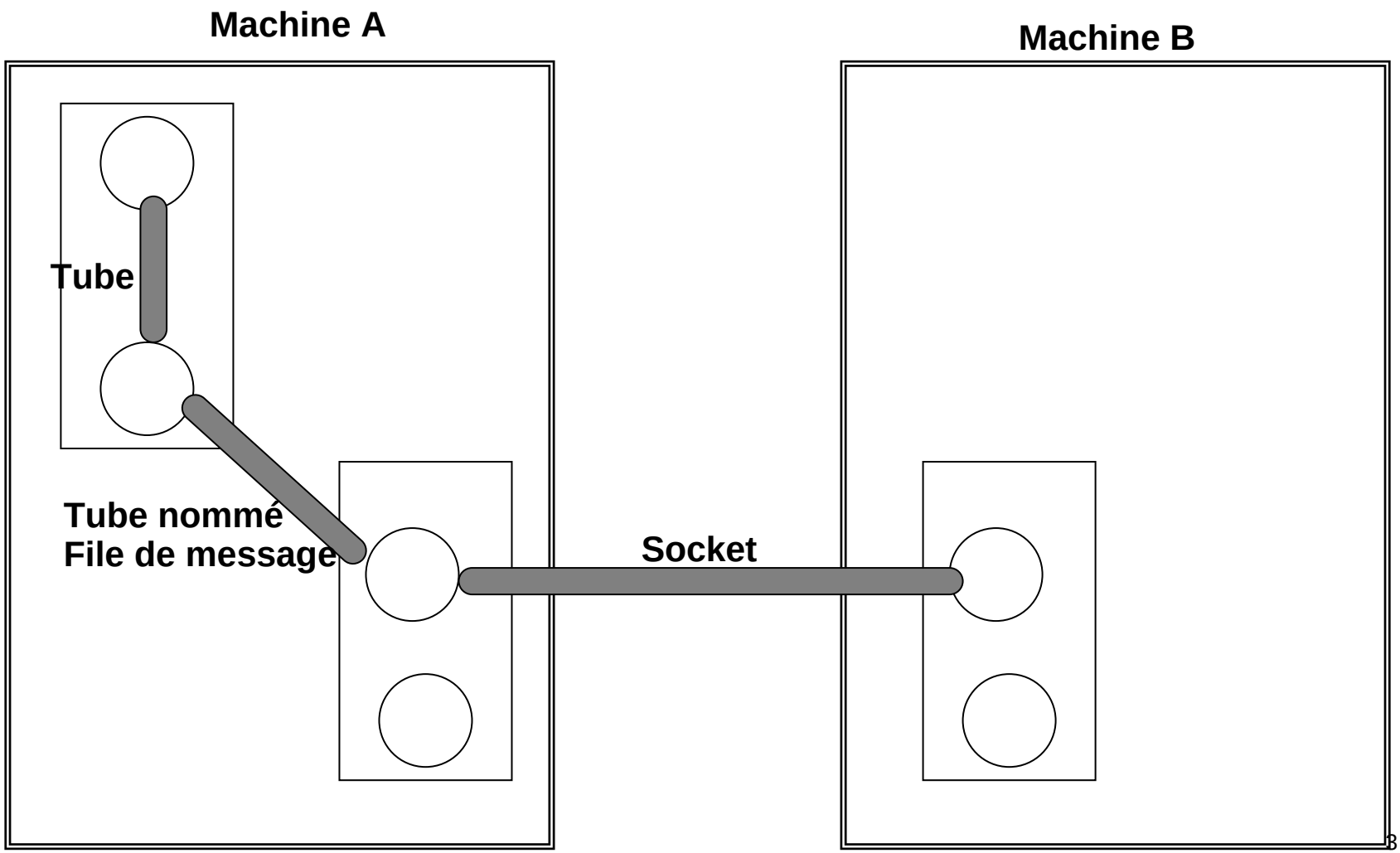
Python et Unix

Yann Thierry-Mieg
Laboratoire d'informatique de Paris 6
EPSI

PARTIE II : Communications Inter-Processus

- Introduction:
- intra-tâche : pipe
- inter-tâche : mkfifo
- inter-machine : socket
  - socket UNIX
  - socket TCP
  - socket UDP
  - multicast

