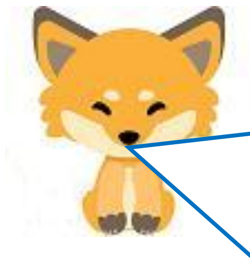
Episode 5 (Fundamentals of SNS Communications)



Tanuki, I hope you understand the point of the computer.
Next time you'll be going through the trees (detail items).
I'm sure that raccoon dogs also use their smartphones to email, chat, and even remotely conference with each other. The basis for this is called inter-process communication. A process is a program running in a computer. It's like the apps running on your smartphone.
It's something you use every day, so you're probably somewhat interested in it.

There is. I've always wondered, like, how many raccoons can attend a remote meeting?
I wonder if an infinite number of raccoons can participate in a remote meeting, etc.
Fox, you tell me.

So, let's begin loosely!
The networks we use mainly use a protocol (communication protocol) called TCP/IP. The TCP/IP protocol used differs for each stage of communication, and the DARPA (four-layer) model is based on a four-layer hierarchy. This is illustrated in the figure below.

[TCP/IP hierarchy considered in the DARPA (4-tier) model]

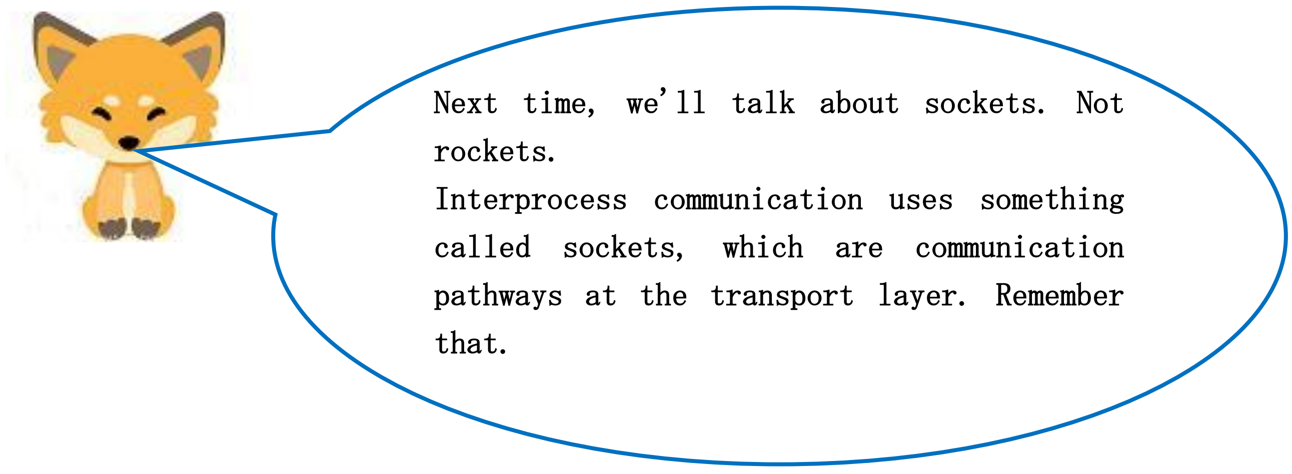
Application layer Telenet、Http、Ftp SMTP、NFS、 X Window	
Transport layer TCP、UDP、ICMP	Sockets: Interfaces to the transport layer TCP port, UDP port
Internet layer IP v 4、IP v 6	
data link layer (Individual network layer) WAN、LAN、 wireless	

・ＴＣＰ/ＩＰ

ＩＰ	: Internet Protocol
ＴＣＰ	: Transmission Control Protocol (meaning "connection-type")

・ＵＤＰ/ＩＰ

ＵＤＰ	: User Datagram Protocol (datagram type, meaning connectionless (datagram type, meaning connectionless))

