FONAMENTOS de SISTEMAS OPERATIVOS

Práctica 2:

THREADS, SEMÁFOROS Y BUZONES

Departament d'Enginyeria

[DEIM]

Informàtica i Matemàtiques

Escola Tècnica Superior d'Enginyeria

Universitat Rovira i Virgili

Índice

1. Fase 1 y 2

1.1. Enunciado resumido Fase 1

En esta fase se deberán modificar las funciones de la Fase 0 que involucran el movimiento de la paleta y de la pelota con el objetivo de que funcionen como hilos de ejecución independientes.

El primer paso recomendado es modificar la función de movimiento de la pelota. Para ello lo primero es modificar la cabecera de la función a :

```
void * mou_pilota(void * index)
```

El valor index es un entero que indica el orden de creación de las pelotas. Este sirve para acceder a la tabla global que contiene la información sobre estas.
La función de movimiento a partir de ahora deberá ejecutarse de manera independiente al bucle principal del programa por lo que se debe implementar su propio bucle. Las condiciones de finalización de este serán las mismas que las del bucle principal. Por ello, estas variables deberán hacerse globales.

El segundo paso recomendado es modificar la función de movimiento del usuario que es la que se encarga de mover la paleta. Lo primero, como en el caso anterior es modificar la cabecera:

```
void * mou_paleta(void * nul)
```

Esta función de movimiento también deberá tener su bucle propio.

El tercero es modificar la función principal del programa. En el main se tendrán que crear los hilos de la pelota y paleta y eliminar la invocación a sus funciones del bucle principal. Además, se tendrá que añadir un contador del tiempo relativo de la partida. Finalmente, la última acción del programa principal será escribir el tiempo total del juego y los mensajes de finalización.

El cuarto paso es modificar la función **comprovar_bloc** para que si la pelota que se mueve toca un bloque del tipo B se cree una nueva pelota creada en un nuevo thread a la que se le pasará por parámetro su identificador.
Además, la función **mou_pilota** debe poder gestionar los datos de la pelota que se le asigna. Por ello cada pelota debe tener su propia información lo que implica que se tengan que convertir las variables globales *'f_pil', 'c_pil', 'pos_f', 'pos_c', 'vel_f' y 'vel_c*' a vectores de máximo 9 posiciones.

1.2. Enunciado resumido Fase 2

En esta fase se deben establecer las secciones críticas para evitar los problemas de concurrencia de la fase anterior. Para ello se incluirán semáforos para hacer la exclusión mútua entre hilos.

Las secciones críticas impiden el acceso concurrente a determinados recursos por parte de los diferentes hilos de ejecución. En nuestro caso se necesita establecer secciones críticas para acceder al entorno de escritura y a algunas variables globales.

Para facilitar el desarrollo se recomienda añadir las dos siguientes líneas al *makefile* para permitir la compilación de la nueva fase:

```
mur2 : mur2.c winsuport.o winsuport.h
      gcc Wall mur2.c winsuport.o o mur2 lcurses
      lpthread
```

De esta manera se establecen las dependencias entre los ficheros de salida y los de entrada además de especificar el comando que se debe invocar en el caso de que la fecha de modificación de algún fichero de entrada sea posterior a la de última modificación del de salida. Este control lo realiza el comando *make*.

Si no se especifica ningún parámetro, *make* verificará las dependencias del primer fichero de salida que encuentro dentro del *makefile* del directorio de trabajo actual.

1.3. Diseño

Solución para el primer paso:

Primero se ha cambiado la cabecera para cumplir con las normas del enunciado.

Para la implementación de su propio bucle, la solución pensada es la de añadir dentro de la función *mou_pilota()* un bucle con las condiciones de fin: (!fi1 && !fi2). Esta condición permite que se salga del bucle en caso de que se pulse la tecla RETURN o de que la pelota salga del tablero.

Para poner las variables *'fi1'* y *'fi2'* a globales se ha copiado y pegado la definición de estas a la parte de las declaraciones de variables globales.

Para facilitar la implementación de los threads se ha incluido la librería *pthread.h*.

Además, se ha añadido una tabla de threads llamada *tid* en la que se posiciona la paleta en la posición 0 y la primera pelota en la 1.

Solución para el segundo paso:

Se ha realizado los mismos cambios que en el paso anterior pero en la función *mou_paleta()*.

Solución para el tercer paso:

Se ha quitado del bucle principal del main la llamada a las funciones *mou_pilota()* y *mou_paleta()*.