Schéma général : communications Unix



Les Communications Entre Processus UNIX

- Intra-tache ou entre threads :
  - partage de variables globales,
  - tas
- Inter processus sur une même machine :
  - signaux,
  - pipes (processus d'une même famille)
  - IPC = Inter Process Communication (Unix system V)
    - Files de messages
    - segment de memoire partagée,
    - sémaphores

Les Communications Entre Processus UNIX (2)

- Inter taches distantes :
  - Sockets (TCP, UDP , IP)
  - Appels de procédures distantes (RPC)
  - Architectures bus logiciel (Corba,Java RMI...)
- Outils de haut niveau
  - Systèmes de fichiers répartis (NFS, RFS, AFS)
  - Bases de données réparties (NIS..)

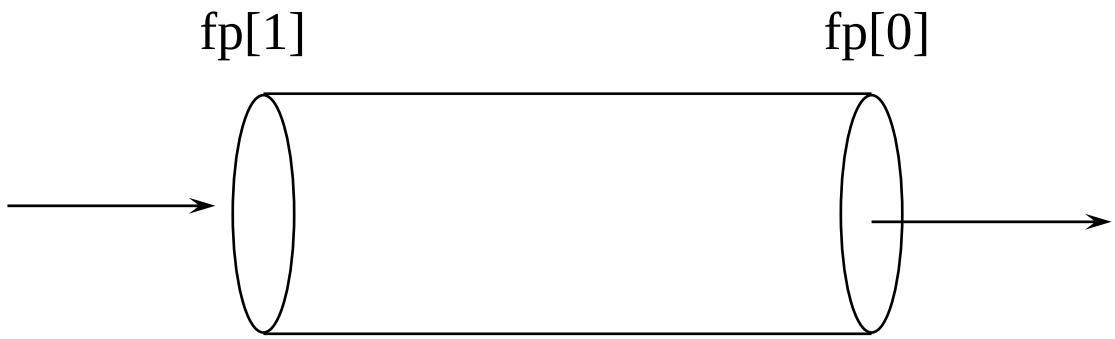
Communications sur une machine

- Communication entre plusieurs processus sur une même machine

- Utilisation des entrées-sorties
  - Extension de la notion de fichiers,
  - Tube de communication
- Partage de données communes
  - Messages,
  - Mémoire,
  - sémaphores

Pipe : communication par tubes

- Canal unidirectionel (une entrée, une sortie)
  - Deux entrées dans la table des descripteurs de fichier,
  - Communications locales de type producteur-consommateur,
- Canal bidirectionnel : deux tubes



pipe : en pratique

Limité à l'arbre de descendance d'un processus

from os import *	importe les API système
from sys import exit	importe exit
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
(r,w) = pipe()	rend deux entiers, référence à des fd
pid = fork()	fork et rend 0 au père, son pid au fils
if (pid == 0) :	cas père
close(w)	on ferme l'extrémité écriture
print read(r,BUFSIZE)	lire qq chose
close(r)	fermer proprement avant de quitter
else :	cas fils
close(r)	fermer l'extrémité lecture
write(w,"Hello world")	ecrire qq chose
close(w)	fermer proprement avant de quitter
exit(0)	terminaison

Tubes nommés : mkfifo

- Utilisable entre processus quelconques
- S'appuie sur un pseudo-fichier (inode en mode S_IFIFO)
- Ouverture du fichier en lecture ou écriture

- Implémente un modèle producteur consommateur simple

mkfifo : En pratique: mkfifoL.py

#! /usr/bin/python	rend executable
from os import *	importe les API système
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
mkfifo("/tmp/montube")	cree un tube nommé
fd = open("/tmp/montube",O_RDONLY)	ouvrir en read only
lu = read(fd,BUFSIZE)	lire qq chose
print "Lu : ",luclose(fd)	imprimer message
unlink("/tmp/montube")	effacer le tube nommé

mkfifo : En pratique: mkfifoE.py

#! /usr/bin/python from os import *	rend executable importe les API système
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
fd = open("/tmp/montube",O_WRONLY) write(fd,"hello")	ouvrir le tube en write only ecrire qq chose
close(fd)	ferme proprement le file descriptor

```
>mkfifoL.py &
[1] 3564
>ls -l /tmp/montube
prw-r--r--    1 thierry  rech          0 Jan 20 13:48 /tmp/montube
>mkfifoE.py
Lu :  hello
[1]  + Done                    mkfifoL.py
```

