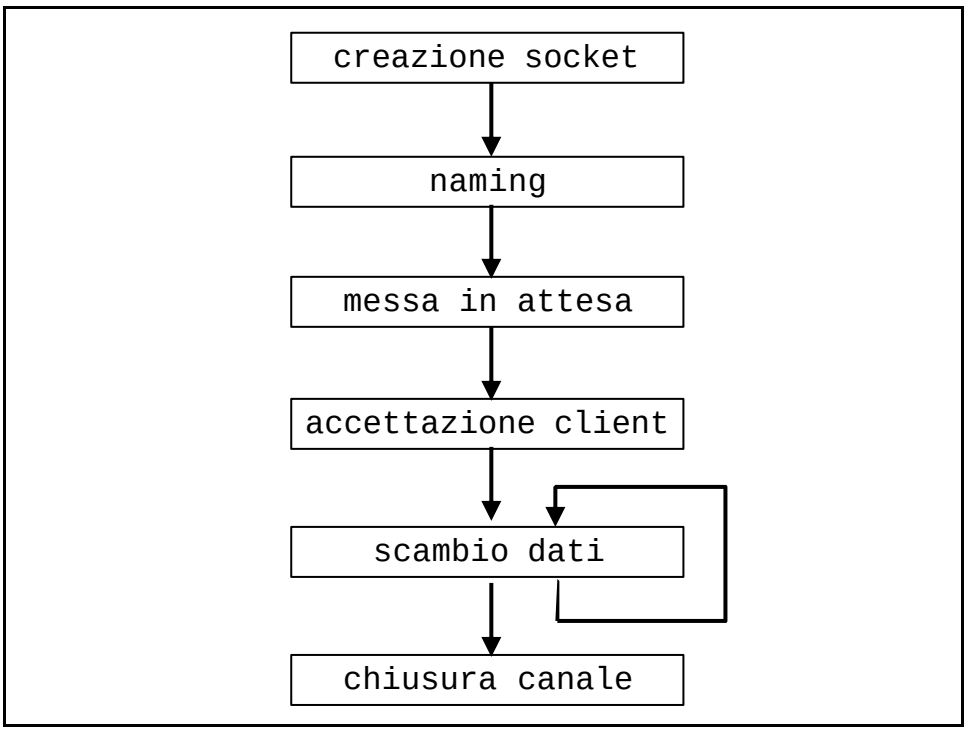
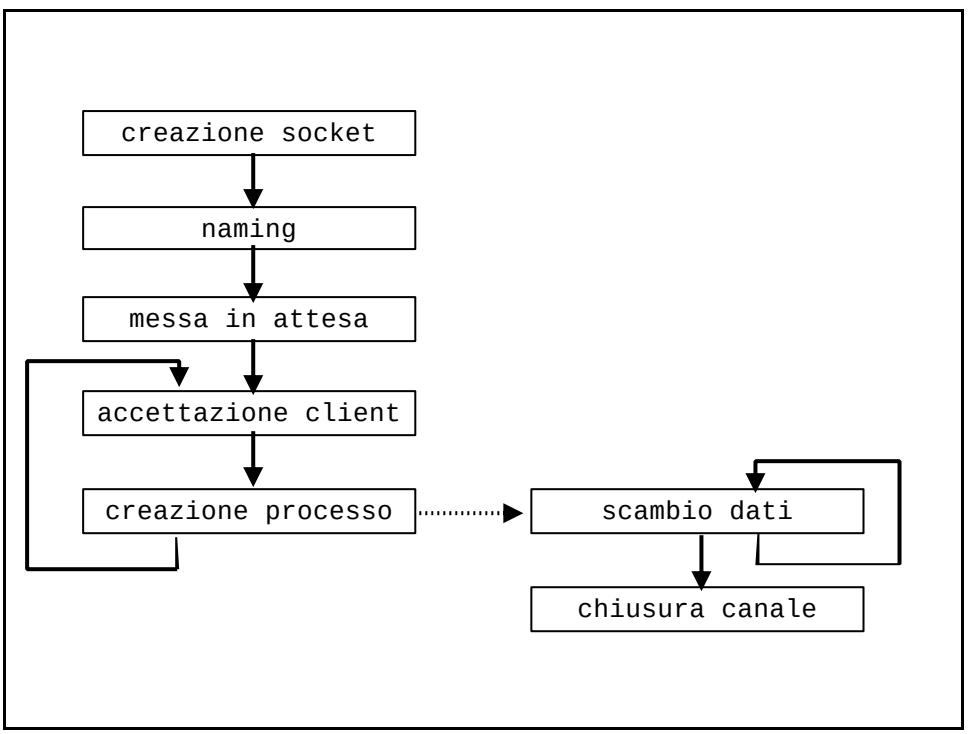
Implementazione di un server concorrente o multiprocesso

