



Universidad Nacional Autónoma de México

Facultad de Ingeniería



División de Ingeniería Eléctrica (DIE)

Arquitectura Cliente Servidor (2946)

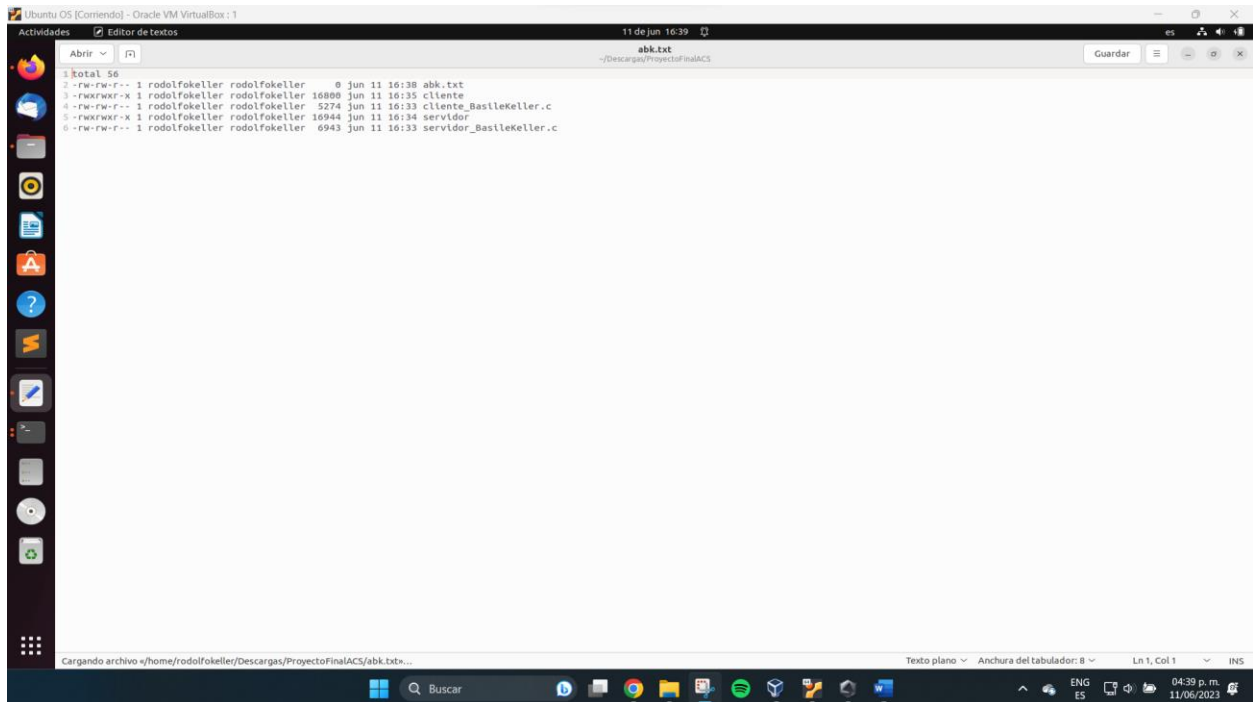
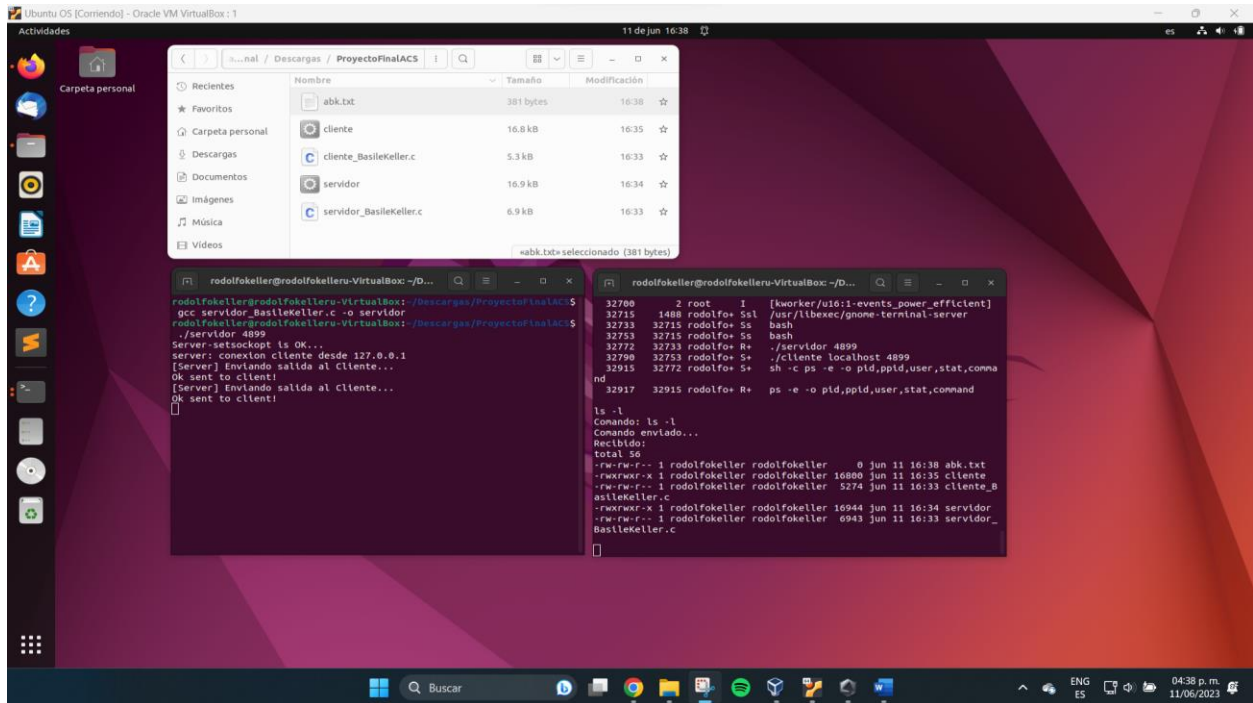
Semestre 2023-2

