ENSTA Bretagne

Initiation aux réseaux

Fabrice LE BARS

Présentation générale de la matière

Présentation générale de la matière

- Objectifs
  - Savoir préparer des ordinateurs embarqués et stations de contrôle ainsi que les périphériques réseaux nécessaires

Présentation générale de la matière

- Planning
  - **32C**
  - 2 parties :

    Initiation rapide (4+2C) et application sur ordinateurs embarqués, manipulations de matériel, etc. (3x4C) en tout début de semestre

    Compléments C/C++ (2C), programmation C/C++ multithread (4C), sockets (4C), mini-projet (3C) en fin de semestre

    1C d'évaluation écrite

Présentation générale de la matière

- Planning
  - Evaluation :

    Rendre tous les **travaux** faits sur Moodle à la fin de **chaque séance** (selfie du binôme/groupe, notes et données techniques, codes, schémas, photos et vidéos des tests, etc.), sera utilisé comme **note de rattrapage***

    Un **mini-projet** en C/C++ sera à rendre à la fin

    1C d'évaluation écrite (connaissance du vocabulaire, des commandes courantes, compréhension de la logique, éventuellement pseudo-code)

*Rattrapage : des travaux/épreuves complémentaires pourront être demandés pour que la note de rattrapage soit du même niveau de difficulté que les épreuves initiales

Présentation générale de la matière

- Organisation
  - Amener ses PC portables, smartphones, câbles USB et de recharge, écouteurs, souris USB
  - Pour limiter les achats de matériel, il y aura parfois des sujets de TD à se répartir en équipes
  - Respect du matériel, rangement à la fin, tout bien rebrancher les PC des salles infos avant de repartir…

Présentation générale de la matière

- Organisation
  - Suggestions pour la prise de notes, la synchronisation de documents, visio, etc.

    Microsoft OneNote : prendre des notes rapidement sur son smartphone ou ses PCs, synchronisées en temps réel et accessibles offline

    Microsoft OneDrive/SharePoint ou équivalents/compléments de Google : synchroniser et partager rapidement en temps réel des documents (e.g. photos prises par smartphones, fichiers Excel, Word, etc.) entre appareils, personnes, etc.

    SyncToy 2.1 (https://www.microsoft.com/en-us/download/details.aspx?id=15155) : synchroniser hors ligne le contenu d'un disque dur externe avec celui d'un PC (e.g. 2 TB de données peuvent être synchronisées en moins de 15 min s'il y a peu de changements et que les disques sont rapides)

    WinMerge : voir rapidement les différences entre des fichiers ou dossiers

Présentation générale de la matière

- Organisation
  - Suggestions pour la prise de notes, la synchronisation de documents, visio, etc.

    Microsoft Teams (https://helpdesk.ensta-bretagne.fr/enseignement-a-distance/teams/) : outil pour les cours et visio à distance préconisé à l'école

    WhatsApp : constituer un groupe de discussion à partir de numéros de téléphones et partager des photos et courtes vidéos e.g. lors d'expérimentations en extérieur

    Microsoft Photos Legacy : montage vidéo de base et conversion vers .mp4 (voir https://www.ensta-bretagne.fr/lebars/Share/Windows.txt pour les problèmes connus)

    Da Vinci Resolve : montage vidéo avancé (voir https://www.ensta-bretagne.fr/lebars/tutorials/davinci_resolve_tuto.pdf), compatible Windows et Ubuntu mais nécessite une bonne carte graphique

    FormatFactory, **ffmpeg** : conversions de formats audio/video

    VLC : lecteur multimedia compatible avec de nombreux formats

    Voir aussi https://sill.code.gouv.fr/

Présentation générale de la matière

- Organisation
  - Suggestions pour la prise de notes, la synchronisation de documents, visio, etc.

    Windows 10/11 Pro : certains de vos PCs portables pouvant être livrés avec une version **Home** (qui a des limitations) ou sans Windows, l'école fournit des licences personnelles pour les étudiants et personnels, voir https://offres-informatiques.ensta-bretagne.fr

    Ubuntu 22.04 : version à privilégier cette année

    Voir aussi https://www.ensta-bretagne.fr/lebars/computer_prereq_rob.pdf

Présentation générale de la matière

- Notations
  - Certaines slides ou points dans des slides seront marqués avec le symbole

pour indiquer des informations a priori **non indispensables** à la compréhension des parties importantes du cours et/ou **pas à apprendre par cœur** à ce stade, mais qu'il faudra peut-être **savoir retrouver** rapidement quand le besoin se présentera e.g. au cours d'un projet dans l'année

Par analogie avec le concept de pointeur en C/C++ : mémoriser le pointeur (i.e. l'emplacement de la slide) quand il y a le symbole, mémoriser les données (i.e. ce qui est dans la slide) quand il n'y est pas!

Sommaire

Types de communications

Types de communications

- Prérequis/Rappels :
  - **Hardware** : matériel
  - **Software** : logiciel
  - **Firmware** : désigne souvent le logiciel complet d'un micro-contrôleur (e.g. dans un périphérique réseau, capteur, carte électronique embarquée, etc.), on ne peut en général que le remplacer dans son ensemble par e.g. une mise à jour
  - **OS** (Operating System/système d'exploitation) : ensemble d'éléments logiciels permettant à des développeurs et utilisateurs d'accéder à un grand nombre de fonctionnalités sur un ordinateur. Exemples : Windows, Ubuntu, macOS

En rapport avec ces notions: « to **brick a device** » signifie rendre inutilisable de manière irréversible un appareil à cause d'une mauvaise configuration logicielle interne (sans qu'il y ait eu de dommages physiques)

Types de communications

- Prérequis/Rappels :
  - **Middleware** : terme assez vague pouvant représenter un ensemble d'éléments logiciels (reposant sur un OS) proposant des outils cohérents (pour développeurs et utilisateurs) pour un ensemble de tâches liées. Exemple dont le nom est trompeur : ROS
  - **Framework** : terme assez vague similaire à middleware mais plus destiné aux développeurs qu'aux utilisateurs standards. Exemple : Microsoft .NET Framework
  - **API** (Application Programming Interface) : ensemble de fonctions dans un langage informatique donné qui permet d'accéder à des fonctionnalités données. Exemple : Windows API

Types de communications

- Prérequis/Rappels :
  - Bas niveau vs haut niveau :

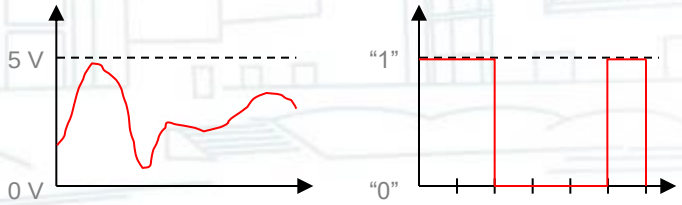
    Bas niveau = le plus proche du matériel

    Haut niveau = le plus proche de l'utilisateur

  - Analogique vs numérique :

    Analogique : e.g. pin dont la tension varie de 0 à 5 V de manière continue dans le temps

    Numérique : e.g. pin dont la tension prend des valeurs de 0 V ou 5 V par pas de temps constants, ces successions de valeurs étant à interpréter comme des niveaux logiques (bits) de 0 ou 1 pour chaque pas de temps



Types de communications

- Prérequis/Rappels :
  - Canal de communication : milieu dans lequel passent les signaux

    Avec fil : via un ou plusieurs **fils de cuivre**, via **fibre optique**, etc.

    Sans fil : via des **ondes électromagnétiques** (habituellement dans l'**air**), **acoustiques** (habituellement dans l'**eau**), dans une ou plusieurs bandes de fréquences constantes ou variables, etc.

  - Multiplexage : faire passer plusieurs informations à travers un seul support de transmission

  - Bande passante : quantité d'informations qu'on peut transmettre par seconde, e.g. 54 Mbps pour du Wi-Fi 802.11g

  - Latence/ping : délai de réception d'une nouvelle donnée

Types de communications

- Prérequis/Rappels :
  - Modulations :

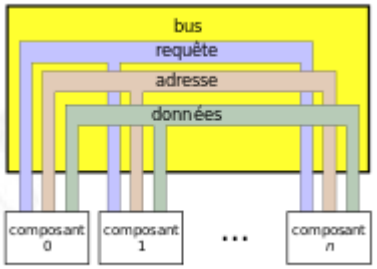
    Analogiques : utilisation d'une ou plusieurs fréquences porteuses adaptées au canal de communication et modulation d'amplitude (AM), fréquence (FM), phase (AM)…

    Numériques : chacun son tour, valeurs discrètes d'amplitudes (ASK), fréquences (FSK), phase (PSK)…

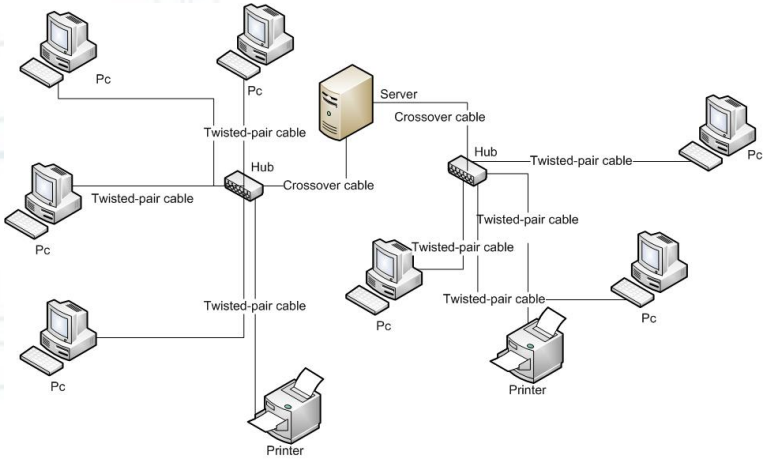
    => En général en robotique on a rarement à se soucier du type de multiplexage ou modulation utilisé, sauf peut-être pour évaluer plus en détail à quel point on pourra avoir plusieurs appareils communicants correctement en même temps…

Types de communications

- Communications numériques :
  - **Liaison point à point (P2P)** : 2 périphériques sont en liaison exclusive
  - **Bus** : canal de communication partagé entre plusieurs périphériques, souvent avec des concepts de maitre et esclaves



  - **Réseau** : plusieurs périphériques et plusieurs canaux



Types de communications

- Communications numériques (en bas niveau) :
  - **Point à point** :

    0 ou 1 sur un ou plusieurs bits en parallèle

    Liaison série RS232/UART

  - **Bus** :

    RS485/RS422

    CAN

    I2C/SMBus

    SPI

    USB

  - **Réseau** :

    Ethernet, Wi-Fi, 4G, BlueTooth, ZigBee, LoRa, some acoustic modems

Types de communications

- Communications numériques :
  - Protocole :

    Ensemble de règles portant sur l'envoi, la réception, et l'interprétation de données entre plusieurs appareils ou programmes pour qu'ils se comprennent

    Ces données sont en général regroupées en paquets/trames/messages, eux-mêmes souvent divisés en sous-parties, dont le protocole doit définir le format, le rôle, l'interprétation, etc.

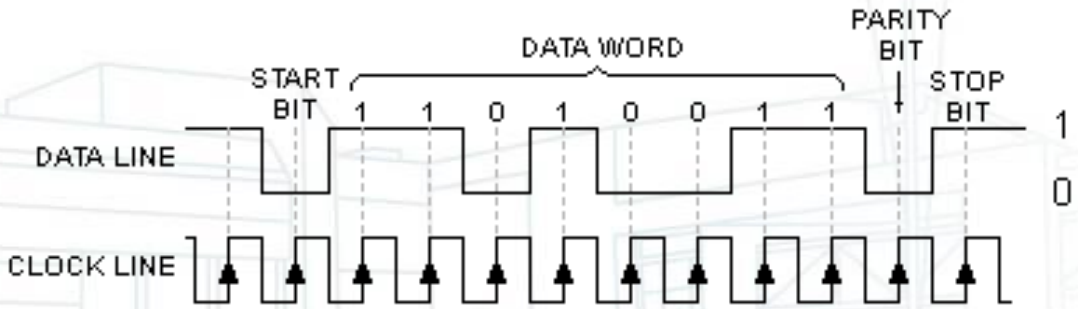
    Les règles peuvent être physiques (e.g. tension min/max des signaux, type de connecteur et câble) et/ou logicielles (interprétation des signaux)

Types de communications

- Communications numériques :
  - Protocoles, paquets/trames de données, protocoles liés à une liaison physique, protocoles purement logiciels… :

    Bit de start (aide à la synchro pour trouver le début du message dans un flot de données), header, taille des données utiles, numéro de paquet, données utiles/payload, footer, parité/checksum, bit de stop…

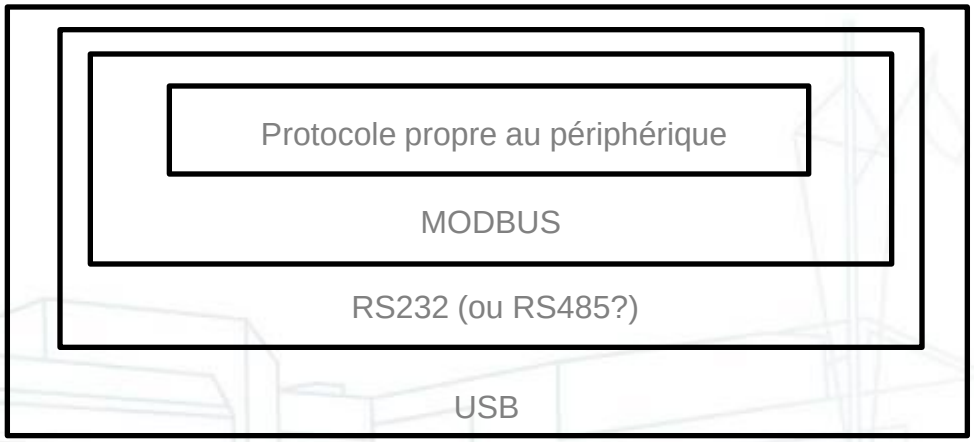
    Exemple : RS232



Types de communications

- Communications numériques :
  - Souvent couches englobantes de types de liaisons, protocoles...