Communications inter machine

- Connecté :
  - similaire à un pipe (i.e. TCP)
  - Acquittements (augmente la charge réseau)
  - Ni pertes, ni désequencement
- Non connecté :
  - Envois non bloquant (asynchrone) (i.e. UDP)
  - Possibilités de désequencement
  - pertes et désequencement possibles
  - multicast possible efficacement

Des exemples de service :

- Session de connexion: telnet, ssh, ftp, http
  - Connexion fiable (TCP)
- Service de streaming : video stream, radio shoutcast ...
  - connexion non fiable
- Services de diffusion : prospectus electronique, application dediée...
  - non connecté, non fiable (datagramme)

Modèle de référence ISO

- ISO 1 et 2 : Physique et Liaison :
  - ARPANET, SATNET, LAN ethernet
- ISO 3 : Réseau
  - Internet Protocol (IP)
- ISO 4 : Transport
  - TCP (Transmission Control Protocol)
  - UDP (User Datagram)
  - ICMP (Internet Control Message)
  - IGMP (Internet Group Management)

Modèle de référence ISO (2)

- ISO 5 : Session :
  - RPC
- ISO 6 : Présentation
  - XDR (eXternal Data Representation)
  - ASN.1 (Abstract Syntax Notation number One)
- ISO 7 : Application
  - telnet, ssh
  - ftp, http, smtp
  - dns,nfs

Sockets

- Extension de la notion de tube nommé
  - associée à un file descriptor
- Socket = Point extreme de communication
- Caractéristiques :
  - Bidirectionelle
  - Associé à un nom unique
  - Associé à un *domaine*
  - Possède un type

Domaine de socket

- Les plus courants :
  - AF_INET : Socket IP, domaine internet IPv4
  - AF_UNIX : Socket de communication locale
  - AF_INET6 : IPv6
- D'autres domaines :
  - AF_X25 (protocole X25),
  - AF_NETROM, AF_ROSE, AF_AX25 (paquets-radio dérivé d'X25),
  - AF_IPX (Novell),
  - AF_APPLETALK (réseau appletalk),
  - AF_ATMPVC (réseau ATM),
  - AF_DECNET (réseau DEC),
  - AF_NETBEUI (réseau Microsoft),

Types de service

- Les plus courants :
  - SOCK_STREAM :
    - i.e. TCP (IPPROTO_TCP)
    - Mode connecté
    - Contrôle de flux (congestion)
    - bidirectionnel
  - SOCK_DGRAM :
    - i.e. UDP (IPPROTO_UDP) ou IGMP (IPPROTO_IGMP)
    - Non connecté
    - Pas de contrôle de flux
    - Transmission par paquet

Types de Service (2)

- D'autres services :
  - SOCK_RAW :
    - i.e. ICMP (IPPROTO_ICMP), protocoles ad hoc (IPPROTO_RAW)
    - Accès direct à la couche IP
    - Réservé au super-utilisateur
  - SOCK_SEQPACKET, SOCK_RDM ...
    - i.e. UDP, ICMP
    - Garantissent différentes propriétés: intégrité, controle de flux ...

Choix d'un protocole : UDP ou TCP ?

- TCP : STREAM
  - Ni perte, ni désequencement
  - Mode flux (taille non bornée)
  - Coûteux (connexion, contrôle d'erreur...)
- UDP : DGRAM
  - Pertes et désequencements possibles (sur un LAN, le problème se pose peu)
  - Taille limitée par paquet
  - Performant (surcouche par rap. à IP)
  - Multicast possible

Modèle client serveur

- Le serveur crée une socket, la nomme (bind) et attend des connexions
- Le client crée sa propre socket et se connecte à celle du serveur (doit connaître le nom de la socket serveur)
- En UDP ou TCP, nom de socket = numéro de port
- En AF_UNIX, nom de socket = nom de fichier
- Attention : Serveur = Serviteur, il offre des services au client, le client est celui qui *initie la connexion*

Socket AF_UNIX

- Utilisée pour communiquer dans le domaine local uniquement
- Nom = nom de fichier
- Extension bidirectionnelle de tube nommé
- Portage aisé vers d'autres protocoles (TCP) réseau
- Largement utilisé par les services internes du noyau
- Uniquement sur plateforme UNIX

