Universidad Nacional Experimental de Guayana

Vice-Rectorado Académico

Coordinación de pregrado

Proyecto de carrera de Ingeniería Informática

Asignatura: Sistemas de Operación

Sección: 01

# Cuatro PID en línea

Profesor:

Caniumilla Andrés

Elaborado Por: C.I.:

Carrasco Tomás 23.506.608
Sánchez Stalin 24.183.684

Ciudad Guayana, Enero de 2015

# Marco Teórico

## Sistema Operativo

Un Sistema Operativo (SO) es el software básico de una computadora que provee una interfaz entre el resto de programas del ordenador, los dispositivos hardware y el usuario.

Las funciones básicas del Sistema Operativo son administrar los recursos de la máquina, coordinar el hardware y organizar archivos y directorios en dispositivos de almacenamiento.

Los Sistemas Operativos más utilizados son Dos, Windows, Linux y Mac. Algunos SO ya vienen con un navegador integrado, como Windows que trae el navegador Internet Explorer.

El sistema operativo es el programa (o software) más importante de un ordenador. Para que funcionen los otros programas, cada ordenador de uso general debe tener un sistema operativo. Los sistemas operativos realizan tareas básicas, tales como reconocimiento de la conexión del teclado, enviar la información a la pantalla, no perder de vista archivos y directorios en el disco, y controlar los dispositivos periféricos tales como impresoras, escáner, etc.

En sistemas grandes, el sistema operativo tiene incluso mayor responsabilidad y poder, es como un policía de tráfico, se asegura de que los programas y usuarios que están funcionando al mismo tiempo no interfieran entre ellos. El sistema operativo también es responsable de la seguridad, asegurándose de que los usuarios no autorizados no tengan acceso al sistema.

## Procesos

Un proceso de unix es cualquier programa en ejecución y es totalmente independiente de otros procesos. El comando de unix ps nos lista los procesos en ejecución en nuestra máquina. Un proceso tiene su propia zona de memoria y se ejecuta "simultáneamente" a otros procesos. Es totalmente imposible en unix que un proceso se meta, a posta o por equivocación, en la zona de memoria de otro proceso. Esta es una de las características que hace de unix un sistema fiable. Un programa chapucero o malintencionado no puede fastidiar otros programas en ejecución ni mucho menos a los del sistema operativo. Si el programa chapucero se cae, se cae sólo él.

Dentro de un proceso puede haber varios hilos de ejecución (varios threads). Eso quiere decir que un proceso podría estar haciendo varias cosas "a la vez". Los hilos dentro de un proceso comparten todos la misma memoria. Eso quiere decir que si un hilo toca una variable, todos los demás hilos del mismo proceso verán el nuevo valor de la variable.

Un proceso es, por tanto,  más costoso de lanzar, ya que se necesita crear una copia de toda la memoria de nuestro programa. Los hilos son más ligeros.

## Señales

Una señal es un "aviso" que puede enviar un proceso a otro proceso. El sistema operativo unix se encarga de que el proceso que recibe la señal la trate inmediatamente. De hecho, termina la línea de código que esté ejecutando y salta a la función de tratamiento de señales adecuada. Cuando termina de ejecutar esa función de tratamiento de señales, continúa con la ejecución en la línea de código donde lo había dibujado.

El sistema operativo envía señales a los procesos en determinadas circunstancias.

Las señales van identificadas por un número entero.

## Memoria Compartida

La memoria compartida es un recurso compartido que pone unix a disposición de los programas para que puedan intercambiarse información.

En C para unix es posible hacer que dos procesos (dos programas) distintos sean capaces de compartir una zona de memoria común y, de esta manera, puede ser accedida por múltiples programas, ya sea para comunicarse entre ellos o para evitar copias redundantes. La memoria compartida es un modo eficaz de pasar datos entre aplicaciones. Dependiendo del contexto, los programas pueden ejecutarse en un mismo procesador o en procesadores separados.

# Algoritmos Utilizados. Descomposición Modular

Al ejecutar el programa se debe pasar como parámetro la cantidad de filas del tablero (solo se admiten valores pares y que sean mayores a 5 y menores a 13), en caso de no pasarse este parámetro la cantidad de filas por defecto será igual a 6, cabe destacar que la cantidad de filas puede cambiarse en la ventana principal. La cantidad de columnas serán iguales a la cantidad de filas +1.

