Url 轉換 getaddrinfo

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char *argv[]) {

	while(--argc > 0) {
		int i = argc;
		if(isdigit(argv[i][0])) {
			struct sockaddr_in ssaa, *sain;
			struct sockaddr *sa;
			char hbuf[NI_MAXHOST], sbuf[NI_MAXHOST];

			sain = &ssaa;
			sain->sin_family = AF_INET;
			sain->sin_port = htons(80);
			inet_aton(argv[i], &sain->sin_addr);
			sa = (struct sockaddr *)sain;
			if(getnameinfo(sa, sizeof(struct sockaddr), hbuf, sizeof(hbuf), sbuf,
sizeof(sbuf), 0)) {
				printf("could not get hostname\n");
			} else {
				printf("hostname: %s, serv: %s\n", hbuf, sbuf);
			}
		} else {
			struct addrinfo hints, *servinfo, *p;
			int rv;

			memset(&hints, 0, sizeof(hints));
			hints.ai_family = AF_INET;
			hints.ai_socktype = SOCK_STREAM;

			if((rv = getaddrinfo(argv[i], "http", &hints, &servinfo)) != 0){
				printf("error no is: %d\n\n", rv);
				continue;
			}

			for(p = servinfo; p != NULL; p = p->ai_next) {
				char IP[INET_ADDRSTRLEN];
				strcpy(IP, inet_ntoa(((struct sockaddr_in *)p->ai_addr)->sin_addr));
				printf("IP address is %s\n", IP);
			}
		}
		printf("\n");
	}
	return 0;
}
```

==========
udp server

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <sys/types.h>
#include <netdb.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/signal.h>
#include <errno.h>
#include <fcntl.h>
#include <sys/ioctl.h>

#define QSIZE 8
#define MAXDG 4096

int sockfd, iget, iput, nqueue;
socklen_t clilen;
static long cntread[QSIZE+1];

struct DG {
        void *dg_data;
        size_t dg_len;
        struct sockaddr *dg_sa;
        socklen_t dg_salen;
}dg[QSIZE];

static void sig_io(int signo) {
        ssize_t len;
        int nread;
        struct DG *ptr;

        for(nread = 0;;) {
                if(nqueue >= QSIZE)
                        printf("receive overflow\n");

                ptr = &dg[iput];
                ptr->dg_salen = clilen;
                len = recvfrom(sockfd, ptr->dg_data, MAXDG, 0, ptr->dg_sa, &ptr->dg_salen);
                if(len < 0) {
                        if(errno = EWOULDBLOCK)break;
                        else {
                                printf("recv error\n");
                                exit(1);
                        }
                }
                printf("recv msg: %s\n", ptr->dg_data);
```

```
                ptr->dg_len = len;
                nread++;
                nqueue++;
                if(++iput >= QSIZE)
                        iput = 0;
        }
        cntread[nread]++;
}

void echo_server(int sockfd_arg, struct sockaddr *pcliaddr, socklen_t clilen_arg) {
        int i;
        const int on = 1;
        sigset_t zeromask, newmask, oldmask;

        sockfd = sockfd_arg;
        clilen = clilen_arg;

        for(i = 0; i < QSIZE; i++){
                dg[i].dg_data = (void *)malloc(MAXDG);
                dg[i].dg_sa = (struct sockaddr *)malloc(clilen);
                dg[i].dg_salen = clilen;
        }

        iget = iput = nqueue = 0;
        signal(SIGIO, sig_io);
        fcntl(sockfd, F_SETOWN, getpid());
        ioctl(sockfd, FIOASYNC, &on);
        ioctl(sockfd, FIONBIO, &on);

        sigemptyset(&zeromask);
        sigemptyset(&newmask);
        sigemptyset(&oldmask);
        sigaddset(&newmask, SIGIO);

        sigprocmask(SIG_BLOCK, &newmask, &oldmask);
        for(;;) {
                while(nqueue == 0)
                        sigsuspend(&zeromask);

                sigprocmask(SIG_SETMASK, &oldmask, NULL);

                sendto(sockfd, dg[iget].dg_data, dg[iget].dg_len, 0, dg[iget].dg_sa,
dg[iget].dg_salen);
                if(++iget >= QSIZE)
                        iget = 0;

                sigprocmask(SIG_BLOCK, &newmask, &oldmask);
                nqueue--;
        }
}

int main(int argc, char *argv[]) {
```

```
	int serverPORT;
	struct sockaddr_in server, sender;

	serverPORT = atoi(argv[1]);

	sockfd = socket(AF_INET, SOCK_DGRAM, 0);
	bzero(&server, sizeof(server));
	server.sin_family = AF_INET;
	server.sin_port = htons(serverPORT);
	server.sin_addr.s_addr = htonl(INADDR_ANY);

	if(bind(sockfd, (struct sockaddr *)&server, sizeof(server)) < 0) {
		printf("bind failed\n");
		exit(1);
	}

	echo_server(sockfd, (struct sockaddr *)&server, sizeof(server));

	return 0;
}
```