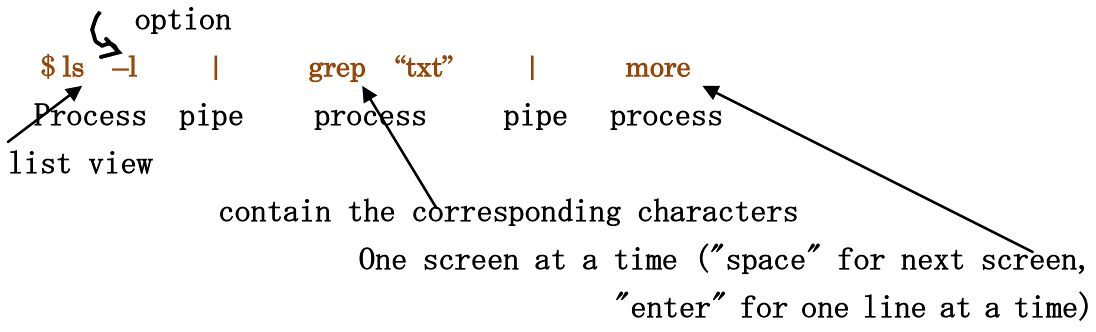

Next time, we'll talk about sockets. Not rockets.
Interprocess communication uses something called sockets, which are communication pathways at the transport layer. Remember that.

Tanuki, let's start with some practice!
I want you to boot a Linux operating system called CentOS, or if you don't have CentOS, Ubuntu or Fedora will do.
An OS is a program (software) that is the core of a computer. Microsoft Windows is the most famous OS, but Linux is also an OS. I like CentOS. Boot up that OS.

Next, can you start up a terminal or something like that?
Each command used on the command line is also a program. It is a small program, though.
So, if you execute a command, that command is a process. If multiple commands are executed and data is sent and received between commands, it is inter-process communication. Here is a simple example.

[Example of simple interprocess communication.]

・Use commands in the shell (limited commands available)
・The pipe is the communication channel for interprocess communication.

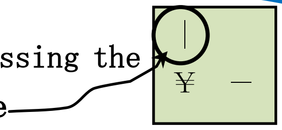


```
$ ls –l | grep "txt" | more
```

option
Process
list view
pipe
process
contain the corresponding characters
pipe
process
One screen at a time ("space" for next screen, "enter" for one line at a time)

How do you enter a pipe?

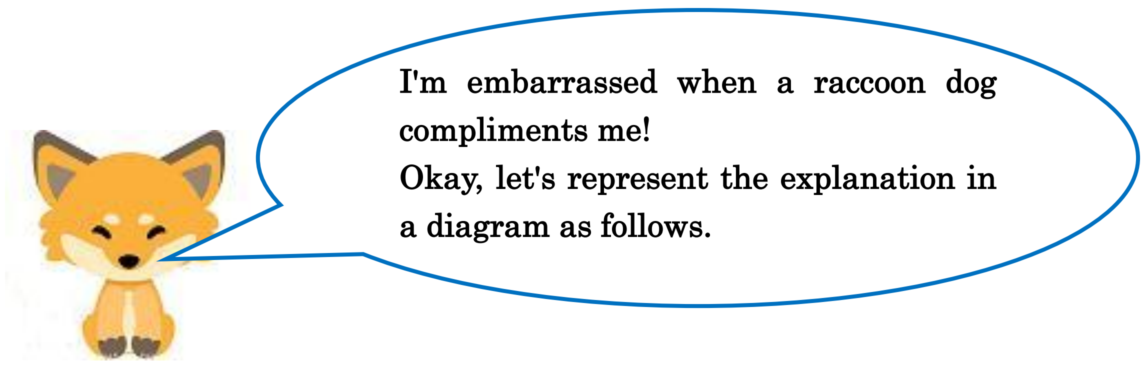
You can type by pressing the



key on the upper right side of the keyboard while holding down the [Shift] key!

