UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES
ESCUELA INGENIERÍA CIVIL INFORMÁTICA

Sistemas Operativos

“Llamadas al sistema”

INTEGRANTES:

Jeorge Cea

Camilo Jiménez

Docentes:

Luis Gajardo Díaz

Fernando Santolaya Franco

Sistemas Operativos

Chillán - Chile

Contenido

Introducción

Dentro de la asignatura de sistemas operativos es importante abordar lo que es las llamadas al sistema, las cuales se pueden definir de manera breve como una interfaz de programación para los servicios proveídos por sistema operativo, se utilizan para transferir el control entre el código del usuario y el del sistema.

Una llamada al sistema se invoca mediante una función, que siempre retorna un valor con información del servicio o del error producido. Existen varios tipos de llamadas al sistema y en esta tarea nos enfocaremos en las de control de procesos y comunicación.

En el control de procesos nos centraremos en la llamada al sistema fork, la cual crea un proceso “hijo” o copia, que es idéntico al proceso original o “padre” y se pueden diferenciar porque retorna un identificador del proceso el cual, si es 0 es identificador del proceso “hijo”, sino, es el del proceso “padre”, además estos procesos comparten descriptores de archivos abiertos, información del entorno, valores de variables hasta ese punto,etc.

La comunicación la llevaremos a cabo con sockets que nos permite la intercomunicación de procesos, que puede ser en un mismo o distinto sistema, unidos mediante una red.

Enunciado

Para esta tarea, usted debe implementar el juego del Gato para dos usuarios conectados en red. Para esto debe implementar un pequeño software cliente-servidor. El servidor será el que se inicie primero, y esperará una conexión desde el cliente. La interfaz del juego será por consola. Se evaluará el orden y claridad de la interfaz de juego.

a) Funcionalidades del servidor:

- Aceptar conexiones desde procesos clientes (jugadores) que residen en otro computador. El servidor considera las dos primeras conexiones (dos jugadores) como una partida (un juego). De inmediato el servidor llama a la función fork() para crear un proceso hijo y el proceso clonado atiende a esos jugadores para llevar cabo la partida. El servidor "padre" continúa escuchando nuevas conexiones.
- Aceptar hasta 3 partidas o juegos simultáneos (que juegan al mismo tiempo). Esto quiere decir hasta 6 jugadores (3 juegos) interactuando con el servidor al mismo tiempo.
- Establecer aleatoriamente el primer jugador que comienza.
- Recibir las jugadas de cada cliente (jugador) y mandárselas al jugador contrario (cliente) enviándole la matriz actualizada con la última jugada.
- Retornar a cada cliente el estado actual del tablero, para que cada cliente lo pueda mostrar al usuario por pantalla (cada vez que se realiza una jugada).
- El servidor debe mostrar por pantalla los mensajes de conexión/desconexión de nuevos clientes y mensajes de la interacción cuando juegan.
- Desconectar a los jugadores cada vez que se termina el juego.

b) Funcionalidades del cliente:

- Al ejecutar el programa cliente (un jugador) se debe aportar en la misma línea de comandos la IP del servidor como argumento de entrada y el nombre del jugador para luego en los mensajes incluir su nombre cuando le corresponda jugar
- Mostrar por pantalla el estado actual del tablero (recibido desde el servidor)
- Solicitar al jugador que ingrese una jugada (cuadrante, "X" o "O").
- El cliente cuenta con una función que muestra el tablero con el formato correspondiente al juego
- Verificar la jugada, el cliente cuenta con una función en donde se verifica si el movimiento que ingreso es válido o no.
- En el cliente se implementó una función para verificar ganador que compara los elementos y ve si se cumple una fila, columna o diagonal con los mismos caracteres.

Consideraciones:

- La tarea debe ser implementada usando el lenguaje C y Sockets.

- Debe utilizar entrada/salida no bloqueante mediante la llamada al sistema select.(Opcional)
- Debe clonar el servidor para que atienda a cada cliente (jugador) que se conecte.
- El código fuente debe estar completamente comentado.
- El computador asigna el dibujo (”O”, ”X”) de forma aleatoria
- El trabajo puede ser realizado de forma individual, o en grupos de dos personas máximo

Solución

Para implementar nuestra solución es necesario utilizar un cliente y un servidor, en donde existan una comunicación, el servidor se encarga de recibir la conexión de los clientes los cuales serán un máximo de seis conexiones y cada juego se realiza entre dos clientes.

