SISTEMAS OPERATIVOS I
PRÁCTICA 3. EXÁMENES

Examen 1.

ESPECIFICACIÓN: Utilizando las llamadas al sistema UNIX, construya un programa C estándar que concatene el contenido de una lista de archivos en un archivo de destino. La lectura y escritura de los archivos se deberá realizar con un buffer de 4096 bytes. Si en la lista de archivos o en el archivo de destino se indica el nombre – se entenderá que se debe leer y/o escribir de la entrada estándar y/o salida estándar. La sintaxis del programa sera:

`concatenar [<lista de archivos>]+ <archivo destino>`

EJECUCIÓN: Anote en la sección **RESULTADO** la suma MD5 del contenido del archivo de destino salida.txt correspondiente al resultado de ejecutar el programa con los siguientes argumentos concatenar /var/test/num* salida.txt y escriba luego el listado del programa en la sección denominada **LISTADO**:(si es necesario continúe al dorso).

```
 #include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#define MAXBUFFER 4096

void concatenar(char *origen,  char *destino, int flag){

  int fd_entrada, fd_salida;
  char buffer[MAXBUFFER];
  struct stat s;
  int numLeidos = 0;
  int numLectura;

  /*Abrimos el archivo de entrada*/
  if((fd_entrada = open(origen, O_RDONLY)) == -1){
    fprintf(stderr, "Error al abrir el archivo\n");
    exit(1);
  }

  /*Abrimos el archivo de salida*/
  if(destino != NULL){
    if((fd_salida = open(destino, O_WRONLY|O_RDONLY, 0664)) == -1){
      fprintf(stderr, "Error de apertura de %s\n", destino);
      exit(1);
    }
  }
  /*Obtenemos los atributos del archivo de entrada para poder asÃ
  sacar su tamaÃ±o y manejar el buffer de lectura */
  if(fstat(fd_entrada, &s) == -1){
    fprintf(stderr, "Error al obtener atributos\n");
    exit(1);
  }

  /*Bucle de control de lectura/escritura*/
  while(numLeidos < s.st_size){
    /*Calculo del numero de bytes que hay que leer */
    if((s.st_size - numLeidos) >= MAXBUFFER) numLectura = MAXBUFFER;
    else numLectura = s.st_size - numLeidos;

    /*Posicionamiento en el archivo de salida, nos colocamos al final*/
    if(destino != NULL){
      if((lseek(fd_salida,SEEK_SET,SEEK_END)) == -1){
```


Julio Alberto López Gómez

```
        fprintf(stderr, "Error al posicionar\n");
        exit(1);
      }
    }

    /*Lectura/Escritura*/
    if(read(fd_entrada, buffer, numLectura) != numLectura){
      fprintf(stderr, "Error al leer la entrada\n");
      exit(1);
    }

    if(flag == 0){

      if(write(fd_salida, buffer, numLectura) != numLectura){
        fprintf(stderr, "Error al escribir la entrada\n");
        exit(1);
      }
    }

    if(flag == 1){
      if(write(1, buffer, numLectura) != numLectura){
         fprintf(stderr, "Error de escritura\n");
         exit(1);
      }
    }

    numLeidos += numLectura;

  }

  close(fd_entrada); close(fd_salida);
}

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

  int i;
  int flag = 0;

  if(argc < 3){
    fprintf(stderr, "Error en la linea de ordenes. Faltan comandos.\n");
    exit(1);
  }

  if((argv)[1][0] == '-'){
    flag = 1;
    for(i=1; i<(argc--); i++) concatenar(argv[i+1], NULL, flag);
  }

  if((argv)[argc-1][0] == '-'){
    flag = 1;
    for(i=1; i<argc-1; i++) concatenar(argv[i], NULL, flag);

  }else for(i=1; i<argc-1; i++) concatenar(argv[i], argv[argc-1], flag);

  exit(EXIT_SUCCESS);
}
```

Examen 2.

