Evaluación del indicador AE2.CD1.I1

Oscar Herrera Alcántara	1151018	Sistemas Operativos
AE2	Aplicar fundamentos de ciencias básicas e ingeniería para analizar y desarrollar procesos de diseño de ingeniería que resulten en proyectos que cumplen las necesidades especificadas.	
CD1	El alumno utiliza conceptos fundamentales de ciencias básicas e ingeniería en la solución de problemas.	
I1	75% de los alumnos utiliza los conceptos fundamentales de ingeniería en la solución de problemas.	
Calificación	100	
Actividad: Práctica con programa y reporte sobre la comunicación de tres procesos para intercambiar información de cambios en un árbol de directorios de GNU/Linux.		

REPORTE

AVANCE DE PROYECTO FINAL

*Codifique un programa en C que comunica **tres** procesos a efecto de:*

1. *Solicitar al usuario un archivo o directorio al cual se realicen operaciones de creación, eliminación, renombrar o consultar dentro de una carpeta dada del directorio home*
2. *Ejecutar la instrucción de crear, eliminar, renombrar un archivo o directorio*
3. *Monitorear los cambios en el sistema de archivos*

Genere un reporte de la actividad realizada y responda las siguientes preguntas:

1. **¿Cuántos procesos creó con la llamada fork?**
   Se crearon dos subprocesos del proceso padre, esto es, tres procesos se ejecutan.

2. **¿Cuáles mecanismos de intercomunicación de procesos usó?**
   Para las CRUD del proceso del hijo 2 se utilizó una tubería, mientras que el proceso inotify del hijo 1 manda la información mediante señales.