Una vez hecho esto se le mostrara al usuario una ventana con varias opciones a elegir modo de juego, dificultad y número de filas.

El modo de juego tiene 3 opciones J1 VS J2 (Jugador VS Jugador), J1 VS PC (Jugador VS Maquina), PC VS PC (Maquina).

En caso de elegir el modo de juego J1 VS PC (Jugador VS Maquina) o PC VS PC (Maquina) se le permitirá a el usuario escoger la dificultad fácil o difícil. En fácil la maquina jugara siempre de forma aleatoria, y si escoge difícil la maquina jugara de forma aleatoria  hasta que vea 3 bolas de mismo color que sean adyacentes en esta caso jugara al lado (si es del mismo color lograra 4 en línea y sino bloqueara la jugada de su contrincante) de estas ya sea horizontal, vertical o diagonal.

El tablero será el proceso padre y los procesos bolas y marco de bola serán hijos del mismo.

Una vez escogido lo anteriormente mencionado, se le mostrara al jugador una pantalla que incluye el tablero que a su lado izquierdo le mostrara al usuario los puntos que lleva cada jugador, los botones de pausar partida, reiniciar partida y salir de la partida. En el tablero se indicara el número de cada columna y encima de ellas un botón para que sea presionado por el jugador si este desea jugar su bola en esa columna.

El primer turno será escogido de forma aleatoria.

Se le permitirá al jugador pausar y reiniciar la partida. Cabe destacar que de pausar la partida esto no afectara las probabilidades de una jugada de ser escogida como mejor tiempo, ya que el tiempo que el juego este pausado no se cuenta.

Cada vez que jugador escoja una columna a jugar se creara un proceso bola con el color que le corresponde a ese jugador, este proceso bola guardara su pid y color en la memoria compartida a la cual el proceso marco correspondiente accederá para poder guardar los datos (pid y color) del proceso bola.

Cada vez que se hace cuatro en línea se añade se actualiza una lista encargada de guardar los puntos de ambos jugadores. Y los procesos marcos

de bola eliminan la información de la bola que alojan (quedando libres) y las bolas que se encuentran arriba de ellas caen.

Una vez el tablero este lleno (Es decir una vez no queden marcos de bolas desocupados) el juego finalizara. Una vez finalizando se mostrará al ganador o empate en caso de suceder. También se mostraran los mejores tiempos para cada tipo de línea (Vertical, Horizontal, Diagonal Principal o Secundaria).

# Descripción de la Estructura de Datos Utilizada

Estructuras

CASILLA: representa una casilla del tablero y guarda el pid del marco, el pid de la bola que esta almacenando (es -1 en caso de no estar ocupada por una bola), el color de la bola que lo está ocupado (en caso de estar ocupado es una cadena vacía) y una variable booleana que indica si está ocupada o no.

NODO: lista que se encarga de almacenar los puntos de los dos jugadores, cada vez que se hace un punto se crea otro nodo que guarda la puntuación actualizada.

RECORDS: tiene cuatro variables de tipo reales que se encarga de guardar los mejores tiempos para hacer una línea horizontal, vertical, diagonal principal y diagonal secundaria.

PUNTEROS: contiene punteros a distintas variables que se necesitan pasar a otras funciones.

## Especificaciones de Entrada

Número de Filas: se pasa como parámetro del juego, aunque después de entrar en la pantalla inicial se puede cambiar, solo admite número pares y mayores a 5 y menores a 13, en caso de no pasarlo por parámetro se tomara por defecto el valor de 6 filas.

Modo de Juego: hay tres opciones a elegir J1vsPC (Jugador versus Maquina), J1vsJ2 (Jugador versus Jugador), PCvsPC (Máquina versus Máquina). En caso de elegir J1vsPC o PCvsPC habrá que elegir el grado de dificultad (principiante o difícil).

Nivel de Dificultad: como se dijo anteriormente en caso de elegir J1vsPC o PCvsPC habrá que elegir el grado de dificultad (principiante o difícil), esto representa el nivel de dificultad con el que jugara la computadora.