ESPECIFICACIÓN: Utilizando las llamadas al sistema UNIX, construya un programa C estándar que cambie el nombre de cada archivo de la línea de órdenes por otro nombre que se la concatenación del nombre del archivo original, el carácter . y el sufijo indicado en la línea de comandos. Si no se indica un sufijo en la línea de órdenes se utilizará el sufijo `old`. La sintaxis del programa será:

`cambiar [-<sufijo>] <nombre>`[x]

EJECUCIÓN: Desde el directorio home del usuario alumno copie el contenido del directorio `/var/test` el subdirectorio `test` y ejecute el programa con los siguientes argumentos `cambiar –dos test/rfc*` comprobando que realiza la funcionalidad requerida y escriba luego el listado del programa en la sección denominada **LISTADO:**(Si es necesario continúe al dorso).

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PATH_MAX 4096

int renombrar(char *viejo, char *ext){
  char nuevo[PATH_MAX+1];
  sprintf(nuevo, "%s.%s", viejo, ext);

  if(strlen(viejo) + strlen(ext)+2 > sizeof(nuevo)){
    fprintf(stderr, "Error: El nombre supera el maximo");
    exit(EXIT_FAILURE);
  }

  if(rename(viejo, nuevo) == -1){
    fprintf(stderr, "Error al renombrar el archivo");
    exit(EXIT_FAILURE);
  }
  return 0;
}

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

  char *ext;

  if(argv[1][0] == '-'){
    ext = ++argv[1];
    --argc;
    ++argv;
  }else{
    ext = "old";
  }

  while((--argc) > 0){
    renombrar(*++argv, ext);
  }

  exit(EXIT_SUCCESS);

}
```

Examen 3.

ESPECIFICACION: Utilizando las llamadas al sistema UNIX, construya un programa C estándar que cree un enlace duro a cada archivo de la línea de órdenes con un nombre para el nuevo enlace duro que sea la concatenación del nombre del archivo original, el carácter . y el sufijo indicado en la línea de comandos. Si no se indica un sufijo en la línea de órdenes se utilizará el sufijo `hard`. La sintaxis del programa será:

`duro[-<sufijo>] [nombre]*`

EJECUCIÓN: Desde el directorio *home* del usuario `alumno` copie el contenido del directorio `/var/test` en el subdirectorio test y ejecute el programa con los siguientes argumentos `duro -hd test/r*` comprobando que realiza la funcionalidad requerida y escribe luego el listado del programa en la sección denominada **LISTADO:**(Si es necesario continúe al dorso).

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

#define PATH_MAX 4096

void enlace(char *archivo, char *sufijo){
  char nombre[PATH_MAX+1];
  if((strlen(archivo) + strlen(sufijo)+2) > PATH_MAX){
    fprintf(stderr, "Error: Nombre demasiado largo");
    exit(EXIT_FAILURE);
  }

  sprintf(nombre, "%s.%s", archivo, sufijo);
  link(archivo, nombre);
}

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

  int i;
  if(argc < 2){
    fprintf(stderr, "Numero de argumentos incorrecto");
    exit(EXIT_FAILURE);
  }

  if(argv[1][0] == '-'){
    for(i=2; i<argc; i++){
      enlace(argv[i], argv[1]+1);
    }
  }else{
    for(i=1; i<argc; i++){
      enlace(argv[i], "hard");
    }
  }
  return EXIT_SUCCESS;
}
```

Examen 4.