En el cliente: en primer lugar, establecemos el puerto y un MAXDATASIZE que nos indica la cantidad máxima de caracteres que se podrá recibir por mensajes desde el servidor, luego creamos una matriz que corresponde al tablero e indicamos las cabeceras de las funciones que usaremos.

```c
#include "./stdinc.h"
const int PORT = 3490; /*puerto de conexion del server, ahi está escuchando*/
const int MAXDATASIZE = 100; /*tamanio del buffer de entrada*/
char matriz[3][3];  /*la  matriz */
void mostrar_matriz(char matriz[3][3]);
void jugada(char matriz[3][3]);
int contarJugadas(char matriz[3][3]);
int verificarGanador(char matriz[3][3]);

int main(int argc, char *argv[]) {
  char *usage = "Usar:  IP del servidor\n nombre";
  int sd; //descriptor de archivo...
  char buf[MAXDATASIZE+1]; //el buffer donde llegan los datos del server
  char *ipServer=argv[1]; //la ip del server
  char *msg = argv[2]; //el Nombre o mensaje

  if (argc != 3){
    Fatal(usage);
  }
  sd = conectate(ipServer, PORT);
  int status= enviaMensaje(msg,sd);  //envia mensaje con el nombre al servidor
  if(status==-1){
    FatalPerror("Fallo el envio del mensaje. DRAMA!");
  }
    printf("BUSCANDO RIVAL.......\n");
    printf("RIVAL ENCONTRADO, EMPEZAMOS EL JUEGO\n");
```

Posteriormente iniciamos un while en donde se recibe la matriz desde el servidor y luego se envía la matriz con la jugada correspondiente que realizo el jugador y verifica cada vez que algún jugador realiza una jugada para ver si hay ganador y finalice el programa.

```
    while(1){

    read(sd, matriz, sizeof(matriz));   //recibe la matriz que le envia el servidor
      mostrar_matriz(matriz);             // funcion para mostrar la matriz con el formato
      if(verificarGanador(matriz)==1){ printf("HAS PERDIDO\n");exit(0);}   //verifica si la
    jugada(matriz);                     // funcion que se encarga de pedir la posicion dond
      mostrar_matriz(matriz);             // muestra la matriz por pantalla con la jugada q
      write(sd, matriz, sizeof(matriz));  //envia la matriz al servidor para que este se la
      if(verificarGanador(matriz)==1){ printf("FELICIDADES HAS GANADO \n");exit(0);} //veri
    }  //fin del while
    close(sd);
    return 0;

  }
```

Se implementan las funciones que se necesitan para el manejo de la matriz como mostrar matriz función que imprime y da formato a la matriz, además de contar jugada que retorna un valor par o impar según el número de jugada para llenar con “X” para los pares y “O” para los impares

```
void mostrar_matriz(char matriz[3][3])//funcion que se encarga de mostrar la matriz por pantalla al jugador
{
  int i;

  for(i=0; i<3; i++) {
    printf(" %c | %c | %c ",matriz[i][0],
            matriz[i][1], matriz[i][2]);
    if(i!=2) printf("\n---|---|---\n");
  }
  printf("\n");
}
```

```
int contarJugadas(char matriz[3][3]){  //esta funcion cuenta las jugadas es importante para la funcion
  //en donde se debe ingresar la jugada ya que nos ayuda a saber si se debe llenar con 'O' u 'X'
 int cont=0;
 int i;
 for(i=0; i<3; i++) {
    if (matriz[i][0] != ' ')cont++;
    if (matriz[i][1] != ' ')cont++;
    if (matriz[i][2] != ' ')cont++;
  }
 return cont;
}
```

```c
void jugada(char matriz[3][3])  //esta funcion se encarga de recibir las coordenadas del tablero
// donde el jugador quiere ingresar su posicion y le indica si la jugada es valida o no, de ser valida se llena la matriz en la posicion x,y
{
  int x, y;
  int contador=0;

  printf("Escribe X,Y para tu movida (separadas por un espacio): \n");
  scanf("%d%*c%d", &x, &y);     //formato para ingresar la coordenada

  x--; y--;

  if(matriz[x][y]!= ' '){

    printf("jugada invalida ,vuelva a intertarlo\n");
      jugada(matriz);
  }
  contador=contarJugadas(matriz);
  if(contador==0|| contador==2 ||contador==4||  contador==6||  contador==8){
     matriz[x][y]='X';
  }else{
     matriz[x][y]='O';
    }
}
```