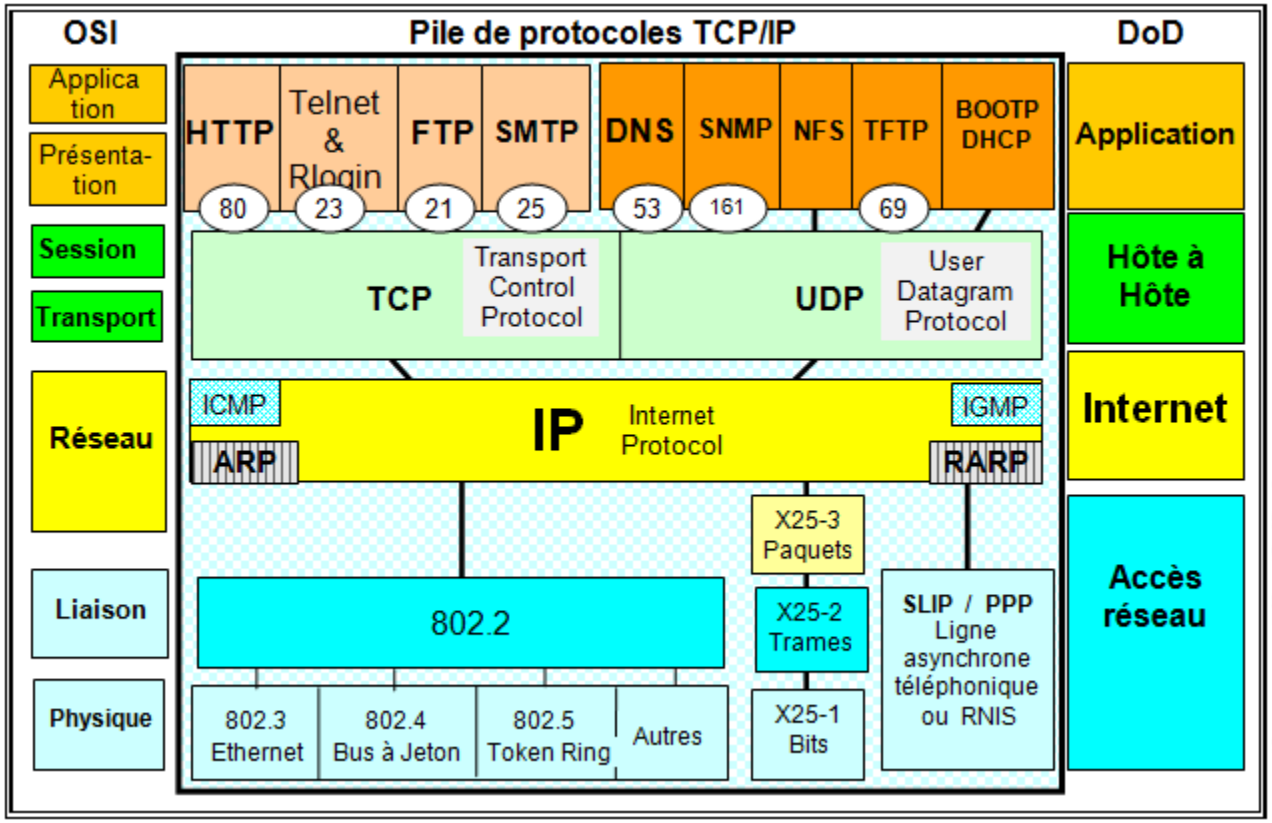
    Exemple : capteur de pression Keller P33x



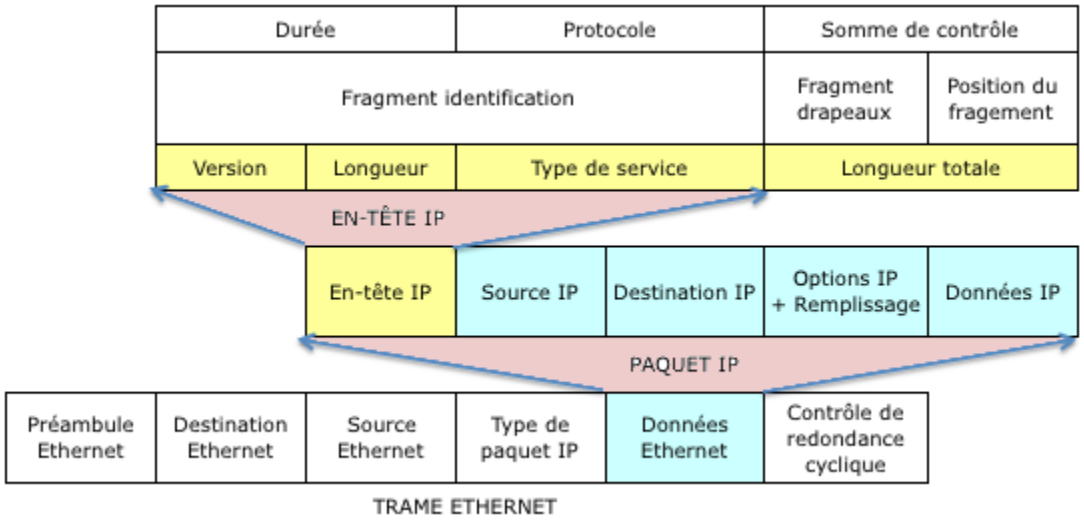
Logiciel PC : ouverture d'un port série (créé par le driver du convertisseur USB-RS232/RS485), envoi et réception de messages via libmodbus

Types de communications

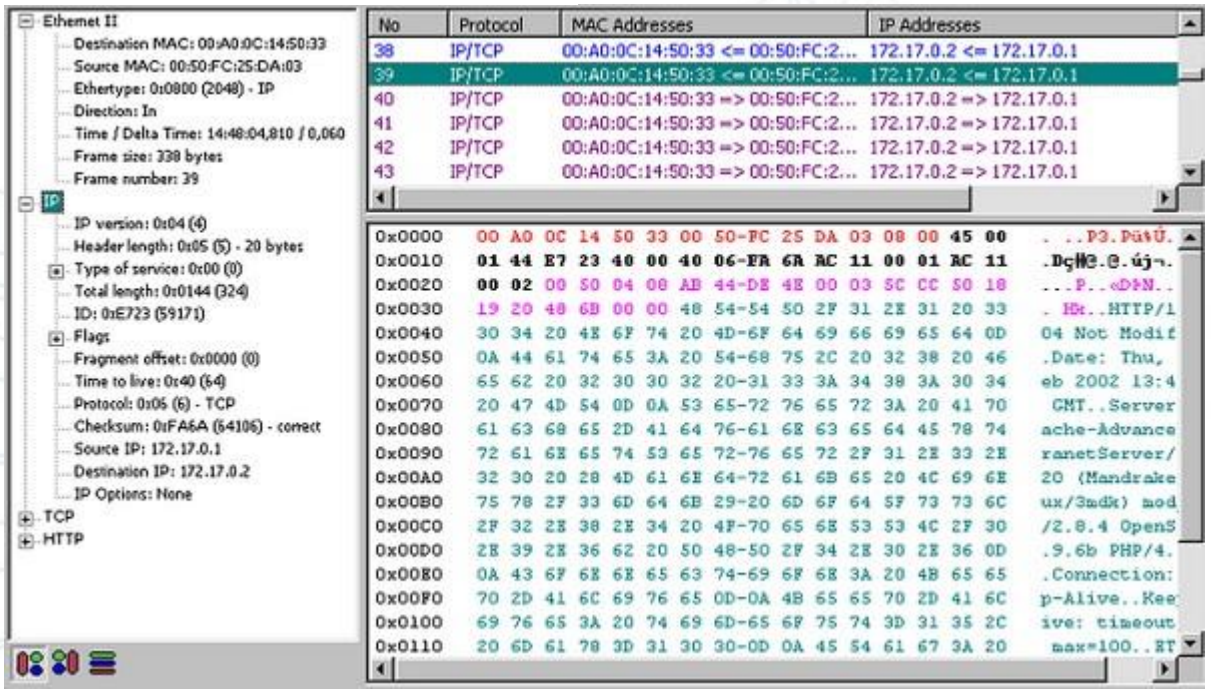
- Communications numériques :
  - Concept de pile de protocoles suivant le modèle OSI pour les réseaux IP :



Types de communications



TRAME ETHERNET



Capture des données qui passent sur une liaison Ethernet, avec des logiciels tels que Wireshark, ou de bons oscilloscopes…

Réseaux IP

Réseaux IP

- Présentation :
  - Les réseaux informatiques locaux d'entreprises ou domestiques et l'accès à Internet sont basés sur le protocole **IP** (Internet Protocol)
  - Par exemple, les **navigateurs web** utilisent principalement le protocole **http**, encapsulé dans le protocole **TCP**, lui-même dans **IP**, lui-même via des couches de protocoles plus basses tels que ceux de la **4G**, du **Wi-Fi**, **Ethernet**, etc. selon les appareils
  - On se limitera à l'**IPv4** (il y a aussi l'IPv6…)

Réseaux IP

- Présentation :
  - Pour bien comprendre leur fonctionnement, du vocabulaire et les concepts associés sont à connaître…
  - Note : il y a beaucoup d'abus de langage en pratique, e.g.

    « Réseau » parfois sous-entend « réseau IP », « réseau Wi-Fi » ou autre selon le contexte…

    Souvent, un appareil ayant été configuré pour effectuer une fonction spécifique sera désigné par le nom de cette fonction (e.g. « routeur », « pare-feu », « serveur DHCP », etc.), i.e. une fonction logicielle sera parfois confondue avec un matériel…

Réseaux IP

- Vocabulaire (bas niveau) :
  - **Interface réseau/NIC (Network Interface Card)** : périphérique d'ordinateur donnant accès à un réseau (typiquement Internet, mais pas forcément), e.g. **carte Wi-Fi** (réseau sans fil), **carte Ethernet** (réseau avec câbles à prises RJ45)
  - **Adresse MAC** (parfois nommée adresse physique) : identifiant unique au monde d'une interface réseau

    Format : 6 nombres hexadécimaux de 00 à FF séparés par des '– 'ou ':', les 3 premiers étant caractéristiques de la marque de l'interface réseau

    e.g. 00-AB-2D-F9-82-BC

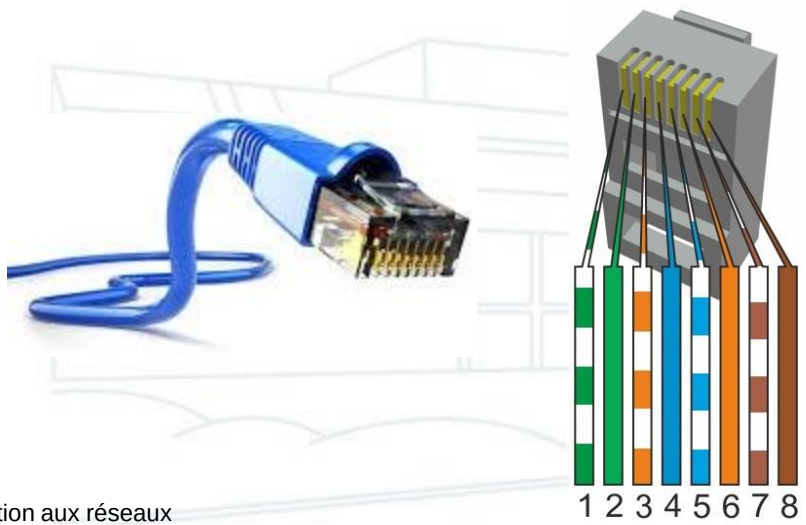
Note : dans l'expression « adresse MAC », « MAC » n'a pas de lien avec les célèbres ordinateurs d'Apple (Mac) et est l'acronyme de Media Access Control

- Vocabulaire (bas niveau, réseaux filaires Ethernet (norme IEEE 802.3)) :
  - Prise **RJ45** : type de prise utilisée pour les liaisons Ethernet
  - Câble Ethernet **droit** vs **croisé**

    On utilise un câble croisé quand on veut relier 2 PC ensembles, l'idée est que les pins **Tx+** et **Tx-** de l'un doivent aller sur les **Rx+** et **Rx-** de l'autre

    Regarder les couleurs des fils dans les connecteurs, s'ils sont dans le même ordre aux 2 extrémités c'est droit

    En pratique, certaines cartes réseaux peuvent automatiquement croiser ou décroiser pour qu'on n'ait pas à se soucier du type de câble…



T-568A

Pin	Description	10base-T	100Base-T	1000Base-T
1	Transmit Data+ or BiDirectional	TX+	TX+	BI_DA+
2	Transmit Data- or BiDirectional	TX-	TX-	BI_DA-
3	Receive Data+ or BiDirectional	RX+	RX+	BI_DB+
4	Not connected or BiDirectional	n/c	n/c	BI_DC+
5	Not connected or BiDirectional	n/c	n/c	BI_DC-
6	Receive Data- or BiDirectional	RX-	RX-	BI_DB-
7	Not connected or BiDirectional	n/c	n/c	BI_DD+
8	Not connected or BiDirectional	n/c	n/c	BI_DD-

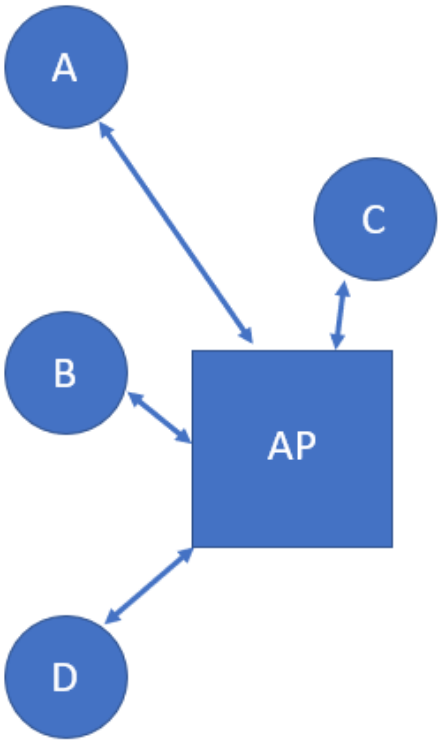
PinoutsGuide.com

Crossover LAN cable					
RJ45		Color		RJ45	
Function	*Pin*	*Color*	*Pair*	*Pin*	*Function*
TX+	1	white/green	3	3	TX+
TX-	2	Green	3	6	TX-
RX+	3	white/orange	2	1	RX+
	4	Blue	1	4	
	5	white/blue	1	5	
RX-	6	Orange	2	2	RX-
	7	white/brown	4	7	
	8	Brown	4	8	

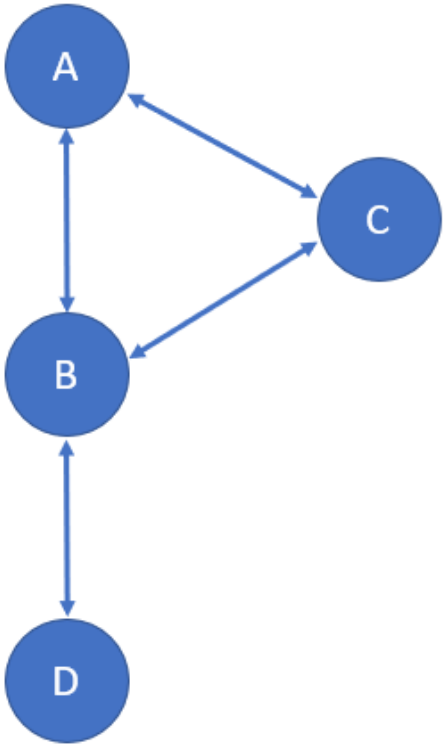
Réseaux IP

- Vocabulaire (bas niveau, réseaux sans fil Wi-Fi (norme IEEE 802.11)) :
  - **SSID** : nom d'un réseau Wi-Fi
  - **AP** (Access Point/Point d'accès/Hotspot) : périphérique générant un réseau Wi-Fi en mode infrastructure
  - **AP client** : périphérique se connectant à un réseau Wi-Fi généré par un AP
  - Wi-Fi **ad-hoc** vs **infrastructure** : **le mode infrastructure est le plus courant**, le mode ad-hoc n'a pas de concept d'AP avec plusieurs AP client connectés dessus, tous les périphériques sont à égalité
  - **Wi-Fi mesh 802.11s** : chaque périphérique peut en plus servir de relais (voir http://www.ensta-bretagne.fr/lebars/Share/mesh_network.txt pour un exemple de configuration pour un Ubiquiti Bullet M2)

