Examen Final SIOP 2021

Ixent Cornella

Exercici 1:

a)
Sense buffering:

```
int main (int argc, char *argv[]){
    int n;
    char c;
    while( (n=read(0, &c, 1)) > 0){ //sizeof(c) == 1
        if(write(1, &c, 1) < 0) perror("jesus: Error al write");
    }
    if (n < 0) perror("jesus: Error al read");
}
```

Amb buffering:

```
int main (int argc, char *argv[]){
        int n;
        char buff[MAX];
        while( (n=read(0, buff, MAX)) > 0){
            if(write(1, buff, n) < 0)
                perror("jesus: Error al write");
        }
        if (n < 0) perror("jesus: Error al read");
    }
```

b)

```
char alarma = 0;

void f_alarma()
{
    alarma = 1;
}

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

    if(argc != 2){
        printf("rei: Error, introdueix un sol argument.\n");
        error(-1);
    }

    char c;
    int i = 0;
```

```
     signal(SIGALRM, f_alarma); //reprogramem alarma perque fagi el que
indiquem a f_alarma.
    while(argv[1][i] != 0){
        alarm(1);   //alarma a 1 segon, pausem durant 1s.
        while (alarma == 0) pause();
        c = argv[1][i]; //per evitar error de tipus
        if(write(1, &c, 1) < 0) perror("rei: Error al write");
        i++;
        alarma = 0;
    }
    if(write(1, "\n", 1) < 0) perror("rei: Error al write");
}
```

c)

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

#define MAX 50

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

    if(argc != 2){
        printf("betlem: Error, introdueix un sol argument.\n");
        error(-1);
    }
    int fd = open(argv[1], O_RDONLY);
    if (fd < 0) perror("betlem: Error al open");

    char regals[3][MAX];
    //OBTENIM CADA LINIA DEL FITXER
    int n, i = 0, regal = 0;
    char c;

    while((n=read(fd, &c, 1)) > 0){
        if(c == '\n'){
            regal++;
            i = 0;
        }
        else{
            regals[regal][i] = c;
```

```
            i++;
        }
    }
    if (n < 0) perror("betlem: Error al read");
    int p[2];
    if(pipe(p) < 0) perror("betlem: Error al pipe");
    int reis[3];
    for(int i = 0; i < 3; i++){
        reis[i] = fork();
        if (reis[i] < 0) perror("betlem: Error al fork");
        if(reis[i] == 0){
            dup2(p[1], 1); //redirigeim stdout a la pipe
            close(p[0]);
             execlp("./rei", "./rei", regals[i], NULL); //executem rei i
li passem el regal corresponent per parametre
        }
    }
    wait(NULL);
    int jesus = fork();
    if (jesus < 0) perror("betlem: Error al fork");
    if(jesus == 0){
        dup2(p[0], 0); //redirigim stdin a la pipe.
        close(p[1]);
           execlp("./jesus", "./jesus", NULL); //executem jesus amb la
nova stdin
    }
    wait(NULL);


}
```

d)

La sortida del programa seran els tres regals, tot i que no necessàriament en el mateix ordre, ja que el ordre d'aquest depen del ordre en que jesus llegeixi els regals que li entren per la pipe.

e)

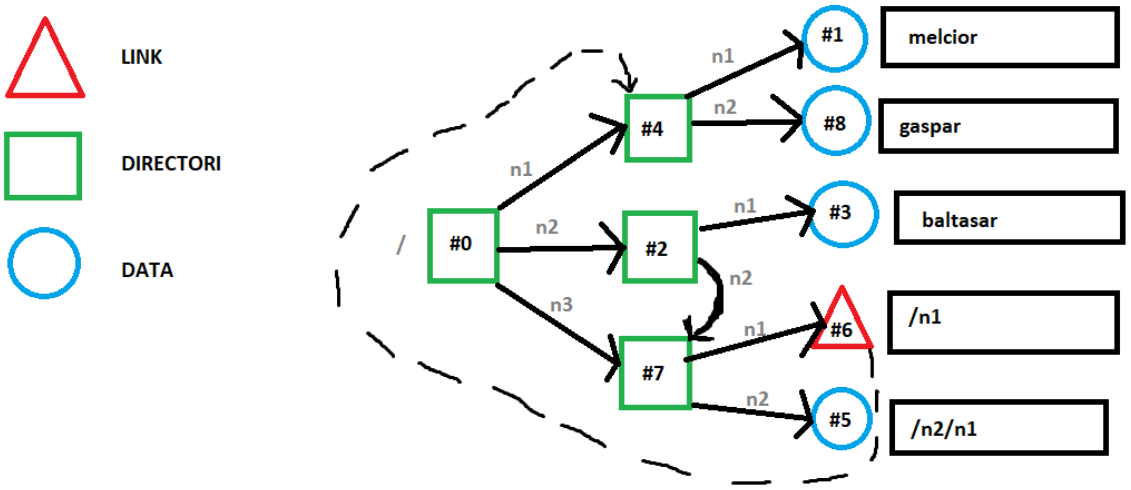
El que faria es canviar el for on es creen els fills, i canvar-li la i:

```
 for(int i = 2; i >= 0; i--){
        reis[i] = fork();
        if (reis[i] < 0) perror("betlem: Error al fork");
        if(reis[i] == 0){
```

```
            dup2(p[1], 1); //redirigeim stdout a la pipe
            close(p[0]);
             execlp("./rei", "./rei", regals[i], NULL); //executem rei i
li passem el regal corresponent per parametre
        }
    }
```

Exercici 2:

a) El graf ha quedat així:



b) El camp de #ref es pot deduir mirant quants cops es repteix el número d'inode als blocs de dades, en aquest cas per al directori pare **#0 -> #ref = 5**, per a **#4 -> #ref = 2,** i finalment **#7 -> #ref = 3.**

c) Si executessim això, ens portaria al inode #5, que té com a contingut al bloc de dades el text "/n2/n1". Tot i que té forma de path, tant sols és text (ho especifica el tipùs d'inode). El resultat és **"/n2/n1".**

d) Aquesta comanda donaria per pantalla "**gaspar**", ja que porta al inode #8.

e) Els accesos serien els següents: open("/n3/n1/n2",O_RDONLY) Sense buffer cache: inode 0 (/) + bloc 5 + inode 7 (n3) + bloc 8 + inode 6 (n1) + bloc 15 (tipus link, seguim) + inode 0 (/) + bloc 5 + inode 4 (n1) + bloc 2 + inode 8 (n2) = **11 accesos a disc.**

f) Els accesos, ara amb caches, serien els següents: open("/n3/n1/n2",O_RDONLY) inode 0 (/) + bloc 5 + inode 7 (n3) + bloc 8 + inode 6 (n1) + bloc 15 (tipus link, seguim) + inode 4 (n1) + bloc 2 + inode 8 (n2) = **9 accesos a disc.** Ens estalviem el segon accés al directori arrel, gràcies a les cache.

Exercici 3:

a)

En el nostre SO tenim adreces lògiques i físiques, i aquestes tenen una gran diferència: les adreces lògiques no existeixen físicament en la memòria, mentre que les físiques son una localització en la memòria, en la qual es pot accedir. Les adreces lògiques s'usen en el SO per accedir a adreces físiques, i enllaçar així la memòria física amb la virtual. El SO genera una adreça lògica, mentre que una adreça física es una localització en la memòria física.
A més, la memòria física existeix en el dispositiu (memòria RAM, disc, etc..) mentre que la virtual és un procés on les dades es van intercanviant amb la memòria física, amb les adreces mencionades anteriorment. La memòria lògica no pot coincidir amb la memòria física perquè la memòria lògica no existeix físicament.

b)

Si el bus d'adreces és de 64 bits, la mida de l'espai lògic del procesador és de 2^64 posicions de memòria.
L'espai físic està determinat per la quantitat de memòria instal·lada en una màquina. L'espai físic d'un procés és el conjunt d'adreces físiques associades al espai d'adreces lògiques del procés.

c)

Si accedim al SO com a privilegiats, vol dir que no hi han restriccions en quant a hardware, es pot accedir a tots els recursos del hardware. El Kernel, per exemple, s'executa en aquest mode. Es pot escriure coses per pantalla en mode usuari perquè el terminal en el que es veuen les coses és un procés creat per aquest mateix usuari.

d)

En memòria virtual, el dirty bit (o bit de modificació) ens indica que la pàgina corresponent ha estat modificada (si el bit val 1) o no (el bit val 0). Si el bit dirty val 0, no cal copiar la pàgina ja que aquesta no ha estat modificada, millorant així l'eficiència en cas de haver de reemplaçar una pàgina a la taula de pàgines.