**Ficha Prática Final :** Sistemas Operativos 2021 :
InterProcess Communiocation IPC – Pipes e Filas

https://www.di.ubi.pt/~operativos/praticos/pdf/11-ipc.pdf

**(1) Implementar a resolução do Exercicio do Exame 1, 8 de Junho de 2016, 14.30 Horas Grupo E: Exercício Prático . Faça uma diagrama para comprender os dois pipes**

Utilizando (duas) pipetas de baixo nível (pipes) é possível criar uma comunicação bidireccional entre dois processos. O objectivo deste exercício é escrever **um** programa que criará dois processos (usando fork()), um cliente e um servidor, onde o processo servidor funcionará como calculador simples. O processo cliente deverá ler (a partir do teclado) dois operandos (números reais) seguido por um operador (+,-,*, / ) e deverá enviar estes valores para o servidor usando um pipe. O servidor calculará o resultado enviando depois o resultado para o cliente usando um pipe, o cliente deverá mostrar no fim o resultado no ecrã:

Exemplos de execução do programa:

| | |
|---|---|
| $./exame | $./exame |
| 2.5 3 + | 3.0 2 / |
| 5.5 | 1.5 |

```
#include <stdio.h> <stdlib.h> <fcntl.h>
#define SR sizeof(float)
int main ()  //ficheiro exame.c
{
    int pid, p1[2], p2[2];
    float op1,op2 , res;
    char op;
    pipe (p1);
    pipe (p2);
    pid = fork ();
    if (0 == pid) /*cliente*/ {
            Read op1,op2,op from user :  usando scanf
            write (p1[1], &op1, SR);
            write (p1[1], &op2, SR);
            write (p1[1], &op, 1);
            read (p2[0], &res, SR);
            printf("Result %f\n",res);
     }
    else /*server*/{
            read (p1[0], &op1, SR);
            read (p1[0], &op2, SR);
            read (p1[0], &op, 1);
            switch ( op ){
                    case '+' : res=op1+op2; break;
                    case '/' : res=op1/op2; break;
            }
            write (p2[1],   &res, SR);
  }
  return 0;
```

**(2) Implementação com Named Pipes**

Usando com base o programa aneirior modifica o programa para criar dois programas seperadas – um Server e um Client. Nota que o Server e Cliente Irá Executar Apenas Uma Vez.

Server+Client necisstam de saber o ficheiro que represent o pipe

```
#define FIFO1 "/tmp/fifo.1"
#define FIFO2 "/tmp/fifo.2"
```

SERVER

```
#define PERMS 0666
umask(0)
mknod(FIFO1, S_IFIFO | PERMS, 0);    mknod(FIFO2, S_IFIFO | PERMS, 0) ;
readfd  = open(FIFO1, 0);
writefd = open(FIFO2, 1);
```

CLIENTE

```
writefd = open(FIFO1, 1);
readfd =  open(FIFO2, 0);
unlink  para remover os pipes
```

Executar : ./server num terminal e a seguir ls -l /tmp para verificar a existencia dos pipes e depois ./cliente num outro terminal.
Nota. Em vez de usar mknod usar de preferência mkfifo(FIFO?, PERMS);

**(3) No seu programa do soshell implementação dum sistema de mensagem para outro utilizador.**

Sintaxe : message user string

- Por exemplo message ruisantos OlaRui

Implementação

- O Programa shell pode enviar mensagens usando um comando embutido
- O programa shell pode escutar mensagens numa thread separada lançada no arranque do shell.
- A comunicação deve ser feito usando um fila de mensagens

A Resolução deve ser feito em grupo.

Notas Importantes

- Como comunicar uma mensagem para um utilizador específica ?
- Resposta utilizar o valor do seu userid !! Este valor pode ser o valor do “mesg_type”

The getuid() function returns the real user ID of the calling process. The real user ID identifies the person who is logged in. returns an unsigned integer (uid_t)
Para obter dum outro utilizador : : https://stackoverflow.com/questions/39157675/how-to-get-linux-user-id-by-user-name

Dica do desenvolvimento.
Para implementar entre shells do mesmo utilizador define um valor inteirio ”instance” da instancia do shell. A seguir

- Lancar thread a escutar mensagens com type “instance”
- Para enviar : message 2 OlaRui (onde 2 seria a instancia com ID 2)