Diciembre-2020

1. The execution of the following code generates at least three processes P1, P2 and P3:

```
...
pipe(fd);/*pipe1*/
pipe(fd2);/*pipe2*/
if(fork() != 0){
  /***Proceso P1 ***/
  dup2(fd[1],STDOUT_FILENO);
  close(fd[0]); close(fd[1]);
  dup2(fd2[0],STDIN_FILENO);
  close(fd2[0]);
close(fd2[1]);
  /* process table P1*/
}else{
  /***Proceso P2 ***/
  dup2(fd[0],STDIN_FILENO);
  close(fd[0]); close(fd[1]);
```

```
  pipe(fd3); /*pipe3*/
  if(fork() != 0){
    close(fd2[0]);
    close(fd2[1]);
    dup2(fd3[1],STDOUT_FILENO);
    close(fd3[0]);
    close(fd3[1]);
    /* process table P2*/
  }else{
     /***Proceso P3 ***/
     dup2(fd3[0],STDIN_FILENO);
     close(fd3[0]);
     close(fd3[1]);
     dup2(fd2[1],STDOUT_FILENO);
     close(fd2[0]);
     close(fd2[1]);
    /* process table P3*/
  }
 }
...
```

a) Write the content of the file descriptor tables for processes P1, P2 and P3, in the code points marked as /*process table Pi*/.

b) Explain the relationship between P1, P2 and P3 as well as the redirection scheme derived from executing the code.

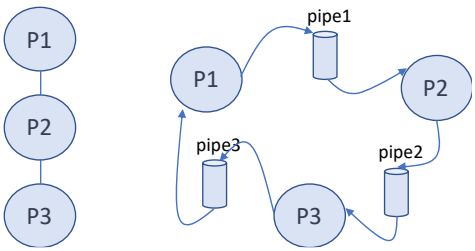
1.

P1	P2	P3

pipe(fd); pipe(fd2);
dup2(fd[1],STDOUT_FILENO);
close(fd[0]); close(fd[1]);
dup2(fd2[0],STDIN_FILENO);
close(fd2[0]); close(fd2[1]);
	pipe(fd); pipe(fd2);
	.... inherited from P0
	dup2(fd[0],STDIN_FILENO);
	close(fd[0]); close(fd[1]);
	pipe(fd3);
	close(fd2[0]); close(fd2[1]);
	dup2(fd3[1],STDOUT_FILENO);
	close(fd3[0]); close(fd3[1]);
		pipe(fd);/*pipe1*/
		pipe(fd2);/*pipe2*/
		.... inherited from P0
		dup2(fd[0],STDIN_FILENO);
		close(fd[0]); close(fd[1]);
		pipe(fd3);
		.... inherited from P1
		dup2(fd3[0],STDIN_FILENO);
		close(fd3[0]);close(fd3[1]);
		dup2(fd2[1],STDOUT_FILENO);
		close(fd2[0]);close(fd2[1]);

P1 | | **0** | **fd2[0]** **1** | **fd[1]** **2** | **stderr**	**P2** | | **0** | **fd[0]** **1** | **fd3[1]** **2** | **stderr**	**P3** | | **0** | **fd3[0]** **1** | **fd2[1]** **2** | **stderr**

2. P1 is parent of P2 and P2 is parent of P3



Diciembre 2020

1. The following listing is the contents of a directory on a POSIX system:

```
drwxr-xr-x   2 user1     grpa              4096 ene  8     2013  .
drwxr-xr-x  11 user1     grpa              4096 ene 10     14:39 ..
-rwsr--r-x   1 user1     grpa           1139706 ene  9     2013  borrar
-rw-------   1 user1     grpa            634310 ene  9     2013  fich
lrwxrwxrwx   1 user1     grpa                 3 ene  9     2013  dat→fich
```

Where borrar is a program that removes a file passed as an argument.

a) Justify if user user2 of the gprb group can delete or not the file fich by executing in that directory the command:

$ ./borrar file

b) Justify if the user user2 of the gprb group can create a file of type link in this directory executing the order:

$ ln -s borrar newborrar

1	a) The borrar executable has the following permissions: `-rwsr--r-x` So, the bit setuid is enabled. This file can be executed by the user1, it is not allowed to other group members, and it is allowed to the rest of the users. So, user2 as member of the other group than user1 can execute borrar, get the identity of user1 and, as consequence, can delete a file using borrar by.
	b) The directory has the following permissions: `drwxr-xr-x` It means that only owner (user1) can write in this directory. So, user2 can not modify the directory by creating a file (regular, link, etc.)