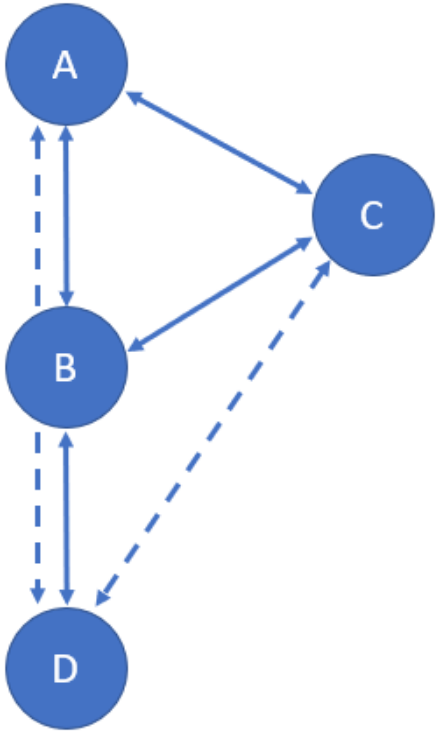
Réseaux IP



Infrastructure mode



Ad-hoc mode



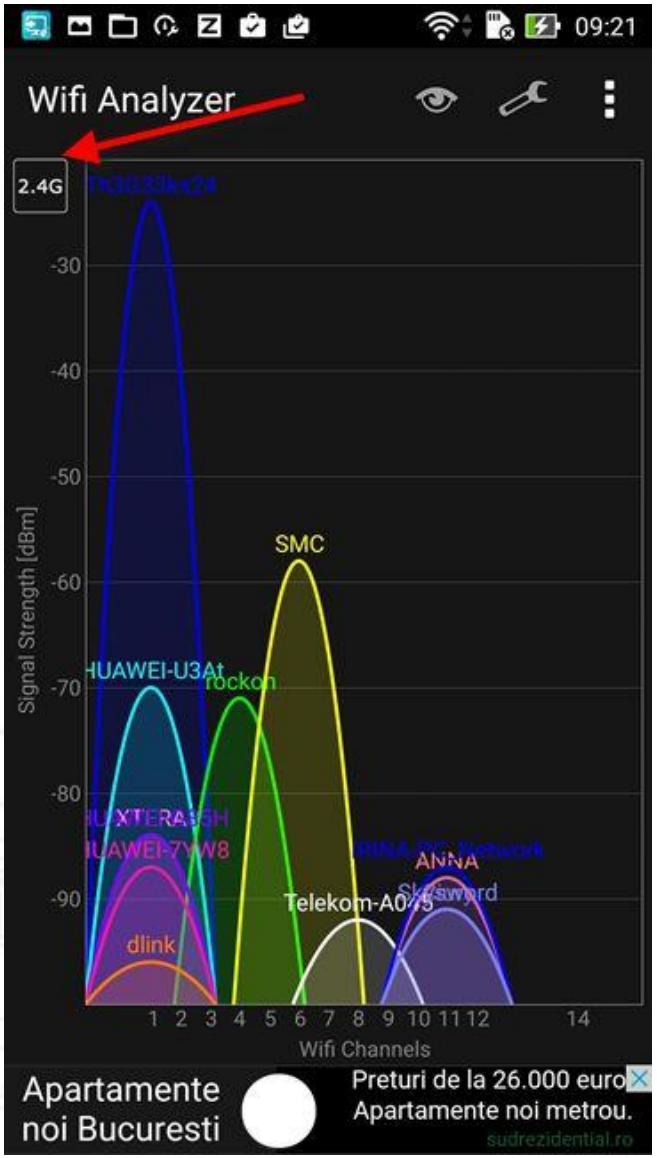
Mesh mode

Note : certaines interfaces Wi-Fi peuvent être configurables dans l'un ou l'autre des modes, voire dans plusieurs modes simultanément, voir e.g. https://www.ensta-bretagne.fr/lebars/wifi_devices.csv . En plus des modes **AP**, **AP client** (parfois appelé **Station** ou encore **Managed**), il y a ceux du **Wi-Fi Direct** (**P2P Group Owner** qui ressemble à **AP**, **P2P client** qui ressemble à **AP client**) qui sont utilisés pour certaines fonctions souvent nommées **Nearby sharing** (e.g. dans **Windows 10 Action Center**, https://www.xda-developers.com/how-to-use-nearby-sharing-in-windows-11/) pour le partage de fichiers rapide (clic droit et **Share** sur le fichier à partager et sélectionner la destination, attention à bien allumer le Wi-Fi au préalable pour qu'il puisse être configuré automatiquement) et **Miracast** pour le partage d'écran et haut-parleurs (e.g. voir **Connect** dans **Windows 10 Action Center**, **Settings\Connection preferences\Cast** sur certains smartphones **Android**).

Réseaux IP

- Vocabulaire (bas niveau, réseaux sans fil Wi-Fi (norme IEEE 802.11)) :
  - **Fréquences**, **canaux,** répartition 1-6-11 ou à la limite 1-5-9-13…

    Voir **Wifi Analyzer** Android app…



Réseaux IP

- Vocabulaire :
  - **Adresse IP** : identifiant d'une interface réseau sur un réseau IP, unique sur ce réseau IP particulier

    Format (pour IPv4) : 4 nombres décimaux de 0 à 255 (i.e. total de 32 bits) séparés par des '.'

    e.g. : 172.20.25.154

    IPv6 : e.g. fe80::e4f9:c270:e9fd:a3be, attention à ne pas confondre avec une adresse MAC!
  - **Masque de sous-réseau/netmask** : combiné avec l'adresse IP (bit à bit, adresse IP & netmask = adresse du réseau), permet de savoir combien de périphériques peuvent être mis sur ce réseau IP

    e.g. pour IPv4 : 255.255.0.0 => pour un réseau d'adresse 172.20.0.0, 256*256-2 IP réellement attribuables avec des IP en 172.20.X.X

    255.255.255.0 => pour un réseau d'adresse 192.168.1.0, 256-2 IP réellement attribuables avec des IP en 192.168.1.X

    Les **2 IP manquantes** dans le nombre correspondent à la 1ère (**adresse du réseau**) et la dernière (**adresse de diffusion/broadcast**). Les paquets envoyés à l'adresse de broadcast sont pris en compte par tous les périphériques de ce sous-réseau

Réseaux IP

- Vocabulaire :
  - **Masque de sous-réseau** : notation **CIDR**

    e.g. **172.20.25.154/16** -> adresse IP **172.20.25.154** avec le masque **255.255.0.0**

    192.168.1.101/24 -> adresse IP **192.168.1.101** avec le masque **255.255.255.0**

    /16 et **/24** correspondent au nombre de 1 si on écrivait le masque de sous-réseau en binaire

    Ainsi, si on a **/X** le nombre d'**IP attribuables** est **2^(32-X)-2**

Réseaux IP

- Vocabulaire :
  - **Passerelle/Gateway** : périphérique (routeur, en général) à contacter si une adresse IP hors du sous-réseau doit être atteinte
  - **DHCP** : protocole de configuration automatique d'adresse IP
  - **Serveur DHCP** : périphérique fournissant des adresses IP aux périphériques compatibles
  - Adresse IP **statique** vs **dynamique** : statique implique de devoir spécifier manuellement au moins l'adresse IP et le masque de sous-réseau, dynamique nécessite qu'un serveur DHCP soit présent sur le réseau pour les configurer automatiquement

Réseaux IP

- Vocabulaire :
  - Gammes d'adresses IP dites « **privées** » classiques :

    127.0.0.1->**localhost** : adresse par défaut du PC local, sert à faire des tests, ne permet pas de communiquer avec l'extérieur

    192.168.0.X, **192.168.1.X**, **10.42.0.X**->réseaux locaux de moins de 254 PCs (voire encore moins s'ils ont plusieurs interfaces réseaux, s'il y a d'autres périphériques réseaux…), e.g. réseaux de Livebox, hotspot Wi-Fi sous Ubuntu, etc.

    172.20.X.X->réseaux d'entreprise de moins de 65534 PCs, e.g. salles infos de l'ENSTA Bretagne

    169.254.X.X->si des PCs en réseau sont réglés en DHCP et qu'il n'y a pas de serveur DHCP, ils vont prendre ce type d'adresse IP au bout de quelques min

    224.0.0.0 à **239.255.255.255**->**multicast** : un peu comme le broadcast mais permet de faire des choses plus propres et optimales

    Voir aussi e.g. https://en.wikipedia.org/wiki/Private_network

    Les adresses ne ressemblant pas à ces exemples sont en général des adresses (dites « **publiques** ») attribuées par un fournisseur d'accès Internet

Réseaux IP

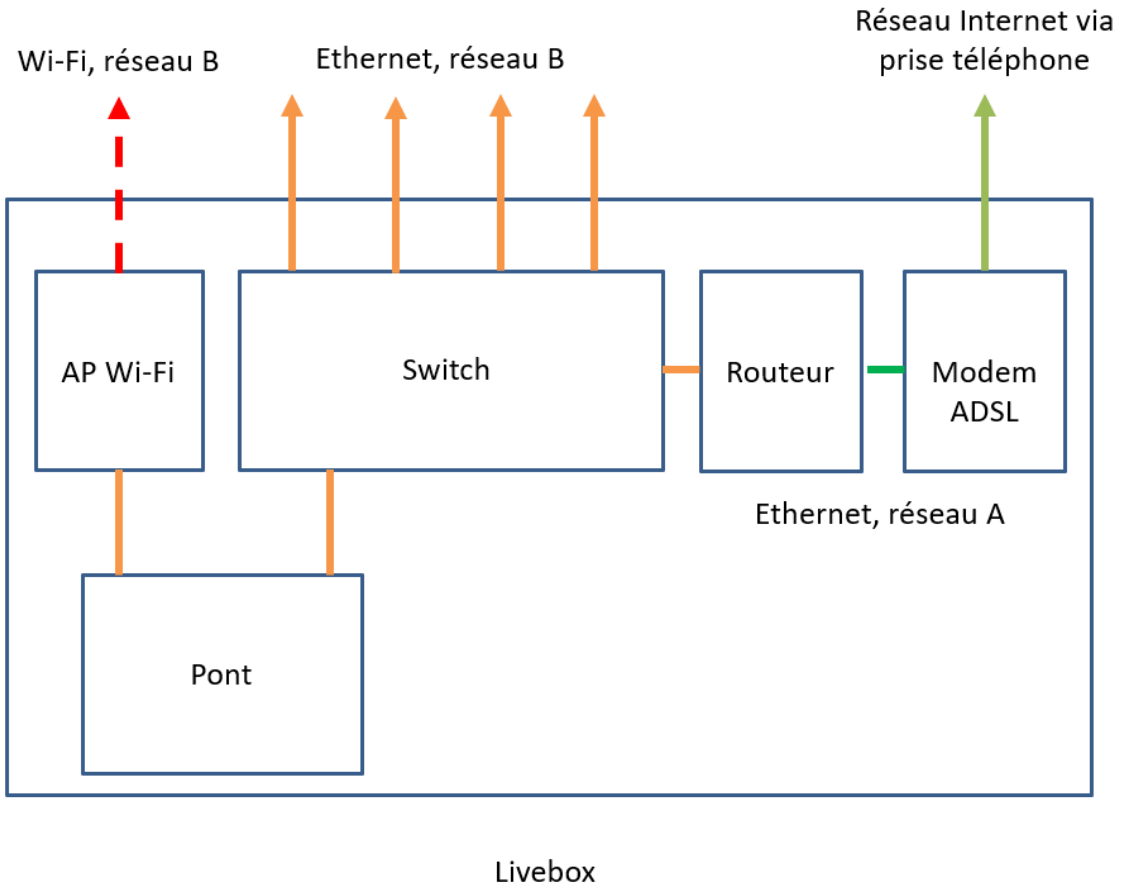
- Vocabulaire :
  - **Pont/Bridge** : fusionne 2 interfaces réseaux différentes en 1 seule (e.g. 1 Wi-Fi et 1 Ethernet) visible sous 1 seule adresse IP (ou parfois invisible du point de vue du protocole IP)
  - **Commutateur/Switch** : périphérique utilisé pour relier plusieurs périphériques sur un même réseau IP
  - **Concentrateur/Hub** : presque comme un switch mais moins intelligent
  - **Routeur/Router** : périphérique utilisé pour relier des réseaux IP différents (gammes d'adresses IP différentes)

Contrairement à un **hub**, un **switch** ne reproduit pas sur tous ses ports Ethernet chaque trame qu'il reçoit : il utilise l'**adresse MAC** de destination dans la trame pour choisir le bon port. Les plus simples ne nécessitent aucune configuration particulière car ils n'ont pas besoin d'interpréter les données des protocoles des couches supérieures du modèle OSI

Un **routeur** utilise les **adresses IP** pour diriger les données entre typiquement 2 réseaux IP notés WAN (Wide Area Network) et LAN (Local Area Network). On appelle **routes** les infos indiquant les liens de passage d'un réseau IP à un autre (souvent unidirectionnel)

Réseaux IP

- Exemple : Livebox



Réseaux IP

- TCP et UDP :
  - Protocoles encapsulés dans IP
  - **TCP** fait le maximum pour que les paquets de données arrivent dans le bon ordre à la destination voulue, il y a un **concept de connexion**
  - **UDP** est plus simple (et donc plus rapide) : les paquets sont envoyés sans **aucune vérification** (c'est donc au protocole encapsulé de s'assurer de leur bonne réception si c'est important)
  - Ces protocoles ont un concept de ports : chaque **port (entre 0 et 65535)** est censé être associé à un **protocole donné** ou une application donnée (ceux <1024 sont en général réservés au système)

Réseaux IP

- TCP et UDP :
  - Ports courants

    TCP 80 : protocole **http** (navigateurs et serveurs web)

    TCP 443 : protocole **https** (navigateurs et serveurs web)

    TCP 21 : protocole **ftp** (échanges de fichiers simples)

    TCP 22 : protocole **ssh** (exécution de commande à distance)

    TCP 22 : protocoles **sftp**, **scp** (échanges de fichiers via **ssh**)

    TCP 23 : protocole **telnet** (version simplifiée de **ssh**, CTRL+] ou CTRL+$ puis Q pour quitter…)

    
    TCP, UDP 3389 : protocole **RDP** (bureau à distance)

    
    TCP, UDP 4000 : protocole **NX NoMachine** (bureau à distance)

    
    TCP 5760, UDP 14550 : protocole **MAVLink** (télémétrie de drones)

- Concept de serveur / client :
  - **Serveur** : programme (ou par abus de langage ordinateur avec un ou des programmes de ce type) attendant des connexions avec un protocole spécifique
  - **Client** : programme (ou par abus de langage ordinateur avec un ou des programmes de ce type) se connectant à un serveur quand il en a besoin (le client initie la connexion)

    Exemple : un **serveur web** est un programme attendant des connexions de clients sur le port réseau **TCP 80**. Il gère toutes les pages web et autres données du site. Lorsqu'un **client navigateur web** (Google Chrome, Internet Explorer, Mozilla Firefox sont des exemples) s'y connecte et demande une page web en particulier (e.g. via le protocole **http**), il va la lui envoyer et le client va l'afficher à l'utilisateur. Typiquement un serveur web est sur un ordinateur puissant capable de gérer des milliers de connexions d'ordinateurs clients.