Socket AF_UNIX : En Pratique SockUnixS.py

#! /usr/bin/python	rend exécutable
from os import *	importe les API système (read,write)
from socket import *	importe API socket
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
s = socket (AF_UNIX,SOCK_STREAM)	Cree une socket AF_UNIX
s.bind ("/tmp/masocket")	lie la socket au fichier (nommage)
s.listen(0)	attente
s2,unused = s.accept()	accepter une connexion unused contient l'adresse client
print s2.recv(BUFSIZE)	imprimer message
unlink("/tmp/masocket")	effacer le fichier/socket

Socket AF_UNIX : En Pratique SockUnixC.py

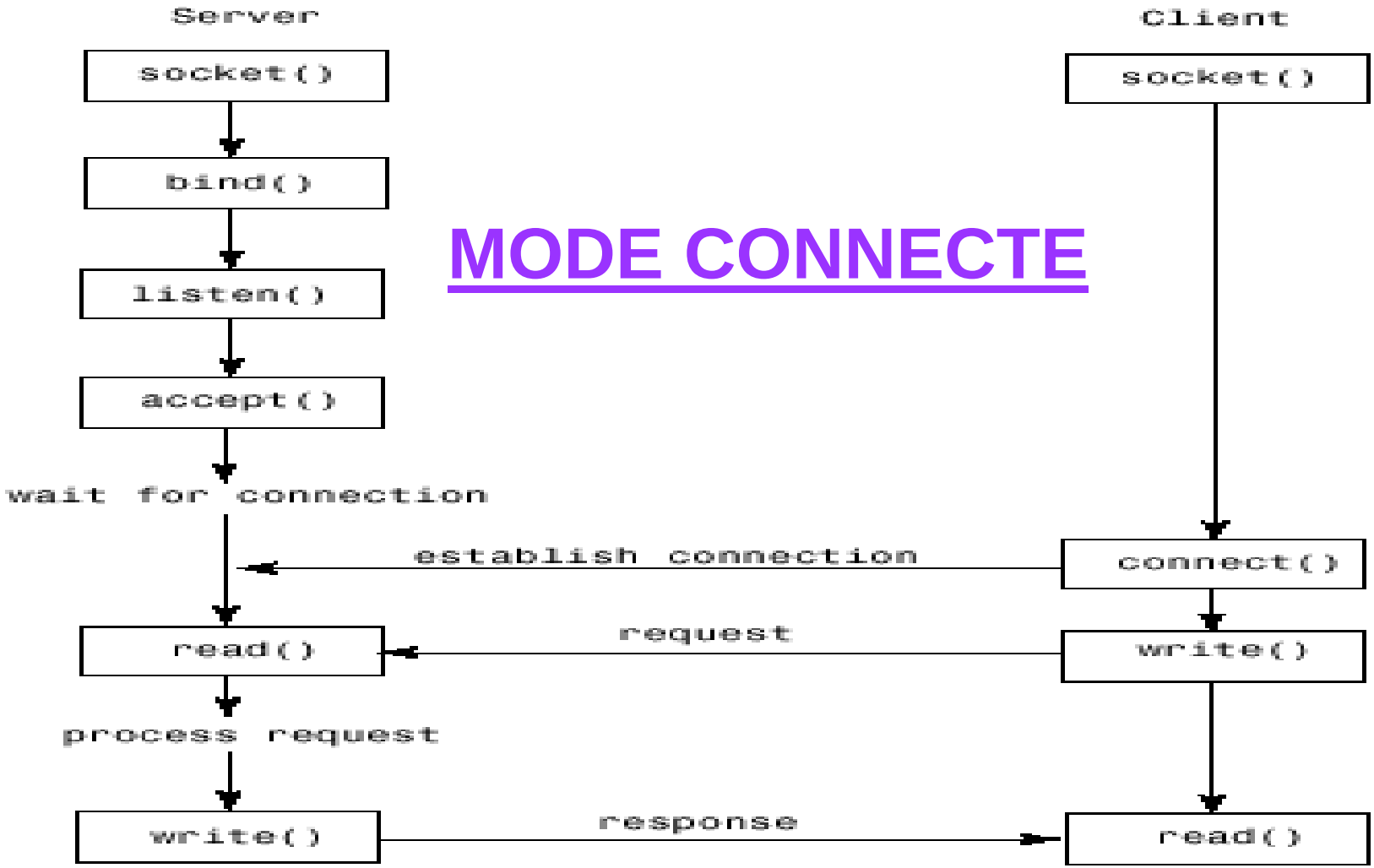
#! /usr/bin/python	rend exécutable
from socket import *	importe les API socket
s = socket (AF_UNIX,SOCK_STREAM) s.connect ("/tmp/masocket") s.send("Hello from client")	crée une socket UNIX connecte a la socket server envoie un message

```
>SockUnixS.py &
[1] 5449
>ls -l /tmp/masocket
Srwxr-xr-x    1 thierry  rech          0 Jan 21 13:01 /tmp/masocket
>SockUnixC.py
Hello from client
[1]  + Done                    SockUnixS.py
```

