

Ricard Sampedro Ochoa

NIU: 1191222

Sergi Cruz Heredia

1281162

Memòria Pràctica 2

Sistemes Operatius

Grup 413-8

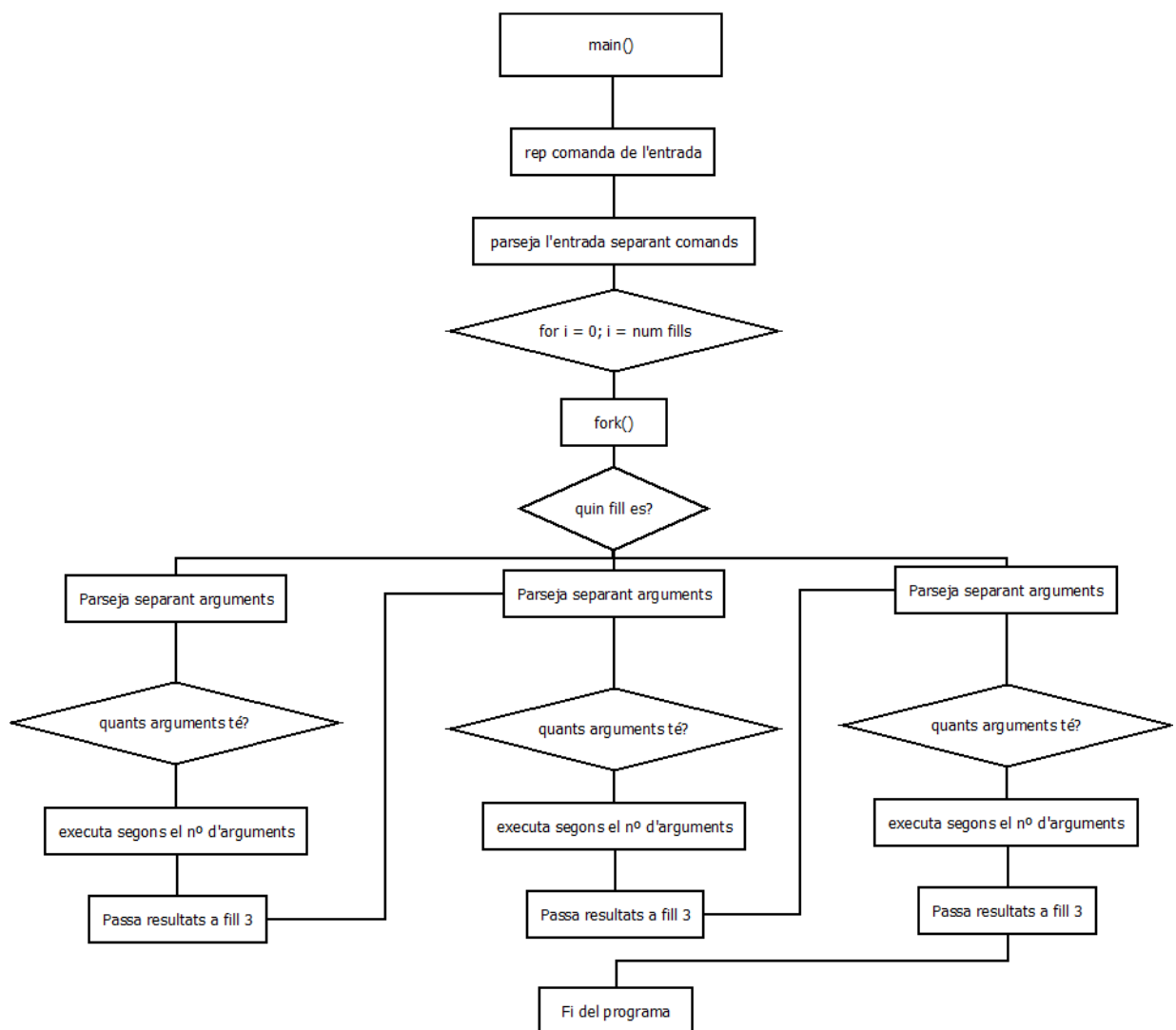
Continguts

1.	<i>Objectiu de la pràctica.....</i>	3
2.	<i>Descripció i plantejament de la pràctica (Organigrama, pseudo codi).....</i>	3
3.	<i>Descripció dels procediments utilitzats a nivell funcional</i>	4
4.	<i>Descripció dels problemes sorgits durant la realització de la pràctica i les solucions trobades.</i>	4
5.	<i>Conclusions extretes de la realització de la pràctica</i>	5
6.	<i>Codi font de la pràctica.</i>	6
7.	<i>Procediments de compilació.....</i>	10

1. Objectiu de la pràctica

Aprendre i habitar-se a l'ús de les crides a sistema en sistemes operatius de tipus UNIX i aprendre els coneixements bàsics de creació de processos mitjançant la comanda `fork()` i la comunicació entre processos per mitjà de pipes. Tot això a través de la creació d'scripts en C-Shell.