==========

udp client

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>


#define MAXLINE 1024

void echo_cli(int sockfd, struct sockaddr_in *recv_addr, socklen_t recvlen) {
	char msg[MAXLINE];
	fd_set rset;
	while(1) {
		FD_ZERO(&rset);
		FD_SET(0, &rset);
		int maxfdp1 = 1;
		select(maxfdp1, &rset, NULL, NULL, NULL);

		if(FD_ISSET(0, &rset)){
			int n = read(STDIN_FILENO, msg, MAXLINE);
			if(strcmp(msg, "-1") == 0) {
				exit(1);
			}
			msg[n-1] = '\0';
			sendto(sockfd, msg, n, 0, (struct sockaddr *)recv_addr, sizeof(*recv_addr));
```

```
                        recvfrom(sockfd, msg, MAXLINE, 0, (struct sockaddr *)recv_addr,
sizeof(*recv_addr));
                        printf("echo msg is: %s\n", msg);
                }
        }
}

int main(int argc, char *argv[]) {
        char serverIP[INET_ADDRSTRLEN];
        int serverPORT;
        int sockfd;
        struct sockaddr_in server;
        struct addrinfo hints, *serverinfo;

        memset(&hints, 0, sizeof(hints));
        hints.ai_socktype = SOCK_DGRAM;
        hints.ai_family = AF_INET;
        if(getaddrinfo(argv[1], NULL, &hints, &serverinfo) != 0) {
                printf("no such that server\n");
                exit(1);
        }

        strcpy(serverIP, inet_ntoa(((struct sockaddr_in *)serverinfo->ai_addr)->sin_addr));
        serverPORT = atoi(argv[2]);

        bzero(&server, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(serverPORT);
        inet_pton(AF_INET, serverIP, &server.sin_addr);

        sockfd = socket(AF_INET, SOCK_DGRAM, 0);
        echo_cli(sockfd, &server, sizeof(server));

        return 0;
}
```

==========

ifconfig-egg ppt16

```
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<netdb.h>
#include<sys/types.h>
#include<sys/ioctl.h>
#include<sys/socket.h>
#include<arpa/inet.h>
#include<net/if.h>
#define MAXIF 10
#define ipsize 16

int main(){
    struct ifreq ifall[MAXIF];
    struct ifconf ioifconf;
```

```
int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

//get if vector
ioifconf.ifc_buf = (void *)ifall;
ioifconf.ifc_len = sizeof(ifall);
//get all interfaces network info
ioctl(sockfd, SIOCGIFCONF, &ioifconf);

struct ifreq *ifpt;
struct ifreq *ifend = (void *)((char *)ifall + ioifconf.ifc_len);
for(ifpt =ifall; ifpt<ifend; ifpt++){
	struct ifreq ifr;
	if(ifpt->ifr_addr.sa_family != AF_INET)
		continue;

	//get interface name
	printf("[%s]\n", ifpt->ifr_name);

	//get mtu
	bzero(&ifr, sizeof(struct ifreq));
	memcpy(ifr.ifr_name, ifpt->ifr_name, sizeof(ifr.ifr_name));
	ioctl(sockfd, SIOCGIFMTU, &ifr);
	printf("MTU: %d\n", ifr.ifr_mtu);

	//get IP address
	uint32_t addri, maski;
	char addrc[ipsize], maskc[ipsize];
	addri = ((struct sockaddr_in *)&ifpt->ifr_addr)->sin_addr.s_addr;
	inet_ntop(AF_INET, &addri, addrc, ipsize);
	printf("IP Address: %s\n", addrc);

