Wrocław University
of Science and Technology

Faculty of Microsystem Electronics and Photonics

# Operating systems

## Lab. 3.

### I. Issues to prepare

- Pipes, processes in Linux
- FIFO, named pipes

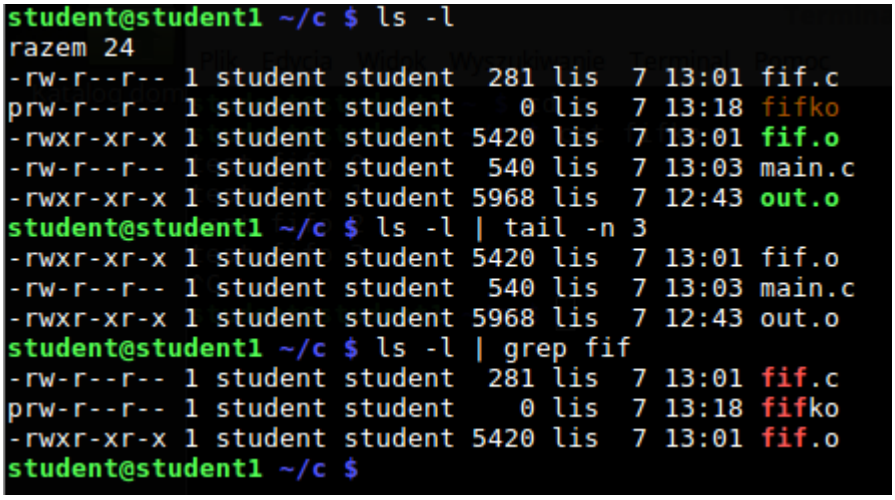
### II. Outline

1. Pipes implementation in C
2. Named pipes
3. Tasks

## III. Tasks

1. Start Linux on virtual machine. Log in and run terminal.
2. Check how Linux pipes work:

Pipes can be used by key "|". Example:




Example analysis:

- command "ls –l" lists content of the current folder,
- command "tail –n X" shows only last X lines of standard input,
- command "grep fif" shows only lines containing "fif" phrase.

By using "|" output of *ls –l* command is redirected as input to *tail* or *grep*.

3. Check out how pipes can be used in C language:

```
int fd[2];
int main() {
  printf("Main process, getpid=%d\n", getpid());
  pipe(fd);
  pid_t pid = fork();
  if(pid == 0) {
    char *msg = "The message from child to parent\n";
    printf("I am child, getpid=%d.\n", getpid());
    close(fd[0]); //close 'read' descriptor
    write(fd[1], msg, strlen(msg)+1);
  } else {
    printf("A am parent, getpid=%d.\n", getpid());
    close(fd[1]); //close 'write' descriptor
    char msg[100];
    read(fd[0], msg, sizeof(msg));
    printf(msg);
  }
  return 0;
}
```

4. How the fork() function works like?
5. Create named pipe FIFO:

```
# mkfifo /somewhere/abc ; ls -l /somewhere/abc
prw-------  1 root root 0 2006-04-03 07:45 abc
```

6. Show FIFO content by basic *cat* command, may be in another terminal:

```
# cat /somewhere/abc
test fifo 0
test fifo 1
test fifo 2
```

7. How to use FIFO in C:

```
int fd, i=0; char buf[32];
int main() {
  fd = open("/somewhere/abc", O_WRONLY);
  for(;;) {
    sprintf(buf, "test fifo %d\n", i++);
    write(fd, buf, strlen(buf));
    sleep(1);
  }
  return 0;
}
```

Useful bash commands:

./program.o **&** - runs program in the background

ctrl+z - stops running present program

bg - wakes up stopped process, let in run in the background

fg - return process to foreground

ps - lists running processes

crtl+c - kills present process

kill PID - kills process no. PID

killall *processname* - kills process with *processname* name

```
student@student1 ~/c $ ps
  PID TTY          TIME CMD
 3208 pts/0    00:00:00 bash
 3494 pts/0    00:00:00 fif.o
 3495 pts/0    00:00:00 ps
student@student1 ~/c $ kill 3494
student@student1 ~/c $ ps
  PID TTY          TIME CMD
 3208 pts/0    00:00:00 bash
 3496 pts/0    00:00:00 ps
[1]+  Zakończony              ./fif.o
```

**Tasks:**

1. Write down a code using the fork() function. Compile and execute the code. Observe the process list (with the *ps -A* command) before and after forking the base process.
2. Modify the above code so the process forks many times. What are the limists for the Linux system?
3. Use the pipes (fd-0, fd-1) for communication between parent and child processes. Use the pipe(), dup(), dup2() functions.
4. Create a FIFO file named /tmp/fifo.
5. Write down a code which forks few times and each copy writes a data to the fifo. Process 'A' should write „AAAAA…", process 'B' „BBBBBB…" respectively and so on, with configurable length of the block written to the fifo.

6. With the ‘cat /tmp/fifo’ read and display the data from fifo. What is the limit of a data before data mixing from various processes occurs?

7. How the process memory is allocated after forking? Fork the process c.a. 10 times. Examine the behavior of the memory usage for process using const char[], char[], malloc() variables.