To create a very simple web server: **python -m http.server --directory . 80**. Adding an **index.html** file would display it on the client instead of the list of files on the server directory.
Simple temporary Python-based web server for quick download and upload of files between 2 computers (read inside **server.bat** for quick info, launch **server.py** manually if needed): https://www.ensta-bretagne.fr/lebars/upload.zip

- Concept de serveur / client :

  Exemple avec le protocole **http** : pour demander la page web par défaut du serveur accessible par l'URL http://checkip.dyndns.org/ (l'adresse IP correspondante est 216.146.38.70), un client peut envoyer (e.g. via des sockets TCP/IPv4 sur le port 80) :

```
GET http://checkip.dyndns.org/ HTTP/1.1\r\n
Host: 216.146.38.70\r\n
Accept: text/html\r\n
\r\n
\r\n
```

  Le serveur répondra :

```
HTTP/1.1 200 OK\r\n
Content-Type: text/html\r\n
Server: DynDNS-CheckIP/1.0\r\n
Connection: close\r\n
Cache-Control: no-cache\r\n
Pragma: no-cache\r\n
Content-Length: 105\r\n
\r\n
\r\n
<html><head><title>Current IP Check</title></head><body>Current IP Address: 86.229.255.67</body></html>\r\n
\r\n
\r\n
```

- Concept de serveur / client :

Exemple avec le protocole **smtp** : pour demander à un serveur mail (e.g. smtp.orange.fr, 80.12.242.10) d'envoyer un mail, un client peut envoyer (e.g. via des sockets TCP/IPv4 sur le port 25) :

```
HELO 86.229.255.67\r\n
MAIL FROM:<sender_example@ensieta.fr>\r\n
RCPT TO:<receiver_example@ensieta.fr>\r\n
DATA\r\n
From: "Sender EXAMPLE" <sender_example@ensieta.fr>\r\n
To: "Receiver EXAMPLE" <receiver_example@ensieta.fr>\r\n
Subject: Test e-mail\r\n
\r\n
Message content test.\r\n
\r\n
.\r\n
QUIT\r\n
```

Le serveur répondra :

```
220 mwinf5d37 ME ESMTP server ready\r\n
250 mwinf5d37 hello [86.229.255.67], pleased to meet you\r\n
250 2.1.0 <sender_example@ensieta.fr> sender ok\r\n
250 2.1.5 <receiver_example@ensieta.fr> recipient ok\r\n
354 enter mail, end with "." on a line by itself\r\n
250 2.0.0 QLf61u00D1U1h4N03Lf8Vb mail accepted for delivery\r\n
221 2.0.0 mwinf5d37 ME closing connection\r\n
```

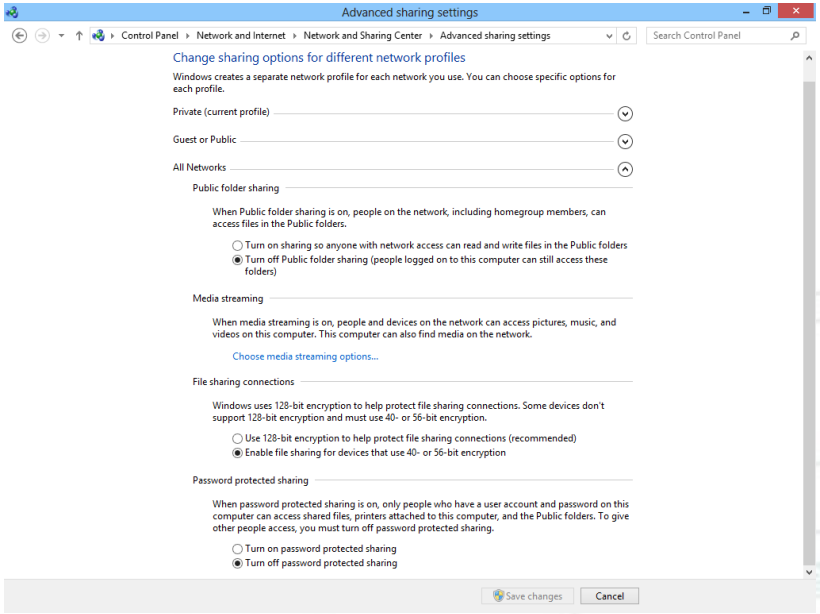
Voir https://www.ensta-bretagne.fr/lebars/tutorials/TD_IP.pdf pour d'autres types de communications entre serveur et client

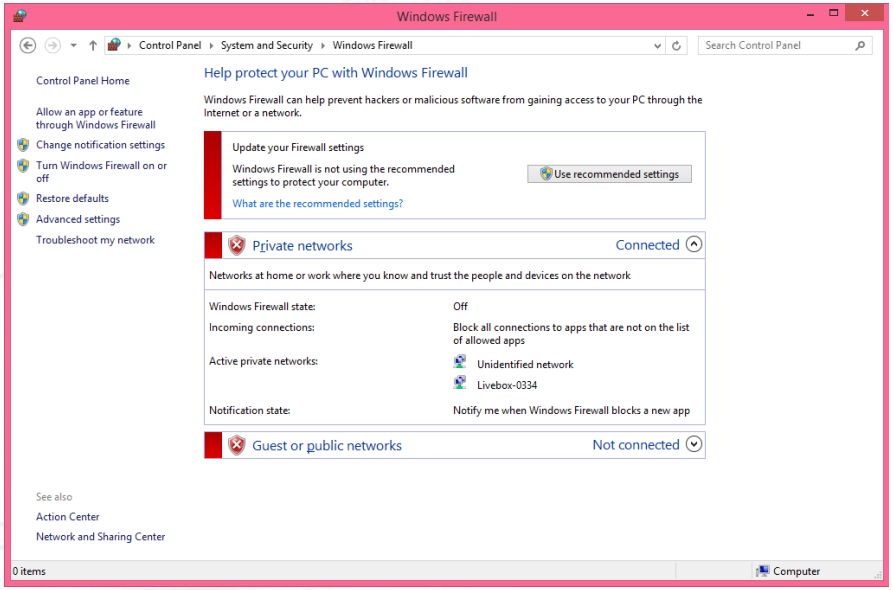
Réseaux IP

- Blocages possibles :
  - **Pare-feu/Firewall** : blocage de port réseaux TCP/UDP spécifiques ou blocage de l'accès réseau pour certaines applications (les routeurs en ont souvent, Windows en a aussi un activé par défaut…). Le blocage peut faire une distinction entre connexions entrantes et sortantes
  - **Antivirus** : blocage de certaines applications
  - **Profils réseaux** : Windows (ou des logiciels tiers) est capable de reconnaitre le réseau sur lequel on est connecté et d'appliquer des paramètres spécifiques (e.g. adresse IP, paramètres du pare-feu) à ce réseau, ces paramètres peuvent changer si le réseau change (voir secpol.msc\Network List Manager Policies). Des fonctions similaires existent sous certains Linux

- Blocages possibles :
  - Profils réseaux Windows 8 :

    2 types définis : **Public** ou **Private**, pour régler on peut e.g. utiliser **regedit** et mettre **Category** à **1** (Private) ou **0** (Public) dans les réseaux listés dans **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkList\Profiles**





Réseaux IP

- Blocages possibles :
  - **Proxy** : les accès réseaux (typiquement Internet) ne se font qu'en passant par l'intermédiaire d'un périphérique spécifique, qui peut ou non accepter de laisser passer certaines informations

    Exemple : configuration de l'ancien proxy de l'école pour divers usages

    http://www.ensta-bretagne.fr/lebars/Share/apt.conf

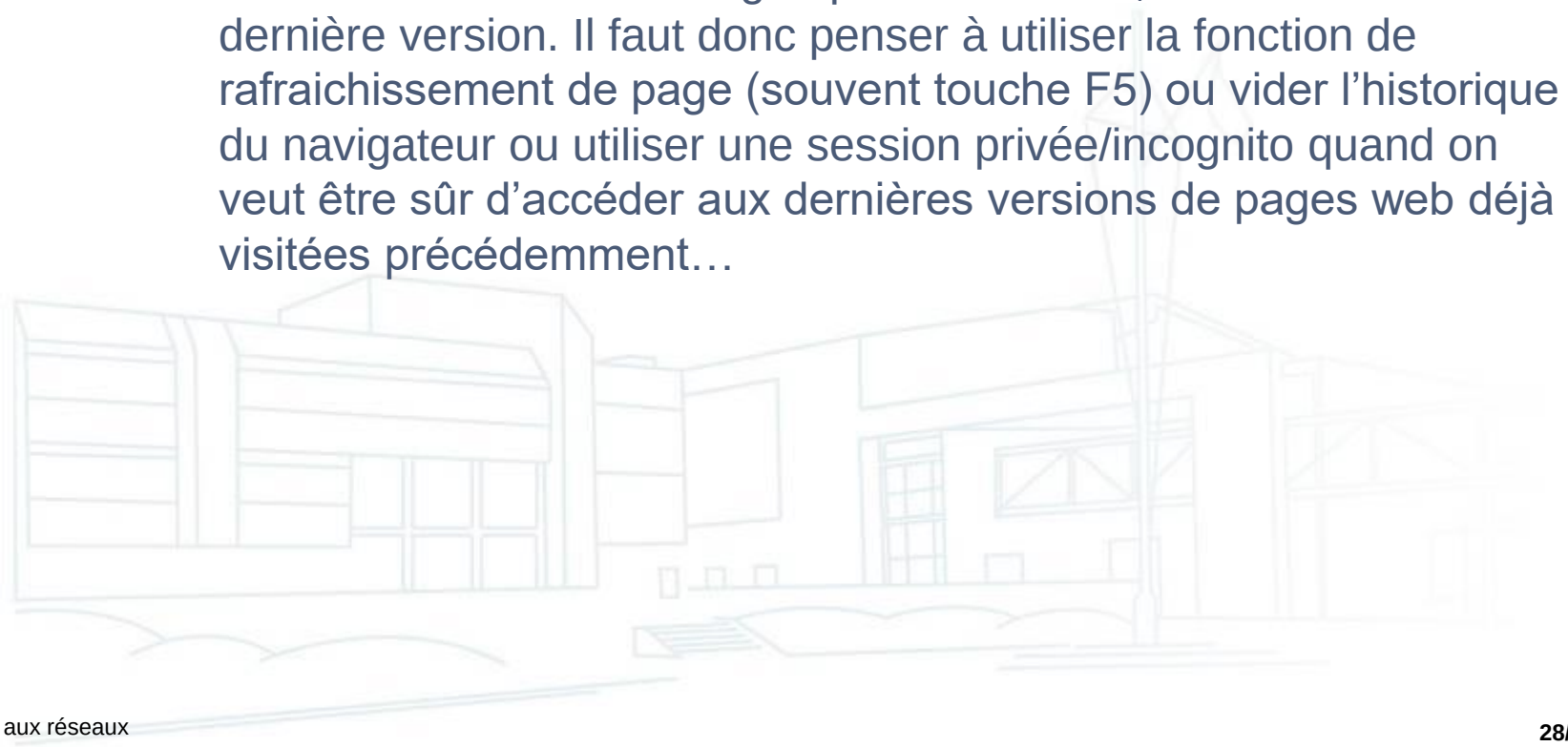
    Commandes Linux

```
export http_proxy=http://192.168.1.17:8080
export https_proxy=https://192.168.1.17:8080
export ftp_proxy=ftp://192.168.1.17:8080
export socks_proxy=socks://192.168.1.10:822
```

    Proxy socks : voir e.g. **Proxifier** (Windows), **tsocks** (Linux) pour pouvoir l'utiliser avec tout protocole basé sur les sockets (contrairement à juste http, https, ftp)

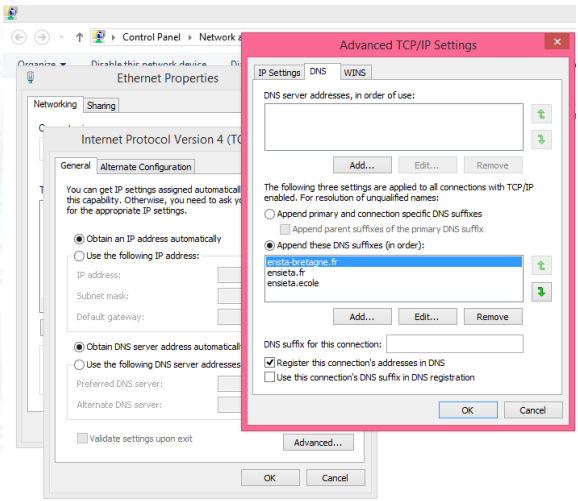
- Blocages possibles :
  - **Navigateurs web** : attention à la **mise en cache** des données

    Si par exemple on clique sur un lien vers cette présentation, il se peut que pour des raisons d'optimisation, le navigateur affiche une ancienne version téléchargée précédemment, et non la toute dernière version. Il faut donc penser à utiliser la fonction de rafraichissement de page (souvent touche F5) ou vider l'historique du navigateur ou utiliser une session privée/incognito quand on veut être sûr d'accéder aux dernières versions de pages web déjà visitées précédemment…



Réseaux IP

- Correspondances entre adresses IP et noms :
  - **Nom de l'ordinateur/computer name/hostname**
    Peut être utilisé à la place de l'IP (sous certaines conditions)
    Cas particulier : sur un ordinateur, le nom **localhost** existe toujours et correspond à **127.0.0.1**, et est donc limité aux communications restant sur l'ordinateur
  - **Nom de domaine/domain, groupe de travail/network group/workgroup** : Concepts assez présents dans les réseaux d'entreprises ou écoles sous Windows
  - **DNS** (Domain Name Server)
    Ce type de serveur s'occupe de gérer la correspondance entre les adresses IP et noms
    Suffixe DNS pour pouvoir juste taper **moodle** et non **moodle.ensta-bretagne.fr** ou **portail** au lieu de **portail.ensieta.ecole**…