Sobre este se ha añadido la creación de los threads de la paleta y de la pelota utilizando la función siguiente:

```
pthread_create(&tid[id], NULL, &funcion_thread, (intptr_t *) id);
```

Donde id es el identificador correspondiente al índice que tomará cada thread en la tabla. Cada vez que se llama a una de estas funciones el contador *id* incrementa en uno.

Para facilitar el paso por parámetro de la variable *id* se ha cambiado el tipo de esta variable a *intptr_t* lo que permite trabajar con enteros de tamaño menor a la vez que se permite hacer un cast correcto a un entero desde un puntero a void *.

Finalmente se ha añadido un contador del tiempo relativo de la partida en formato (minutos:segundos) utilizando funciones propias de relojes en C.

Solución para el cuarto paso:

Primero se ha cambiado los tipos de las variables 'f_pil', 'c_pil', 'pos_f', 'pos_c', 'vel_f' y 'vel_c' a vectores de 9 posiciones.

Para conseguir la creación de más pelotas a la hora de realizar el choque de una pelota con un bloque de tipo *BBB* se ha hecho que cuando se entra en la función *comprovar_bloc()* y la condición de que el tipo de bloque es este, se guarda en los vectores de posición del nuevo thread a crear la posición de la fila y columna que se le han pasado por parámetro con la finalidad de conseguir el efecto de que cuando el bloque se destruye sale una pelota de este.

Conjuntamente a la creación de estas pelotas, se les asigna una velocidad random de rango -1 a 1.

Al crear el thread perteneciente a la nueva pelota aumentará el contador *id* en uno y el número total de pelotas.

Al añadir la funcionalidad de múltiples pelotas, se ha añadido una condición de fin a la función *mou_pilota()* para que el hilo también finalice al salirse su pelota por la portería. Además, se ha cambiado el valor de fi2 que dependerá tanto del número total de bloques de tipo AAA restantes como de la cantidad de pelotas. Esta variable se actualizará a 1 en caso de que el número de pelotas o de bloques de tipo AAA sea igual a 0.

Para que se realizara de manera correcta el conteo de las pelotas, a la hora de la creación del juego, en el momento en el que se establece el número de pelotas creadas, se le ha restado uno debido a que se contaban las líneas del fichero params.txt y este contiene dos líneas. Una de ellas es del tamaño del tablero y la segunda de la pelota por lo que se contaban dos pelotas en vez de 1 que sería lo correcto.

Para conseguir la sincronización de todos los threads creados se ha añadido un lock y unlock alrededor de cada variables global que se modifica y utiliza en diferentes threads.
En concreto, se han añadido antes alrededor de las actualizaciones de las siguientes variables:

- dir_paleta : utiliza la dirección de la paleta para realizar un efecto cuando una pelota la usa para rebotar. Por ello, todos los hilos de tipo pelota la actualiza.
- num_pilotes : se utiliza para finalizar el juego. Al conocer todos los hilos de tipo pelota el número de estas que quedan en el tablero para llegar a su fin.

- fi1 : condición final de los bucles del programa principal, pelotas y paleta. Si esta se actualiza se debe permitir que se haga sin interrupciones.
- fi2 : condición final de los bucles del programa principal, pelotas y paleta. Si esta se actualiza se debe permitir que se haga sin interrupciones.
- nblocs : forma parte de la condición final de los hilos. Informa de la cantidad de bloques de tipo ‘AAA’ que hay restantes en el tablero.
- win_escricar() : función que permite escribir en la pantalla de juego. Cada hilo debe poder escribir en pantalla sin ser interrumpido.
- win_escristr() : función que permite escribir en la última línea de pantalla. Se debe poder escribir sin ser interrumpido.

1.4. Implementación

El código obtenido con los cambios del apartado anterior es el mostrado a continuación. En este se plasmará cada cambio de cada función actualizada a color azul.

Librerías añadidas:

```
#include <stdio.h>              /* incloure definicions de funcions estandard */
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "winsuport.h"          /*incloure definicions de funcions propies */
#include <pthread.h>            /* incloure threads */
```