Socket AF_INET : Internet

- La plus utilisée
- Compatible toutes plateformes
- Deux protocoles en général : UDP et TCP
- Nommage de socket = (Adr. IP, Num Port)
- Bidirectionnelle
- Sert de base aux services les plus courants (http, ftp, telnet, ssh, ...)

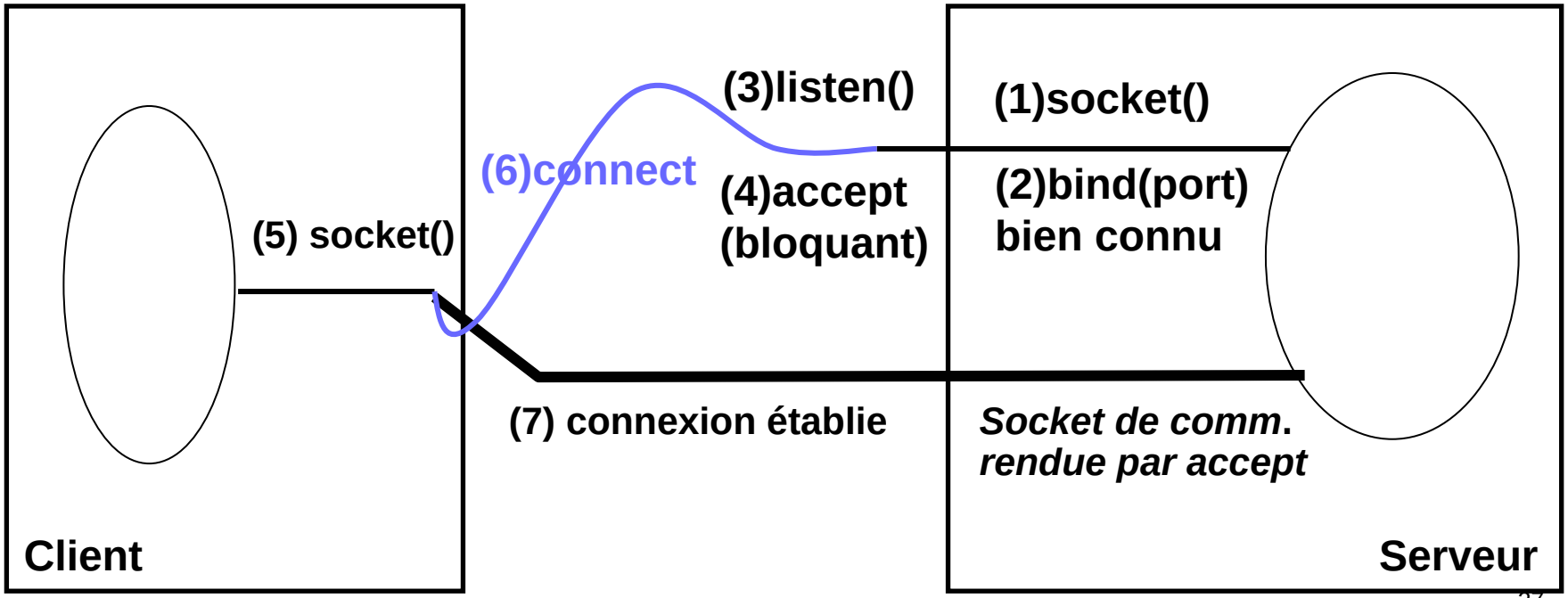
Schéma de communication TCP