Finalmente, la función verifica ganador que revisa la matriz en todas sus filas, columnas y diagonales comparando lo que hay en cada posición y retornar si existe un ganador.

```c
int verificarGanador(char matriz[3][3]){  //funcion que verifica las columnas, filas y diagonales en caso de cumplir
  //La condicion para ser ganador retornara un 1
  int i;
  for(i=0; i<3; i++)  /* verifica filas */
    if(matriz[i][0]==matriz[i][1] &&
       matriz[i][0]==matriz[i][2] && matriz[i][0]!=' ') return 1;
  for(i=0; i<3; i++)  /* verifica columnas  */
    if(matriz[0][i]==matriz[1][i] &&
       matriz[0][i]==matriz[2][i] && matriz[0][i]!=' ') return 1;
  /* verifica diagonales */
  if(matriz[0][0]==matriz[1][1] &&
     matriz[1][1]==matriz[2][2] && matriz[2][2]!=' ')
       return 1;z
  if(matriz[0][2]==matriz[1][1] &&
     matriz[1][1]==matriz[2][0] && matriz[1][1]!=' ')
       return 1;
  return 0;
}
```

En el servidor: Primero se establece el puerto, el número máximo de conexiones, el tamaño máximo de caracteres que se recibirán por mensaje del cliente y luego se procede a declarar las variables como los descriptores de sockets que son dos uno para listen y otro para aceptar la conexión del cliente.

```c
#include "./stdinc.h"
const int MYPORT = 3490;  /*cliente se conecta en este puerto*/
const int BACKLOG = 6;  /*max # conexiones pendientes*/
const int MAXDATASIZE = 100; /*tamagno del buffer de entrada*/
char matriz[3][3];  /*la  matriz */
void iniciaMatriz(char matriz[3][3]); // corresponde a una funcion para inciar la matriz, que sera enviada a los clientes

main() {
  int sd;   /*listen sobre sd*/
  int new_fd; /*nueva conexion*/
    int new_fd2;  // corresponde a una segunda conexion, ya que en la logica de nuestro programa recibe las 2 conexiones y
    char buf[MAXDATASIZE+1]; //buffer donde se almacena el nombre del cliente 1
    char buf2[MAXDATASIZE+1];  //buffer donde se almacenara el nombre del cliente 2
  int sinSz = sizeof(struct sockaddr_in); //tamaño de la estructura de entrada (datos del cliente que se conecta) ...

  struct sockaddr_in clAddr;    /*direccion del cliente*/
  sd=conectaServer(MYPORT,BACKLOG);  // sd recibe el descriptor del socket habilitado para listen
  if(sd ==-1){
    FatalPerror("NO CONECTO EL SERVER!. DRAMA!");
  }
```

Luego se inicia un while en donde se establece la conexión con dos usuarios por vez, asignando descriptores para cada uno, luego iniciamos la matriz vacía y hacemos if con fork () para duplicar procesos y si este es 0 es un proceso hijo e ingresa a otro while el cual contiene los write y read que envían y reciben la matriz entre servidor y cliente

