# Fabrication du fichier

*gcc -Wall -o executable.exe fichier.c*

# Exécution du fichier

*./executable.exe*

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

La fonction main reçoit les arguments mis en paramètres à l’exécution.

Par exemple :  
Avec *./tp1.exe 127.0.1.1 8080*  
**argc** contient le nombre de paramètres (3) et  
**argv** contient un tableau de pointeurs des paramètres saisis :

* « tp1.exe »
* 127.0.1.1
* 8080

# Connexion à un serveur (en tant que client)

1. Création socket
2. Connexion au serveur
3. Echanges
   1. Réception
   2. Envoi
4. Fermeture socket

# Fabrication d’un serveur

1. Création socket
2. Lier le socket à un port
3. Mise en écoute du socket
4. **Traitement des requêtes**
5. Fermeture socket

# Traitement des requêtes (en tant que serveur)

1. Acceptation connexion entrante
2. Réception message (fermeture de sa socket en cas d’échec)
3. Traitement
4. (Envoi de la réponse)
5. Fermeture socket client

# Fonctions utiles

|  |  |
| --- | --- |
| Fonction | Explications |
| perror("message d'erreur"); | Affiche une erreur importante. Passe devant le message par défaut |
| fprintf(stderr,  "message"); | Affiche le message en précisant son type de sortie (fichier, erreur…) stderr : erreur |
| printf("Message %s", params); | Affiche un message standard stdout |
| fgets(message,  sizeof(message),  stdin); | Récupère un message saisi |
| atoi("1234"); 🡪 1234 | String 🡪 int |
| int socket(int domain,  int type,  int protocol); | Pour créer un socket.  Domain : AF\_INET pour ipv4  Type : SOCK\_STREAM pour TCP  Protocol : 0 pour laisser le système choisir en général |
| htons(server\_port); | Traduit un port entier au format de l’ordinnateur |
| bind(serveur\_socket,  (struct sockaddr\*)&server\_adr,  sizeof(server\_adr)); | Associe une @ip et un port à un socket  Renvoie -1 en cas d’erreur |
| connect(client\_socket,  (struct sockaddr \*)&server\_addr,  sizeof(server\_addr)); | Lie un socket client avec un socket serveur |
| close(server\_socket); | Pour fermer la socket |
| listen(server\_socket, 5) | La socket écoute au maximum 5 connexions en attente. |
| inet\_pton(AF\_INET,  server\_ip,  &server\_addr.sin\_addr); | Convertit une @ip string en @ip binaire |
| send(client\_socket,  message,  strlen(message),  0); | Envoie des données à travers un socket |
| recv(client\_socket,  response,  sizeof(response),  0); | Reçoit une réponse d’un socket. On doit indiquer la taille max de données qu’on veut recevoir.  '\0' si rien n’est reçu. |
| accept(server\_socket,  (struct sockaddr \*)&client\_addr,  &client\_len); | Accepte une connexion entrante |
| inet\_ntop(AF\_INET,  &client\_addr.sin\_addr,  client\_ip,  sizeof(client\_ip)); | Convertit une @ip binaire en format réseau stockable en sockaddr\_in. Permet d’afficher de manière lisible l’@ |
| ntohs(client\_addr.sin\_port) ; | Convertit un port du format réseau au format hôte |
| continue; | Saute la boucle du while. Permet de passer a l’utilisateur suivant |
| snprintf((char\*)msg\_formate,  sizeof(msg\_formate),  "Message a formater :%n",  message); | Formate un message dans un tampon :  Le message a formater utilisant les « % » ne dépassera pas la taille indiquée. |
| key\_t ftok(const char \*pathname,  int proj\_id);  ftok("/chemin/vers/monfichier.txt", 'A'); | = « File to Key »  Génère une clé permettant d’accéder et de partager des ressources partagées.  \*pathname : nom du fichier contenant le processus ayant créé le segment.  Proj\_id : entier pour identifier chaque clé d’un même fichier. |
| int shmget(key\_t key,  size\_t size,  int shmflg);  int shmid = shmget(key, 1024, 0666 | IPC\_CREAT); | Permet d’accéder ou de créer un segment de mémoire partagée.  Size : taille du segment de mémoire partagée en octets.  Shmflg : ensemble de droits et d’options. |
| void \*shmat(int shmid,  const void \*shmaddr,  int shmflg);  void \*shm\_ptr = shmat(shmid, NULL, 0); | Permet d’attacher un segment de mémoire à au processus en cours  Shmaddr : Adresse de base à laquelle attacher le segment (Généralement Null pour laisser le système choisir)  Shmflg : ensemble de droits et d’options  Retourne : adresse a laquelle la mémoire s’est attachée |
| int shmdt(const void \*shmaddr); | Détache un segment de mémoire partagée d’une adresse |
| int shmctl(int shmid,  int cmd,  struct shmid\_ds \*buf);  shmctl(shmid, IPC\_RMID, NULL); | Effectuer des opérations de contrôle sur un segment de mémoire partagée telle que la suppression.  Cmd : commande de contrôle  Buf : Buffer qui stocke le resultat s’il y en a un (mettre null si on récupère rien). |
|  |  |

# Structures / types de données

struct sockaddr\_in {// Structure surtout utilisee pour IPV4

    short sin\_family;        // Famille d'adresse (AF\_INET)

    unsigned short sin\_port; // Numéro de port (htons(server\_port))

    struct in\_addr sin\_addr; // Adresse IP : (INADDR\_ANY)

};

struct sockaddr {// Structure plus generique

    unsigned short sa\_family; // Famille d'adresse (AF\_INET, AF\_INET6, etc.)

    char sa\_data[14];         // Adresse IP générique

};

socklen\_t client\_len = sizeof(client\_addr);

char client\_ip[INET\_ADDRSTRLEN]; //INET\_ADDRSTRLEN = longueur max d'une @ IPv4