```c
/*
    Per eseguire un file .c su terminale
        gcc - o run es.c
        ./ run

    Per eseguire un file .c con server e client
        Terminale 1: gcc -o run es.c
                     ./run
        Terminale 2: telnet localhost <porta>

    Per eseguire due istanze di uno stesso programma
        Terminale 1: gcc -o run es.c
                     ./run
        Terminale 2: ./run
*/

// COMANDI FONDAMENTALI

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

// Controllo argomenti passati per invocazione al programma
// N e' il numero di parametri che devono essere inseriti + 1
// Se bisgona inserire solo 1 parametro, N = 2
if (argc != N) {
    fprintf(stderr, "Errore argomenti\n");
    exit(1);
}

int numero = atoi(argv[1]) // Per trasformare un argomento in intero

/* ----------- GESTIONE DEI SEGNALI ----------- */

void handler() {
    serviceAvailable = !serviceAvailable;
    printf("SIGUSR1: servizio %s\n", (serviceAvailable) ? "ATTIVO" : "NOT ATTIVO");
}

// Codice da inserire nel main

// Gestione del segnale SIGUSR1
struct sigaction sig;

sig.sa_handler = handler; // Gestore del SIGUSR1
sigemptyset(&sig.sa_mask);
sig.sa_flags = SA_RESTART;

// Primitiva FONDAMENTALE per la gestione dei segnali affidabili
sigaction(SIGUSR1, &sig, NULL);

// Quando si hanno 2 segnali si fa due volte.
sigaction(SIGUSR2, &sig, NULL);

```

```c
/* ----------- BLOCCO SEGNALE ----------- */
// Processo Ps in attesa di SIGUSR1 quindi blocco il segnale
sigset_t sigmask;
sigemptyset(&sigmask);
sigaddset(&sigmask, SIGUSR1);
sigprocmask(SIG_BLOCK, &sigmask, NULL); // Blocca SIGUSR1 nel gestore

sigsuspend(&zeromask); // Segnale SIGUSR1 viene sbloccato attraverso la maschera, va fatta quando bisogna sbloccare il segnale.

/* ----------- CREAZIONE DI UNA PIPE ----------- */
int piped[2]; // PIPE[0] LETTURA, PIPE[1] SCRITTURA
if (pipe(piped) < 0) {
    perror("Errore creazione pipe");
    exit(4);
}

/* ----------- CREAZIONE DI UN PROCESSO ----------- */
int pid;
if ((pid = fork()) < 0) {
    perror("Errore creazione processo");
    exit(5);
} else if (pid == 0) {              // Processo figlio Ps
    /* [...] -> codice */
} else {                            // Processo padre (pid > 0)
    /* [...] -> codice */
}
```

```c

/* ----------- SERVER CONCORRENTI CON SOCKET  ----------- */

#define SERVERPORT 1234 // Inserire prima del main, dopo #include
int serviceAvailable = 1;
void handler(int signo) {    // Funzione di gestione segnali. Inserire prima del main, dopo #include
    serviceAvailable = !serviceAvailable;
    printf("SIGUSR1: service is %s\n", (serviceAvailable) ? "ACTIVE" : "not ACTIVE");
}


//da qui in poi, mettere nel main.
int sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
    perror("Errore creazione stream socket");
    exit(4);
}
printf("SERVER PID %d\n", getpid());
struct sockaddr_in server, client;

// Imposto i parametri del server
server.sin_family = AF_INET; // Internet protocol
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(SERVERPORT);

int on = 1;
if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0) {
    perror("Errore setsockopt");
    exit(5);
}

if (bind(sock, (struct sockaddr*) & server, sizeof(server)) < 0) {
    perror("Errore binding stream socket");
    exit(6);
}
int length = sizeof(server);
if (getsockname(sock, (struct sockaddr*) & server, (socklen_t*)&length) < 0) {
    perror("Errore getting socket name");
    exit(7);
}
printf("Porta della socket #%d\n", ntohs(server.sin_port));

// Massimo 5 connessioni in attesa
if (listen(sock, 5) < 0) {
    perror("Errore listen");
    exit(8);
}
//PER DUE SOCKET BISOGNA RIPETERE IL CODICE DI PRIMA 2 VOLTE

printf("In attesa di connessioni...\n");
```

```
155
156
157 //IL SERVER CON SOCKET CONTINUA DOPO
158
159 // QUESTA PARTE E'LA CONTINUA DEI SERVER CON SOCKET
160
161 char* serviceUnavaible = "Servizio non disponibile";
162 int msgsock;
163 // Ciclo principale del server concorrente
164 // Crea un figlio per ogni client che si connette
165 do {
166     // Il server attende richieste di connessioni sulla propria socket
167     msgsock = accept(sock, (struct sockaddr*) & client, (socklen_t*)
        &length); //PER SCRIVERE SUL SOCKET MENTRE SEI DENTRO UN PROCESSO
        FIGLIO BISOGNA CREARE UN MSGSOCK E SCRIVERCI DENTRO CON WRITE.
168     if (msgsock < 0) {
169         perror("Errore accept");
170         exit(9);
171     }
172
173     printf("Connessione effettuata con successo con %s\n", inet_ntoa
        (client.sin_addr));
174
175     if (fork() == 0) {
176         close(sock); // SEMPRE
177
178         /* [...] -> codice diverso in base al programma */
179         // E' sempre meglio inserire il codice in una funzione
180
181         // Controllo se il servizio è attivo
182         // serviceAvailable è una variabile globale gestita nella funzione
            handler
183         // vedi esempio "BIBLIOTECA BASATA SU PIPE"
184         if (!serviceAvailable) {
185             write(msgsock, serviceUnavaible, strlen(serviceUnavaible) + 1);
186             close(msgsock);
187             continue;
188         } else {
189             // Fork
190         }
191
192         close(msgsock);
193         exit(0);
194     } else {
195         close(msgsock);
196     }
197 } while (1);
198
199 // FUNZIONI UTILI PER COMUNICARE TRA CLIENT E SERVER
200
201 // Invia al client un messaggio con la stringa "s"
202 write(msgsock, s, strlen(s) + 1);
203
```

```c
int nread;
// Lettura di un messaggio scritto da client
if ((nread = read(msgsock, buffer, 256)) < 0) {
    perror("Lettura messsaggio");
    exit(9);
}

int dato;
sscanf(buffer, "%d", &dato);

// Per stampare un valore o una stringa sul server utilizzo direttamente "printf"
printf("Dato ricevuto = %d\n", dato);

/* ----------- GESTIONE FILE I/O ----------- */

char filename[256];
strcpy(filename, argv[1]); // Nome del file passato come argomento

// Apertura di un file solo in lettura
int file;
if ((file = open(filename, O_RDONLY)) < 0) {
    fprintf(stderr, "Non e' possibile aprire %s: %s\n", filename, strerror(errno));
    exit(2);
}

// Controllo della dimensione di un file
struct stat fbuf;
fstat(file, &fbuf);
int filesize = fbuf.st_size; // filesize in bytes
if (filesize <= 5) {
    fprintf(stderr, "La dimensione del file (%d bytes) non e' accettata\n", filesize);
    exit(3);
}

#define BUFFER_SIZE 356 // Inserire prima del main, dopo #include

char buffer[BUFFER_SIZE];
int nread;

// Lettura di un file
while ((nread = read(file, &buffer, BUFFER_SIZE)) > 0) {
    /* [...] -> codice */
}

if (nread < 0) {
    fprintf(stderr, "Non e' possibile leggere il file %s\n", filename);
    exit(5);
}

close(file);
```

```
254
255 /* ----------- PRIMITIVA SELECT ----------- */
256 FD_ZERO(&sock_fdset);
257 FD_SET(sock1, &sock_fdset);
258
259 int n;
260 do {
261     // Test di lettura sulla socket
262     n = select(sock2 + 1, &sock_fdset, NULL, NULL, NULL);
263 } while (n == -1 && errno == EINTR);
264
265 if (FD_ISSET(sock1, &sock_fdset)) {
266 }
```