Botón Salir: Botón que se encuentra en la ventana principal y permita al usuario salir del programa.

Botón Columna: en cada columna se encontrara encima un botón que al ser presionado por el usuario jugara la bola en la posición más baja disponible de la misma.

Botón Pausar: Botón que se encuentra del lado izquierdo del tablero, cuando es presionado pausa la partida que se esté jugado en el momento.

Botón Reiniciar: Botón que se encuentra del lado izquierdo del tablero, cuando es presionado vuelve a comenzar la partida (perdiéndose lo que se había jugado anteriormente).

## Especificaciones de Salida

Ventana Principal: ventana principal que se le muestra al usuario después de ejecutar el juego. En esta ventana se le permite escoger el modo de juego, dificultad y cambiar el número de filas.

Tablero: ventana en donde transcurre el juego, el usuario llegara  en esta ventana se le permita al jugador elegir la columna en donde quiere jugar, se le mostrara también las puntuaciones y le indicara a que jugador le corresponde el turno.

Bolas: cada vez que el jugador escoja una columna se le mostrara la bola bajando hasta la fila más baja de esa columna.

## Diccionario de Variables

casillaNueva-> inicializarTablero: variable de tipo Casilla que se usa para inicializar las casillas del tablero con valores por defecto.

I-> inicializarTablero: variable de tipo entero que se usa en el primer for para representar las filas del tablero.

j-> inicializarTablero: variable de tipo entero que se usa en el segundo for para representar las columnas del tablero.

pid-> inicializarTablero: variable de tipo entero que representa los pid de cada una de las casillas del tablero.

I-> imprimirCasillasvariable de tipo entero que se usa en el primer for para representar las filas del tablero.

j-> imprimirCasillas: variable de tipo entero que se usa en el segundo for para representar las columnas del tablero.

claveMemoria-> agruparBolaYMarco: variable de tipo key_t representa la clave de la memoria compartida.

idMemoria -> agruparBolaYMarco: variable de tipo entero que guarda el id de la memoria compartida.

buffer -> agruparBolaYMarco: variable de tipo Casilla que guarda el pid del marco.

i-> filaMasBajaLibre: variable de tipo entero que se usa en el for para poder recorrer las filas de una columna.

pid-> crearBola: variable de tipo entero que guarda el pid del proceso bola.

i-> visualizarTablero: variable de tipo entero que se usa para recorrer las filas del tablero.

j-> visualizarTablero: variable de tipo entero que se usa para recorrer las columnas del tablero.

x-> visualizarTablero: variable de tipo entero que representa la posición en horizontal de la pantallla

y-> visualizarTablero: variable de tipo entero que representa la posición en vertical de la pantalla.

pid-> inicializarPuntos: variable de tipo entero que guarda el pid del proceso de los puntos.

pid-> agregarPunto: variable de tipo entero que guarda el pid del proceso de los puntos.

i-> barridoDeBolas: variable de tipo entero que se usa para recorrer las filas del tablero.

j-> barridoDeBolas: variable de tipo entero que se usa para recorrer las columnas del tablero.

faux-> barridoDeBolas: variable de tipo entero que se usa de forma auxiliar para las filas.

caux-> barridoDeBolas: variable de tipo entero que se usa de forma auxiliar para las columnas

i->actualizarTiempo: variable de tipo entero que se usa para recorrer la estructura timeval.

resultado->actualizarTiempo: variable de tipo entero que guarda el tiempo.

i->procesarLineas: variable de tipo entero que se usa para recorrer las filas del tablero.

i->procesarLineas: variable de tipo entero que se usa para recorrer las columnas del tablero.

fi->procesarLineas: variable de tipo entero se usa para guardar la fila donde comience un 4 en línea.

ci->procesarLineas: variable de tipo entero se usa para guardar la columna donde comience un 4 en línea.

tipoDeLinea ->procesarLineas: variable de tipo entero que representa el tipo de línea que se hizo (0 = ninguna, 1 = horizontal, 2 = vertical, 3 = diagonal principal y 4 = diagonal secundaria).

i-> tableroLleno: variable de tipo entero que se usa para recorrer las filas del tablero.

i-> tableroLleno: variable de tipo entero que se usa para recorrer las columnas del tablero.

i-> inicializarTiemp: variable de tipo entero que se usa para recorrer la estructura timeval.

i-> reiniciar: variable de tipo entero que se usa para recorrer las filas del tablero.

i-> t reiniciar: variable de tipo entero que se usa para recorrer las columnas del tablero.

i-> G_salir: variable de tipo entero que se usa para recorrer las filas del tablero.

i-> G_salir: variable de tipo entero que se usa en el forma para recorrer las columnas del tablero.