Proyecto Final

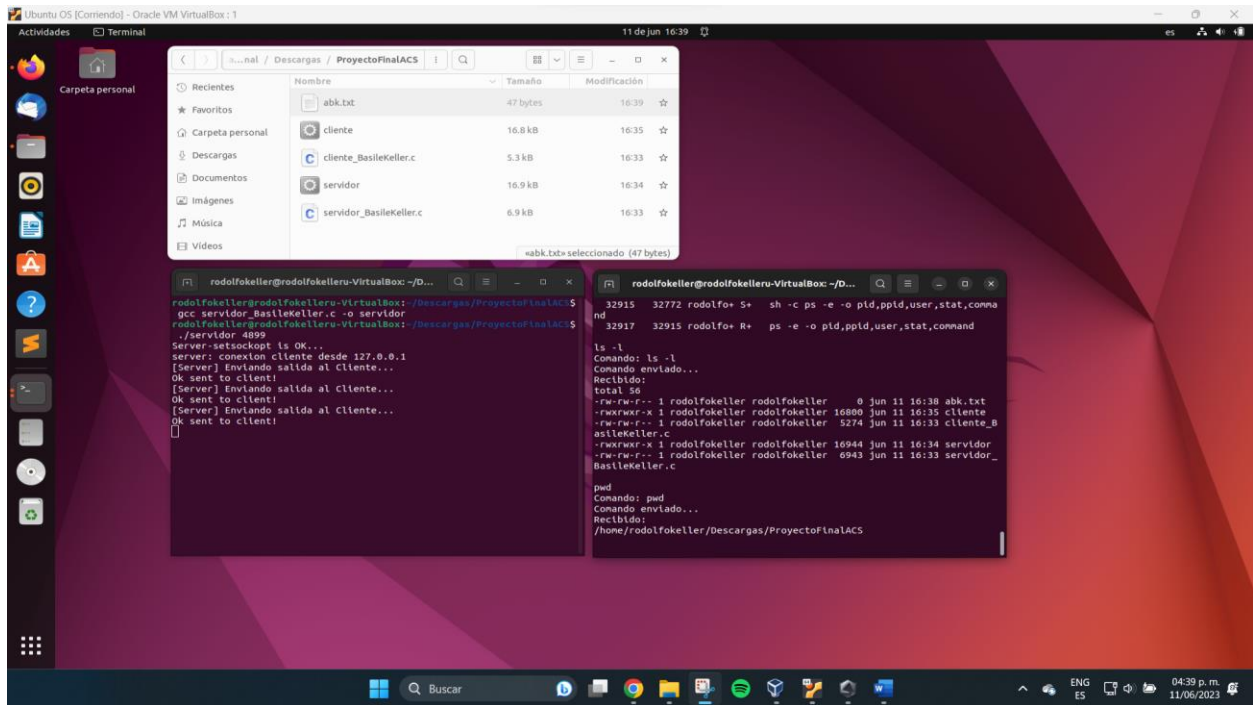
Nombre de los Alumnos:	Basile Álvarez Andrés José (316617187)		
	Keller Ascencio Rodolfo Andrés (316515746)		
Grupo:	1	Profesor: Ing. Carlos Alberto Román Zamitiz	Calificación:
Semestre:	2023-2	Fecha de Entrega: 12. Junio. 2023	

Pruebas de implementación proyecto final:

Comando “ls -l”:



Comando “pwd”:



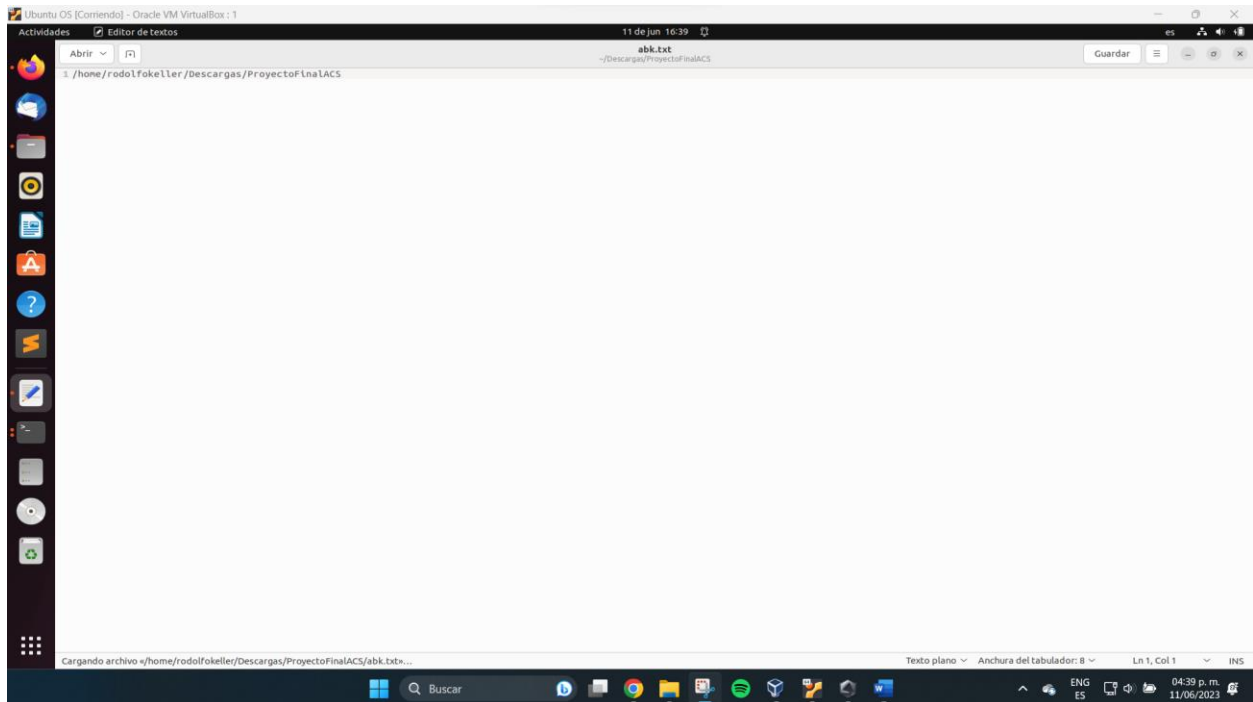
The screenshot shows an Ubuntu VM desktop with a purple background. A file manager window is open at `/Descargas/ProyectoFinalACS`, displaying a list of files:

Nombre	Tamaño	Modificación
abk.txt	47 bytes	16:39
cliente	16.8 kB	16:35
cliente_Basilekeller.c	5.3 kB	16:33
servidor	16.9 kB	16:34
servidor_Basilekeller.c	6.9 kB	16:33

Below the file manager, two terminal windows are open. The left terminal shows the execution of `gcc` and `./servidor`, with output indicating a successful connection to a client at `127.0.0.1`. The right terminal shows the execution of `ls -l`, displaying the permissions and details of the files in the current directory.

```
rodolfo@rodolfo-VirtualBox: ~/D...  
rodolfo@rodolfo-VirtualBox: ~/Descargas/ProyectoFinalACS$ gcc servidor_Basilekeller.c -o servidor  
rodolfo@rodolfo-VirtualBox: ~/Descargas/ProyectoFinalACS$ ./servidor 4899  
Server: setsockopt is OK...  
Server: conexión cliente desde 127.0.0.1  
[Server] Enviando salida al Cliente...  
Ok sent to client!  
[Server] Enviando salida al Cliente...  
Ok sent to client!  
[Server] Enviando salida al Cliente...  
Ok sent to client!  
[Server] Enviando salida al Cliente...  
Ok sent to client!
```

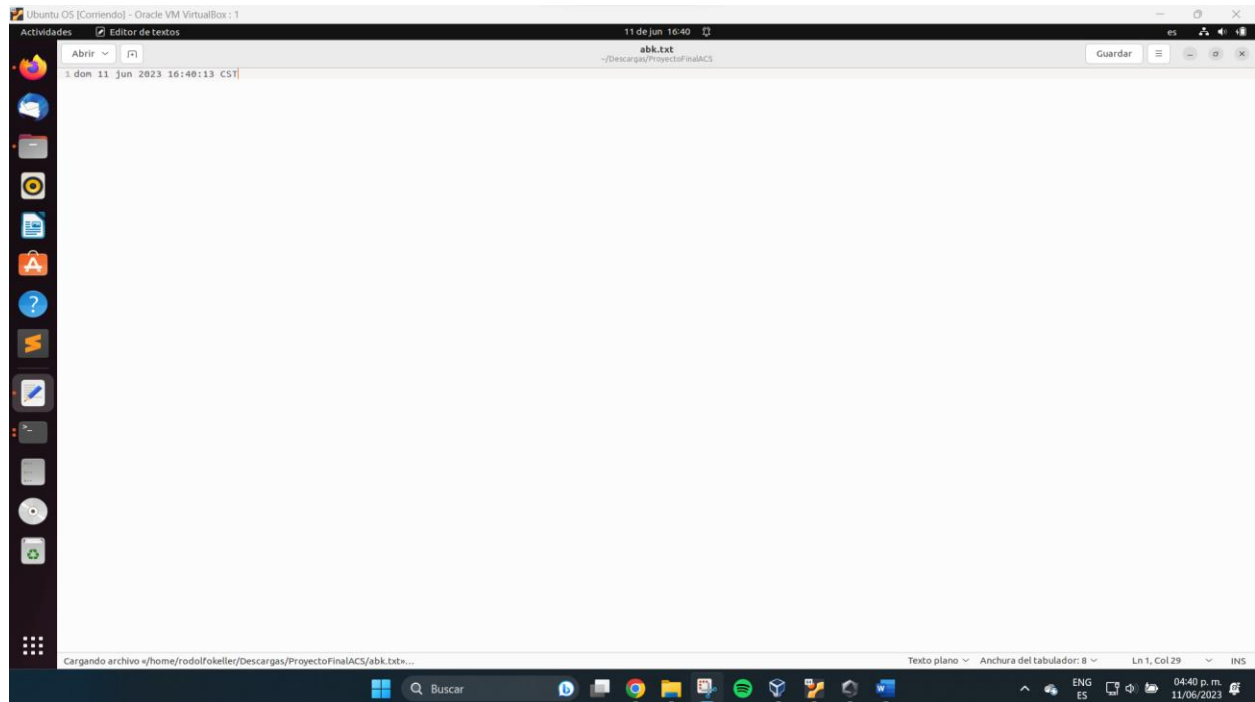
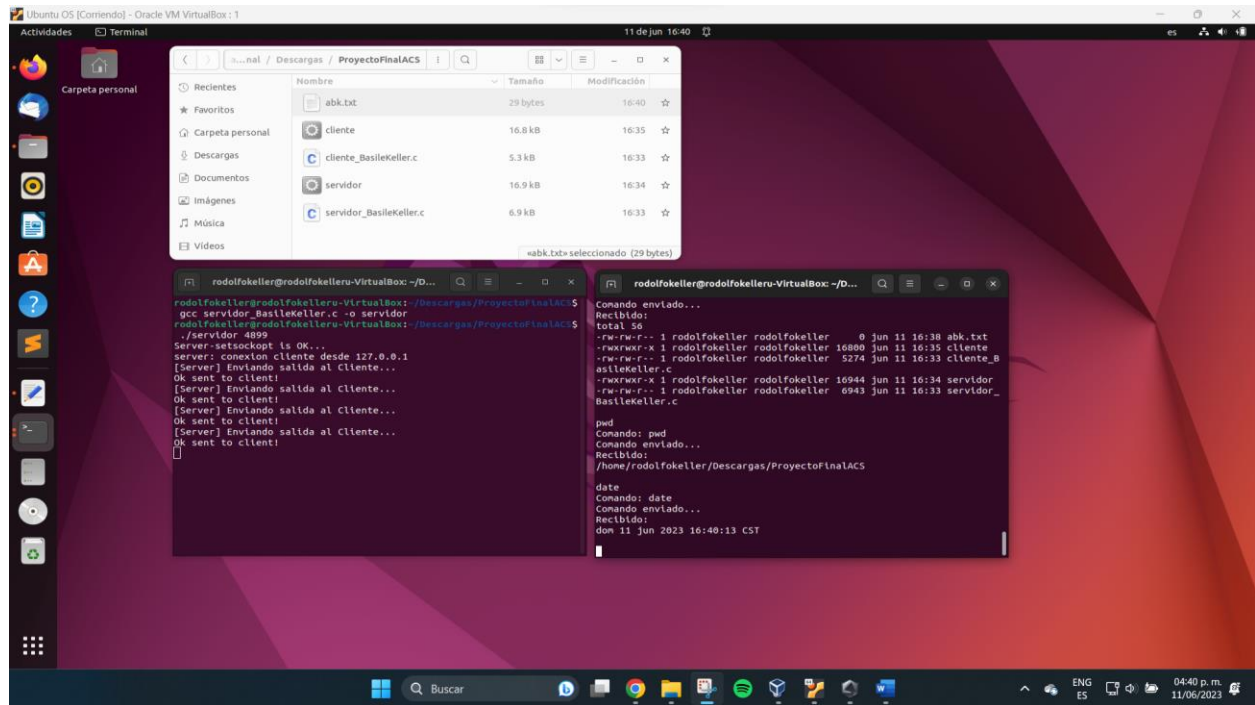
```
rodolfo@rodolfo-VirtualBox: ~/D...  
32915 32772 rodolfo$ sh -c ps -e -o pid,ppid,user,stat,comma  
nd  
32917 32915 rodolfo$ ps -e -o pid,ppid,user,stat,command  
ls -l  
Comando: ls -l  
Comando enviado...  
Recibido:  
total 56  
-rw-rw-r-- 1 rodolfo@rodolfo 0 jun 11 16:38 abk.txt  
-rwxrwxr-x 1 rodolfo@rodolfo 16880 jun 11 16:35 cliente  
-rw-rw-r-- 1 rodolfo@rodolfo 5274 jun 11 16:33 cliente_B  
asilekeller.c  
-rwxrwxr-x 1 rodolfo@rodolfo 16944 jun 11 16:34 servidor  
-rw-rw-r-- 1 rodolfo@rodolfo 6943 jun 11 16:33 servidor_  
Basilekeller.c  
  
pwd  
Comando: pwd  
Comando enviado...  
Recibido:  
/home/rodolfo@rodolfo/Descargas/ProyectoFinalACS
```