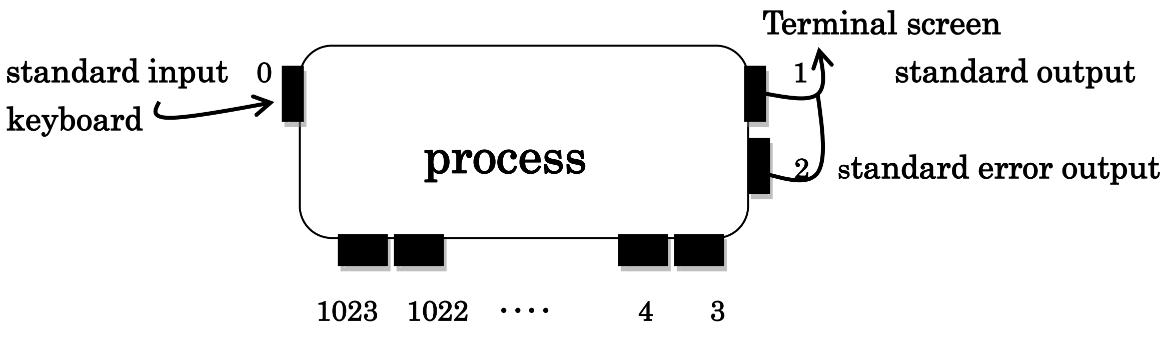
Next, we'll talk about file descriptors that lead to sockets.
A process (a running program) has a doorway to exchange data with an external device. The numbers from 0 to 1023 are assigned to those doorways in sequence. In other words, the integers assigned to these 1024 entry/exit points are file descriptors. Note, however, that not only hardware such as external devices, but also processes (software) other than yourself (process) are considered external.
Of the file descriptors, 0 is fixed to standard input such as keyboard, 1 to standard output such as monitor or printer, and 2 to standard error output. All other numbers are assigned in connection order. Therefore, one process can theoretically communicate with approximately 500 processes simultaneously.

Okay, so in theory, you could have a remote meeting with 500 raccoons at the same time. But I'll use voice at the same time, so maybe 100 to 200 people. That makes some sense. Kitsune, you are awesome!

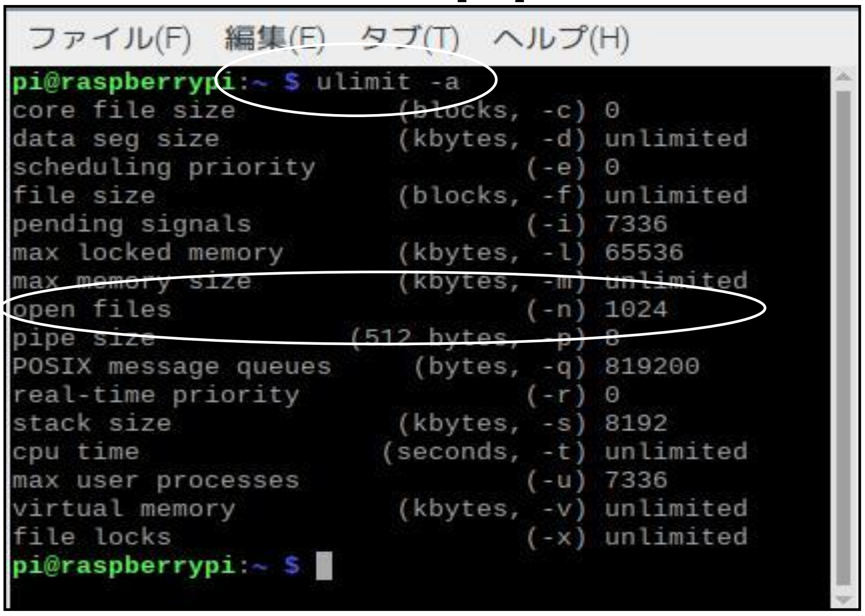

I'm embarrassed when a raccoon dog compliments me!
Okay, let's represent the explanation in a diagram as follows.

