Sisteme de operare
Tema 8

Exercițiul 1

Rulați și inspectați toate exemplele. Asigurați-vă că ați înțeles cum funcționează.

```
[user@fedora destination]$ make lib
gcc -Wall -g -O   -c -o error.o error.c
gcc -Wall -g -O   -c -o tellwait.o tellwait.c
ar rcs liblab8.a error.o tellwait.o
[user@fedora destination]$ make
gcc -o pipe1 pipe1.c liblab8.a
gcc -o pipe2 pipe2.c liblab8.a
gcc -o tellwait2 tellwait2.c liblab8.a
gcc -o popen2 popen2.c liblab8.a
gcc -o myuclc myuclc.c liblab8.a
gcc -o popen1 popen1.c liblab8.a
gcc -o add2 add2.c liblab8.a
gcc -o pipe4 pipe4.c liblab8.a
gcc -o fifo fifo.c liblab8.a
```

Am generat biblioteca liblab8.a și după am compilat exemplele.

```
[user@fedora destination]$ ./pipe1
[user@fedora destination]$ hello world
```

Programul pipe1.

```
#include      <sys/wait.h>
#include      "ourhdr.h"

#define DEF_PAGER       "/usr/bin/more"         /* default pager program */

int main(int argc, char **argv)
{
        int     n, fd[2];
        pid_t   pid;
        char    line[MAXLINE], *pager, *argv0;
        FILE    *fp;

        if (argc != 2)
                err_quit("usage: a.out <pathname>");
        if ( (fp = fopen(argv[1], "r")) == NULL)
                err_sys("can't open %s", argv[1]);

        if (pipe(fd) < 0)
                err_sys("pipe error");

        if ( (pid = fork()) < 0)
                err_sys("fork error");
        else if (pid > 0) {                     /* parent */
                close(fd[0]);           /* close read end */
                        /* parent copies argv[1] to pipe */
                while (fgets(line, MAXLINE, fp) != NULL) {
                        n = strlen(line);
                        if (write(fd[1], line, n) != n)
                                err_sys("write error to pipe");
                }
                if (ferror(fp))
                        err_sys("fgets error");

                close(fd[1]);   /* close write end of pipe for reader */
                if (waitpid(pid, NULL, 0) < 0)
                        err_sys("waitpid error");
                exit(0);

        } else {                                /* child */
                close(fd[1]);   /* close write end */
                if (fd[0] != STDIN_FILENO) {
                        if (dup2(fd[0], STDIN_FILENO) != STDIN_FILENO)
                                err_sys("dup2 error to stdin");
                        close(fd[0]);   /* don't need this after dup2 */
                }

                        /* get arguments for execl() */
                if ( (pager = getenv("PAGER")) == NULL)
                        pager = DEF_PAGER;
```

Programul pipe2.

```
[user@fedora destination]$ ./tellwait2
output from parent
output from child
```

Programul tellwait2.

```
#include        <sys/wait.h>
#include        "ourhdr.h"

#define PAGER   "${PAGER:-more}" /* environment variable, or default */

int main(int argc, char *argv[])
{
        char    line[MAXLINE];
        FILE    *fpin, *fpout;

        if (argc != 2)
                err_quit("usage: a.out <pathname>");
        if ( (fpin = fopen(argv[1], "r")) == NULL)
                err_sys("can't open %s", argv[1]);

        if ( (fpout = popen(PAGER, "w")) == NULL)
                err_sys("popen error");

        /* copy argv[1] to pager */
        while (fgets(line, MAXLINE, fpin) != NULL) {
                if (fputs(line, fpout) == EOF)
                        err_sys("fputs error to pipe");
        }
        if (ferror(fpin))
                err_sys("fgets error");
        if (pclose(fpout) == -1)
                err_sys("pclose error");
        exit(0);
}
```

Programul popen2.

```
[user@fedora destination]$ ./popen1
prompt> ceva
CEVA
prompt> CEVA
CEVA
```

Programul popen1.

Programul pipe4.

Exercițiul 2

Rescrieți exemplul `pipe1` folosind funcția `popen`. Folosiți în loc de programul pager comanda `cat`.

```c
#include	"ourhdr.h"
#include  <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
	FILE* fp;
	char line[1024];

	fp=popen("cat","w");
	if(fp==NULL)
	{
		printf("Failed to run command\n");
		exit(1);
	}

	strcpy(line,"hello world\n");
	fwrite(line,sizeof(char),strlen(line),fp);

	pclose(fp);
	exit(0);
}
```

Am rescris programul pipe1.

```
[user@fedora destination]$ ./pipe1
hello world
```

Am rulat noul program.

Exercițiul 3

Rescrieție exemplul `pipe4` folosind FIFO. Veți avea două procese: unul care citește din fișier, scrie în FIFO, citește din FIFO și afișează, respectiv un al doilea proces care preia din FIFO parametrii, execută adunarea și scrie rezultatul tot în FIFO.

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define FIFO_PATH "/usr/destination"

void executeAddition(int param1, int param2)
{
        int result = param1+param2;
        printf("Result: %d\n",result);
}
int main(void)
{
        int fd;
        pid_t pid;
        int params[2];

        if(mkfifo(FIFO_PATH, 0666)==-1)
        {
                if(errno!=EEXIST)
                {
                        perror("mkfifo");
                        exit(1);
                }
        }

        if((pid=fork())==-1)
        {
               perror("fork");
               exit(1);
        }

        if(pid>0)
        {
                fd=open(FIFO_PATH, O_WRONLY);
                if(fd==-1)
                {
                        perror("open");
                        exit(1);
                }
```

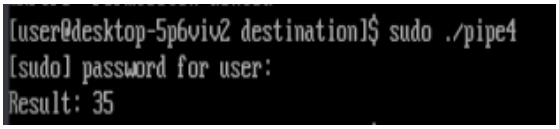
```
        FILE *file=fopen("input.txt","r");
        if(file==NULL)
        {
                perror("fopen");
                exit(1);
        }
        if(fscanf(file, "%d %d", &params[0], &params[1]) !=2)
        {
                perror("fscanf");
                exit(1);
        }

        fclose(file);

        if(write(fd,params, sizeof(params))==-1)
        {
                perror("write");
                exit(1);
        }
        close(fd);
}
else
{
        fd=open(FIFO_PATH, O_RDONLY);
        if(fd==-1)
        {
                perror("open");
                exit(1);
        }

        int param1, param2;
        if(read(fd, &param1, sizeof(int))==-1)
        {
                perror("read");
                exit(1);
        }
        if(read(fd,&param2, sizeof(int))==-1)
        {
                perror("read");
                exit(1);
        }

        close(fd);
}

if(pid>0)
        unlink(FIFO_PATH);
return 0;
```

Programul pipe4 rescris astfel încât să facă ce se cere în cerință.

Programul rescris creează un FIFO, dacă nu există, pe urmă se împarte în două procese, unul părinte și unul copil. Procesul părinte citește din fișierul input.txt și scrie în FIFO. Procesul copil preia din FIFO parametrii, adică îi citește, pe urmă face adunarea și o afișează. După ce procesul copil se execută, procesul părinte elimină FIFO-ul cu funcția unlink.

```
[user@fedora destination]$ ./pipe4
23 12
35
```

Am executat programul pipe4 înainte de modificare.



Am executat programul pipe4 după modificare.