2. Descripció i plantejament de la pràctica (Organigrama, pseudo codi)



3. Descripció dels procediments utilitzats a nivell funcional

fork():	crea un child del procés actual amb pid diferent al del procés pare i que pot executar codi diferent.
sscanf():	Utilitzat per a parsejar cadenes de caràcters, seguint les normes que se li passin retorna punters.
waitpid():	Detén l'execució fins a la finalització d'un child amb el pid corresponent.
close():	Tanca el pipe especificat.
dup():	Duplica el pipe especificat PERO NO ELIMINA L'ORIGINAL
strtok():	Retorna un punter a la primera aparició del caràcter o caràcters especificats dins d'una cadena.
execlp():	Executa el programa especificat amb els arguments especificats.
strcpy():	Copia la cadena de caràcters especificada a l'array especificat.

4. Descripció dels problemes sorgits durant la realització de la pràctica i les solucions trobades.

Un problema que ens hem trobat és que en la primera pràctica si feiem els bucles incorrectament, el procés crearà childs infinitament colapsant l'ordinador. La solució que hi vam trobar va ser simplement tenir més cura en definir loops; més encara quan manipulem coses d'aquestes.

Un altre problema que vam trobar va ser a l'hora de fer els pipes, que moltes vegades en estar tancats i oberts de manera incorrecta els childs no transferien l'informació com s'esperava i podia donar tot una plétora de resultats inesperats, desde no mostrar cap de les funcions per pantalla fins a mostrar-ne de repetides.

Un problema al que vem trobar solució també va ser a l'hora de parsejar les comandes en principi no eliminàvem el primer espai després de cada barra de pipe. Això no ens va donar molts mals de cap, però de totes maneres vem mirar de solucionar-ho, en comptes de fer el parseig amb strtok(), ho vem passar a fer amb sscanf() tot i que va ser més difícil de trobar quins arguments li havíem de passar per a obtenir el que buscàvem.

5. Conclusions extretes de la realització de la pràctica

D'aquesta pràctica hem, a part d'haver entès en part el que es pretenia, ampliat una mica el ventall de possibilitats de coses realitzables amb C-Shell, tot i que a gustos personals no ens acaba de fer el pes, en part pel haver d'escriure el cod i amb gedit estant acostumats a les comoditats de Visual Studio o altres eines més enfocades a l'usuari aprenent. Tot i així considerem que aquesta ha sigut una experiència enriquidora i ens ha aixamplat les perspectives sobre la programació en entorns diferents.

6. Codi font de la pràctica.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <limits.h>
#include <string.h>
#define numchild 3          // numero de hijos
int main(){
    int i,j, listpid[numchild],status, n;
    pid_t child;
    int fd[2];
    int fd1[2];
    typedef char t_command [20];
    t_command commands[3];
    t_command arguments[4];
    t_command tmp;
    char *ptr;
    char entrada[50];
    int numargs, numcommand;

    pipe(fd);
    pipe(fd1);
    printf ("Introdueix la commanda\n");
    gets (entrada);

    /*PARSING DE COMANDES*/
    /* Omple commands[3] amb les comandes que s'hagin fet separades per " |
    */
    i=0;
    ptr = entrada;
    while(sscanf(ptr,"%19[^\n]",tmp,&n) == 1)
    {
        if (numcommand > 3)
        {
            perror("too many commands");
            exit(EXIT_FAILURE);
        }
        ptr = ptr + n;
        strcpy(commands[numcommand],tmp);
        numcommand++;
        if(*ptr != '|')
        {
            break;
        }
        ptr = ptr +2;
    }
    printf ("El numero de comandes és %d \n", numcommand);
    puts ("Commands:");
    for (i=0; i < numcommand; i++){
        puts (commands[i]);
    }
    for(i=0; i< numchild; i++){
        if((child = fork()) == -1){
            perror("fork");
            exit(EXIT_FAILURE);
        }
        else if(child == 0){
            if(i == 0){
                if (numchild > 1){
```