	//get netmask
	bzero(&ifr, sizeof(struct ifreq));
	memcpy(ifr.ifr_name, ifpt->ifr_name, sizeof(ifr.ifr_name));
	ioctl(sockfd, SIOCGIFNETMASK, &ifr);
	maski = ((struct sockaddr_in *)&ifr.ifr_addr)->sin_addr.s_addr;
	inet_ntop(AF_INET, &maski, maskc, ipsize);
	printf("Netmask: %s\n", maskc);

	//get mac address
	ioctl(sockfd, SIOCGIFFLAGS, &ifr);
	if(! (ifr.ifr_flags & IFF_LOOPBACK)){   //skip loopback interface
		bzero(&ifr, sizeof(struct ifreq));
		memcpy(ifr.ifr_name, ifpt->ifr_name, sizeof(ifr.ifr_name));
		ioctl(sockfd, SIOCGIFHWADDR, &ifr);
		printf("Hwaddr: ");
		for (int i=0; i<6; ++i){
			if(i != 0)
				printf(":");
			printf("%02x", (unsigned char) ifr.ifr_addr.sa_data[i]);
		}
		printf("\n");
```

```
            }

            printf("\n");
        }
}
```

==========

ifconfig-r

```
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <ifaddrs.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <linux/wireless.h>


int check_wireless(const char* ifname, char* protocol) {
   int sock = -1;
   struct iwreq pwrq;
   memset(&pwrq, 0, sizeof(pwrq));
   strncpy(pwrq.ifr_name, ifname, IFNAMSIZ);

   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
      perror("socket");
      return 0;
   }

   if (ioctl(sock, SIOCGIWNAME, &pwrq) != -1) {
      if (protocol) strncpy(protocol, pwrq.u.name, IFNAMSIZ);
      close(sock);
      return 1;
   }

   close(sock);
   return 0;
}


int main(int argc, char const *argv[]) {
   struct ifaddrs *ifaddr, *ifa;

   if (getifaddrs(&ifaddr) == -1) {
      perror("getifaddrs");
      return -1;
   }

   /* Walk through linked list, maintaining head pointer so we
        can free list later */
   for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
      char protocol[IFNAMSIZ]  = {0};
```

```
    if (ifa->ifa_addr == NULL ||
        ifa->ifa_addr->sa_family != AF_PACKET) continue;

    if (check_wireless(ifa->ifa_name, protocol)) {
      printf("interface %s is wireless: %s\n", ifa->ifa_name, protocol);
    } else {
      printf("interface %s is not wireless\n", ifa->ifa_name);
    }
  }

  freeifaddrs(ifaddr);
  return 0;
}
```

==========

set socket opt timeout – past1 server

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <string>

int main() {
        int sockFd;
        if ((sockFd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
                perror("socket() error");

        struct timeval tv;
        tv.tv_sec = 3;
        tv.tv_usec = 0;
        if (setsockopt(sockFd, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) < 0)
                perror("setsockopt() error");

        struct sockaddr_in srvAddr;
        bzero(&srvAddr, sizeof(srvAddr));
        srvAddr.sin_family = AF_INET;
        srvAddr.sin_port = htons(57345);
        srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);

        if (bind(sockFd, (const struct sockaddr *) &srvAddr, sizeof(srvAddr)) < 0)
                perror("bind() error");

        while (1) {
                ssize_t len;
                char buf[4096];
                struct sockaddr_in cliAddr;
                socklen_t addrLen = sizeof(cliAddr);

                if ((len = recvfrom(sockFd, buf, 4096, 0, (struct sockaddr *) &cliAddr, &addrLen)) <
0) {
                        if (errno == EWOULDBLOCK)
                                continue;
                        else
```

```
                        perror("recvfrom() error");
                }

                int numA = strtol(buf, NULL, 10);

                if ((len = recvfrom(sockFd, buf, 4096, 0, (struct sockaddr *) &cliAddr, &addrLen)) <
0) {
                        if (errno == EWOULDBLOCK)
                                continue;
                        else
                                perror("recvfrom() error");
                }

                int numB = strtol(buf, NULL, 10);
                int numC = numA + numB;
                sprintf(buf, "%d", numC);

                if (sendto(sockFd, buf, strlen(buf) + 1, 0, (const struct sockaddr *) &cliAddr,
addrLen) < 0)
                        perror("sendto() error");
        }
}
```