［file descriptor］

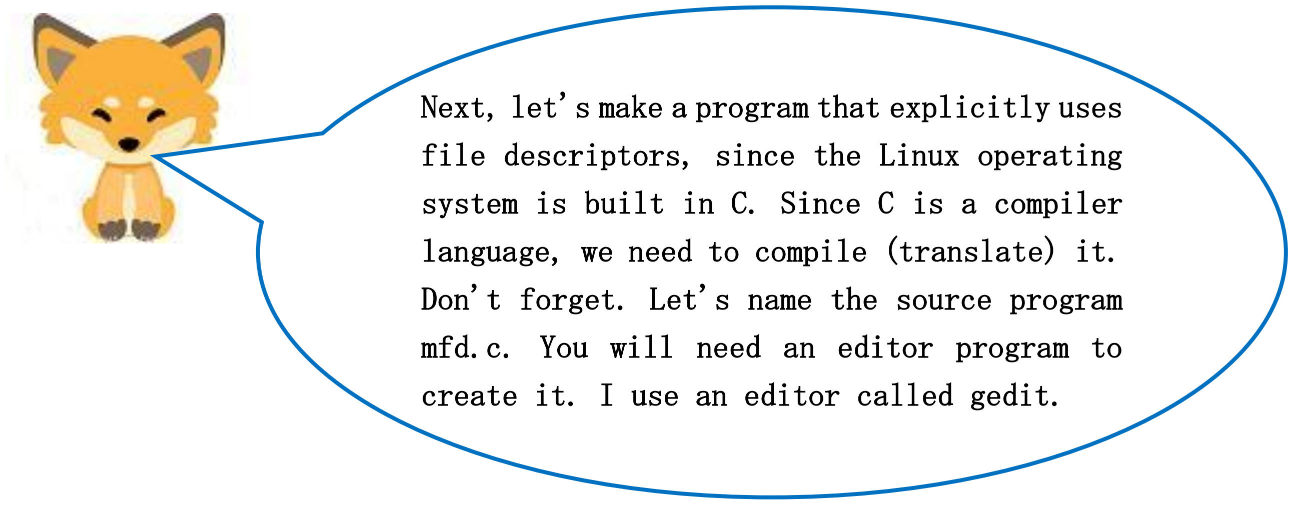
Interaction between the process and external devices

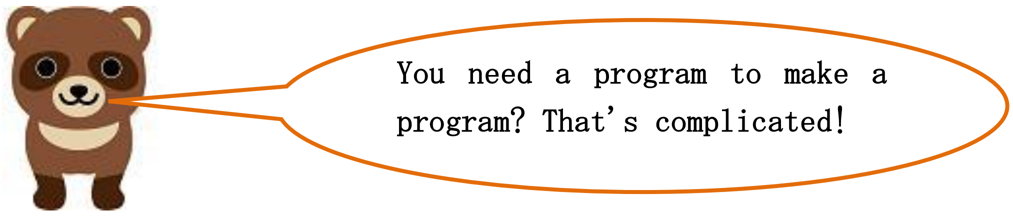


・Number of entrances and exits per process (number of file opens)



```
pi@raspberrypi:~ $ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 7336
max locked memory       (kbytes, -l) 65536
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 7336
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
pi@raspberrypi:~ $
```


Next, let’s make a program that explicitly uses file descriptors, since the Linux operating system is built in C. Since C is a compiler language, we need to compile (translate) it. Don’t forget. Let’s name the source program mfd.c. You will need an editor program to create it. I use an editor called gedit.



[From creation to execution of mfd.c]

・Create mfd.c with gedit and save.　(mfd.c: process at runtime)

```
#include <stdio.h>   /*  standard input/output */
int main()
{
        int fd;
        char buf[10];
        fd = open("data",0);     /* 0:standard input : file descriptor   */
        read(fd,buf,5);    /* fd=3:file descriptor */
        write(1,buf,5);    /* 1:standard output : file descriptor */
        close(fd);
}
```

・Create and save the data file that mfd.c uses in the program with gedit.

```
abcdefghijklmn
```

・compilation (Run on terminal command line)

However, it is assumed that the mfd.c source file is saved in its own directory and that it is the current directory. The mfd.c source file is stored in its own directory and that directory is the current directory.

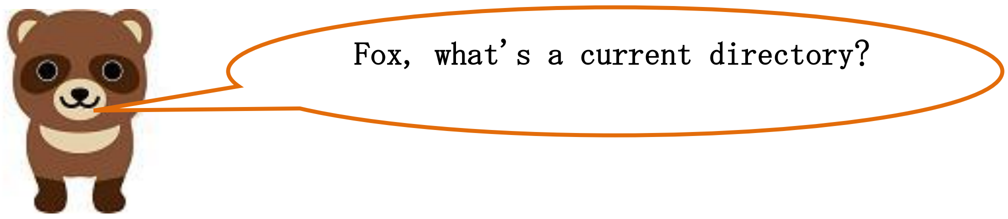
```
$ gcc  –o  mfd  mfd.c
```

・execution (e.g. program)

```
$ sudo ./mfd
```

・result (5 characters read, 5 characters output)

```
abcde
```



A directory is a Windows folder.
The current directory is the folder in which the user currently resides. The current directory is where the contents are displayed when the command "ls" is executed. If the file mfd.c is found there, it can be compiled. If the message "File does not exist" is displayed, it means that the file is not stored in the current directory.