2 The following listing is the contents of a directory on a POSIX system:

```
drwxr-xr-x   2 user1     grpa              4096 ene  8     2013  .
drwxr-xr-x  11 user1     grpa              4096 ene 10     14:39 ..
-rwxr-sr-x   1 user1     grpa           1139706 ene  9     2013  cambia_claves
-rw-------   1 user1     grpa            634310 ene  9     2013  claves_web
-rw----rw-   1 user1     grpa            104157 ene  9     2013  claves_impr
-rw-rw----   1 user1     grpa            634310 ene  9     2013  claves_sala
```

Where **cambia_claves** is a program that allows to edit and modify the content of the keys stored in the data files: **claves_web, claves_impr and claves_sala**. Fill in the table, indicating in case of success which are the permissions that are computed and, in case of error, which is the permission that fails and why.

2

Usuario	Grupo	Orden	¿Funciona?	Observaciones
user3	grpb	`./cambia_clave claves_web`	NO	Can execute but can?t Access to the file (only owner) Euid = user3; eguid= grpa
user2	grpa	`./cambia_clave claves_impr`	NO	User2 can't modifiy the file. (only owner and rest) Euid = user2; eguid= grpa
user2	grpa	`./cambia_clave claves_sala`	YES	Euid = user2; eguid= grpa
user3	grpb	`./cambia_clave claves_sala`	YES	Euid = user3; eguid= grpa

Diciembre-2020

1. A disk with a capacity of 8GB, is formatted with a version of MINIX with sizes different from the standards. The sizes used in the formatting are :

- Block size = 2KBytes
- Zone Size = 2^0 blocks = 1 zone
- Pointers to Zone are 32bits = 4Bytes
- The size of the i-node is 64 bytes (7 direct pointers, 1 indirect, 1 double indirect).
- Each directory entry occupies 32 bytes.
- The boot block and the superblock requires 1 block each
- When formatting, space has been reserved in the header for 4096 i-nodes
- The scheme of the different elements of the disk is as follows

Boot	Super block	Node-i bitmap	Zone bitmap	Nodes- i	Data zones

It is requested:

a) Find the number of blocks that each element of the header occupies: i-node bitmap, zones bitmap and i-nodes.

b) Compute the block that corresponds to the first Data Zone as well as the number of Data Zones.

c) Suppose that on this disk there is only one directory, the root directory, which contains 10 regular files,

c1) Indicate the number of data zones occupied by the root directory

c2) Assume in addition that each of the regular files contains information that occupies 50KBytes and indicate in a justified way the number of occupied data zones for this case, take into account both the data and the metadata of the file.

	a) Block size: 2KB. In 1 block there are (2 * 1024) = 2^{11} = 2048 B or (2 * 1024 * 8) = 16384 bits There are 4096 i-nodes. Number of blocks for i-node map: ceiling(16384 / 4096) = 1 block Number of blocks for i-nodes => ceiling (size i-node * n.i-nodes / bytes in block) = ceiling (64 * 4096 / 2048) = 128 blocks Disk size = 8GB = 2^{33}; Number of blocks = 2^{33} / 2^{11} = 2^{22} blocks Number of zones = 2^{22} / 1 = 2^{22} zones Zone bipmap = ceiling (2^{22} / 16384) = 256 blocks Disk structure: 1 + 1 + 1 + 256 + 128 + (2^{22} – (3 + 256 +128)) = 1 + 1 + 1 + 256 + 128 + 4193917
	b) The first data zone will be allocated in block = 1 + 1 + 1 + 256 + 128 = 387 The first block is block 0, so 1st zone block is 387 Number of Data zones = (2^{22} – (3 + 256 +128)) = 4193917
	c1) root directory has 10 regular files. Each directory entry uses 32 bytes. Root directory will have 10 entries + 2 (. and ..). Number of data blocks = ceiling (12 * 32 / 2048) = 1 block = 1 zone
	c2) Each file contains 50KB, the number of blocks for each is ceiling (50K / 2K) = 25 areas Each i-node has to point to 50 blocks => 7 direct pointers + 1 indirect pointer (uses 18 of 64) Each file requires 25 data areas for data and 1 data area for indirect pointers => 26 data areas 10 regular files require 26 * 10 = 260 data areas. The i-node of the root directory requires 12 entries => 1 data area (uses 12 of 64) Total = 261 data areas (data blocks).