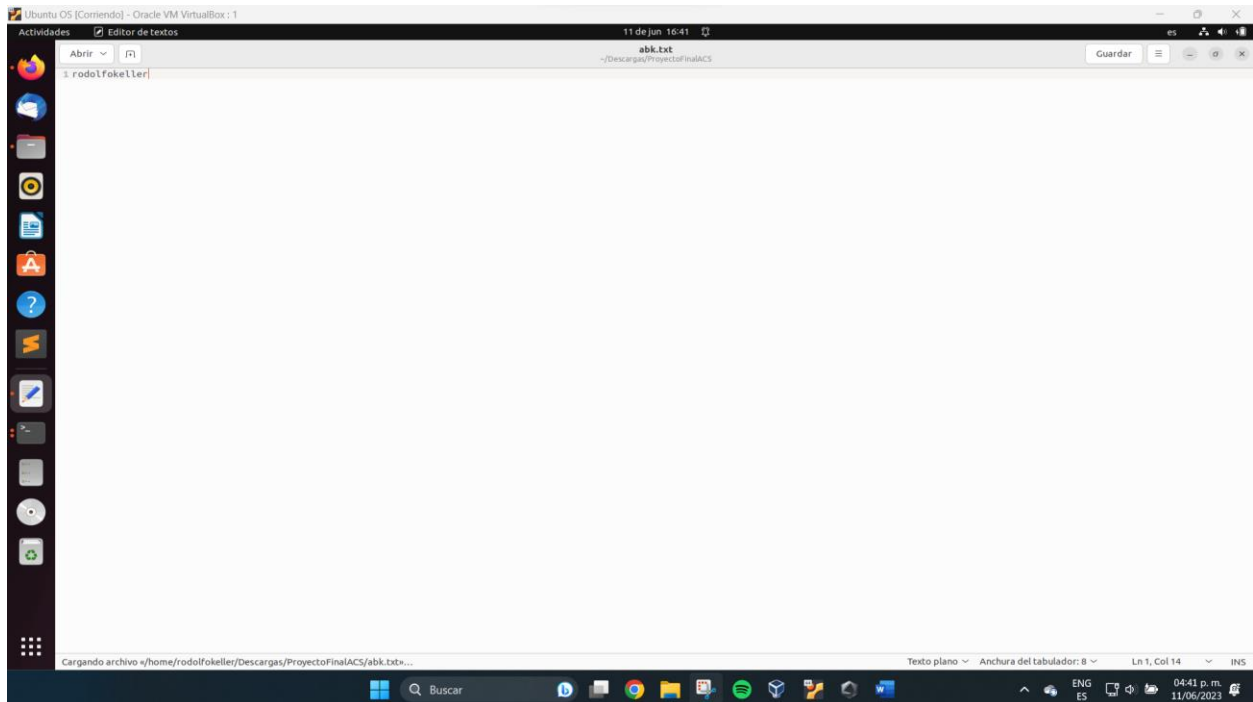
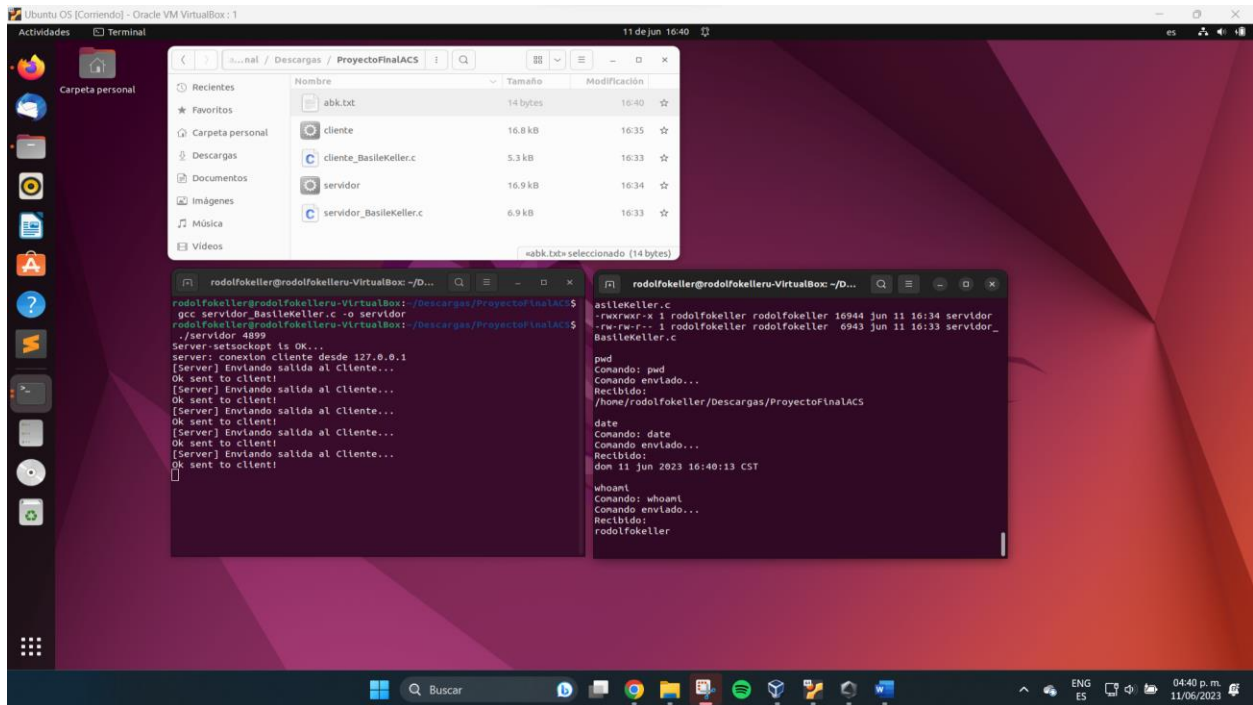
The screenshot shows the same Ubuntu VM desktop, but with a text editor window open. The editor is displaying the contents of `abk.txt`, which contains the path `/home/rodolfo@rodolfo/Descargas/ProyectoFinalACS`. The status bar at the bottom indicates the file is being loaded from `/home/rodolfo@rodolfo/Descargas/ProyectoFinalACS/abk.txt`.

```
1 /home/rodolfo@rodolfo/Descargas/ProyectoFinalACS
```

