University of Athens
Dept. of Informatics and Telecommunications *K22*

Named-Pipes

Introduction
A named-pipe is a special type of file (FIFO) that is stored in the local file-system and allows for inter-process communication through writing to and reading from this file. Every such named-pipe is designated by a path (in a way similar to a regular file). Hence, every time two processes want to communicate, they can "open" this FIFO file and while one writes the other reads. Name-pipes must be explicitly removed as they do not disappear at the end of the process communication.

In order to have two processes communicate, they both have to open the corresponding FIFO file. Opening up this file for writing, a process can send messages while opening the FIFO for reading, another process can receive (read) messages lined up in the file. Evidently, the two processes must be part of the same computing system.

Creating a FIFO file
A name-pipe can be created in your program by invoking the following call:

```
int mkfifo(const char *pathname, mode_t mode);
```

The first parameter is the (absolute or relative) path to the file under creation and the second parameter offer the access rights that the FIFO will have.

Opening a FIFO file
A named-pipe can be opened in a way very simialr to the one used in opening a regular file in Unix. The `open()` library system call is used as follows:

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

As is the case with opening regular files, if the above call is successful it returns a file descriptor (integer number). Similarly the following call could be used:

```
FILE *fopen(const char *path, const char *mode);
```

When `fopen()` is invoked a file pointer is returned (instead of the file descriptor). The file descriptor/pointer is used to either read or write to the named-pipe.

Reading and Writing a FIFO file
The writing and reading of a FIFO file is identical to that of writing and reading of a regular file. The library calls that can be used to accomplish this are:

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

The differences between traditional files and named-pipes are as follows:

1. A named-pipe cannot be opened for both reading and writing at the same time. A name-pipe can be opened with either `open()` or `fopen()` by a single process. Should you require bidirectional communication between two processes then two FIFO files have to be established witch each one implementing a unidirectional channel of communication.
2. Both reading and writing are by default *blocking*. This means that if a process tries to read from a named-pipe that does not have data, it will block. Similarly, if a process write into a named-pipe that has not yet been opened by another process, the writer will block.
3. Movements of the file "current" position as is the case in regular files is not allowed in named-pipes.

If we want to have a process not-blocking until data appears for reading, the `poll()` call can be used:

```
#include <poll.h>

int poll(struct pollfd *fds, nfds_t nfds, int timeout);
```

Below find an example of how to use this call. With the help of `poll()`, the program checks when there is something available for reading. The program reads from the named-pipe only after makes sure there is something to read.

```
#define  MSG_BUF        256

#include <poll.h>
#include <signal.h>
#include <stdio.h>
#include <fcntl.h>
#include <ctype.h>
#include <stdlib.h>

main(int argc, char *argv[]){
char buf[MSG_BUF];
int bytes_in, fd;
struct pollfd fdarray[1];
int rc, i;

if (argc!= 2) {
        printf("Usage:␣%s␣<name␣of␣pipe>␣\n",argv[0]);
        exit(1);
        }

if (mkfifo(argv[1], 0666) < 0){
        perror("Error␣creating␣the␣named␣pipe");
        exit(1);
        }

fd=open(argv[1], O_RDONLY, 0);

for(;;){
        /* initialize poll parameters */
        fdarray[0].fd = fd;
        fdarray[0].events = POLLIN;

        /* wait for incomign data or poll timeout */
        rc = poll(fdarray, 1, 300);

        if (rc == 0) {
                printf("Poll␣timed-out.\n");
                exit(1);
                }
        else if ( (rc == 1) && (fdarray[0].revents == POLLIN)  ){
                if ( fdarray[0].fd == fd ){
                        bytes_in = read(fd, buf, MSG_BUF);
                        buf[bytes_in]='\0';

                        printf("\nRead␣from␣the␣pipe␣:␣%s\n", buf);
                        fflush(stdout);
                        }
                }
        }
}
```

Removing a FIFO file

A named-pipe can be removed as any other regular file.

Using a FIFO file

In what follows, we provide an example of how a (blocking) FIFO can be used between a program (termed *server*) that awaits some input from another program (called *client*). The client provides a string of characters and the server as soon as it reads the line of input turns all characters to upper case. A snapshot of the work of the server is shown below:

```
ad@serifos:~/Pitt-CS1550/NamedPipes/src$ ./server MyNAMEDpipe1
Read from the pipe : This IS the MESSAGE - Alex Delis
Converted String : THIS IS THE MESSAGE - ALEX DELIS
ad@serifos:~/Pitt-CS1550/NamedPipes/src$
```

The server program that needs to be run first is:

```
#define  MSG_BUF        256

#include <poll.h>
#include <signal.h>
#include <stdio.h>
#include <fcntl.h>
#include <ctype.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

main(int argc, char *argv[]){
        int fd, ret_val, count, numread;
        char buf[MSG_BUF];

        if (argc!= 2) {
                printf("Usage:␣%s␣<name␣of␣(server)␣pipe>␣\n",argv[0]);
                exit(1);
                }

        ret_val = mkfifo(argv[1], 0666);

        if ((ret_val == -1) && (errno != EEXIST)){
                perror("Error␣creating␣the␣named␣pipe");
                exit(1);
                }

        /* open for reading only */
        fd=open(argv[1], O_RDONLY);

        numread = read(fd, buf, MSG_BUF);
        buf[MSG_BUF]='\0';

        printf("Read␣from␣the␣pipe␣:␣%s\n", buf);

        printf("Converted␣String␣:␣");
        count =0;
        while( count < numread ){
                buf[count]=toupper(buf[count]);
                putchar(buf[count]);
                count++;
                }
        putchar('\n');
        }
```

The source code for the client-program that needs to be run in a different `tty` from the server (so that confusion is avoided) is:

```
#include <stdio.h>
#include <ctype.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

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

        if (argc!= 3) {
                printf("Usage:␣%s␣<name␣of␣named-pipe>␣<message>␣\n",argv[0]);
                exit(1);
                }

        /* open for writing only */
        fd = open(argv[1], O_WRONLY);

        /* write into the named-pipe */
        write(fd, argv[2], strlen(argv[2]) );
        }
```