==========

fork – past 1 client

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>

void handler(int signum) {
        while (waitpid(-1, NULL, WNOHANG) > 0);
        return;
}

int main(int argc, char **argv) {
        int sockFd;
        if ((sockFd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
                perror("socket() error");

        struct sockaddr_in srvAddr;
        bzero(&srvAddr, sizeof(srvAddr));
        srvAddr.sin_family = AF_INET;
        srvAddr.sin_port = htons(57345);
        if (inet_pton(AF_INET, argv[1], &srvAddr.sin_addr) <= 0)
                perror("inet_pton() error");

        if (connect(sockFd, (const struct sockaddr *) &srvAddr, sizeof(srvAddr)) < 0)
                perror("connect() error");

        struct sigaction act;
```

```
	act.sa_handler = handler;
	sigemptyset(&act.sa_mask);
	act.sa_flags = SA_RESTART;
	if (sigaction(SIGCHLD, &act, NULL) < 0)
		perror("sigaction() error");

	//signal(SIGCHLD, sig_fork);
	// void sig_fork(int signo){
	//     pid_t pid;
	//     int status;
	//     while((pid = waitpid(-1, &status, WNOHANG)) > 0){
	//     };
	//     return;
	// }

	pid_t pid;
	if ((pid = fork()) < 0)
		perror("fork() error");

	if (pid == 0) {
		while (1) {
			char buf[4096];
			if (recv(sockFd, buf, 4096, 0) < 0)
				perror("recv() error");
			printf("%s\n", buf);
		}
	}