Buf->itoa: variable de tipo vector de caracteres.

i->itoa: variable de tipo vector de enteros.

i-> G_inicializarTablero: variable de tipo entero que se usa para recorrer las filas del tablero.

i-> G_inicializarTablero: variable de tipo entero que se usa para recorrer las columnas del tablero.

color->G_inicializarTablero: variable de tipo GdkColor que representa el color del tablero.

i-> columnaATKoDEF: variable de tipo entero que se usa para recorrer las filas del tablero.

i-> columnaATKoDEF: variable de tipo entero que se usa para recorrer las columnas del tablero.

resultado-> columnaATKoDEF: variable de tipo entero que representa la columna a jugar.

i-> G_juegaPC: variable de tipo entero que representa la columna a jugar.

i-> G_reiniciar: variable de tipo entero que se usa para recorrer las filas del tablero.

j-> G_reiniciar: variable de tipo entero que se usa para recorrer las columnas del tablero.

color-> G_reiniciar: variable de tipo GdkColor que representa el color del tablero.

i-> G_inicialiarVentana: variable de tipo entero que se usa para recorrer las filas del tablero.

j-> G_inicialiarVentana: variable de tipo entero que se usa para recorrer las columnas del tablero.

ventana-> G_inicialiarVentana: variable de tipo gtk_window_new.

contenedor-> G_inicialiarVentana: variable de tipo gtk_table_new.

ventana-> G_configuracionInicial: variable de tipo gtk_window_new.

contenedor-> G_configuracionInicial: variable de tipo gtk_table_new.

color->G_ponerBola: variable de tipo GdkColor que representa el color del tablero.

i-> G_actualizarTablero: variable de tipo entero que se usa para recorrer las filas del tablero.

j-> G_actualizarTablero: variable de tipo entero que se usa para recorrer las columnas del tablero.

color-> G_actualizarTablero: variable de tipo GdkColor que representa el color del tablero.

nombre1->G_actualizarTablero: variable de tipo vector de caracteres.

nombre2->G_actualizarTablero: variable de tipo vector de caracteres.

columnaAJugar ->manejarJugada: variable de tipo entero que representa la columna que donde se va a poner la bola.

filaAJugar ->manejarJugada: variable de tipo entero que representa la fila que donde se va a poner la bola.

color->manejarJugada: variable de tipo char representa el color de la bola.

i->G_conectarBotonesDeJuego: variable de tipo entero que se usa para recorrer las columnas del tablero.

fila->main: variable entera que representa  la cantidad de filas de el tablero.

columna->main: variable entera que representa  la cantidad de filas del tablero.

turno->main: variable entera que indica a quien le corresponde el turno.

modo->main: variable entera que representa el modo de juego.

colorJ1->main: variable entera que representa el color de las bolas del jugador 1.

claveMemoria->main: variable de tipo key_t que representa la clave de acceso a la memoria compartida.

idMemoria->main: variable de tipo entero que guarda la dirección de la memoria compartida.

mejorTiempo->main: variable de tipo Records que guarda los mejores tiempos para cada jugada.

inicio->main: variable de tipo timeval.

inicio->main: variable de tipo timeval.

puntos->main: variable de tipo Nodo que guarda los puntos de cada jugador.

ventana->main: variable de tipo GtkWidget.

contenedor->main: variable de tipo GtkWidget.

hilo->main: variable de tipo pthread_t sirve para crear un hilo.

finalizarHilo->main: variable de tipo booleana que indica cuando debe finalizar el hilo.

dificultad->main: variable de tipo entera que representa el grado de dificultad.

