6 (RFC 1886)
  - Basic Socket Interface Extensions for IPv6 (RFC 2553)
  - Internet Transparency (RFC 2775)
- Internet Drafts:
  - The Case for IPv6
  - A Guide to the Introduction of IPv6 in the IPv4 World

Further Information (cont.)

- POSIX Standards:
  - IEEE: Protocol Independent Interfaces, IEEE Std 1003.1g, Draft 6.6, March 1997
- Important Web Sites:
  - <URL:http://www.ipv6forum.com/>
  - <URL:http://www.ipv6.org/>
  - <URL:http://www.6bone.net/>
  - <URL:http://www.join.uni-muenster.de/>
- Linux Specific Web Sites:
  - <URL:http://bofh.st/ipv6/apps.shtml/>

Thanks for listening!

<URL:http://www.mew.org/mgp/>