MODE CONNECTE

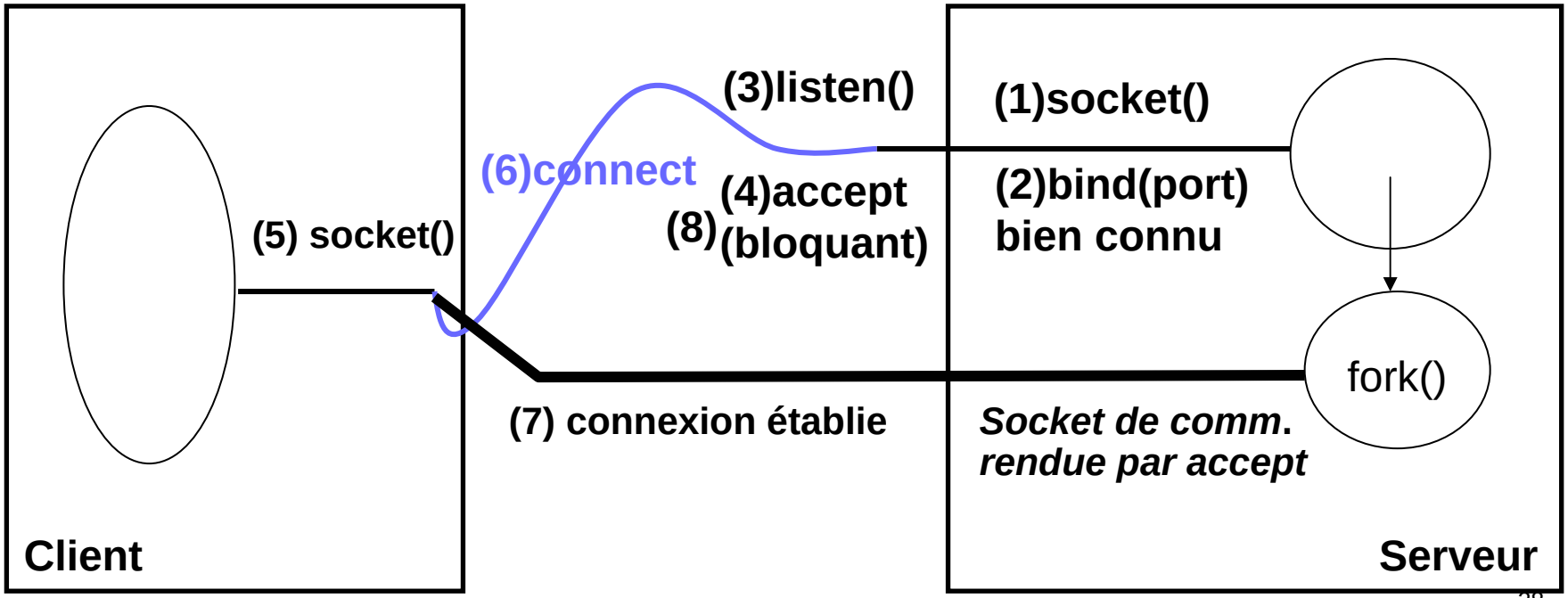
Etablissement de connexion TCP (2)

- Côté serveur : 1 socket pour demande de connexion, 1 pour communication
- Côté client : 1 socket



Etablissement de connexion TCP (3)

- Problème : multi-client
- Solution : listen(taille de file d'attente)
- Multi thread