## Tabla de Funciones Definidas

| Nombre | Tipo | Parámetros | ¿Qué hace? |
|---|---|---|---|
| establecerDimensiones | Booleana | int argc, char **argv, int *fila, int *columna, GtkWidget *cboNroFilas | Dado el número filas calcula el número de columnas a utilizar de la siguiente manera Nº Columnas = Nº filas + 1. El número de filas solo puede ser par y no puede ser menor a 5 mayor a 12, de lo contrario retornara falso. Si el número de filas es válido retornara verdadero. Si no se da el número de filas las dimensiones por defecto serán de 6 filas y 7 columnas. |
| aislar | Vacía | | Se encarga de pausar o asilar los procesos, con el fin de no saturar el procesador. |
| inicializarTablero | Vacía | int fila, int columna,Casilla tablero[fila][columna | Se encarga de llenar el tablero de procesos casilla que después son aislados inmediatamente. |
| agruparBolaYMarco | Vacía | int pidMarco | Agrupa la bola y al marco a través de la memoria compartida |
| filaMasBajaLibre | Entera | int fila, int columna, Casilla tablero[fila][columna], int columnaDeseada | Se encarga de buscar la fila más baja libre en una columna del tablero. |
| imprimirCasillas | Vacía | int fila, int columna,Casilla tablero[fila][columna] | Imprime la información de todas y cada una de las casillas. |

| | | | |
|---|---|---|---|
| liberarBuffer | Vacía | int idMemoria,Casilla *buffer | Se encarga de vaciar la memoria compartida |
| actualizarTablero | Vacía | int fila, int columna, Casilla tablero[fila][columna], int filaDeseada, int columnaDeseada, Casilla *buffer | Pasa el pid del proceso bola almacenado en la memoria compartida para que sea guardado en el marco |
| inicializarMemoriaCompartida | Vacía | key_t *claveMemoria, int *idMemoria, Casilla **buffer | Crea la memoria compartida |
| crearBola | Vacia | int fila, int columna, Casilla tablero[fila][columna], int filaDeseada, int columnaDeseada, Casilla *buffer, char color | Crea un proceso bola y guarda su pid en la memoria compartida para después ser aislado |
| clrscr | Vacía | | Limpia la pantalla |
| gotoxy | Vacía | int x,int y | Permite escribir en un lugar determinado de la pantalla |
| visualizarTablero | Vacía | int fila, int columna, Casilla tablero[fila][columna] | Permite visualizar el tablero en modo consola |
| inicializarPuntos | Vacía | Nodo **puntos | Inicializa los puntos de ambos jugadores en 0 |
| agregarPunto | Vacía | int puntoParaJ1, int puntoParaJ2, Nodo **puntos | Suma los puntos correspondientes a cada jugador |
| liberarPuntos | Vacía | Nodo **puntos | Se vacía la lista encargada de guardar los puntos de ambos jugadores |
| barridoDeBolas | Vacía | barridoDeBolas | Elimina las bolas que hayan formado una línea |
| calcularTiempo | Real | struct timeval *inicio, struct timeval *fin | Calcula y devuelve el tiempo transcurrido entre dos períodos de tiempo |

| | | | |
|---|---|---|---|
| procesarLineas | Entera | int fila, int columna, Casilla tablero[fila][columna], Nodo** puntos, struct timeval inicio[5], struct timeval fin[5], Records *mejorTiempo, GtkWidget* G_tablero[fila+1][columna+1] | Nos dice que tipo de línea se acaba de formar. Devuelve 1,2,3,4 o 0 si la línea formada es horizontal, vertical, diagonal principal, diagonal secundario o no se ha formado una línea respectivamente. |
| tableroLleno | Booleana | int fila, int columna, Casilla tablero[fila][columna] | Determina si el tablero está lleno. Devuelve Verdadero si está lleno, de lo contrario retornara falso |
| pausar | Vacía | struct timeval inicio[5], struct timeval fin[5], Records *mejorTiempo | Se encarga de pausar el juego |
| inicializarTiempo | Entera | struct timeval inicio[5], struct timeval fin[5], Records *mejorTiempo | Inicializa el tiempo de inicio y fin de la partida y de las mejores jugadas. Devuelve 1. |
| reiniciar | Vacía | int fila, int columna, Casilla tablero[fila][columna],Nodo **puntos, struct timeval inicio[5], struct timeval fin[5], Records *mejorTiempo | Inicia otra vez la partida, |
| mostrarGanadorYRecords | Vacía | Nodo *puntos, Records *mejorTiempo | Muestra el ganador de la partida y los mejores tiempos en los distintos tipos de jugada. |

