PROBLEM 1

What are the main functions of an operating system?

SOLUTION

The operating system is a program that is responsible for covering the hardware to facilitate the use of computers, which it carries out by performing three main functions:

1. Management of system resources.
2. Execution of services for programs.
3. Execution of programs by users.

1. **Hardware resource management**. This function is performed by the kernel of the operating system, which must take care of allocating computer resources to running programs, as well as preventing some programs from accessing the resources of others and keeping an accounting of resources consumed. for each program. Its main task is to manage the different processes that may be running on the machine simultaneously. The most important thing is to know, through the appropriate structures, which resources are free and which are occupied. In view of this, it must assign certain resources to the running processes, depending on the priority of each one, and recover, for its use, those that have been left free because a process has ended. The operating system must also ensure that the processes do not access the resources assigned to others, thus ensuring the security of the system. Finally, the system must measure the amount of resources that are being used at all times, which is frequently used for monitoring.

2. **Execution of services for the programs**. This function is carried out in the layer called services or system calls that allows the execution of the programs to be more comfortable. These calls take care of:

a. Run and destroy processes.

b. Facilitate input / output and operations on files.

c. Detect and deal with errors detected by the hardware.

The operating system mainly offers four types of services, or system calls; services to launch programs and turn them into processes, stop them and abort them. Due to the complexity of input / output operations with peripherals, the S.O. provides the necessary services to facilitate communication between processes and external devices (open, close, read, etc ...). To work with files, the operating system offers services similar to those of I / O, but with a higher level of abstraction. In addition, the system is in charge of treating possible errors that the hardware may detect.

3. **Execution of user commands**. The S.O. It is also responsible for the interaction of the computer with the users, this is carried out through shells. Shells can be command line, such as UNIX bash, or graphical such as Windows. The command line shells force the user to learn all the commands, while the graphs are much more intuitive. The shell behaves like an infinite loop by repeating the following sequence:

a. Wait for an order from the user.

b. Once an order is received, it analyzes it and, if it is correct, executes it.

c. Once the order is completed, it returns to waiting.

PROBLEM 2

Write a UNIX C function that copies one file to another. The original name and the destination are parameters of the function. The destination file must have protections that allow it to be read by any user, but only to be modified by the owner.

The error conditions to be logged are as follows:

- The file cannot be opened

- The destination file cannot be created

- The copy process did not finish well

Solution

Solution: copiarfichero.c

```
#include <fcntl.h>
#define BUFSIZE    512         /*tamaño de las tiras a leer*/
#define PERM       0644        /*permisos para creación de archivo*/
int copyfile (char *name1, char *name2)   /*copia name1 a name2*/
{
    int infile, outfile, nread;
    char buffer [BUFSIZE]
    if ( (infile = open(name1, O_RDONLY) ) < 0 )
        return(-1); /* no puede abrirse */
    if ( (outfile = creat (name2, PERM) ) < 0 ) {
        close (infile);
        return (-2); /* no puede crearse */
        }
    /*lectura de name1 BUFSIZE caracteres de una vez*/
    while( (nread = read(infile, buffer, BUFSIZE) ) > 0 )
    {
        /*escribir el buffer al archivo de salida*/
        if (write (outfile, buffer, nread) < nread)
        {
            close(infile);
            close(outfile);
            return (-3);           /* error al escribir */
        }
    }
    if (nread==-1)
        return (-4);    /* error al leer */
    close(infile);
    close(outfile);
    return(0);
}
```

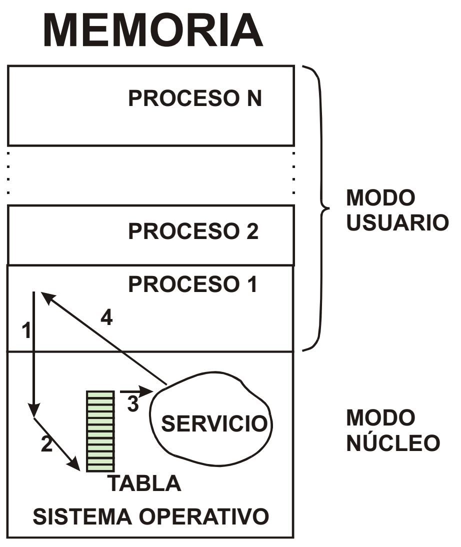
PROBLEM 3

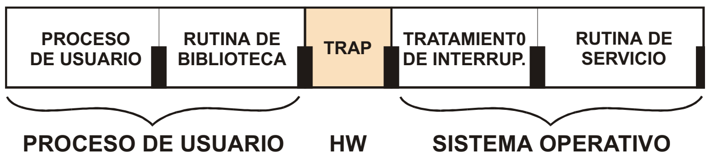
¿How do a program request a system call?

Solution

It is requested through an interrupt mechanism. When requested by a running process, it uses a TRAP instruction that generates an interrupt. As shown in Figure, the TRAP interrupt handling routine uses an internal table of the S.O. to determine which routine to activate depending on which call is requested.