Programa que permite listar todos los ficheros regulares menores que un tamaño dado.

```c
/*Listar todos los ficheros regulares menores que un tamano dado*/
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define MAXIMO 4096

void listarArchivos(const char *nodo, int size){
  DIR *d;
  struct dirent *entr;
  struct stat s;
  char nuevo[MAXIMO+1]; /* +1 por el caracter nulo*/

  /*se abre el stream del directorio nodo*/
  if((d=opendir(nodo))==NULL){
    fprintf(stderr,"Error al abrir el stream del directorio\n");
    exit(EXIT_FAILURE);
  }
  while((entr=readdir(d))!=NULL){
    /*se omite directorio actual y directorio anterior*/
    if(strcmp(entr->d_name,".")==0 || strcmp(entr->d_name,"..")==0){
```

```c
    /*se omite directorio actual y directorio anterior*/
    if(strcmp(entr->d_name,".")==0 || strcmp(entr->d_name,"..")==0){
      continue;
    }
    /*se comprueba que no se excede el tamano maximo de el path*/
    if(strlen(nodo)+strlen(entr->d_name)+2>sizeof(nuevo)){
      continue;
    }
    sprintf(nuevo,"%s/%s",nodo,entr->d_name);
    /*se obtienen los atributos de los nodos del directorio*/
    if(stat(nuevo,&s)==-1){
      continue;
    }
    /*se comprueba que es un fichero regular*/
    if(S_ISREG(s.st_mode)&&((int)s.st_size)<size)&&(*(entr->d_name)!='.')){
      fprintf(stdout,"%s\tTotal Size: %d Bytes\n",nuevo,(int)s.st_size);
    }
    else if(S_ISDIR(s.st_mode)){
      listarArchivos(nuevo,size); /*llamada recursiva para listar
                                    los archivos de los subdirectorios*/
    }
  }
  closedir(d);
} /*fin programa*/
```

```c
int main(int argc, char *argv[]){
  int tamano=0;
  /*se revisa si la linea de ordenes es correcta*/
  if(argc<3){
    fprintf(stderr,"Linea de ordenes incorrecta\n");
    exit(EXIT_FAILURE);
  }
  /*se revisa la opcion del tamano*/
  if(argv[1][0]=='-'){
    tamano=atoi(++argv[1]);
    --argc;
    ++argv;
  }
  else{
    fprintf(stderr,"No se ha especificado el tamano\n");
    exit(EXIT_FAILURE);
  }
  /*se realiza el procesamiento*/
  while(--argc>0){
    listarArchivos(*++argv,tamano);
    printf("----------------------------------\n");
  }
  return EXIT_SUCCESS;
}
```

Examen 5.

ESPECIFICACIÓN: Utilizando las llamadas al sistema UNIX, construya un programa C estándar que imprima en la salida estándar los nombres de los archivos regulares ocultos de un directorio o de sus subdirectorios (incluyendo aquellos subdirectorios que no se puedan acceder) y tengan un tamaño menor que uno dado. La salida consistirá en líneas con los nombres de los archivos regulares ocultos y sus respectivos tamaños (un archivo por línea) y la sintaxis del programa será:

`ocultos <numero> <directorio>`

EJECUCIÓN: Anote en la sección **RESULTADO** el resultado de ejecutar el programa con los siguientes argumentos `ocultos 1000 /etc` y escriba luego el listado del programa en la sección denominada **LISTADO:**(si es necesario continúe al dorso).

```
/* Examen practica 3. Programa que muestre los archivos ocultos de un
directorio dado y que tengan un tamaÃ±p menor que el indicado, para ejecutarlo seria:
ocultos <numero> <directorio> */

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PATH_MAX 4096

void buscarOculto(char *dir, int tam){

	DIR *d; // Manejador del directorio
	struct stat st;
	struct dirent *sd:
	char nomOcu[PATH_MAX]; // Directorio + nombre del nodo
	char nombre[PATH_MAX];

	/* Apertura del directorio */

	if((d = opendir(dir)) == NULL){

		fprintf(stderr, "Error en el direcotio %s\n", dir);
		exit(EXIT_FAILURE);
	}

	/* Bucle de lectura de nodos del directorio */

	while((sd = readdir(d)) != NULL){
		/* Se omite el directorio actual y el padre */
		if(strcmp(sd->d_name, ".") == 0 || strcmp(sd->d_name, "..") == 0){
			continue;
		}

		/* Ruta absoluta del fichero */
		sprintf(nomOcu, "%s%s", dir, sd->d_name);

		/* Copiamos el nombre del fichero */
		strcpy(nombre, sd->d_name);

		if(stat(nomOcu, &st) == -1){
			fprintf(stderr, "Error en %s\n", nomOcu);
			exit(EXIT_FAILURE);
		}
```

```
			if(S_ISREG(st.st_mode) && st.st_size < tam){
				if(nombre[0] == '.'){
					printf("%s\t%1d\n", nomOcu, st.st_size);
				}
			}

			else if(S_ISDIR(st.st_mode)){
				buscarOcultos(nomOcu, tam);
			}
		}

		closedir(d);
}

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

		if(argc != 3){
			fprintf(stderr, "Error en la linea de ordenes\n");
			return EXIT_FAILURE;
		}
		buscarOcultos(argv[2], atoi(argv[1]));
		return EXIT_SUCCESS;
}
```