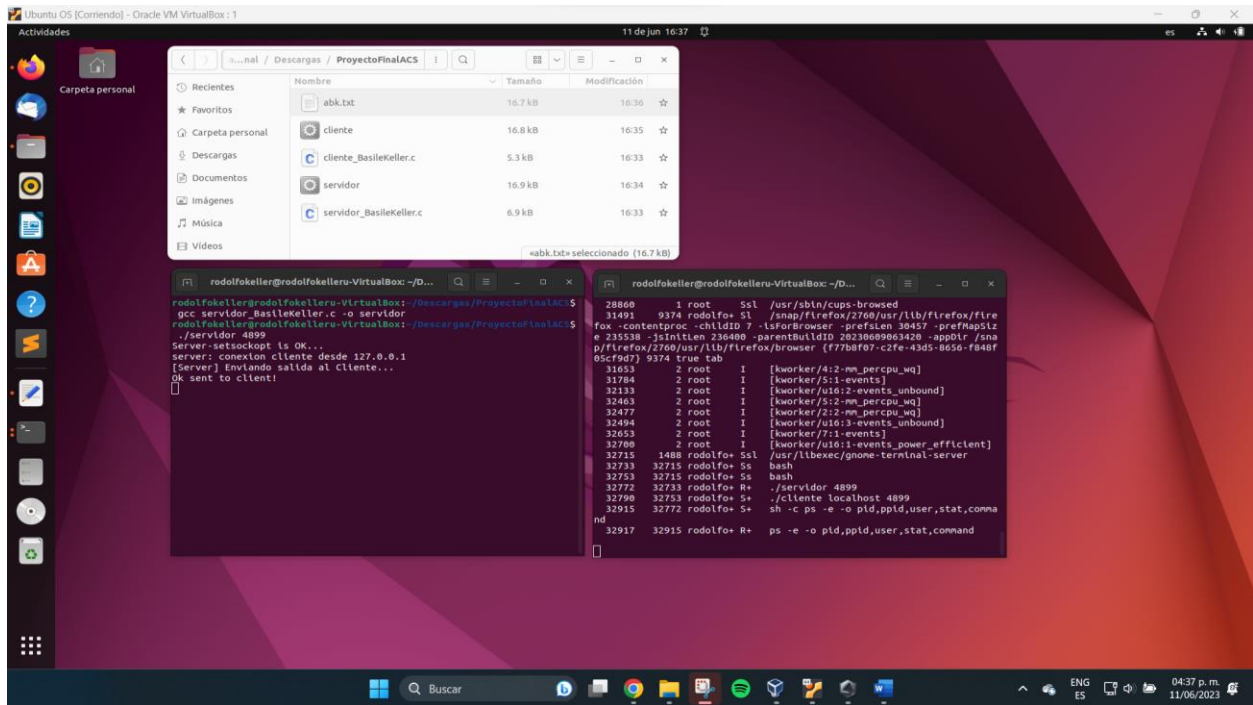
Comando “date”:

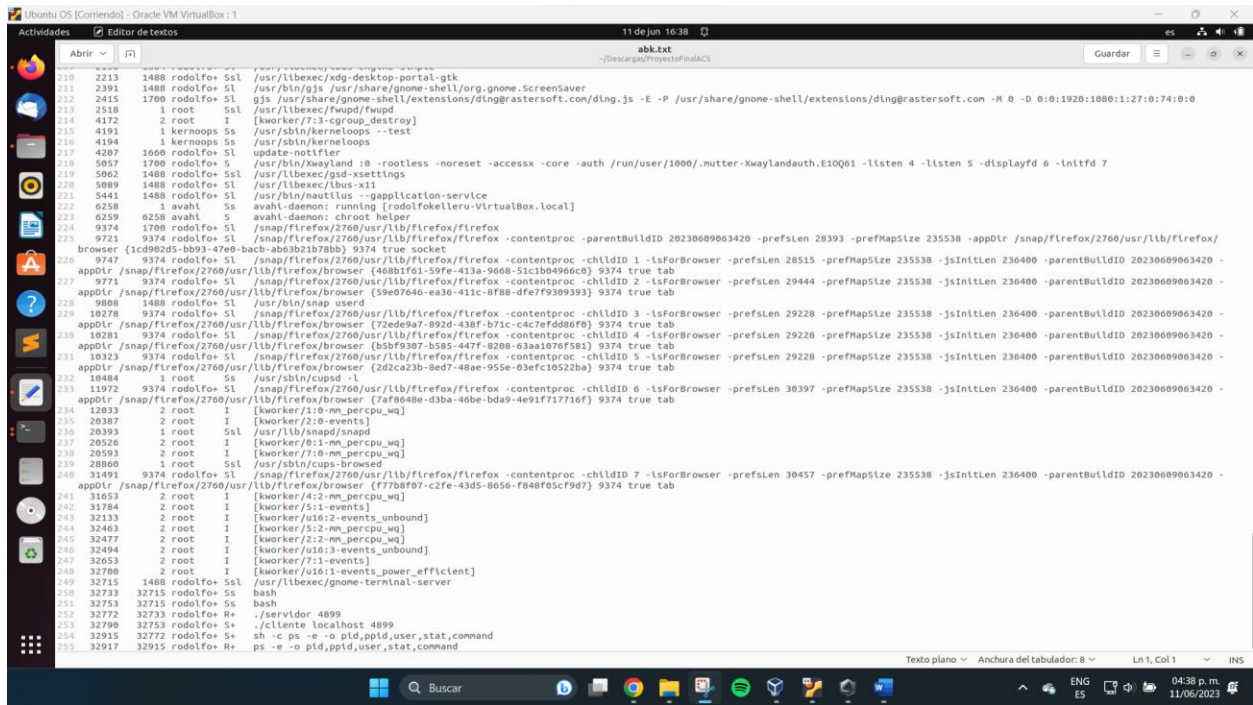


Comando “whoami”:

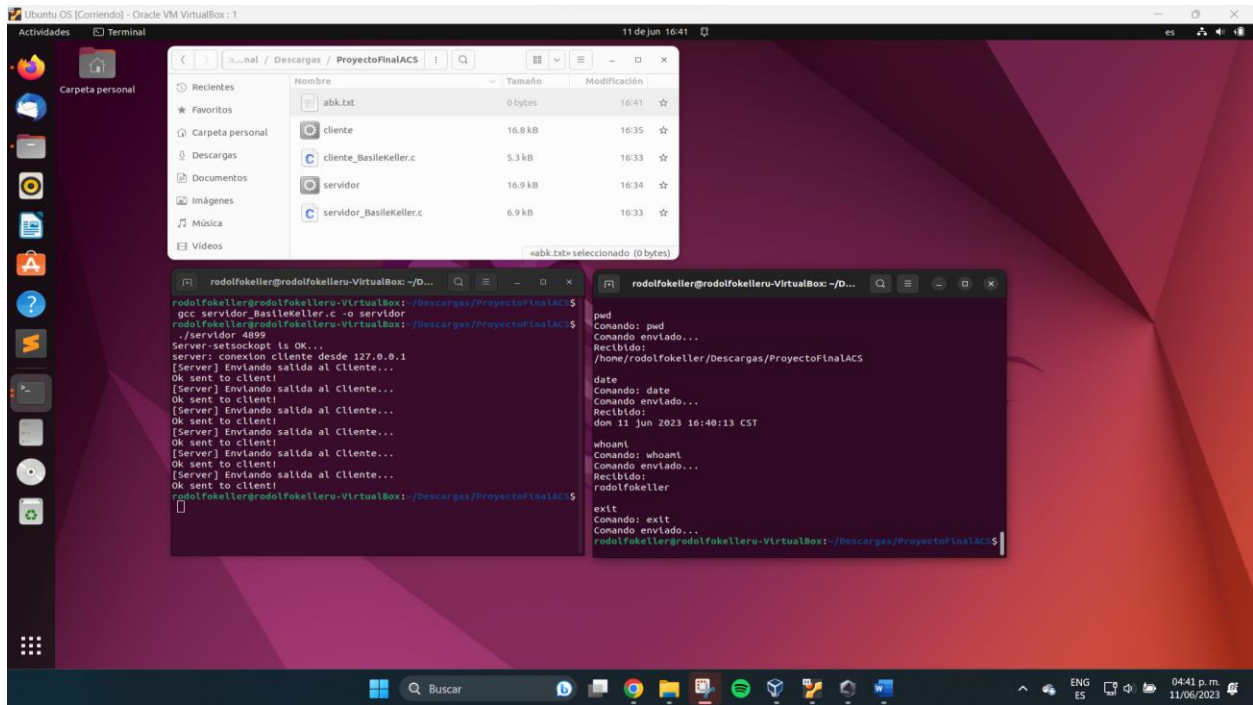


Comando “ps -e -o pid,ppid,user,stat,command”:





Comando “exit”:



Código servidor_BasileKeller.c

```
//////////////////////////////////////////////////////////////////

//                                UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
//                                FACULTAD DE INGENIERIA
//                                ARQUITECTURA CLIENTE-SERVIDOR

//                                PROYECTO FINAL
//                                PROGRAMA SERVIDOR

//                                ALUMNOS:
//                                - BASILE ALVAREZ ANDRES JOSE
//                                - KELLER ASCENCIO RODOLFO ANDRES

//////////////////////////////////////////////////////////////////

// Se importan las bibliotecas a utilizar para el cliente
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
//Indica el tamaño de caracteres del comando recibido, se quedo en 100
#define MAXDATASIZE 100
//Indica el tamaño de caracteres de la respuesta en la terminal, pasa de 20000 a 60000
#define LENGTH 60000

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

    int numbytes;                //Almacena el numero de bytes recibidos por parte
del cliente como comando
    char buf[MAXDATASIZE];       //Almacena el comando recibido por parte del cliente

    // Tenemos dos estructuras sockaddr_in:
    // Una para el propio server y otra para la conexion cliente
    // Necesitamos dos file descriptor
    int server_fd, cliente_fd;

    // Estas son las dos estructuras, la primera llamada servidor, que se asociara a
server_fd
    // y la segunda estructura llamada cliente que se asocia a cliente_fd
    struct sockaddr_in servidor;  //Información sobre direccion del servidor
    struct sockaddr_in cliente;   //Información sobre dirección del cliente

    // Variables que almacenan el tamaño de servidor y de cliente
    int sin_size_servidor;
    int sin_size_cliente;
```



```

//Se crea un socket, donde el descriptor se almacena en server_fd
if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    perror("socket");
    exit(1); //Si hay un error salimos del programa
}

//Se establecen las opciones del socket
if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &(int){1}, sizeof(int)) == -1)
{
    perror("Server-setsockopt() error!");
    exit(1);
}else printf("Server-setsockopt is OK...\n");

//Se configura la direccion y puerto del servidor
servidor.sin_family = AF_INET; // Ordenación de bytes de la máquina
servidor.sin_port = htons( atoi(argv[1]) ); // short, Ordenación de bytes de la
red
servidor.sin_addr.s_addr = INADDR_ANY; // Establece mi dirección IP
memset(&(servidor.sin_zero), '\0', 8); // Poner a cero el resto de la
estructura

sin_size_servidor = sizeof( servidor ); //Calcula el tamaño del servidor y
lo almacena

//Enlaza el socket del servidor a su direccion y puerto
if (bind(server_fd, (struct sockaddr *)&servidor, sin_size_servidor) == -1){
    perror("bind");
    exit(1);
}

//Se pone al servidor en modo de escucha para buscar conexiones, y establece la cola
de conexiones a 1
//espera una unica conexion
if (listen(server_fd, 1) == -1){
    perror("listen");
    exit(1);
}

sin_size_cliente = sizeof( cliente ); //Se calcula el tamaño del cliente
y lo almacena

//Al establecer una conexion con un cliente, se crea un socket para la conexion y le
añade la informacion del cliente
if ((cliente_fd = accept(server_fd, (struct sockaddr *)&cliente, &sin_size_cliente))
== -1){
    perror("accept");
    exit(1);
}

//Se muestra la direccion IP del cliente que se conecto al servidor
printf("server: conexion cliente desde %s\n", inet_ntoa(cliente.sin_addr));

```

```

////////////////////////////////////
//MODIFICACIONES DEL CODIGO PROPORCIONADO

//CICLO PARA QUE SE SIGA EJECUTANDO TRAS VARIOS COMANDOS HASTA RECIBIR "EXIT"

while(1){
    //Recibe los datos enviados por el cliente a traves de su socket y los almacena en
buf
    if ((numbytes=recv(cliente_fd, buf, 100-1, 0)) == -1) {
        perror("recv");
        exit(1);
    }

    //AÑADIMOS COMANDO PARA SALIR EN CASO DE QUE EL COMANDO INGRESADO SEA "EXIT"
    if (strcmp(buf,"exit") == 0) break;

    buf[numbytes] = '\0'; //Agrega un caracter nulo al final del comando para trabajar
con una cadena de caracteres valida

    // Se abre el archivo "abk.txt" en modo escritura
    FILE* file = fopen("abk.txt", "w");
    if (file == NULL) {
        perror("fopen failed");
        exit(1);
    }

    // Ejecutamos el comando almacenado en buf y redirigimos la salida a pipe
    FILE* pipe = popen(buf, "r");
    if (pipe == NULL) {
        perror("popen failed");
        exit(1);
    }

    // Leemos la salida del comando desde el pipe y escribimos en el archivo
    char buffer[1024];
    size_t bytesRead;
    while ((bytesRead = fread(buffer, 1, sizeof(buffer), pipe)) > 0) {
        fwrite(buffer, 1, bytesRead, file);
    }
    // Cerramos el archivo y el pipe
    fclose(file);
    pclose(pipe);

    //considerar comando "ps -e -o pid,ppid,user,stat,command"

    // Se lee el archivo abk.txt y su informacion se envia al cliente
    char* fs_name="abk.txt";
    char sdbuf[LENGTH]; //Se prepara el envio de caracteres del tamaño
definido 60000
    printf("[Server] Enviando salida al Cliente...\n");
    FILE *fs = fopen(fs_name, "r");
    if(fs == NULL){
        printf("ERROR: File %s not found on server.\n", fs_name);
        exit(1);
    }
}