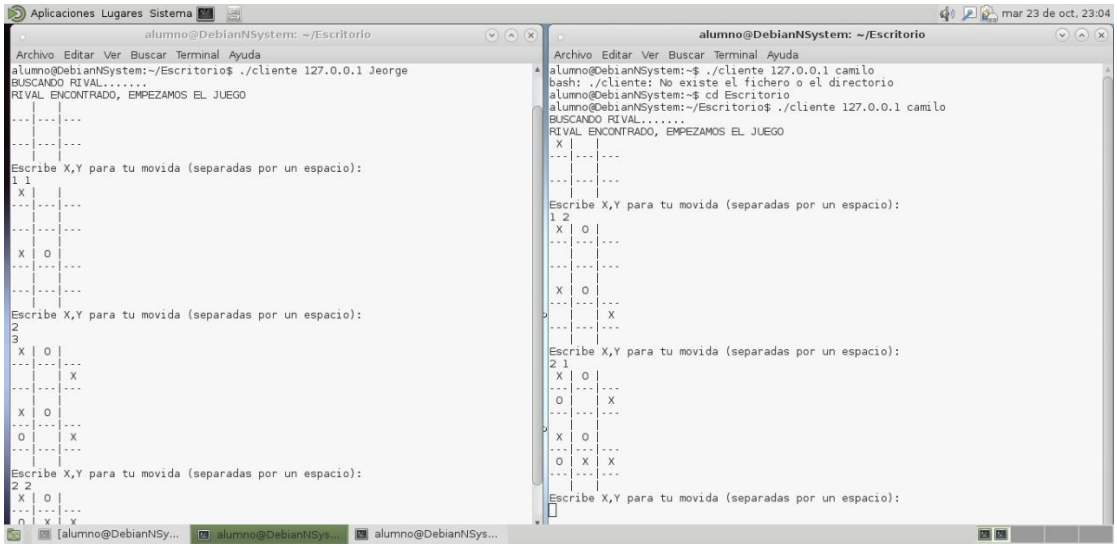
```c
while(1) { // loop principal para el  accept()
  new_fd = accept(sd, (struct sockaddr *) &clAddr, &sinSz);  //new_fd es llenado con el descriptor obtenido de la primera conexion
  if (new_fd == -1)
    FatalPerror("accept");
  int statusRespuesta=esperaRespuesta(new_fd,buf);
  if(statusRespuesta==-1) {
    FatalPerror("Fallo la recepción del mensaje. DRAMA!");
    close(new_fd);
    break;
  }
      new_fd2=accept(sd, (struct sockaddr *) &clAddr, &sinSz);  //new_fd2 es llenado con el descriptor obtenido de la segunda conexion (para no alarg
      //en este espacio se debe comprobar algun error en el accept como en el primer descriptor
      esperaRespuesta(new_fd2,buf2);
      //en este espacio se debe ver si se pudo recibir el mensaje como en la primera conexion (para no alargar mucho el codigo nos saltamos esta veri
  iniciaMatriz(matriz);  //inicia la matriz para empezar el juego entre los 2 clientes conectados
      if(fork()==0){   //uso del fork para crear un proceso hijo que entre al ciclo while para manejar el juego de los 2 clientes y el proceso padre
   while(1){
   printf("SERVIDOR: Obtuvo conexión de: %s, y su nombre es: %s\n",inet_ntoa(clAddr.sin_addr),buf);  // mensaje que muestra la direccion de la conexi
       printf("SERVIDOR: Obtuvo conexión de: %s, y su nombre es: %s\n",inet_ntoa(clAddr.sin_addr),buf2); // mensaje que muestra la direccion de la c
   write(new_fd, matriz, sizeof(matriz));     //manda la matriz al primer jugador quien iniciara la partida
       read(new_fd, matriz, sizeof(matriz));      //recibe la matriz actualizada con la jugada del primer jugador
       write(new_fd2, matriz, sizeof(matriz));   //manda la matriz al segundo jugador con la jugada que realizo su rival
   read(new_fd2, matriz, sizeof(matriz));    //recibe la matriz actualizada con la jugada del segundo jugador
  }
      }
  close(new_fd);
}//fin while
printf("Apagando el server...");
close(sd);
```

Y por último se realizan las funciones necesarias para manejar la matriz, aquí se emplea la función iniciar matriz la cual inicia la matriz con espacios vacíos y lista para utilizar.