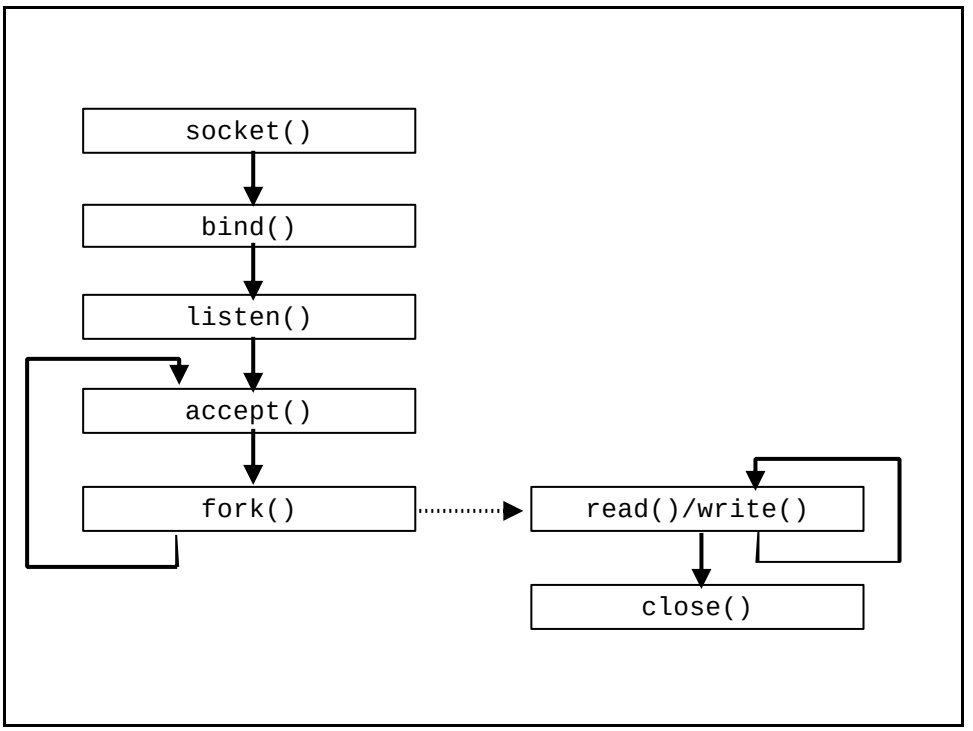
Dobbiamo scrivere un programma in C che offre un servizio, ma non vogliamo tenere gente "*in coda*"

Server multiprocesso

- Offro servizio ad più client contemporaneamente
- È una variante di poco più complicata del server iterativo


creazione socket
naming
messa in attesa
accettazione client
scambio dati
chiusura canale


creazione socket
naming
messa in attesa
accettazione client
creazione processo
scambio dati
chiusura canale



fork()

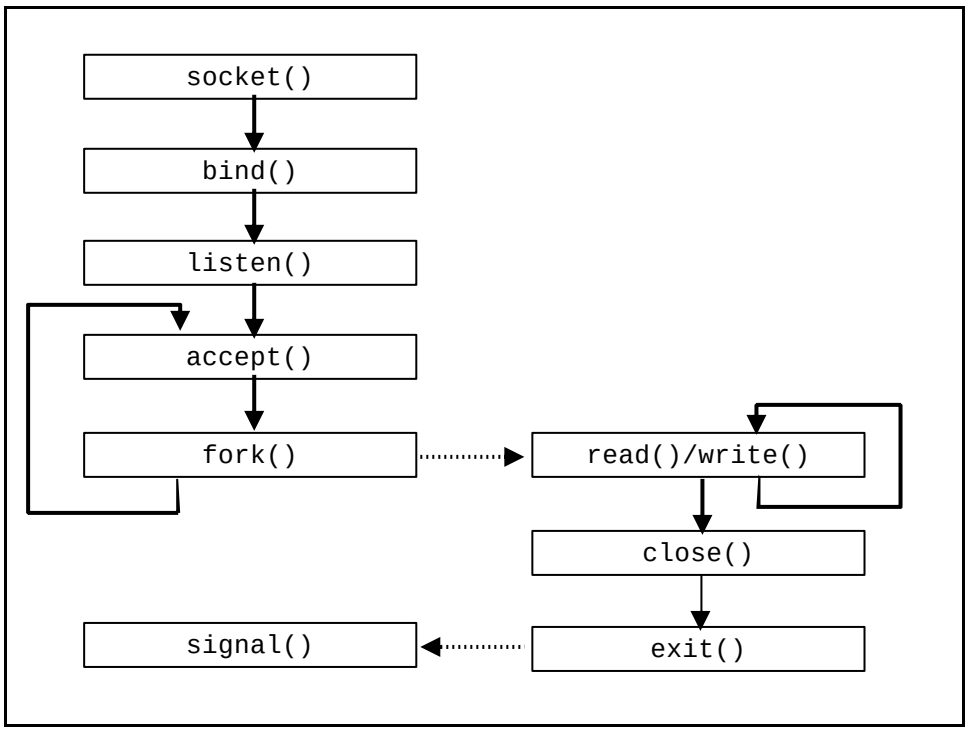
```
int pid;
pid = fork();
if (pid == 0)
    printf(“figlio\n”);
else
    printf(“padre\n”);
```