```

```

        //Envia la salida del comando al cliente
        bzero(sdbuf, LENGTH);          //Establece todos los bytes a utilizar del buffer
a cero
        int fs_block_sz;
        //Se lee la informacion del archivo que contiene la salida del comando ejecutado
        while((fs_block_sz = fread(sdbuf, sizeof(char), LENGTH, fs))>0){
            //Se envia el contenido del bufer al cliente
            if(send(cliente_fd, sdbuf, fs_block_sz, 0) < 0){
                printf("ERROR: al enviar la salida del comando al cliente\n");
                exit(1);
            }
            bzero(sdbuf, LENGTH);
        }
        //Se cierra el archivo nuevamente
        fclose( fs );
        printf("Ok sent to client!\n");
    }

    //////////////////////////////////////

    //Cierra los descriptores del cliente y del servidor
    close(cliente_fd);
    close(server_fd);

    //Deshabilita la recepcion y el envio de datos a traves del socket del servidor
    //Se cierra tanto la lectura como la escritura del socket del servidor
    shutdown(server_fd, SHUT_RDWR);

    // Termina con exit(0) que significa terminacion exitosa
    exit(0);

    return 0;
}

```

Código Cliente_BasileKeller.c

```
//////////////////////////////////////////////////////////////////

//                                UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO
//                                FACULTAD DE INGENIERIA
//                                ARQUITECTURA CLIENTE-SERVIDOR

//                                PROYECTO FINAL
//                                PROGRAMA CLIENTE

//                                ALUMNOS:
//                                - BASILE ALVAREZ ANDRES JOSE
//                                - KELLER ASCENCIO RODOLFO ANDRES

//////////////////////////////////////////////////////////////////

// Se importan las bibliotecas a utilizar para el cliente
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
//Indica el tamaño de caracteres del comando recibido, se quedo en 100
#define MAXDATASIZE 100
//Indica el tamaño de caracteres de la respuesta en la terminal, pasa de 20000 a 60000
#define MAXDATASIZE_RESP 60000

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

    // Variables para el comando ingresado por parte del cliente
    char comando[MAXDATASIZE];        //Almacena el comando ingresado por el usuario
    int len_comando;                  //Indica la longitud del tamaño

    // Variables para la respuesta recibida por parte del servidor
    char buf[MAXDATASIZE_RESP];        //Almacena el buffer con la respuesta recibida del
servidor
    int numbytes;                      //Indica el numero de bytes recibidos como respuesta

    int sockfd;                       //Entero para el descriptor del socket
    struct hostent *he;                //Almacena informacion del servidor
    struct sockaddr_in cliente;        //Información de la dirección de destino del cliente

    //Se verifica que se hayan recibido tres argumentos al momento de ejecutar el
programa, para tener el host y el puerto
    if (argc != 3) {                  //Si no se recibieron tres argumentos se manda un
mensaje y se sale del programa
        fprintf(stderr,"usage: client hostname puerto\n");
        exit(1);
    }
}
```

```

//Se almacena la informacion del servidor, en caso de que sea NULL, se muestra error
y se sale del programa
if ((he=gethostbyname(argv[1])) == NULL) { // obtener información de host servidor
    perror("gethostbyname");
    exit(1);
}

//Se crea un socket que se almacena en sockfd, si hay un error en su creacion se
sale del programa
if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
}

//Se configura la direccion y puerto del servidor al que se conectara el cliente
cliente.sin_family = AF_INET; // Ordenación de bytes de la máquina
cliente.sin_port = htons( atoi(argv[2]) ); // short, Ordenación de bytes de la red.
Puerto a conectarse
cliente.sin_addr = *((struct in_addr *)he->h_addr); //Servidor a conectarse
memset(&(cliente.sin_zero), '\0',8); // poner a cero el resto de la estructura

//Se establece la conexion con el servidor mediante el descriptor y la informacion
del servidor.
if (connect(sockfd, (struct sockaddr *)&cliente, sizeof(struct sockaddr)) == -1) {
    perror("connect");
    exit(1);
}

////////////////////////////////////
//MODIFICACIONES DEL CODIGO PROPORCIONADO

//CICLO PARA QUE SE SIGA EJECUTANDO TRAS VARIOS COMANDOS HASTA RECIBIR "EXIT"
while(1){

    //Se lee el comando como entrada del teclado y se almacena en la variable comando
    fgets(comando,MAXDATASIZE-1,stdin);
    len_comando = strlen(comando) - 1; //No tomamos en cuenta el salto de linea
    comando[len_comando] = '\0'; //Agrega un caracter nulo al final del
comando para trabajar con una cadena de caracteres valida
    //Se imprime el comando que ingreso el cliente
    printf("Comando: %s\n",comando);

    //Mandamos el comando al servidor usando la funcion send() y el descriptor del
socket
    if(send(sockfd,comando, len_comando, 0) == -1) {
        perror("send()");
        exit(1); //Si hay un error en el envio sale del
programa
    } else printf("Comando enviado...\n");

    // SI EL COMANDO DE ENTRADA ES LA PALABRA "EXIT", CERRAMOS LA SESIÓN DEL CLIENTE.
    if(strcmp(comando,"exit")==0)
        break;
}

```

```

//Si el send no devuelve error el programa continua y lee la respuesta del servidor
//mediante el uso de recv y lo almacena en buf
if ((numbytes=recv(sockfd, buf, MAXDATASIZE_RESP-1, 0)) == -1) {
    perror("recv");
    exit(1);
}
buf[numbytes] = '\0'; //Agrega un caracter nulo al final de la
respuesta para trabajar con una cadena de caracteres valida
printf("Recibido:\n%s\n",buf); //Imprime la respuesta recibida del servidor
}

////////////////////////////////////

//Tras haber trabajado con todos los comandos y haber obtenido un mensaje "exit" o
un error,
//cerramos el file descriptor del cliente
close(sockfd);

return 0;
}

```