When programmed in a high-level language, the request for services to the operating system is made through a call to a specific function, which is responsible for generating the system call and the corresponding trap.





PROBLEM 4

Which of the following hardware mechanisms is not a requirement to build a multiprogrammed operating system with protection between users? Give reasons for your answer.

A.- Virtual memory.

B.- Memory protection.

C.- I / O instructions that can only be executed in kernel mode.

D.- 2 modes of operation: core and user.

Solution

The correct answer is the first: Virtual Memory, since although it is necessary to have a large amount of main memory, this does not make it necessary to use that service.

On the other hand, the remaining options are necessary. Memory protection is necessary when we want to protect the memory image of a process against the intrusion of another, which may be from a different user. The execution of I / O instructions in kernel mode is necessary for several reasons, one of them is that we need these processes to be executed by the OS so that while they are being carried out, it can launch another process, and thus make the most of the execution time. Finally, the two modes of operation are necessary for safety, since when several users work, as well as various processes, if there was only one execution mode, a process could execute any instruction on any element of the computer with the consequences in terms of security that this would imply.

PROBLEM 5

Write a C program that creates a file, writes something, jumps 100 bytes further from the end of the file, and writes something. Does it end correctly?

Solution

Solución: writefile.c

```
#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>



int main(void)

{

  int fd;

  char buf1[]="abc";

  char buf2[]="ABC";


  if ( (fd=creat("file_hole",0666))<0)

  {

    perror("error creating the file");

    exit(1);

  }
  if ( write(fd,buf1,3)< 0) {
    perror("write error");
    exit(1);
  }

  if( lseek(fd,100,SEEK_CUR) < 0) {
     perror("seek error");
     exit(1);
  }

  if ( write(fd,buf2,3) <0) {
    perror("write error");
    exit(1);
  }



  if (close(fd)<0){
    perror("close error");
    exit(1);
  }

  return 0;
}
```

PROBLEM 6

Write a C program that opens a file, whose name the student can give, checks that it exists and reads all its content and displays it on the screen.

Solution

Solución: readfile.c

```
#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>


#define N 1


int main()

{

  int fd,n;

  char buf[N];


  /* Open an existing file read only */


  if ((fd=open("file_read",O_RDONLY,0666))<0) {

      perror("File does not exist\n");

      exit(1);

  }




  while ((n=read(fd,buf,N))>0)

    printf("Read from file character %c \n",buf[0]);
```

```
 if (n<0) {
    perror("Read error occured");
    return -1;
  }
  else
    close(fd);
    return 0;
}
```

PROBLEM 7

Write a program in C that for a number of students, which is requested by keyboard, asks for each one of them: name and 3 notes (which can have decimals). Once the list of students has been entered, it calculates the average of the three grades for each student and displays the name and average grade of the student on the screen.

Solution

Solución: ficheronotas.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>


#define NUMALUMNOS 10

#define NUMNOTAS 3


struct Alumno {
  char nombre[20];

  float notas[NUMNOTAS]; };


typedef struct Alumno tAlumno;


void pedirAlumno(tAlumno *alumno);

void escribirAlumno(tAlumno alumno, int fd);


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

  tAlumno clase[NUMALUMNOS];

  tAlumno Al;

  float *media;

  int i, numAl, fichid;


  printf("Numero de alumnos (max 10) = ");

  scanf("%d",&numAl); getchar();

  printf("Numero de alumnos del grupo = %d\n", numAl);


  fichid = open(argv[1], O_RDWR|O_CREAT, 0660);


  for (i=0; i< numAl; i++) {

     pedirAlumno (&Al);

     escribirAlumno (Al, fichid);

  }


  close(fichid);
```

```
}

void pedirAlumno(tAlumno *alumno){

  int j;


   printf ("Nombre del alumno: ");

   fgets (alumno->nombre,20,stdin);

   alumno->nombre[strlen(alumno->nombre)-1]='\0';  //elimino el
salto de lÌnea

   getchar();


   printf("\n");

   for (j=0; j<NUMNOTAS;j++) {

      printf ("Dar la nota %d del alumno %s(decimales con
.):",j,alumno->nombre );

      scanf ("%f", &(alumno->notas[j]) ); getchar();

    }

}

void escribirAlumno(tAlumno alumno, int fd){

  char linea[80];

  sprintf (linea, "%s %4.2f %4.2f %4.2f \n", alumno.nombre,
alumno.notas[0], alumno.notas[1], alumno.notas[2]);

  write (fd, linea, strlen(linea)) ;

}
```

Problema 6

Ejemplo sencillo del uso de `fork`, donde un proceso crea una copia y a continuación se hace un execl para hacer un comando "ls". Ejecútelo y vea la salida. ¿Qué ocurre?

Solución

Solución: fork-exec.c

```
main()
```

```
{
        int valor_regr=0;

        printf("Bifurcando el proceso\n");

        valor_regr=fork();

        printf("El id del proceso es %d y el valor devuelto es %d\n",
                getpid(), valor_regr);

        execl("/bin/ls","ls","-l",0);

        printf("Esta linea no es impresa\n");
}
```