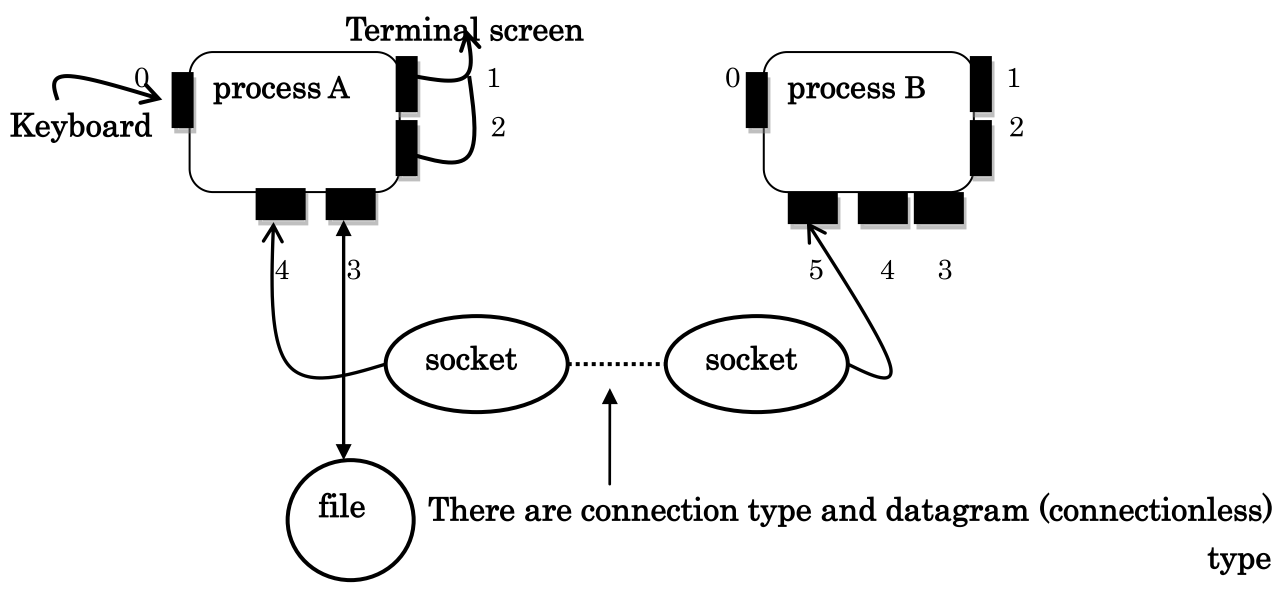
In the source file mfd.c, before "close(fd);" insert a line "printf("print file number %d",fd);" to display the positive value of the file descriptor used to read the data file!
If you insert a line "printf("print file number %d",fd);" before "close(fd);" in the source file "mfd.c", you can print the positive value of the file descriptor used to read the data file!
Go ahead, Tanuki, give it a try, it's a great way to practice compiling and executing the C language.

OK, I'll try.

Finally, let's talk about sockets, which are at the heart of Episode 5. Sockets are software interfaces (communication specifications) that processes in an OS can use to communicate with each other. Recently, browsers have been equipped with an interface function for inter-process communication, making it possible to have remote meetings and chats on the browser, which is very convenient.
Now, let me show you a diagram of a socket!

[What is a socket?]

File I/O and Interprocess Communication



Connection type: Communication path is secured (image of telephone line)
Datagram (connectionless) type: no permanent ties (postal image)
Client-server model
    Server: Process waiting for a connection
        Server program: Program responsible for accepting socket connections
    Client: Process requesting a connection
        Client program: A program that connects a socket to a server

Process A and Process B in the above diagram can be running in one computer, or if they are networked, they can be remote, with Process A running on a computer at home and Process B running on a computer at work.
So, in Episode 6, let's try to actually program the chat. ・・・・・.

Continue to Episode 6

Translated at DeepL