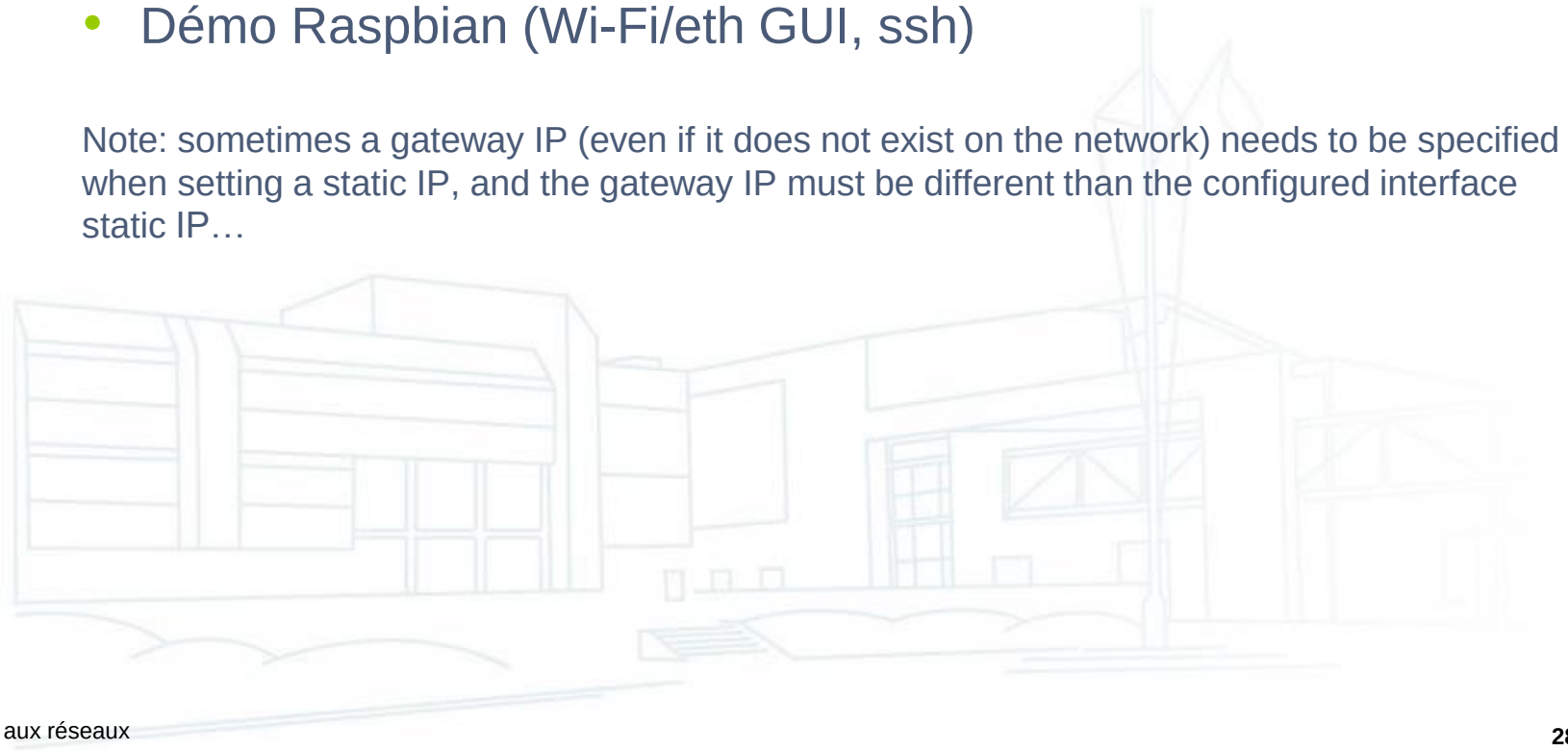


Configuration du réseau pour l’accès à distance

Configuration du réseau pour l'accès à distance

- Configuration via GUI, commandes, fichiers de config, etc.
  - Démo Windows (Wi-Fi/eth classic UI, modern UI, RDP, ssh)
  - Démo Ubuntu (Wi-Fi/eth GUI, terminal-only GUI, ssh)
  - Démo Raspbian (Wi-Fi/eth GUI, ssh)

Note: sometimes a gateway IP (even if it does not exist on the network) needs to be specified when setting a static IP, and the gateway IP must be different than the configured interface static IP…



Configuration du réseau pour l'accès à distance

- Commands and files for advanced configuration
  - Ubuntu alternate GUI: **nm-connection-editor** (e.g. useful to configure an Ethernet bridge)
  - Ubuntu terminal-only: **nmtui**
  - Ubuntu Server: **netplan apply** in combination with files in **/etc/netplan/** or **/run/netplan** (consider running **sudo apt install network-manager** as soon as you get the Internet to be able to use **nmtui**)
  - Raspbian: **/etc/dhcpcd.conf**, **/etc/wpa_supplicant/wpa_supplicant.conf**, **/etc/network/interfaces**
  - Windows: **netsh** (rarely necessary)

More info in https://www.ensta-bretagne.fr/lebars/Share/Ubuntu.txt, https://www.ensta-bretagne.fr/lebars/Share/Raspbian.txt, https://www.ensta-bretagne.fr/lebars/Share/Windows.txt, https://www.ensta-bretagne.fr/lebars/tutorials/TD_wifi_rtk_multiboot_rpi.pdf

Configuration du réseau pour l'accès à distance

- Tests, diagnostics :
  - **ipconfig /all** (Windows), **ifconfig**, **ip a** (Linux) : voir ses interfaces réseaux et leurs adresses IP, MAC, etc. A noter que certains logiciels rajoutent des interfaces réseaux virtuelles et qu'il y a aussi toujours l'interface virtuelle **localhost**
  - **ping IP_OR_NAME** : tester la communication avec une adresse IPv4 (rajouter paramètre **-4**) ou v6 (rajouter paramètre **-6**), ou un nom réseau (dans ce cas l'adresse IPv4 ou v6 sera retrouvée, sous Windows parfois rajouter un point (**ping IP_OR_NAME.**) à la fin du nom (ou spécifier le nom complet avec le domaine e.g. NUC.ensieta.ecole) peut aider s'il indique que le nom n'a pas été trouvé, essayer aussi **ipconfig /flushdns**…)
  - **Look@LAN** (Windows), **nmap 192.168.0.1-110** : scanner un réseau pour retrouver les différents périphériques connectés

Configuration du réseau pour l'accès à distance

- Tests, diagnostics :
  - **netstat -abn** or **TCPView** (Windows), **netstat -apn** (Linux) : Lister les serveurs TCP et UDP
  - **tracert IP_OR_NAME** (Windows), **traceroute IP_OR_NAME** (Linux) : donne plus de détails que ping

Configuration du réseau pour l'accès à distance

- Tests, diagnostics :
  - Cohérence adresses IP, DNS, adresses MAC

    hostname : nom réseau du PC

    nslookup 192.168.0.1 : trouver nom à partir d'IP (délais possibles lors de changements)

    arp -a : table de correspondances adresses IP et MAC

Configuration du réseau pour l'accès à distance

- Configuration du réseau pour l'accès à distance
  - Exécuter des commandes à distance :

    ssh user@192.168.0.1

    telnet 192.168.0.1
  - Transmettre des fichiers à distance :

    scp filetosend user@192.168.0.1:downloadsfolder

    sftp user@192.168.0.1

    Lecteurs réseaux (samba/smb)

    Exemple : lecteurs réseaux sur PC de salle info

    http://www.ensta-bretagne.fr/lebars/Share/Network%20drives%20ENSIETA.bat

    http://www.ensta-bretagne.fr/lebars/Share/remote_ensieta.sh

    wget http://192.168.0.1/filetodownload

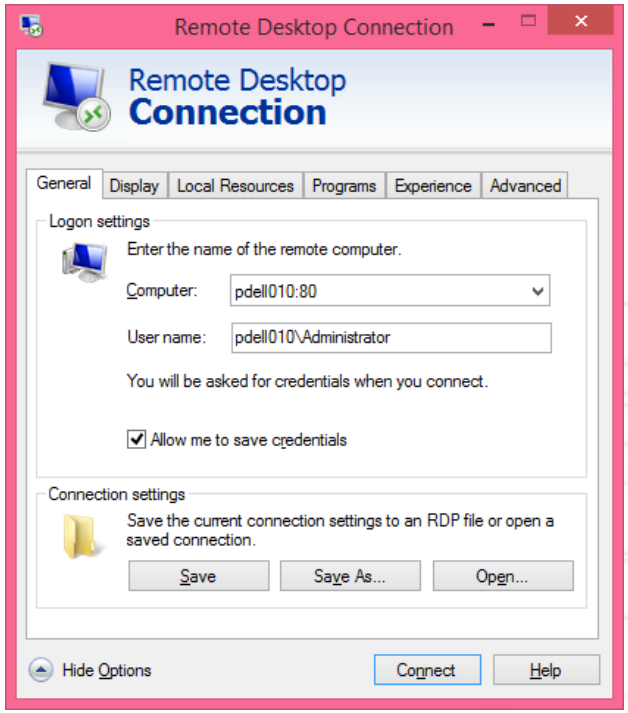
    Télécharge un fichier disponible sur un serveur web

To create a very simple web server: **python -m http.server --directory . 80**. By default, it makes available to download the files on the server directory. Simple temporary Python-based web server for quick download and upload of files between 2 computers (read inside **server.bat** for quick info, launch **server.py** manually if needed): https://www.ensta-bretagne.fr/lebars/upload.zip

Configuration du réseau pour l'accès à distance

- Configuration du réseau pour l'accès à distance
  - Bureau à distance :

    RDP, **NoMachine**, Team Viewer, VNC, ssh+X



Note : Sur certains PC pour **mstsc /admin** il faut préciser le nom de domaine (nom du serveur si pas dans un domaine)…

Configuration du réseau pour l'accès à distance

- Configuration du réseau pour l'accès à distance
  - **Port série** via serveur/client **TCP** : **socat** sous Linux, **Virtual Serial Port Emulator** (VSPE)+**com0com** sous Windows, permettent aussi de créer des ports série virtuels, etc.

    Peut servir par exemple à récupérer les données d'un capteur branché sur un PC embarqué et leurrer un programme de test propriétaire lancé sur un PC portable en faisant comme si le capteur était branché sur le PC portable

Configuration du réseau pour l'accès à distance

Configuration du réseau pour l'accès à distance

- Virtual serial port and TCP client with debug :

  sudo socat -d -d PTY,raw,echo=0,link=/dev/ttyVUSB0,mode=666 tcp:192.168.0.56:5555

- Real serial port and TCP server with debug :

  socat -d -d /dev/ttyACM0,b9600,raw,echo=0,mode=666 tcp-listen:5555,reuseaddr,fork

fork : multiclient

-u, -U : unidirectionnel

VSPE (Virtual Serial Port Emulator) Connector+TCP client/server for Windows

com0com : si les ports virtuels de **VSPE** ne conviennent pas (e.g. une application propriétaire ne les reconnait pas bien), ceux de **com0com** ont plus d'options

Configuration du réseau pour l'accès à distance

- Configuration du réseau pour l'accès à distance
  - **str2str** (GUI **strsvr** sous Windows, de https://rtklib.com/) a aussi des fonctionnalités similaires à **socat**
  - **mux_server.py** (voir https://github.com/lebarsfa/mux_serial) peut aussi être utile si plusieurs applications ont besoin d'accéder en lecture au même port série (attention, en écriture il peut y avoir d'autres problèmes)

    S'assure que toutes les données venant du port série sont bien redistribuées à toutes les applications, sinon des données prises par une application ne sont en général pas récupérables par les autres

    Sous Windows, **VSPE** Connector+Splitter+TCP server peut être utilisé à la place

Configuration du réseau pour l'accès à distance

- Configuration du réseau pour l'accès à distance
  - Rappel : blocages possibles :

    Pare-feu/Firewall, **Antivirus**, **Profils réseaux**, **Proxy**, etc.->voir précédemment…

  - Si le PC a plusieurs interfaces réseaux configurées, il se peut qu'il y ait des ambiguités sur l'interface par laquelle le trafic réseau voulu doit passer…

Configuration du réseau pour l'accès à distance

- Configuration du réseau pour l'accès à distance
  - Multiples interfaces réseaux configurées

    Pour limiter les problèmes :

    Mettre en DHCP puis **désactiver** celles qui ne seraient **pas utiles**

    Essayer de n'avoir qu'**une seule interface avec une passerelle/gateway** (typiquement, ce sera celle qui est configurée en DHCP (qui configurera automatiquement l'adresse IP, le masque de sous-réseau, la gateway, le serveur DNS, etc.) et connectée à Internet, **les autres** interfaces réseaux étant avec des adresses IP **statiques** de réseaux IP différents (et le masque de sous-réseau correspondant))

    Autrement, il est possible de définir des priorités pour chaque interface (**metric**, correspond à un coût)

    Redéfinir certaines routes manuellement peut aussi parfois aider selon les situations

Configuration du réseau pour l’accès à distance

- Configuration du réseau pour l’accès à distance
  - Info/rappel : fonctionnement, montage et installation d’un PC :

    Voir 2nde partie dans https://www.ensta-bretagne.fr/lebars/robotique_pratique.pdf

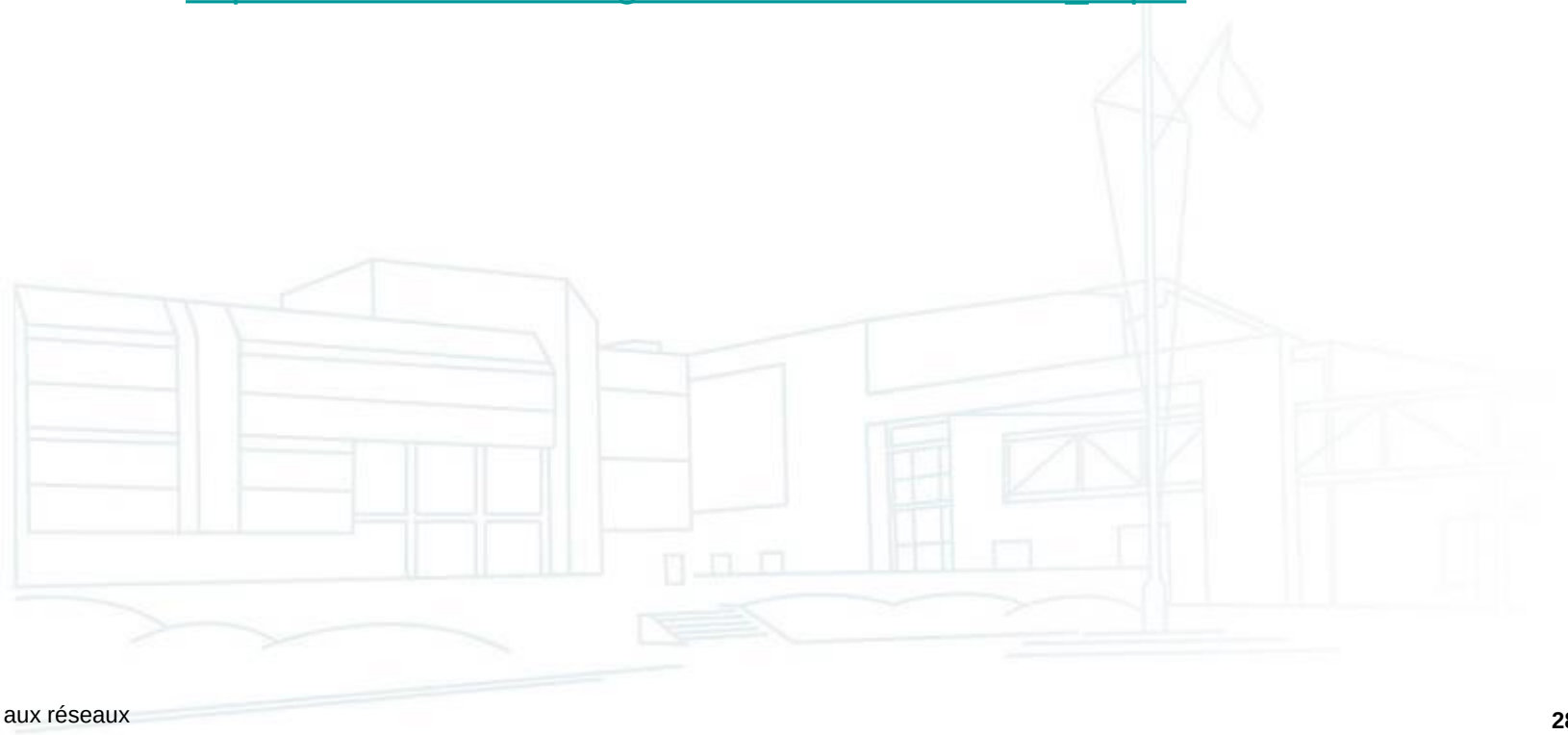
Réseaux et C/C++

Réseaux et C/C++

- C/C++
  - Plus d'infos sur les différences entre C et C++, compilateurs, IDE, utilisation de bibliothèques, etc. :

    http://www.ensta-bretagne.fr/lebars/tutorials/Complements_C-C++.pdf

    https://www.ensta-bretagne.fr/lebars/tutorials/TD_C.pdf



- Processus
  - Lorsqu'un OS (Operating System) est lancé, un certain nombre d'applications et services sont en cours d'exécution, qu'ils soient lancés automatiquement par le système ou manuellement par l'utilisateur : ils génèrent chacun 1 ou plusieurs **processus/process** (parfois aussi appelés **tâches/tasks**) en cours d'exécution dans la mémoire