fork()

```
int data_socket;
int pid;

while(1) {
       data_socket = accept(s, &accept_addr,
                                  &accept_addrlen);
       pid = fork();
       if (pid == 0)
              give_service(data_socket);
       close(data_socket); /* codice del padre */
}

...

void give_service(int socket) {
       ...
       close(socket);
       exit(0);
}
```

La morte del processo figlio

- Il sistema operativo notifica l'avvenuta morte al padre
- Un segnale viene messo in coda al PCB del processo padre ed il figlio viene tolto dalla process table solo quando il processo è stato gestito
- Quindi, il server deve prendersi cura di gestire il segnale, altrimenti avremmo una proliferazioni di processi "*zombie*"

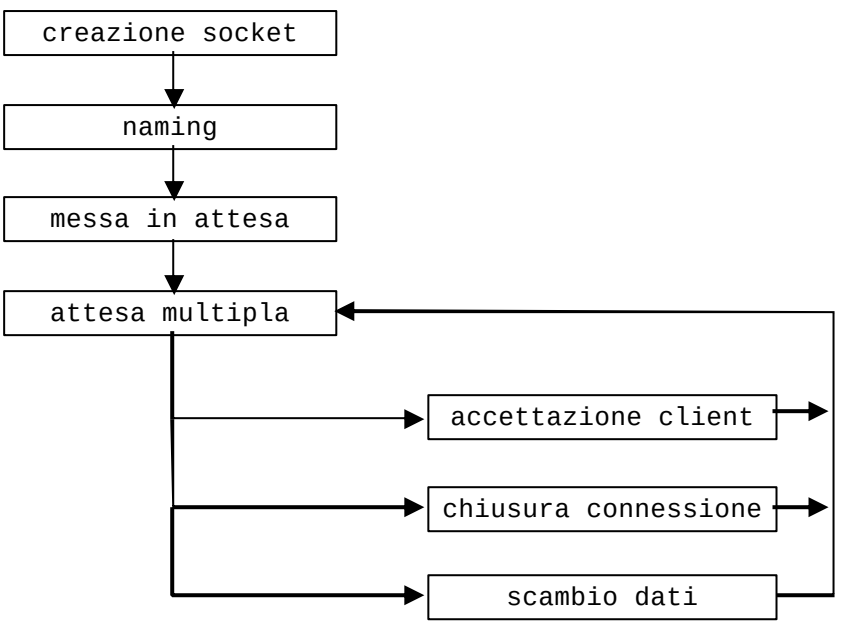


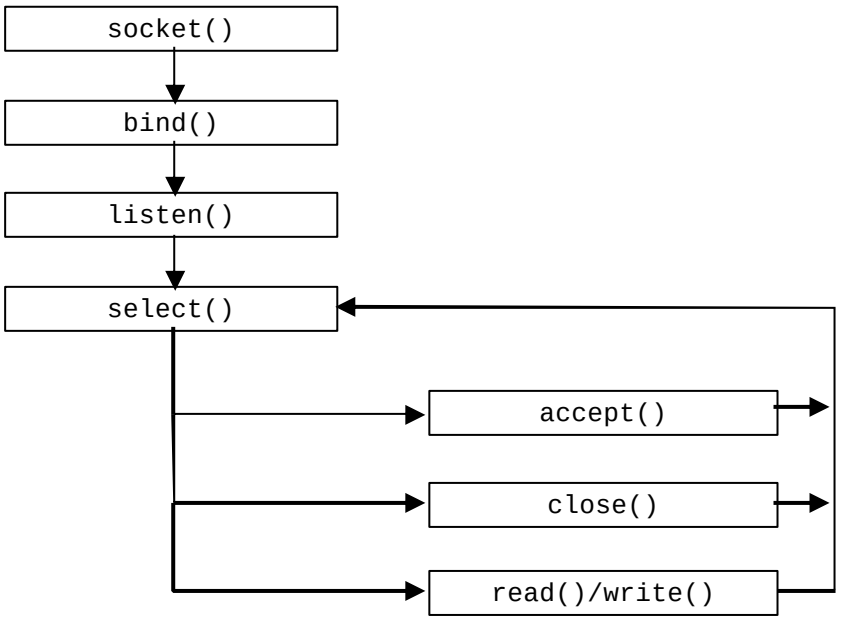
signal()/exit()

```
int reaper();

void main(int argc, char ** argv) {
       ...
       signal(SIGCHLD, reaper);
       ...
       while (1) { ... }
}

int reaper() {
    int status;
    while ( wait3 (&status, WNOHANG, NULL) >= 0);
    (void) signal(SIGCHLD, reaper);
}
```