3. **Incluya el código fuente del programa solución e indique claramente las llamadas al sistema utilizadas.**

```
//Programa A

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

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/inotify.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define MAX 256
#define MAX_EVENTS 1024 /*Max. number of events to process at one go*/
#define LEN_NAME 1024 /*Assuming length of the filename won't exceed 16
bytes*/
#define EVENT_SIZE  ( sizeof (struct inotify_event) ) /*size of one
event*/
#define BUF_LEN     ( MAX_EVENTS * ( EVENT_SIZE + LEN_NAME )) /*buffer to
store the data of events*/

#define PORT 3550
#define MAXDATASIZE 100

void conectar(char* direccion){
      int fd, numbytes;
      char buf[MAXDATASIZE];
      struct hostent *he;
      struct sockaddr_in server;

      if ((he=gethostbyname(direccion))==NULL){
            /* llamada a gethostbyname() */
            printf("gethostbyname() error\n");
                  exit(-1);
      }

      if ((fd=socket(AF_INET, SOCK_STREAM, 0))==-1){
                  /* llamada a socket() */
                  printf("socket() error\n");
                  exit(-1);
      }

      server.sin_family = AF_INET;
      server.sin_port = htons(PORT);
      /* htons() es necesaria nuevamente ;-o */
      server.sin_addr = *((struct in_addr *)he->h_addr);
      /*he->h_addr pasa la informaciÃ³n de ``*he'' a "h_addr" */
      bzero(&(server.sin_zero),8);

      if(connect(fd, (struct sockaddr *)&server,sizeof(struct
sockaddr))==-1){
                  /* llamada a connect() */
                  printf("connect() error\n");
                  exit(-1);
      }

      if ((numbytes=recv(fd,buf,MAXDATASIZE,0)) == -1){
                  /* llamada a recv() */
                  printf("Error en recv() \n");
```

```
                exit(-1);
        }

        buf[numbytes]='\0';

        printf("Mensaje del Servidor: %s\n",buf);
        /* muestra el mensaje de bienvenida del servidor =) */

        close(fd);    /* cerramos fd =) */
}

char menu(){
        char opcion;
        printf("\n*** Bienvenido al UAM-DROPBOX ***\n");
        printf("Eliga la opcion que desea realizar\n");
        printf("a) Crear Archivo\n");
        printf("b) Modificar Archivo\n");
        printf("c) Eliminar Archivo\n");
        printf("d) Crear Directorio\n");
        printf("e) Eliminar Directorio\n");
        printf("f) Salir\n");
        scanf("%s", &opcion);

        return opcion;
}

void crearArchivo(char* nombre){
        int arc;
        arc = creat(nombre, 0644);
        if (arc==-1){
                perror("Error al crear el fichero:");
                exit(1);
        }
        close(arc);
}

void crearDirectorio(char* nombre){
        char com[MAX];
        strcpy(com, "mkdir ");
        strcat(com, nombre);
        printf("%s\n", com);
        system(com);
}

void modificarArchivo(char* nombre){
        char com[MAX];
        strcpy(com,"touch -a -c ");
        strcat(com,nombre);
        //printf("%s\n", com);
        system(com);
}

void eliminarArchivo(char* nombre){
        int arc;
        arc = remove(nombre);
        if(arc != 0){
                printf("Error al eliminar el archivo");
```

```
	}
}

void eliminarDirectorio(char* nombre){
	char com[MAX];
	strcpy(com, "rmdir ");
	strcat(com, nombre);
	printf("%s\n", com);
	system(com);
}

void watchdog(int fd){
	char buffer[BUF_LEN];
	int length, i = 0;

	length = read( fd, buffer, BUF_LEN );
	if (length < 0) {
		perror( "read" );
	}

	while (i < length) {
		struct inotify_event *event = ( struct inotify_event * )
&buffer[ i ];
		if (event->len) {
		if ( event->mask & IN_CREATE) {
		    if (event->mask & IN_ISDIR)
			printf( "El directorio %s fue creado.\n", event->name
);
		    else
			printf( "El archivo %s fue creado.\n", event->name);
		}

		if ( event->mask & IN_MODIFY) {
		    if (event->mask & IN_ISDIR)
			printf( "El directorio %s fue modificado.\n", event-
>name);
		    else
			printf( "El archivo %s fue modificado.\n", event-
>name);
		}

		if ( event->mask & IN_DELETE) {
		    if (event->mask & IN_ISDIR)
			printf( "El directorio %s fue eliminado.\n", event-
>name);
		    else
			printf( "El archivo %s fue eliminado.\n", event-
>name);
		}
		i += EVENT_SIZE + event->len;
	    }
	}
}

int main(int argc, char** argv){
	int PROC = 2;		// PROC guarda la cantidad de procesos a
crear
```

```
	int status;
	int procNum;			// procNum almacena el numero del
proceso
	int readbytes;
	char mensaje[MAX];
	char nombre[MAX];
	char arr[MAX];
	int tuberiaA[2];
	pid_t pid;
	pid_t pid1;
	pid_t pid2;

	int bytel;

	/*Entablar conexion con el servidor*/
	/*char direccion[MAX];
	printf("Ingrese su direccion IP: \n");
	scanf("%s", direccion);
	conectar(direccion);
	*/
	pipe(tuberiaA);

	pid1 = fork();	// se hace fork()

	if(pid1 == 0){	//proceso hijo 1
			char directorio[MAX];
			strcpy(directorio,
"/media/luis/JAPETH_UAM/S.O/Proyecto");
			printf("Soy el proceso hijo1 %d = %d con proceso padre
%d \n", getpid(), pid2, getppid());

			int wd, fd;
			fd = inotify_init1(0);
			if ( fd < 0 ) {
				perror( "No se pudo inicializar inotify");
			}

			wd = inotify_add_watch(fd, directorio, IN_CREATE |
IN_MODIFY | IN_DELETE);
			if (wd == -1) { /*Llamada mediante señales*/
				printf("No se pudo observar a: %s\n",directorio);
			}else{
				printf("Observando:: %s\n",directorio);
			}

			/* do it forever*/
			while(1) {
				watchdog(fd);
			}

			/* Clean up*/
			inotify_rm_watch(fd, wd);
			close( fd );
			exit(0);

	}else{			//proceso padre
```

```
            pid2 = fork();
            if(pid2 == 0){    //proceso hijo2
                  char sigue = 's';
                  char op;
                  printf("Soy el proceso hijo2 con pid= %d y con proceso
padre ppid= %d\n", getpid(),getppid());
                  do{
                  op = menu();
                  if(op == 'f'){
                        exit(0);
                  }
                  mensaje[0] = op;
                  write(tuberiaA[1], mensaje, strlen(mensaje));/*Llamada
atreves de una tubería*/
                  printf("Ingrese el nombre del archivo\n");
                  scanf("%s", nombre);
                  write(tuberiaA[1], nombre,strlen(nombre));
                  printf("Desea realizar otra operacion? [s/n]\n");
                  scanf("%s", arr);
                  write(tuberiaA[1], arr, strlen(arr));
                  sigue = arr[0];
                  }while(sigue != 'n');

                  exit(0);

            }else{
                  printf("Soy el proceso padre %d  con proceso inicial %d
y un hijo2= %d\n", getpid(), getppid(), pid2);

                  char next = 's';
                  char opcion;
                  do{ /*Recibiendo el mensaje de la tubería*/
                  bytel = read(tuberiaA[0], &mensaje, MAX);
                  printf("Bytes leidos: %d\n", bytel);
                  printf("Mensaje recibido: %s\n", mensaje);
                  opcion = mensaje[0];

                  bytel = read(tuberiaA[0], &nombre, MAX);
                  printf("Bytes leidos: %d\n", bytel);
                  printf("Mensaje recibido: %s\n", nombre);

                  bytel = read(tuberiaA[0], &arr, MAX);
                  printf("Bytes leidos: %d\n", bytel);
                  printf("Mensaje recibido: %s\n", arr);

                  next = arr[0];

                  switch(opcion){
                        case 'a':
                              crearArchivo(nombre);
                              break;
                        case 'b':
                              modificarArchivo(nombre);
                              break;
                        case 'c':
                              eliminarArchivo(nombre);
                              break;
```

```
                        case 'd':
                              crearDirectorio(nombre);
                              break;
                        case 'e':
                              eliminarDirectorio(nombre);
                              break;
                        default:
                              break;
                  }
                  }while(next != 'n');
            }

      }

      for (int i=0; i<2; i++){            // esperar a que todos los hijos
terminen
                  if ((pid = wait(NULL)) >= 0){
                      printf("\t\tEl proceso padre con pid=%d espera... y
ha terminado el proceso %d \n", getpid(), pid);
                  }
      }

    return 0;           // Fin del programa
}
```

4. Haga un diagrama ilustrativo de la arquitectura del programa implementado.