  - Exemple du **Bloc-notes** sous Windows :

    Lorsque l'utilisateur clique sur le raccourci **Bloc-notes** le fichier **C:\Windows\system32\notepad.exe** est alors exécuté, ce qui crée le processus **Bloc-notes**, visible dans le **Gestionnaire de Tâches**. Si l'utilisateur clique une autre fois sur le raccourci, un 2ème processus **Bloc-notes** sera alors créé

Note : certaines applications ou services peuvent volontairement empêcher la création d'un 2ème processus si on tente de les lancer 2 fois simultanément, e.g. un lecteur multimedia peut préférer rajouter à la suite de sa playlist actuelle le nouveau fichier audio avec lequel on a voulu le lancer une 2ème fois plutôt que de le jouer en même temps que ce qu'il jouait déjà…

Réseaux et C/C++

- Processus
  - Chaque processus a en général :
    - Des **paramètres/arguments** d'appel qui lui sont propres (e.g. **notepad.exe file.txt**)
    - Un **dossier courant** (qui peut être différent de celui où se trouve son fichier exécutable principal)
    - Un processus **parent**, des processus **enfants** (e.g. utiliser **Process Explorer** (https://docs.microsoft.com/en-us/sysinternals/) pour voir l'arbre des processus sous Windows)
    - Une **date de démarrage**
    - Une **priorité**, qui peut changer au cours de l'exécution, gérée par l'**ordonnanceur/scheduler** de l'OS
    - Un **état**, e.g. working, suspended, terminated, etc.
    - Des **droits d'accès** spécifiques qui dépendent de la façon dont il a été lancé (e.g. hérités de ceux de l'utilisateur ou processus qui l'a lancé)
    - Sa **zone mémoire réservée**, etc.

Réseaux et C/C++

- Processus et C/C++
  - Ecrire un fichier **test.c**/cpp avec dedans une fonction **main()** puis le compiler en un **test.exe** permet de créer un fichier exécutable, double-cliquer dessus ou le lancer via le terminal permet de créer un processus
  - Dans le code C/C++, on peut créer des processus enfants avec la fonction **CreateProcess()** sous Windows ou **fork()** sous Linux

Réseaux et C/C++

- Processus, threads et C/C++
  - A l'intérieur d'un processus, toutes les instructions se succèdent normalement les unes après les autres, un appel à une fonction bloquant la fonction appelante jusqu'à ce qu'elle soit terminée
  - **Créer un processus enfant** permet de lancer l'exécution d'autres **instructions en parallèle** de celles du processus parent…

Note : même si l'ordinateur n'a qu'un processeur avec un seul cœur, les OS sont capables de simuler des exécutions d'instructions en parallèle (grâce à leur partie ordonnanceur/scheduler), de plus même sur un système multiprocesseur les différents types de mémoire ne sont pas forcément accessibles simultanément…

Réseaux et C/C++

- Processus, threads et C/C++
  - La communication inter-process est cependant un peu compliquée et coûteuse en ressources : utilisation de **signaux** (assez utilisé sous Linux, e.g. **CTRL-C** envoie le signal **SIGINT** au processus, ce qui par défaut le termine, mais on peut changer ce comportement), de **pipe**, de **files de messages**, de **fichiers** (sans doute le plus simple, voir https://www.ensta-bretagne.fr/lebars/Share/comm_file.zip), de **sockets** (étudiées plus en détails dans la suite car permet communication inter-ordinateur), etc.

  - Pour simplifier un peu l'exécution d'instructions en parallèle et notamment le partage de variables, le concept de **thread** a été développé
    - On parle parfois de **processus léger** pour désigner un thread, par opposition aux **processus lourds** qui sont ceux définis précédemment

Réseaux et C/C++

- Processus, threads et C/C++
  - Un **thread** est une sorte de **fonction spéciale** qui une fois créée **ne va pas bloquer la fonction appelante**, les instructions dans le thread s'exécutant toujours de manière successive comme pour les autres fonctions
  - On peut dire que la fonction **main()** est le **thread principal** d'un processus
  - Comme les processus, les threads ont un concept de **priorité** et d'**état** (e.g. working, suspended, terminated), par contre ils partagent la même zone mémoire que le processus qui les a créés

Réseaux et C/C++

- Processus, threads et C/C++
  - Dans un robot autonome, on a souvent un code qui ressemble globalement à

```
int main(int argc, char* argv[]) {
    while (!end) {
        data = get_sensors_data();
        process(data);
        send_receive_telemetry(data);
        update_ui(data);
        set_actuators(data);
    }
    return 0;
}
```

  - Les fonctions incluant des communications avec des périphériques bloquent souvent pendant plusieurs dizaines de ms (attentes de réponse dans un protocole de communication avec un capteur, etc.) donc en cumulé il y a un risque que cette boucle de régulation soit trop lente et que le robot ne se comporte pas bien => on utilise souvent 1 thread par périphérique pour paralléliser les communications…

Réseaux et C/C++

- Processus, threads et C/C++
  - Plusieurs spécifications et implémentations du concept de thread existent :

    Standard C++11 (ex **C++0x**) : bien que ce soit à privilégier dans le futur, c'est arrivé un peu tard et la majorité des programmes actuels ne l'utilisent pas encore pour des raisons de compatibilité avec d'anciens programmes, bibliothèques, etc.

    pthreads : probablement la plus simple, **portable** et répandue (surtout sous Linux) car au moins compatible C99 voire plus ancien

    Win32 API threads : Windows a les fonctions **CreateThread()** ou **_beginthreadex()**, etc. (à noter que l'implémentation de pthreads sous Windows se base probablement sur ces fonctions)

    Dans le cadre du cours, on va utiliser **pthreads** (voir doc sur http://man7.org/linux/man-pages/), le passage aux threads du C++11 s'en déduisant facilement

- Threads et C/C++
  - Création d'un thread avec **pthreads**

```c
#include <pthread.h>

void* thread0(void* param) {
    // ...
    return NULL;
}

int main(int argc, char* argv[]) {
    pthread_t t0;
    pthread_create(&t0, NULL, thread0, NULL);
    // ...
    pthread_join(t0, NULL);
    return 0;
}
```

Réseaux et C/C++

- Threads et C/C++
  - Création d'un thread avec **pthreads**

    Exemple : affichage de la touche appuyée par l'utilisateur pendant un long calcul dans le main()…

```c
#include <stdio.h>
#include <math.h>
#include <pthread.h>

void* thread0(void* param) {
    int c = 0;
    while (c != ' ') {
        c = getchar();
        printf("User pressed : %c\n", (char)c);
    }
    printf("Will stop as soon as possible...\n");
    return NULL;
}

int main(int argc, char* argv[]) {
    double d = 1;
    pthread_t t0;
    printf("Beginning.\n");
    pthread_create(&t0, NULL, thread0, NULL);
    while (d > 0.00000001) {
        d = fabs(log(exp(d))/1.000001);
        sched_yield();
    }
    pthread_join(t0, NULL);
    printf("End.\n");
    return 0;
}
```

- Threads et C/C++
  - Gestion de la fin du thread :

    pthread_join() permet d'attendre la fin du thread, récupérer sa valeur de retour et libérer les ressources liées à la création du thread

    pthread_detach() configure le thread pour qu'il libère lui-même les ressources liées à sa création lorsqu'il se termine

  - Contrôle du scheduler : avec **sched_yield()**, on demande au **scheduler** (dont le rôle est de décider quel thread est actif) de placer le thread en bas de la file d'attente des threads de même priorité, pour éviter de monopoliser le processeur

    Ceci n'est peut-être pas obligatoire selon ce que fait le thread (certaines fonctions impliquant une attente passive y font souvent déjà appel implicitement), la configuration du scheduler, etc.

- Threads et C/C++
  - Communication et synchronisation entre threads

    Comme toute fonction, la fonction d'un thread a **accès aux variables globales** du programme, ce qui facilite le partage de données entre threads

    La grande **difficulté** est de s'assurer que les **accès concurrents** aux mêmes variables, périphériques, etc. par différents threads en même temps ne causent pas d'effets inattendus!

    C'est d'autant plus difficile que le comportement des **schedulers** des OS n'est pas toujours bien documenté, ou qu'il est complexe…

    A noter que les OS dits « **temps réel** » suivent des règles plus strictes et prévisibles, mais la notion de temps réel n'est pas liée à une plus grande rapidité d'exécution…

Réseaux et C/C++

- Threads et C/C++
  - Communication et synchronisation entre threads

Opération atomique : opération **indivisible**. Un + entre nombres entiers n'est notamment pas une opération atomique par défaut en C/C++, il faut utiliser des fonctions spéciales (si disponibles) pour cela…

Section critique : opérations qui doivent être exécutées **sans être interrompues**

Exclusion mutuelle : des threads sont en exclusion mutuelle si seul l'un d'entre eux à la fois peut accéder à une ressource partagée

Réseaux et C/C++

- Threads et C/C++
  - Communication et synchronisation entre threads

Changement de contexte/context switch : sauvegarde de l'état d'un thread pour restaurer à la place celui d'un autre, géré par le scheduler. Sa durée peut être à prendre en compte…

Attente active vs **passive** : une attente **active vérifie de manière répétée une condition** sans laisser explicitement la possibilité au scheduler de passer à autre thread, alors qu'une attente **passive met le thread en attente** et utilise e.g. les fonctionnalités d'interruption du processeur pour vérifier l'état d'une condition alors que d'autres threads sont actifs

- Threads et C/C++
  - Communication et synchronisation entre threads

Deadlock : des threads sont **en attente** d'un évènement que seul l'un d'entre eux peut réaliser, celui-ci ne pouvant donc jamais se réaliser

Livelock : des threads **changent continuellement d'état** en fonction de l'action des autres

Illustration :

Livelock : 2 personnes arrivent face à face et se décalent dans le même sens pour passer. Tant qu'elles se décaleront du même côté au même moment elles ne pourront pas passer…

Deadlock : 4 voitures arrivent en même temps à une intersection avec 4 stops…

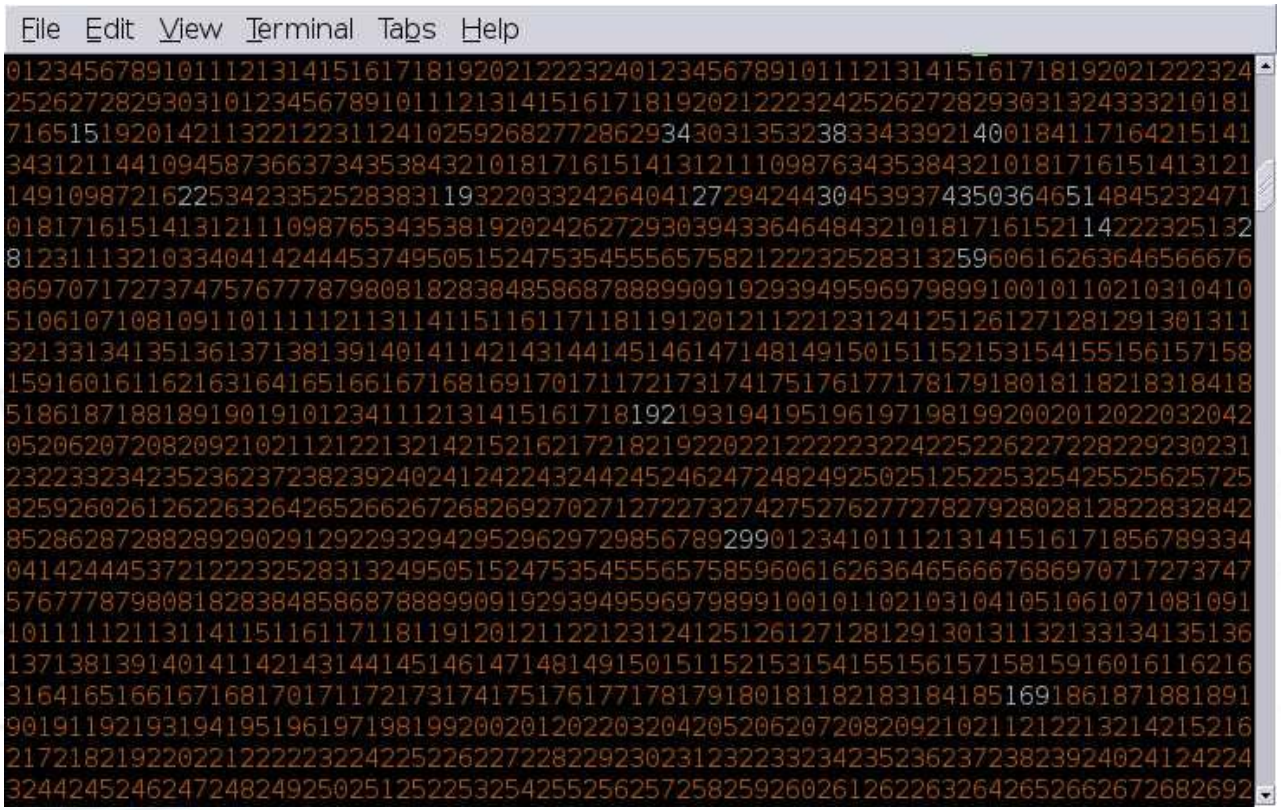
- Threads et C/C++
  - Communication et synchronisation entre threads