Socket TCP : En Pratique SockTCPS.py

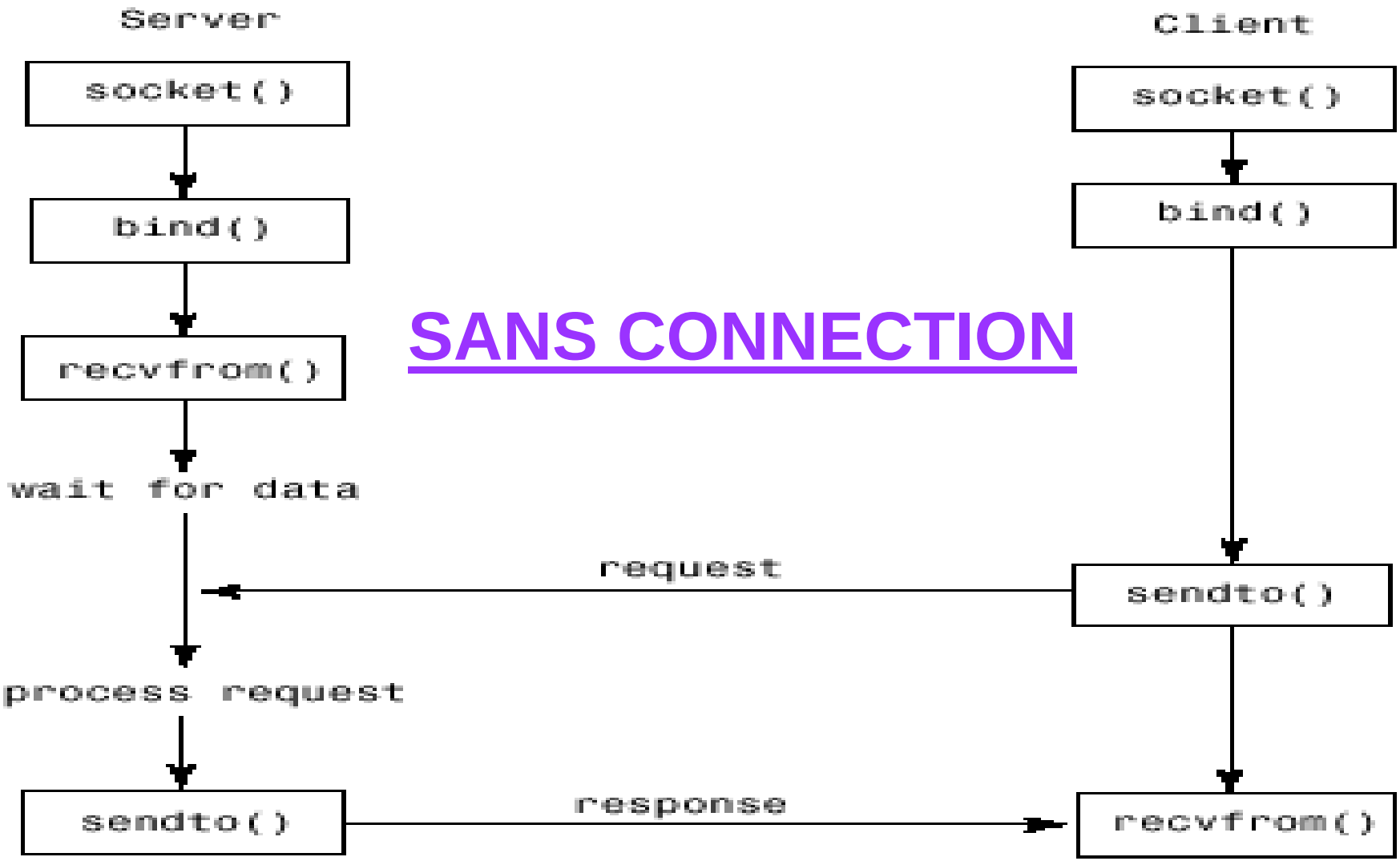
#! /usr/bin/python	rend exécutable
from socket import *	importe API socket
PORT = 1664	Un port (0-1023 = réservé système)
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
s = socket(AF_INET,SOCK_STREAM)	Cree une socket AF_INET,TCP
s.bind(("localhost",PORT))	lie la socket adresse loopback,PORT
s.listen(1)	attente (1 connexion en attente possible)
s2,adr = s.accept()	accepter une connexion (appel bloquant) s2 =nouvelle socket connectée au client
print "Incoming from" + `adr`	exhibe l'adresse client
print s2.recv(BUFSIZE)	imprimer message
s2.send("ACK")	renvoyer une réponse
s2.close()	fermer la socket de communication
s.close()	fermer la socket d'attente de connexions

Socket TCP : En Pratique SockTCPC.py

from socket import *	importe les API socket
PORT = 1664	Le port serveur (bien connu)
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
s = socket(AF_INET,SOCK_STREAM)	crée une socket TCP
s.connect(("127.0.0.1",PORT))	connecte a la socket server
s.send("Hello !!")	envoie un message
s.close()	ferme la socket de communication

```
>SockTCPS.py &
[1] 5489
>netstat -a | grep 1664
tcp        0      0 localhost.localdom:1664 *:*                    LISTEN
>SockTCPC.py
Incoming from('127.0.0.1', 32982)
Hello !!
[1]  + Done                    SockTCPS.py
```