	else {
		char buf[4096];
		while (fgets(buf, 4096, stdin) != NULL) {
			if (send(sockFd, buf, strlen(buf) + 1, 0) < 0)
				perror("send() error");
		}
		sprintf(buf, "kill %d", pid);
		system(buf);
		return 0;
	}
}
```

==========

ping subnet

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
```

```
#include <errno.h>
#include <sys/time.h>

int dataLen = 56;
int sockFd;
pid_t pid;
char sendBuf[1500];
int nSent = 0;
int verbose = 0;
char tempWa[200];

struct proto {
	struct hostent *host;
	struct sockaddr *saSend;
	struct sockaddr *saRecv;
	socklen_t saLen;
	int icmpProto;
} sock;

struct addrinfo *hostServ(const char *host, const char *serv, int family, int sockType);
char *sockNtopHost(const struct sockaddr *sa, socklen_t saLen);
void tvSub(struct timeval *out, struct timeval *in);
int procIcmp(char *ptr, ssize_t len, struct msghdr *msg, struct timeval *tvRecv);
void sendIcmp();
uint16_t inCksum(uint16_t *addr, int len);

int main(int argc, char **argv) {
	char *host;
	struct addrinfo *addrIn;
	char *h;

	host = argv[1];
	pid = getpid() & 0xFFFF;

	int subIp;
	for (subIp = 1; subIp <= 254; ++subIp) {
		char ip[128];
		sprintf(ip, "%s.%d", host, subIp);

		addrIn = hostServ(ip, NULL, 0, 0);
		h = sockNtopHost(addrIn->ai_addr, addrIn->ai_addrlen);

		sock.saSend = addrIn->ai_addr;
		sock.saRecv = calloc(1, addrIn->ai_addrlen);
		sock.saLen = addrIn->ai_addrlen;
		sock.icmpProto = IPPROTO_ICMP;
		in_addr_t ipAddr = inet_addr(ip);
		//sock.host = gethostbyaddr(&ipAddr, sizeof(ipAddr), AF_INET);

		struct sockaddr_in tempSa;
		tempSa.sin_family = AF_INET;
		inet_pton(AF_INET, ip, &tempSa.sin_addr);
```

```
		getnameinfo((const struct sockaddr *) &tempSa, sizeof(tempSa), tempWa,
sizeof(tempWa), NULL, 0, 0);

		char recvBuf[1500];
		char ctrlBuf[1500];
		struct msghdr msg;

		struct iovec iov;
		ssize_t n;
		struct timeval tval, sendTval;

		sockFd = socket(sock.saSend->sa_family, SOCK_RAW, sock.icmpProto);
		if (sockFd < 0)
			perror("socket() error");
		int size = 60 * 1024;
		setsockopt(sockFd, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size));
		struct timeval timeoutTv;
		timeoutTv.tv_sec = 3;
		timeoutTv.tv_usec = 0;
		setsockopt(sockFd, SOL_SOCKET, SO_RCVTIMEO, &timeoutTv,
sizeof(timeoutTv));

		sendIcmp(&sendTval);

		iov.iov_base = recvBuf;
		iov.iov_len = sizeof(recvBuf);
		msg.msg_name = sock.saRecv;
		msg.msg_iov = &iov;
		msg.msg_iovlen = 1;
		msg.msg_control = ctrlBuf;

		int result;
		struct timeval waitTv;
		do {
			msg.msg_namelen = sock.saLen;
			msg.msg_controllen = sizeof(ctrlBuf);
			n = recvmsg(sockFd, &msg, 0);

			if (n < 0) {
				if (errno == EINTR) {
					continue;
				}
				else if (errno == EAGAIN || errno == EWOULDBLOCK) {
					result = 0;
					break;
				}
				else {
					perror("recvmsg() error");
					exit(2);
				}
			}
```

```
                        gettimeofday(&tval, NULL);
                        result = procIcmp(recvBuf, n, &msg, &tval);

                        waitTv = sendTval;
                        tvSub(&waitTv, &tval);
                } while (!result && waitTv.tv_sec < 3);

                if (!result)
                        printf("%s (%s) no response\n", ip, tempWa/*sock.host->h_name*/);

                sleep(1);
        }
        return 0;
}

struct addrinfo *hostServ(const char *host, const char *serv, int family, int sockType) {
        int n;
        struct addrinfo hints, *res;
        memset(&hints, 0x0, sizeof(struct addrinfo));
        hints.ai_flags = AI_CANONNAME;
        hints.ai_family = family;
        hints.ai_socktype = sockType;

        if ((n = getaddrinfo(host, serv, &hints, &res)) != 0) {
                fprintf(stderr, "getaddrinfo() error\n");
                exit(1);
        }

        return res;
}

char *sockNtopHost(const struct sockaddr *sa, socklen_t saLen) {
        static char str[128];
        if (sa->sa_family == AF_INET) {
                struct sockaddr_in *sin = (struct sockaddr_in *) sa;
                if (inet_ntop(AF_INET, &sin->sin_addr, str, sizeof(str)))
                        return str;
        }

        return NULL;
}

void tvSub(struct timeval *out, struct timeval *in) {
        if ((out->tv_usec -= in->tv_usec) < 0) {
                --out->tv_sec;
                out->tv_usec += 1000000;
        }
        out->tv_sec -= in->tv_sec;
}

int procIcmp(char *ptr, ssize_t len, struct msghdr *msg, struct timeval *tvRecv) {
        int hLen1, icmpLen;
```

```
	double rtt;
	struct ip *ip;
	struct icmp *icmp;
	struct timeval *tvSend;
	char recvIp[128];
	inet_ntop(AF_INET, &((struct sockaddr_in *) sock.saRecv)->sin_addr, recvIp,
sizeof(recvIp));

	ip = (struct ip *) ptr;
	hLen1 = ip->ip_hl << 2;
	if (ip->ip_p != IPPROTO_ICMP) {
		fprintf(stderr, "IPPROTO error\n");
		return 0;
	}

	icmp = (struct icmp *) (ptr + hLen1);
	if ((icmpLen = len - hLen1) < 8) {
		fprintf(stderr, "hLen1 error\n");
		return 0;
	}

	if (icmp->icmp_type == ICMP_ECHOREPLY) {
		if (icmp->icmp_id != pid) {
			fprintf(stderr, "pid error\n");
			return 0;
		}
		if (icmpLen < 16) {
			fprintf(stderr, "icmpLen error\n");
			return 0;
		}

		tvSend = (struct timeval *) icmp->icmp_data;
		tvSub(tvRecv, tvSend);
		rtt = tvRecv->tv_sec * 1000.0 + tvRecv->tv_usec / 1000.0;

		printf("%s (%s) RTT=%.3fms\n", recvIp, tempWa/*sock.host->h_name*/, rtt);
		return 1;
	}
	else {
		return 0;
	}
}

void sendIcmp() {
	int len;
	struct icmp *icmp;

	icmp = (struct icmp *) sendBuf;
	icmp->icmp_type = ICMP_ECHO;
	icmp->icmp_code = 0;
	icmp->icmp_id = pid;
	icmp->icmp_seq = nSent++;
```

```
        memset(icmp->icmp_data, 0xA5, dataLen);
        gettimeofday((struct timeval *) icmp->icmp_data, NULL);

        len = 8 + dataLen;
        icmp->icmp_cksum = 0;
        icmp->icmp_cksum = inCksum((u_short *) icmp, len);

        sendto(sockFd, sendBuf, len, 0, sock.saSend, sock.saLen);
}

uint16_t inCksum(uint16_t *addr, int len) {
        int nLeft = len;
        uint32_t sum = 0;
        uint16_t *w = addr;
        uint16_t answer = 0;

        while (nLeft > 1) {
                sum += *w++;
                nLeft -= 2;
        }
        if (nLeft == 1) {
                *(u_char *) (&answer) = *(u_char *) w;
                sum += answer;
        }

        sum = (sum >> 16) + (sum & 0xFFFF);
        sum += (sum >> 16);
        answer = ~sum;
        return answer;
}
```