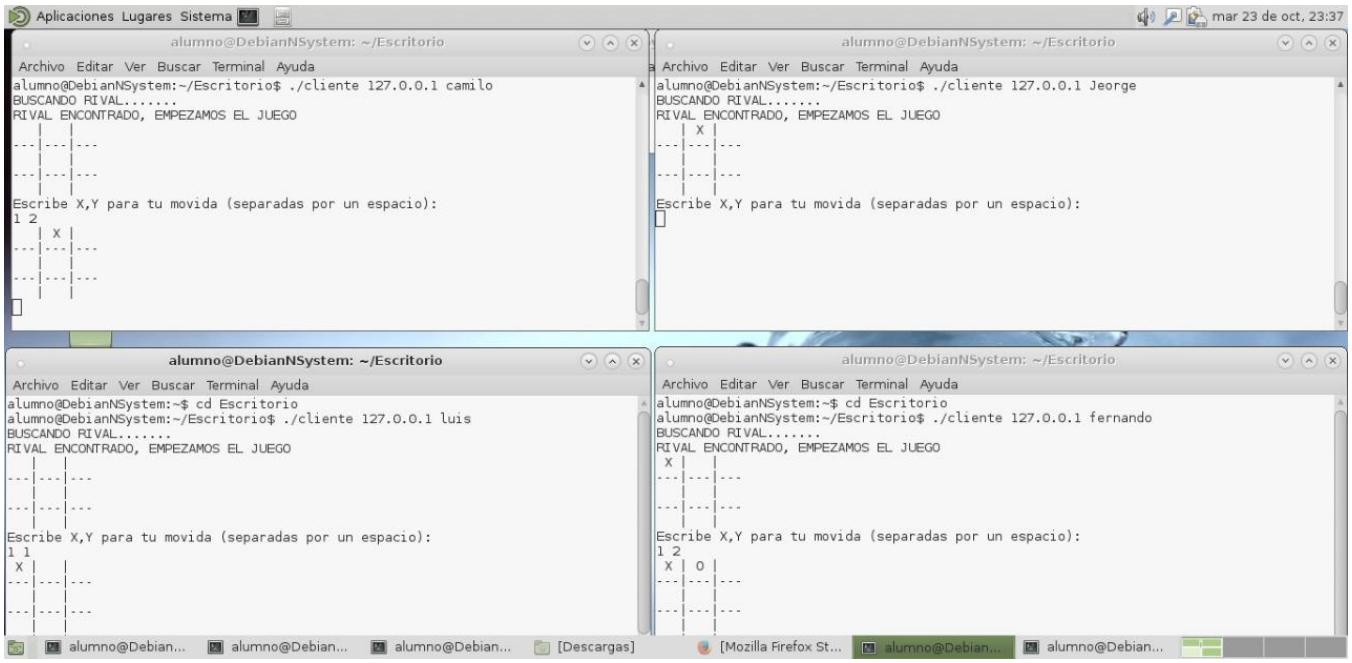
```
51  void iniciaMatriz(char matriz[3][3]){//iniciar la  matriz
52    int i, j;
53    for(i=0; i<3; i++)
54      for(j=0; j<3; j++) matriz[i][j] =  ' ';
55  }
```

Ejemplo de salida

Ejemplo se salida de una partida de dos jugadores, donde el primero que se conecta debe esperar que se conecte un oponente para comenzar el juego.



En el siguiente ejemplo se puede observar cuatro jugadores jugando en simultaneo, asignado cada uno con su pareja y en procesos distintos, sin mezclar las jugadas con las de la otra pareja.



Conclusión

Durante el transcurso de esta tarea se ha podido comprender mejor sobre el uso de sockets y como poder utilizarlos para comunicar un servidor con múltiples clientes, en un principio el primer objetivo fue comunicar un servidor con un cliente y establecer una conversación como si se tratara de un chat e imprimir los mensajes por pantalla, esto ayuda a entender cómo realizar las llamadas de las funciones send y recv (en el código de la solución se muestran como read y write, que internamente son lo mismo), luego se procede a establecer una segunda conexión para poder iniciar el juego entre los dos clientes, con esto se puede entender como usar los descriptores de sockets y que se pueden aceptar múltiples conexiones asignándolas a distintos descriptores para no perder el descriptor de la conexión anterior.

En el transcurso de la tarea una parte importante fue la implementación del fork(), debido a que se debe manejar el juego de las dos primeras conexiones y el servidor debe seguir esperando nuevas conexiones, es evidente que se debe implementar un fork(), el cual luego de analizar y estudiar la mejor forma de implementarlo en este caso en particular era después de aceptar las conexiones de los dos primeros jugadores y asignar a un proceso hijo que se encargue de la comunicación entre estos clientes, esto nos da una idea clara de como se dividen los procesos y que estos pueden operar en “simultaneo” (entre comillas porque en realidad no es así pero a nivel de usuario se aprecia de esta manera), luego del uso del fork() como es una comunicación continua entre estos dos jugadores hasta que uno gane o uno se retire (desconecte del servidor) se debe utilizar un ciclo en este caso se usó un while en donde solo ingresa el proceso hijo para manejar el juego.

Finalmente se debe destacar que las funcionalidades que nos prestan las llamadas al sistema son una herramienta necesaria y que requiere de un buen manejo y conocimiento sobre su uso, ya que la comprensión de estas no es sencilla y la implementación debe ser meticulosa para no cometer errores, debido a que en el proceso de desarrollo estos errores nos dificultaron en gran medida el avance en encontrar una solución.