Famine : un ou plusieurs threads ne peuvent plus progresser car leur(s) ressource(s) critique(s) sont monopolisée(s) par un ou plusieurs threads concurrents. Similaire au **livelock**, sauf que seuls les threads lents seront affectés…

Thread-safe : a thread-safe function is one that can be safely (i.e., it will deliver the same results regardless of whether it is) called from multiple threads at the same time (see also https://en.wikipedia.org/wiki/Reentrancy_(computing))

- Threads et C/C++
  - Communication et synchronisation entre threads

Exemple de problème : 300 threads qui passent l'affichage du terminal de blanc vers orange, puis affichent un nombre, puis repassent le terminal de orange à blanc…



Source : cours d'OS d'Irvin PROBST 2006-2007

- Threads et C/C++
  - Communication et synchronisation entre threads

S'il n'y a pas de protection sur la ressource « affichage », voici ce qu'il se passe probablement par moments :

Thread1 : Passer l'affichage du terminal de blanc vers orange
SCHEDULER : INTERROMPT THREAD1 ET ACTIVE THREAD 2
Thread2 : Passer l'affichage du terminal de blanc vers orange
Thread2 : Afficher nombre
Thread2 : Passer l'affichage du terminal de orange vers blanc
SCHEDULER : FIN THREAD 2 ET ACTIVE THREAD 1
Thread1 : Afficher nombre
Thread1 : Passer l'affichage du terminal de orange vers blanc

=> **Conflit d'accès** sur la ressource « affichage », il faudrait définir le code gérant l'affichage comme une **section critique** et faire en sorte que les threads soient en **exclusion mutuelle**…

- Threads et C/C++
  - Communication et synchronisation entre threads

Autre exemple de problème courant :

Dans section critique d'un **Thread1** : prendre ressource A et la rendre quand B libre

Dans section critique d'un **Thread2** : prendre ressource B et la rendre quand A libre

Thread1 : Bloquer accès à A
Thread1 : Prendre A
Thread2 : Bloquer accès à B
Thread2 : Prendre B
Thread1 : Attente libération de B...
Thread2 : Attente libération de A...
=> **DEADLOCK**, il faudrait repenser le programme…

- Threads et C/C++
  - Communication et synchronisation entre threads

    Outils de synchronisation principaux fournis par **pthreads** :

    Mutex : pthread_mutex_init(), pthread_mutex_lock(), pthread_mutex_trylock(), pthread_mutex_unlock(), pthread_mutex_destroy(), etc.

    Exemple d'utilisation d'un mutex : affichage de la touche appuyée par l'utilisateur pendant un long calcul dans le main(), avec indication de l'état du calcul (« ready » si fini)…

Réseaux et C/C++

```c
#include <stdio.h>
#include <math.h>
#include <pthread.h>

double d = 1;
pthread_mutex_t m;

void* thread0(void* param) {
    int c = 0;
    while (c != ' ') {
        c = getchar();
        pthread_mutex_lock(&m);
        if (d > 0.00000001) printf("User pressed : %c (not ready)\n", (char)c);
        else printf("User pressed : %c (ready)\n", (char)c);
        pthread_mutex_unlock(&m);
    }
    printf("Will stop as soon as possible...\n");
    return NULL;
}

int main(int argc, char* argv[]) {
    pthread_t t0;
    printf("Beginning.\n");
    pthread_mutex_init(&m, NULL);
    pthread_create(&t0, NULL, thread0, NULL);
    while (d > 0.00000001) {
        pthread_mutex_lock(&m);
        d = fabs(log(exp(d))/1.000001);
        pthread_mutex_unlock(&m);
        sched_yield();
    }
    pthread_join(t0, NULL);
    printf("End.\n");
    return 0;
}
```

- Threads et C/C++
  - Communication et synchronisation entre threads

    Déroulement possible dans les boucles :

    main() : **lock()** va vérifier si un autre **lock()** est actuellement en cours, s'il n'y en a pas d est modifié et **unlock()** termine le **lock()**, sinon il va se mettre en attente passive au moins jusqu'à ce que thread0() appelle **unlock()**

    main() : **yield()** va laisser le scheduler activer thread0()

    thread0() : la plupart du temps, **getchar()** attend un caractère de l'utilisateur de manière passive donc le scheduler va activer le main(), sinon **lock()** va vérifier si un autre **lock()** est actuellement en cours, s'il y en a un thread0() va se mettre en attente passive au moins jusqu'à ce que le main() appelle **unlock()**, sinon il va vérifier la valeur de d et appeler **unlock()**

- Threads et C/C++
  - Communication et synchronisation entre threads

Remarques sur le code précédent :

Des lectures concurrentes (a priori non atomique) de la variable **d** ne sont pas protégées, dans l'idéal il faudrait aussi protéger par le mutex la vérification de la condition sur d dans le main…

En pratique, il semble que ce ne soit pas un problème en général, contrairement aux mélanges d'écritures et lectures concurrentes…

- Threads et C/C++
  - Communication et synchronisation entre threads

    Mélanges d'écritures et lectures concurrentes :

    En C/C++, l'opération d'affectation ou d'utilisation de valeur pour un entier signé de 4 octets est peut-être décomposée en plusieurs instructions pour le processeur, e.g. 1 par octet

    Exemple de problème : **Thread1** veut écrire -1 dans une variable qui était initialement à 0 pendant que **Thread2** la lit, sans mutex…

- Threads et C/C++
  - Communication et synchronisation entre threads

Mélanges d'écritures et lectures concurrentes :

Thread1 : Ecriture de l'octet 1
Thread2 : Lecture de l'octet 1
Thread2 : Lecture de l'octet 2
Thread2 : Lecture de l'octet 3
Thread2 : Lecture de l'octet 4
Thread1 : Ecriture de l'octet 2
Thread1 : Ecriture de l'octet 3
Thread1 : Ecriture de l'octet 4

FF	00	00	00
FF			
FF	00		
FF	00	00	
FF	00	00	00
FF	FF	00	00
FF	FF	FF	00
FF	FF	FF	FF

Par contre, pour des lectures concurrentes sans aucune écriture, il n'y aurait pas de problème…

- Threads et C/C++
  - Communication et synchronisation entre threads

Pour des mélanges de lectures concurrentes, si maintenant on suppose qu'une utilisation d'un entier signé de 4 octets (e.g. 0xFF000001) se fait par lecture de l'octet initial, lecture du suivant, lecture du suivant, lecture du suivant (utilisation d'adresses relatives au lieu d'absolues) :

Thread1 : Lecture de l'octet initial	FF			
Thread1 : Lecture de l'octet suivant	FF	00		
Thread2 : Lecture de l'octet initial	FF			
Thread2 : Lecture de l'octet suivant	FF	00		
Thread2 : Lecture de l'octet suivant	FF	00	00	
Thread2 : Lecture de l'octet suivant	FF	00	00	01
Thread1 : Lecture de l'octet suivant	FF	00	??	
Thread1 : Lecture de l'octet suivant	FF	00	??	??

Réseaux et C/C++

- Threads et C/C++
  - Communication et synchronisation entre threads

    Même si cette dernière possibilité ne correspond probablement pas à ce qu'il se passe dans la réalité, dans le doute il vaudrait donc mieux utiliser un code similaire au suivant…

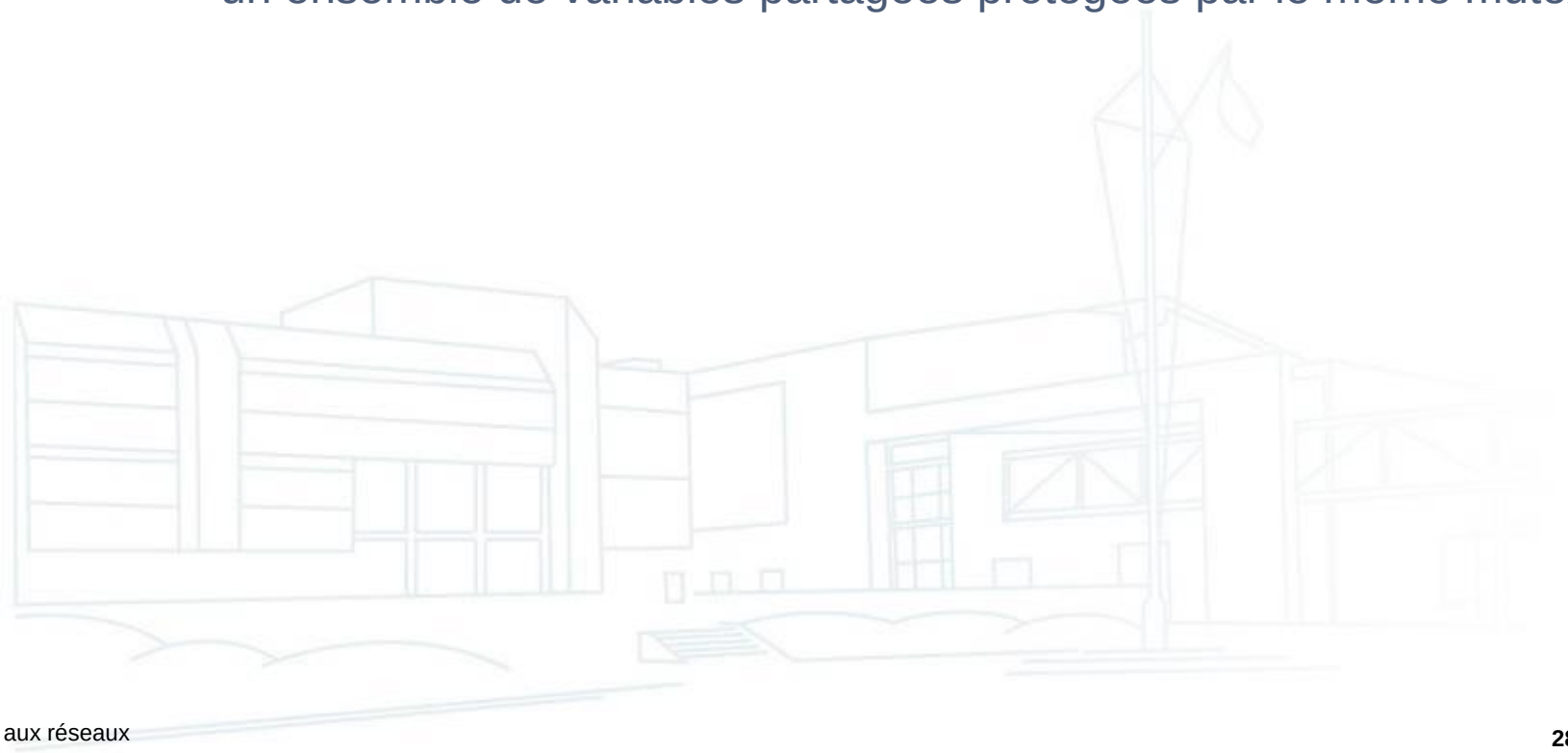
Réseaux et C/C++

```c
#include <stdio.h>
#include <math.h>
#include <pthread.h>

double d = 1;
pthread_mutex_t m;

void* thread0(void* param) {
    int c = 0;
    while (c != ' ') {
        c = getchar(); // Need to validate with ENTER...
        pthread_mutex_lock(&m);
        if (d > 0.00000001) printf("User pressed : %c (not ready)\n", (char)c);
        else printf("User pressed : %c (ready)\n", (char)c);
        pthread_mutex_unlock(&m);
    }
    printf("Will stop as soon as possible...\n");
    return NULL;
}

int main(int argc, char* argv[]) {
    pthread_t t0;
    printf("Beginning.\n");
    pthread_mutex_init(&m, NULL);
    pthread_create(&t0, NULL, thread0, NULL);
    pthread_mutex_lock(&m);
    while (d > 0.00000001) {
        d = fabs(log(exp(d))/1.000001);
        pthread_mutex_unlock(&m);
        sched_yield();
        pthread_mutex_lock(&m);
    }
    pthread_mutex_unlock(&m);
    pthread_join(t0, NULL);
    pthread_mutex_destroy(&m);
    printf("End.\n");
    return 0;
}
```

Réseaux et C/C++

- Threads et C/C++
  - Communication et synchronisation entre threads

    Blackboard : comme il n'est pas forcément optimal d'associer à chaque variable partagée un mutex, on appelle parfois **blackboard** un ensemble de variables partagées protégées par le même mutex



- Threads et C/C++
  - Communication et synchronisation entre threads

    Valeur de retour d'un thread

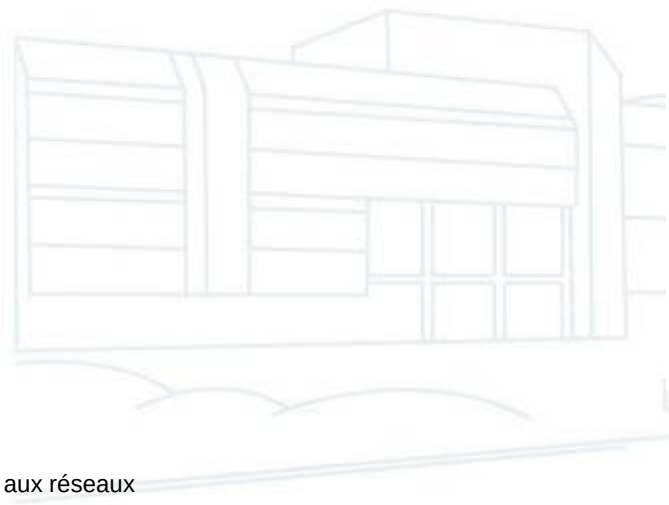
```c
#include <pthread.h>

void* thread0(void* param) {
    // ...
    return NULL;
}

int main(int argc, char* argv[]) {
    pthread_t t0;
    void* ret = NULL;
    pthread_create(&t0, NULL, thread0, NULL);
    // ...
    pthread_join(t0, &ret);
    // Use ret...
    return 0;
}
```

Possibly return other values than NULL here…

Réseaux et C/C++

- Threads et C/C++
  - Communication et synchronisation entre threads
    - Spécification d'un **paramètre** pour le thread

```c
#include <stdlib.h>
#include <pthread.h>

struct EXAMPLE
{
    int i;
    double d;
};
typedef struct EXAMPLE EXAMPLE;

void* thread0(void* param) {
    EXAMPLE* p_e = (EXAMPLE*)param;
    // p_e->i...
    return NULL;
}

int main(int argc, char* argv[]) {
    pthread_t t0;
    EXAMPLE* param = (EXAMPLE*)calloc(1, sizeof(EXAMPLE));
    param->i = 1; param->d = 3.14;
    pthread_create(&t0, NULL, thread0, (void*)param);
    // ...
    pthread_join(t0, NULL);
    free(param);
    return 0;
}
```



Réseaux et C/C++

- Threads et C/C++
  - Voir les exemples de code : http://www.ensta-bretagne.fr/lebars/pthread_samples.zip
  - Pour compiler manuellement :

    gcc -Wall Main.c -lpthread

    (ou -pthread pour les versions récentes de gcc/g++…)
  - Documentation : see e.g. http://man7.org/linux/man-pages/
  - TD : https://www.ensta-bretagne.fr/lebars/TD_threads.pdf