| | | | |
|---|---|---|---|
| conectarPunteros | Vacía | Punteros *misPunteros, int *fila, int *columna, int *turno, int *modo, int *colorJ1, Casilla tablero[*fila][*columna], key_t *claveMemoria, int *idMemoria, Casilla *buffer, Records *mejorTiempo, struct timeval inicio[5], struct timeval fin[5], Nodo* puntos, GtkWidget* ventana, GtkWidget* contenedor, GtkWidget *G_tablero[*fila][*columna+1], pthread_t *hilo,bool *finalizarHilo, int *dificultad | Se usa para trasladar ciertas variables a otras funciones |
| G_salir | Vacía | int fila, int columna, Casilla tablero[fila][columna], Nodo* puntos | Manda a liberar los puntos y matar todos los procesos marco y bola |
| manejarSalida | Booleana | GtkWidget *widget, Punteros* misPunteros | Llama a la función G_salir y después sale del programa |
| manejarSalidaConfig | Booleana | GtkWidget *widget, gpointer data | Se encarga de salirse del programa |
| itoa | Carácter | int val, int base | Transforma un entero a un caracter |
| G_actualizarTurno | Vacía | int fila, int columna, GtkWidget* G_tablero[fila+1][columna+1], int *turno | Se encarga de mostrar a quien le toca jugar |

| | | | |
|---|---|---|---|
| G_inicializarTablero | Vacía | int fila, int columna, GtkWidget* G_tablero[fila+1][columna+1], int modo, int *turno, bool *finalizarHilo | Inicializa el tablero en modo gráfico |
| manejarPause | Vacía | GtkWidget* btnPausar, Punteros *misPunteros | Se encarga de manejar la pausa |
| columnaATKoDEF | Entera | int fila, int columna, Casilla tablero[fila][columna] | Determina y devuelve en qué columna debe jugar la maquina si hay 3 bolas del mismo color consecutivo esta de forma horizontal, vertical o diagonal, jugara en la columna contiguo, sino elegirá una columna al azar |
| G_juegaPC | Vacía | int fila, int columna, GtkWidget* G_tablero[fila+1][columna+1],Casilla tablero[fila][columna], int dificultad | Se encarga de elegir en donde jugar la maquina dependiendo de su dificultad, si es fácil será al azar, sino se usa la función columnaATKoDEF |
| G_PCvsPC | Vacía | Punteros *misPunteros | Se encarga de ejecutar el juego en el modo PC vs PC |
| G_reiniciar | Vacía | int fila, int columna, GtkWidget* G_tablero[fila+1][columna+1],int *turno, int modo, int colorJ1, pthread_t *hilo, bool *finalizarHilo, Punteros *misPunteros | Reinicia el juego en modo grafico |
| G_cargarValoresPredeterminados | Vacía | Punteros *misPunteros | Llama a las funciones que se encargan de inicializar valores |

| | | | |
|---|---|---|---|
| G_configuracionInicial | Vacía | int argc, char **argv, int* fila, int* columna, int*modo, int* colorJ1, int* dificultad | Muestra la ventana principal donde se elige el número de filas, dificultad, modo de juego, color de las fichas,etc. |
| G_ponerBola | Vacía | int fila, int columna, GtkWidget* G_tablero[fila+1][columna+1],int filaAJugar, int columnaAJugar, int turno, GtkWidget* ventana | Pone la bola en la columna deseada en modo grafico |
| G_actualizarTablero | Vacía | int fila, int columna, Casilla tablero[fila][columna], GtkWidget* G_tablero[fila+1][columna+1], Nodo* puntos, int *turno,int filaAJugar, int columnaAJugar | Actualiza el tablero en modo gráfico |
| manejarJugada | Entera | GtkWidget* btnPulsado, Punteros *misPunteros | Se encarga de administrar la jugada que se quiere hacer, sea un jugador o la PC. |
| G_conectarBotonesDeJuego | Vacía | int fila, int columna, GtkWidget *G_tablero[fila+1][columna+1], Punteros *misPuntero | Se encarga de conectar los botones del juego con la acción que tienen que realizar una vez sean presionados |
| manejador | Vacía | int numSignal | Se encarga de manejar las señales, ya sea para quitar la pausa o agrupar a la bola y al marco. |