Schéma de Communication UDP



SANS CONNECTION

Socket UDP : En Pratique SockUDPS.py

#! /usr/bin/python	rend exécutable
from socket import *	importe API socket
PORT = 1664	Un port (0-1023 = réservé système)
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
s = socket(AF_INET,SOCK_DGRAM)	Cree une socket AF_INET,UDP
s.bind(("localhost",PORT))	lie la socket adresse loopback,PORT
msg,adr = s.recvfrom(BUFSIZE)	attente (appel bloquant), adr = client
print msg	imprimer message
s.sendto("ACK OK",adr)	renvoyer une réponse
s.close()	fermer la socket de communication

Socket UDP : En Pratique SockUDPC.py

from socket import *	importe les API socket
PORT = 1665	Un port Libre
BUFSIZE = 1024	on peut choisir d'autres valeurs, ici 1ko
s = socket(AF_INET,SOCK_DGRAM)	Cree une socket UDP
s.bind(("localhost",PORT))	la nomme pour le système
s.sendto("Hello",("localhost",1664))	connecte a la socket server (Port connu)
msg,adr = s.recvfrom(BUFSIZE)	envoie un message
print msg	reçoit une réponse
s.close()	ferme la socket de communication

```
>SockUDPS.py &
[1] 5518
>netstat -a | grep 1664
udp        0      0 localhost.localdom:1664 *:*
>SockUDPC.py
Hello
ACK OK
[1]  + Done                    SockUDPS.py
```

Socket UDP : Multicast

- Multicast : implementation efficace coté tables de routage pour la multi-diffusion
- Une classe d'adresses dédiées :
  - Classe D : 224.0.0.1 à 239.255.255.255
- Réception :
  - Abonnement à un groupe = adresse de classe D
  - recvfrom
  - Désabonnement
- Emission :
  - connect sur l'adresse du groupe
  - send

Socket UDP Multicast : En Pratique SockMCReceive.py

from socket import *	importe API socket
PORT = 1664	Un port
BUFSIZE = 1024	ici 1ko
myAdr = gethostbyname(gethostname())	stocke l'adresse IP de l'hôte
s=socket(AF_INET,SOCK_DGRAM)	crée une socket UDP
s.setsockopt(SOL_SOCKET,\	permet de faire tourner plusieurs
SO_REUSEADDR,1)	processus sur le même port
s.bind(('',PORT))	Nomme la socket
s.setsockopt(SOL_IP,IP_ADD_MEMBERSHIP,\	Rejoint le groupe
inet_aton('235.0.50.1')+inet_aton(myAdr))	adresse de groupe + mon adr
msg,adr = s.recvfrom(BUFSIZE)	bloquant, lit un message
print "Received :" ,msg	imprimer message
s.setsockopt(SOL_IP,IP_DROP_MEMBERSHIP,	quitte le groupe
inet_aton("235.0.50.1")+inet_aton(myAdr))	
s.close()	ferme la socket

Socket UDP Multicast: En Pratique SockMCSend.py

from socket import *	importe les API socket
PORT = 1664	Un port Libre
BUFSIZE = 1024	ici 1ko
s = socket(AF_INET,SOCK_DGRAM)	Cree une socket UDP
s.connect(("235.0.50.1",PORT))	se connecte au groupe
s.setsockopt(SOL_IP,\	mets un TTL suffisant pour un LAN
IP_MULTICAST_TTL,8)	le TTL diminue a chaque réémission
s.send ("hello ALL!!",BUFSIZE)	envoie un message
s.close()	ferme la socket de communication

```
>SockMCReceive.py &
[1] 5640
>SockMCReceive.py &
[2] 5641
>SockMCSend.py
Received : hello ALL!!
Received : hello ALL!!
[2]  + Done                    SockMCReceive.py          ....
```

Bibliographie

- Système : Noyau, Pierre Sens, Supports de maîtrise d'informatique Paris 6
- Python 2.1 Bible,Dave Brueck & Stephen Tanner, IDG
- *UNIX Network Programming, Volume 1: Networking APIs - Sockets and XTI* , W. Richard Stevens , Prentice Hall
- *TCP/IP Illustré vol.2 : La mise en Oeuvre,*G. Wright & W. Stevens, Vuibert
- + biblio première partie