```

        close(1);
        dup(fd[1]);
    }
    close(fd[1]);
    close(fd[0]);
    close(fd1[0]);
    close(fd1[1]);
    fprintf(stderr, "\nrunning %s, in first child process with pid: %d.
\n", commands[0], getpid());
    j=0;
    ptr = strtok(commands[j], " ");
    strcpy (arguments[0], ptr);
    while ((ptr = strtok( NULL, " ")) != NULL)
    {
        j++;
        strcpy (arguments[j], ptr);
    }
    numargs = j+1;
    printf ("\nEl numero d'arguments és %d \n", numargs);
    for (i=0; i < numargs; i++){
        puts (arguments[i]);
    }
    switch (numargs){
        case (1):
            execlp(arguments[0], arguments[0],
(char*) 0);
            break;
        case (2):
            execlp(arguments[0], arguments[0],
arguments[1], (char*) 0);
            break;
        case (3):
            execlp(arguments[0], arguments[0],
arguments[1], arguments[2], (char*) 0);
            break;
        case (4):
            execlp(arguments[0], arguments[0],
arguments[1], arguments[2], arguments[3], (char*) 0);
            break;
        default:
            puts("Alguna cosa ha fallat");
            break;
    }
    exit(EXIT_SUCCESS);
}
else if(i == 1){
    fprintf(stderr, "\nrunning %s, in second child process with
pid: %d. \n", commands[1], getpid());
    sleep(1);
    dup(fd[0]);
    if (numchild > 2){
        close(1);
        dup(fd1[1]);
    }
    close(fd[1]);
    close(fd[0]);
    close(fd1[0]);
    close(fd1[1]);
j=0;

    ptr = strtok(commands[j], " ");
    strcpy (arguments[0], ptr);
    while ((ptr = strtok( NULL, " ")) != NULL)

```

```

        {
            j++;
            strcpy (arguments[j], ptr);
        }
        numargs = j+1;
        printf ("\nEl numero d'arguments és %d \n", numargs);
        for (i=0; i < numargs; i++){
            puts (arguments[i]);
        }
        switch (numargs){
            case (1):
                execlp(arguments[0], arguments[0],
(char*) 0);
                break;
            case (2):
                execlp(arguments[0], arguments[0],
arguments[1], (char*) 0);
                break;
            case (3):
                execlp(arguments[0], arguments[0],
arguments[1], arguments[2], (char*) 0);
                break;
            case (4):
                execlp(arguments[0], arguments[0],
arguments[1], arguments[2], arguments[3], (char*) 0);
                break;
            default:
                puts("No d'arguments invàlid");
                break;
        }
        exit(EXIT_SUCCESS);
    }
    else if(i == 2){
        fprintf(stderr, "\nrunning %, in third child process with
pid: %d. \n", commands[2], getpid());
        j=0;
        sleep(2);
        close(0);
        dup(fd1[0]);
        close(fd[1]);
        close(fd[0]);
        close(fd1[0]);
        close(fd1[1]);
        ptr = strtok(commands[j], " ");
        strcpy (arguments[0], ptr);
        while ((ptr = strtok( NULL, " ")) != NULL)
        {
            j++;
            strcpy (arguments[j], ptr);
        }
        numargs = j+1;
        printf ("\nEl numero d'arguments és %d \n", numargs);
        for (i=0; i < numargs; i++){
            puts (arguments[i]);
        }
        switch (numargs){
            case (1):
                execlp(arguments[0], arguments[0],
(char*) 0);
                break;
            case (2):

```



```

arguments[1], (char*) 0);
                                execlp(arguments[0], arguments[0],
                                break;
                                case (3):
arguments[1], arguments[2], (char*) 0);
                                execlp(arguments[0], arguments[0],
                                break;
                                case (4):
arguments[1], arguments[2], arguments[3], (char*) 0);
                                execlp(arguments[0], arguments[0],
                                break;
                                default:
                                puts("Nº d'arguments no valid");
                                break;
                                }
                                exit(EXIT_SUCCESS);
                                }
                                }
                                else{
                                listpid[i]=child;
                                }
                                }
                                close(fd1[0]);
                                close(fd1[1]);
                                close(fd[0]);
                                close(fd[1]);
                                sleep(1);
                                for(i = 0; i < numchild; i++){
                                waitpid(listpid[i], &status, 0);
                                printf("\n[%d] TERMINATED (Status: %d)\n",listpid[i], status);
                                }
                                exit(EXIT_SUCCESS);
                                }

```

7. Procediments de compilació

1. Anar mitjançant cd al directori que contenen els codis font a compilar.
2. Executar l'ordre corresponent al codi a compilar utilitzant la següent estructura:

gcc <nom del codi Font> -o <nom de l'executable>

Pràctica 1	gcc my-shell1.c -o my-shell1
Pràctica 2	gcc my-shell2.c -o my-shell2
Pràctica 3	gcc my-shell3.c -o my-shell3
Pràctica 4	gcc my-shell4.c -o my-shell4
Pràctica 5	gcc my-shell5.c -o my-shell5