| | | | |
|---|---|---|---|
| actualizarTiempo | Vacía | int tipoLinea, struct timeval inicio[5], struct timeval fin[5], Records *mejorTiempo | Calcula el tiempo necesitado para realizar una línea desde que se hizo la ultima y determina si es el mejor tiempo para hacer ese tipo de línea |
| manejarReinicio | Vacía | GtkWidget* btnPausar, Punteros *misPunteros | Se encarga de manejar el reinicio del juego |
| G_inicialiarVentana | Vacía | GtkWidget* ventana, GtkWidget* contenedor, int fila, int columna, GtkWidget* G_tablero[fila+1][columna+1], Punteros *misPunteros | Inicializa la ventana de inicio |

## Consideraciones

En la ventana principal se podrá cambiar el número de filas.

El juego iniciará mediante un sorteo entre los dos (02) jugadores para saber quien inicia.

Cuando una bola se aloje en un marco de bola el proceso bola guarda su pid y color en la memoria compartida  y se envía una señal del proceso bola al proceso marco indicándole que vaya a la memoria compartida a buscar el pid y el color del proceso bola que va a alojar.

El juego finalizará cuando el tablero este lleno.

Se maneja un reloj que permite almacenar los mejores tiempos para formar líneas según su tipo: horizontal, vertical o diagonal principal o secundaria. También se indica a que jugador le corresponde su turno según su color de bola.

El juego tiene dos niveles de dificultad: principiante y difícil.

Tanto el ganador o empate como los mejores tiempos se muestran en la terminal.

Para quitar el pause se debe pulsar ctrl+c en la terminal.

## Restricciones

No se podrá pausar el juego durante una partida en modo PCVSPC.

No se podrá compilar desde una pc con Debian ya que presentará fallas en el manejo las cadenas, debido a la incompatibilidad en el formato de codificación de caracteres utilizado para desarrollar el proyecto en Ubuntu, aunque si se podrá ejecutar y funcionar correctamente en ambos SO.

## Conclusiones

Un proceso de unix es cualquier programa en ejecución y es totalmente independiente de otros procesos. Un proceso tiene su propia zona de memoria y se ejecuta "simultáneamente" a otros procesos. Es totalmente imposible en unix que un proceso se meta, a posta o por equivocación, en la zona de memoria de otro proceso. Esta es una de las características que hace de unix un sistema fiable. Un programa chapucero o malintencionado no puede fastidiar otros programas en ejecución ni mucho menos a los del sistema operativo. Si el programa chapucero se cae, se cae sólo él.

Dentro de un proceso puede haber varios hilos de ejecución (varios threads). Eso quiere decir que un proceso podría estar haciendo varias cosas "a la vez". Los hilos dentro de un proceso comparten todos la misma memoria. Eso quiere decir que si un hilo toca una variable, todos los demás hilos del mismo proceso verán el nuevo valor de la variable.

Un proceso, en el momento de lanzarlo, se hace independiente del nuestro, así que no deberíamos tener ningún problema, salvo que necesitemos comunicación entre ellos, que nos llevaría a programar memorias compartidas (con sus correspondientes semáforos), colas de mensajes, sockets o cualquier otro mecanismo de comunicación entre procesos unix.

Una señal es un aviso que puede enviar un proceso a otro proceso. El sistema operativo unix se encarga de que el proceso que recibe la señal la trate inmediatamente. De hecho, termina la línea de código que esté ejecutando y salta a la función de tratamiento de señales adecuada. Cuando termina de ejecutar esa función de tratamiento de señales, continúa con la ejecución en la línea de código donde lo había dibujado.

El sistema operativo envía señales a los procesos en determinadas circunstancias.