==========

tcp connect port

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <vector>

std::vector<int> fds;
std::vector<int> ports;
int maxFd = 0;

int main(int argc, char **argv) {
        for (int port = 10000; port <= 11000; ++port) {
                int sockFd;
                if ((sockFd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
                        perror("socket() error");
```

```
		int flags = fcntl(sockFd, F_GETFL, 0);
		if (fcntl(sockFd, F_SETFL, flags | O_NONBLOCK) < 0)
			perror("fcntl() error");

		struct sockaddr_in srvAddr;
		bzero(&srvAddr, sizeof(srvAddr));
		srvAddr.sin_family = AF_INET;
		srvAddr.sin_port = htons(port);
		if (inet_pton(AF_INET, argv[1], &srvAddr.sin_addr) <= 0)
			perror("inet_pton() error");

		if (connect(sockFd, (const struct sockaddr *) &srvAddr, sizeof(srvAddr)) < 0) {
			if (errno == EINPROGRESS) {
				fds.push_back(sockFd);
				ports.push_back(port);
				maxFd = sockFd > maxFd ? sockFd : maxFd;
			}
		}
		else
			printf("TCP port number %d is open.\n", port);
	}

	while (1) {
		fd_set fs;
		FD_ZERO(&fs);
		for (auto fd: fds)
			FD_SET(fd, &fs);

		if (select(maxFd + 1, NULL, &fs, NULL, NULL) < 0)
			perror("select() error");

		for (int i = 0; i < fds.size(); ++i) {
			int fd = fds[i];
			if (FD_ISSET(fd, &fs)) {
				int ret;
				socklen_t len;
				getsockopt(fd, SOL_SOCKET, SO_ERROR, &ret, &len);
				if (ret == 0)
					printf("TCP port number %d is open.\n", ports[i]);
				fds.erase(fds.begin() + i);
				ports.erase(ports.begin() + i);
			}
		}

		if (fds.empty())
			break;
	}

	return 0;
}
```

==========

local MTU

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <string>
#include <sys/ioctl.h>
#include <net/if.h>
#include <string>

int main(int argc, char **argv) {
	int sockFd;
	if ((sockFd = socket(AF_LOCAL, SOCK_DGRAM, 0)) < 0)
		perror("socket() error");

	struct ifconf ifc;
	struct ifreq *ifr;
	char buf[2048];

	ifc.ifc_len = 2048;
	ifc.ifc_buf = buf;
	ifr = (struct ifreq *) buf;

	if (ioctl(sockFd, SIOCGIFCONF, &ifc) < 0)
		perror("ioctl() error");

	if (argc == 2) {
		unsigned newMtu = strtoul(argv[1], NULL, 10);
		ifr->ifr_mtu = newMtu;

		if (ioctl(sockFd, SIOCSIFMTU, ifr) < 0)
			perror("ioctl() error");
	}

	else {
		if (ioctl(sockFd, SIOCGIFMTU, ifr) < 0)
			perror("ioctl() error");

		printf("MTU of loopback (lo): %d\n", ifr->ifr_mtu);
	}

	return 0;
}
```