Réseaux et C/C++

- Sockets et C/C++
  - **Socket** : concept existant dans la plupart des langages de programmation et OS pour faire de la **communication inter-processus** entre des ordinateurs **sur un même réseau**
  - Les sockets **TCP/IPv4** (mode « connecté » i.e. retournera une erreur si des données sont perdues) et **UDP/IPv4** (mode « non-connecté » i.e. ne vérifie pas si les données sont arrivées à destination) sont les plus couramment utilisées
  - Elles peuvent être configurées en mode **bloquant** ou non bloquant, avec **timeouts** (pour les send() et recv(), en général par défaut bloquant avec timeout infini), etc.
  - En robotique, les middleware tels que **ROS** se basent souvent sur de la communication par sockets

Réseaux et C/C++

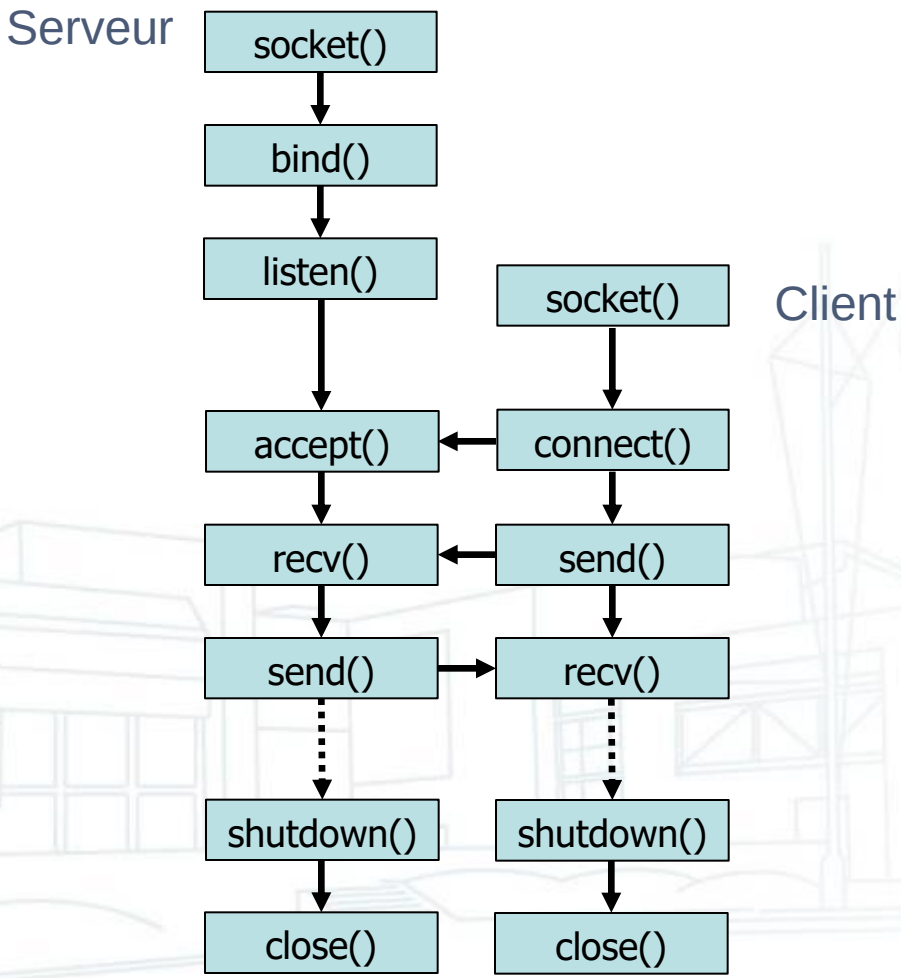
- Sockets TCP et C/C++
  - En **TCP**, 2 types de programmes différents sont à écrire : un **serveur** et un **client**

- Sockets TCP et C/C++
  - Etapes principales pour le serveur :
    - Création de l'objet socket et spécification du protocole (**socket()**)
    - Configuration d'options éventuelles (**setsockopt()**, **ioctlsocket()/ioctl()**, etc.)
    - Association de la socket avec une adresse IP et un port TCP (**bind()**)
    - Active la file d'attente de demandes de connexion (**listen()**)
    - Attente d'une connexion et création d'une socket pour représenter le client accepté (**accept()**)
    - Envoi et réception de données avec la socket cliente (d'autres acceptations de connexion à la socket serveur et communications peuvent être faites en même temps e.g. dans des threads parallèles) (**send()**, **recv()**)
    - Indication au client que les communications sont terminées (**shutdown()**)
    - Destruction de la socket cliente (**closesocket()/close()**)
    - Destruction de la socket serveur (**closesocket()/close()**)

Réseaux et C/C++

- Sockets TCP et C/C++
  - Etapes principales pour le client :
    - Création de l'objet socket et spécification du protocole (**socket()**)
    - Configuration d'options éventuelles (**setsockopt()**, **ioctlsocket()/ioctl()**, etc.)
    - Demande de connexion à un serveur avec une adresse IP et un port TCP (**connect()**)
    - Envoi et réception de données (**send()**, **recv()**)
    - Indication au serveur que les communications sont terminées (**shutdown()**)
    - Destruction de la socket (**closesocket()/close()**)

- Sockets TCP et C/C++



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef _WIN32
#include <winsock2.h>
#include <ws2tcpip.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <unistd.h>

typedef int SOCKET;
#define SD_BOTH SHUT_RDWR
#define closesocket(sock) close((sock))
#endif // _WIN32

int main(int argc, char* argv[])
{
    SOCKET s, s_cli;
    struct sockaddr_in sa;
    char buf[256];
#ifdef _WIN32
    WSADATA wsaData;

    WSAStartup(MAKEWORD(2,2), &wsaData);
#endif // _WIN32
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons((unsigned short)atoi("32768"));
    sa.sin_addr.s_addr = inet_addr("0.0.0.0");
    bind(s, (struct sockaddr*)&sa, sizeof(sa));
    listen(s, 1);
    s_cli = accept(s, NULL, NULL);
    recv(s_cli, buf, sizeof(buf), 0);
    send(s_cli, buf, sizeof(buf), 0);
    shutdown(s_cli, SD_BOTH);
    closesocket(s_cli);
    shutdown(s, SD_BOTH);
    closesocket(s);
#ifdef _WIN32
    WSACleanup();
#endif // _WIN32
    getchar();
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef _WIN32
#include <winsock2.h>
#include <ws2tcpip.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <unistd.h>

typedef int SOCKET;
#define SD_BOTH SHUT_RDWR
#define closesocket(sock) close((sock))
#endif // _WIN32

int main(int argc, char* argv[])
{
    SOCKET s;
    struct sockaddr_in sa;
    char buf[256];
#ifdef _WIN32
    WSADATA wsaData;

    WSAStartup(MAKEWORD(2,2), &wsaData);
#endif // _WIN32
    s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    memset(&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons((unsigned short)atoi("32768"));
    sa.sin_addr.s_addr = inet_addr("127.0.0.1");
    connect(s, (struct sockaddr*)&sa, sizeof(sa));
    send(s, "Hello", strlen("Hello")+1, 0);
    recv(s, buf, sizeof(buf), 0);
    printf("%.255s", buf);
    shutdown(s, SD_BOTH);
    closesocket(s);
#ifdef _WIN32
    WSACleanup();
#endif // _WIN32
    getchar();
    return 0;
}
```

- Sockets TCP et C/C++
  - Dans l'exemple, l'adresse 0.0.0.0 signifie que le serveur est accessible via toutes les interfaces réseaux actuellement connectées
  - send() et recv(), valeurs de retour des fonctions :

    Il est particulièrement important de **vérifier les valeurs de retour** des différentes fonctions appelées pour savoir à quel point toutes les données ont bien été envoyées ou reçues, **les données** de plusieurs dizaines d'octets **ne peuvent pas toujours être transmises en une seule fois**…

    Ainsi, la plupart des protocoles de communication classiques définissent un octet de fin particulier permettant de signaler la fin des données utiles échangées si le nombre d'octets attendu est inconnu, pour pouvoir vérifier si elles sont complètes et sinon continuer à recevoir jusqu'à ce qu'elles le soient

- Sockets TCP et C/C++
  - select()

    La fonction **select()** permet de **surveiller l'état de plusieurs sockets** en même temps (pendant un timeout à spécifier) et indique lorsqu'elle retourne celles qui sont prêtes à recevoir des données ou si elles essayent d'envoyer des données

    Note : une socket serveur en train de recevoir une demande de connexion d'un client est considérée comme prête en lecture, ceci peut être utile pour contourner le fait que la fonction **accept()** n'a pas de timeout configurable

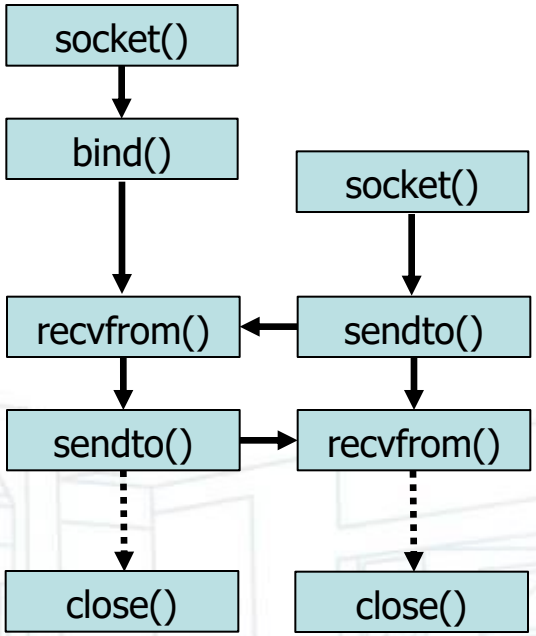
Réseaux et C/C++

```c
#ifdef _WIN32
#else
#include <sys/select.h>
#endif // _WIN32

int test_select(SOCKET s, int timeout) {
    fd_set sock_set;
    int iResult = -1;
    struct timeval tv;

    tv.tv_sec = (long)(timeout/1000);
    tv.tv_usec = (long)((timeout%1000)*1000);

    // Initialize a fd_set and add a socket to it.
    FD_ZERO(&sock_set);
    FD_SET(s, &sock_set);

    // Wait for the readability of the socket in the fd_set, with a timeout.
    if (timeout != 0) iResult = select((int)s+1, &sock_set, NULL, NULL, &tv);
    else iResult = select((int)s+1, &sock_set, NULL, NULL, NULL);

    switch (iResult) {
    case -1: printf("select() failed\n"); return EXIT_FAILURE;
    case 0: printf("select() timeout"); return EXIT_FAILURE;
    default:
        if (FD_ISSET(s, &sock_set) != 0) {
            printf("s is readable\n");
            // ...
            FD_CLR(s, &sock_set); // Remove the socket from the set.
            return EXIT_SUCCESS;
        }
    }
    return EXIT_FAILURE;
}
```

- Sockets UDP et C/C++
  - En **UDP**, la distinction entre un programme serveur et un programme client est un peu moins claire et dépend beaucoup de ce qu'on souhaite faire (on peut éventuellement considérer que le client est celui qui initie le transfert de données et le serveur serait celui qui attend)…
  - Les fonctions principales un peu spécifiques à l'UDP sont **sendto()** et **recvfrom()**, qui permettent de **spécifier ou connaitre directement l'adresse IP et le port UDP** contrairement à **send()** et **recv()**
    - Il est cependant possible de faire un **connect()** avant l'utilisation de **send()** et **recv()** pour qu'elles utilisent par défaut l'adresse IP et le port UDP spécifiés lors du **connect()**…
    - En UDP, ces fonctions vont en général échouer si les données font plus de **65507 octets**, il faut donc découper en plus petits paquets

Réseaux et C/C++

- Sockets UDP et C/C++



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef _WIN32
#include <winsock2.h>
#include <ws2tcpip.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <unistd.h>

typedef int SOCKET;
#define SD_BOTH SHUT_RDWR
#define closesocket(sock) close((sock))
#endif // _WIN32

int main(int argc, char* argv[])
{
    SOCKET s;
    struct sockaddr_in sai;
    struct sockaddr_storage sa;
    socklen_t salen = 0;
    char buf[256];
#ifdef _WIN32
    WSADATA wsaData;

    WSAStartup(MAKEWORD(2,2), &wsaData);
#endif // _WIN32
    s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    memset(&sai, 0, sizeof(sai));
    sai.sin_family = AF_INET;
    sai.sin_port = htons((unsigned short)atoi("32768"));
    sai.sin_addr.s_addr = inet_addr("0.0.0.0");
    bind(s, (struct sockaddr*)&sai, sizeof(sai));
    memset(&sa, 0, sizeof(sa));
    salen = sizeof(sa);
    recvfrom(s, buf, sizeof(buf), 0,
             (struct sockaddr*)&sa, (socklen_t*)&salen);
    sendto(s, buf, sizeof(buf), 0,
           (struct sockaddr*)&sa, sizeof(sa));
    closesocket(s);
#ifdef _WIN32
    WSACleanup();
#endif // _WIN32
    getchar();
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef _WIN32
#include <winsock2.h>
#include <ws2tcpip.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <unistd.h>

typedef int SOCKET;
#define SD_BOTH SHUT_RDWR
#define closesocket(sock) close((sock))
#endif // _WIN32

int main(int argc, char* argv[])
{
    SOCKET s;
    struct sockaddr_in sai;
    struct sockaddr_storage sa;
    socklen_t salen = 0;
    char buf[256];
#ifdef _WIN32
    WSADATA wsaData;

    WSAStartup(MAKEWORD(2,2), &wsaData);
#endif // _WIN32
    s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    memset(&sai, 0, sizeof(sai));
    sai.sin_family = AF_INET;
    sai.sin_port = htons((unsigned short)atoi("32768"));
    sai.sin_addr.s_addr = inet_addr("127.0.0.1");
    sendto(s, "Hello", strlen("Hello")+1, 0,
           (struct sockaddr*)&sai, sizeof(sai));
    memset(&sa, 0, sizeof(sa));
    salen = sizeof(sa);
    recvfrom(s, buf, sizeof(buf), 0,
             (struct sockaddr*)&sa, (socklen_t*)&salen);
    printf("%.255s", buf);
    closesocket(s);
#ifdef _WIN32
    WSACleanup();
#endif // _WIN32
    getchar();
    return 0;
}
```

Réseaux et C/C++

- Sockets et C/C++
  - Voir les exemples de code : http://www.ensta-bretagne.fr/lebars/sockets_samples.zip
  - Pour compiler manuellement sous Windows avec MinGW, l'option **-lWS2_32** est nécessaire (pas sous Linux) :

    gcc -Wall Main.c -lWS2_32
  - Pour trouver rapidement la description officielle de la plupart des fonctions, types, constantes, etc. sur Google, rechercher :

    Linux : man function_name

    Windows : MSDN function_name
  - TD : https://www.ensta-bretagne.fr/lebars/TD_sockets.pdf