Examen 6.

ESPECIFICACIÓN: Utilizando las llamadas al sistema UNIX, construya un programa C estándar que imprima en la salida estándar los nombres de los directorios de la línea de órdenes cuya suma total de los archivos regulares que contenga ese directorio (y de sus subdirectorios) sea mayor que uno dado. La salida consistirá en líneas con los nombres de los directorios y los respectivos tamaños totales (un archivo por línea) y la sintaxis del programa será:

`tamaño -<número>[<directorio>]+`

EJECUCIÓN: Anote en la sección **RESULTADO** el resultado de ejecutar el programa con los siguientes argumentos `tamaño – 1000000 /var/log/packages /var/test /sbin` y escriba luego el listado del programa en la sección denominada **LISTADO**:(Si es necesario continúe al dorso).

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <stdlib.h>

#define PATH_MAX 4096

void SumaRecursiva(const char *nodo, int *suma){

  DIR *d;
  struct dirent *entr;
  struct stat s;
  char nuevo[PATH_MAX+1];
  /*se abre el directorio*/
  if((d = opendir(nodo)) == NULL){
    fprintf(stderr, "Error al abrir el archivo");
    exit(EXIT_FAILURE);
  }
  /*se van leyendo las entradas para sumar los tamaños*/
  while((entr = readdir(d)) != NULL){
    if(strcmp(entr->d_name, ".") == 0 || strcmp(entr->d_name, "..") == 0){
      continue;
    }

    if(strlen(nodo) + strlen(entr->d_name)+2 > sizeof(nuevo)){
      fprintf(stderr, "Nombre demasiado largo");
      continue;
    }
/*Creamos el nuevo nodo*/
    sprintf(nuevo, "%s/%s", nodo, entr->d_name);
    if(stat(nuevo, &s) == -1){
      fprintf(stderr, "Error al obtener atributos del archivo %s", nuevo);
      exit(EXIT_FAILURE);
    }

    /*Se comprueba si es un archivo regular o un directorio*/
    if(S_ISREG(s.st_mode)){ /*archivo regular*/
      *suma = *suma + (int)s.st_size;
    }

    if(S_ISDIR(s.st_mode)){ /*directorio*/
      *suma = *suma + (int)s.st_size;
```

```
      SumaRecursiva(nuevo, suma);
    }
  }
  closedir(d);
}

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

  int suma = 0;/*para contar el tamaño maximo del directorio*/
  int tamano = 0;
  /*comprobación de la línea de órdenes*/
  if(argc < 3){
    fprintf(stderr, "Error en la lÃnea de ordenes");
    exit(EXIT_FAILURE);
  }
  /*se tratan las opciones*/
  if(argv[1][0] == '-'){
    tamano = atoi(*++argv[1]);
    ++argv;
    --argc;
  }else{
    fprintf(stderr, "No hay opcion de tamano");
    exit(EXIT_FAILURE);
  }
 /*se procesa la orden*/
  while(--argc > 0){
    SumaRecursiva(*++argv, &suma);
    if(suma > tamano){
      printf("%s\tTotal Size: %d Bytes\n", *argv, suma);
    }
  }
  exit(EXIT_SUCCESS);
}
```