Server concorrente

- Offro servizio ad più client contemporaneamente
- È la variante più complessa da gestire





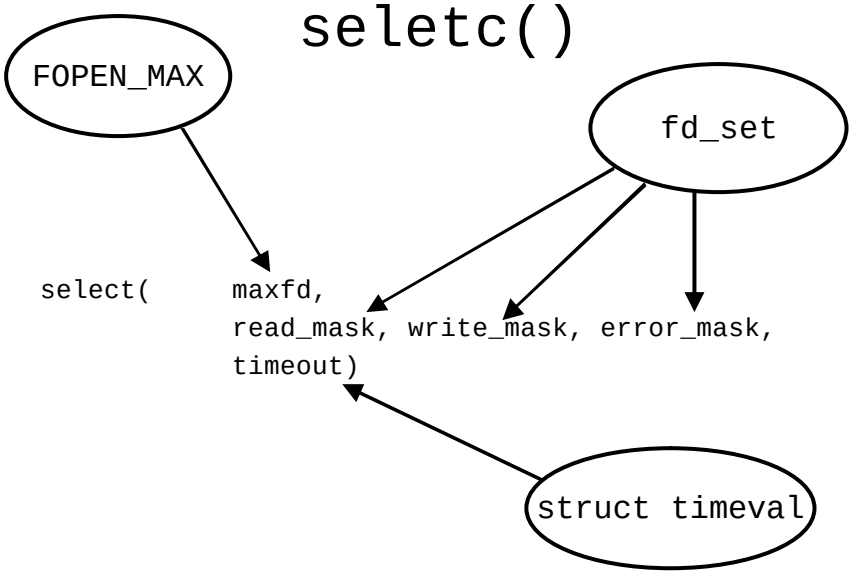
Maschere di bit

- Servono ad elencare dei gruppi di file descriptor
- Questo non vuol dire che ci limitiamo alle socket
- Potremmo voler scrivere un server che gestisce delle socket e contemporaneamente un operatore da tastiera

select()

- Ci permette di stare in attesa su più canali contemporaneamente
- La `select` termina quando è possibile fare una operazione su uno qualunque dei canali, secondo certi criteri di selezione

seletc()



Maschere di bit

ATTENZIONE

Dopo essere stata usata come parametro per la select, una maschera di bit non è più utilizzabile.

Dentro ci saranno le informazioni su quali canali sono disponibile per effettuare operazioni

Manipolazione delle maschere

- `FD_ZERO(maschera)`
  Azzeramento della maschera
- `FD_SET(fd, maschera)`
  Impostazione di un canale
- `FD_CLR(fd, maschera)`
  De-impostazione di un canale
- `FD_ISSET(fd, maschera)`
  Controlle se è possibile fare operazioni sul canale

```
fd_set fds;
fd_set temp_fds;
int nfds;
...
FD_ZERO(&afds);
FD_SET(s, &afds);
...
while(1) {
       bcopy((char *)&fds,(char *)&temp_fds, sizeof(fds));
       if (select(nfds, temp_fds, NULL, NULL, NULL) < 0) {
              perror(“select()”);
              exit(1);
       }
       ...
}
```

```
int fd;

while(1) {
       bcopy((char *)&fds,(char *)&temp_fds, sizeof(fds));
       select(nfds, temp_fds, NULL, NULL, NULL);
       if(FD_ISSET(s, &temp_fds)) {
              data_sock = accept(s, ...);
              FD_SET(data_sock,&afds);
       }
       else {
              for(fd = 0; fd <= nfds; fd += 1) {
                     if(FD_ISSET(fd, &temp_fds)) {
                            do_read_write(fd);
                     }
              }
       }
}
```

```
int fd;

while(1) {
	...
	else {
		for(fd = 0; fd <= nfds; fd += 1) {
			if(FD_ISSET(fd, &temp_fds)) {
				do_echo(fd);
				FD_CLR(fd, &fds);
				close(fd);
			}
		}
	}
}
```