Declaración de variables globales:

```
pthread_t tid[MAX_THREADS];  /* tabla de identificadors de threads */
/* per inicialitzar el mutex es pot posar:
int pthread_mutex_init(pthread_mutex_t * mutex, const pthread_mutexattr_t * attr)
o bé es pot fer la següent assignació, que inicialitza el mutex amb els valors per defecte*/
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

intptr_t id = 1;                /* identificador de thread*/
int num_pilotes = 1;            /* numero de pelotas */
int n_fil, n_col;               /* numero de files i columnes del taulell */
int m_por;                      /* mida de la porteria (en caracters) */
int f_pal, c_pal;               /* posicio del primer caracter de la paleta */
int f_pil[MAXBALLS], c_pil[MAXBALLS];       /* posicio de la pilota, en valor  enter */
float pos_f[MAXBALLS], pos_c[MAXBALLS];     /* posicio de la pilota, en valor real */
float vel_f[MAXBALLS], vel_c[MAXBALLS];     /* velocitat de la pilota, en valor real */
int nblocs = 0;
int dirPaleta = 0;
int retard;                     /* valor del retard de moviment, en mil.lisegons */
int fi1, fi2;                   /* valor de condicions finals */
char strin[LONGMISS];           /* variable per a generar missatges de text */
```

Función que se encarga de controlar el impacto entre pelotas y paleta:

```
void control_impacte(void)
{
	int i;
	for(i = 1; i <= num_pilotes; i++)
	{
		if (dirPaleta == TEC_DRETA)
		{
			if (vel_c[i] <= 0.0)			/* pilota cap a l'esquerra */
				vel_c[i] = -vel_c[i] - 0.2;	/* xoc: canvi de sentit i reduir velocitat */
			else
			{					/* a favor: incrementar velocitat */
				if (vel_c[i] <= 0.8)
					vel_c[i] += 0.2;
			}
		}
		else
		{
			if (dirPaleta == TEC_ESQUER)
			{
				if (vel_c[i] >= 0.0)		/* pilota cap a la dreta */
					vel_c[i] = -vel_c[i] + 0.2;  /* xoc: canvi de sentit i reduir la
									velocitat */
				else
				{				/* a favor: incrementar velocitat */
					if (vel_c[i] >= -0.8)
						vel_c[i] -= 0.2;
				}
			}
			else
			{					/* XXX trucs no documentats */
				if (dirPaleta == TEC_AMUNT)
					vel_c[i] = 0.0;			/* vertical */
				else
				{
					if (dirPaleta == TEC_AVALL)
						if (vel_f[i] <= 1.0)
							vel_f[i] -= 0.2;	/* frenar */
				}
			}
		}
		pthread_mutex_lock(&mutex);
		dirPaleta=0;					/* reset perque ja hem aplicat l'efecte */
		pthread_mutex_unlock(&mutex);
	}
}
```

Función que comprueba la colisión entre la pelota y los bloques:

```
/*Si hi ha una col.lisió pilota-bloc esborra el bloc */
void comprovar_bloc(int f, int c)
{
	int col;
	char quin = win_quincar(f, c);
	if (quin == BLKCHAR || quin == FRNTCHAR)
	{
		col = c;
		while (win_quincar(f, col) != ' ')
		{
			pthread_mutex_lock(&mutex);
			win_escricar(f, col, ' ', NO_INV);
			pthread_mutex_unlock(&mutex);
			col++;
		}
		col = c - 1;
		while (win_quincar(f, col) != ' ')
		{
			pthread_mutex_lock(&mutex);
			win_escricar(f, col, ' ', NO_INV);
			pthread_mutex_unlock(&mutex);
			col--;
		}
		pthread_mutex_lock(&mutex);
		nblocs--;
		pthread_mutex_unlock(&mutex);
		/*  generar nova pilota que té com a posició inicial la fila i columna del bloc
		BBB destruït*/
		if (quin == BLKCHAR)
		{
			pos_f[id] = f;
			pos_c[id] = c;
			vel_f[id] = (float)rand()/(float)(RAND_MAX/2)-1;
			vel_c[id] = (float)rand()/(float)(RAND_MAX/2)-1;
			pthread_create(&tid[id],NULL, &mou_pilota , (intptr_t *) id);
			pthread_mutex_lock(&mutex);
			id++;
			num_pilotes++;
			pthread_mutex_unlock(&mutex);
		}
	}
}
```

Función que se encarga del movimiento de las pelotas:

```
void * mou_pilota(void * index)
{
	int f_h, c_h;
	char rh, rv, rd, no;
	int fora = 0, fi3=0;
	int in = (intptr_t)index;
	do{                                                                    /* Bucle pelota */
		f_h = pos_f[in] + vel_f[in];            /* posicio hipotetica de la pilota (entera) */
		c_h = pos_c[in] + vel_c[in];
		rh = rv = rd = ' ';
		if ((f_h != f_pil[in]) || (c_h != c_pil[in]))
		{
		/* si posicio hipotetica no coincideix amb la posicio actual */
			if (f_h != f_pil[in])                    /* provar rebot vertical */
			{
				rv = win_quincar(f_h, c_pil[in]);   /* veure si hi ha algun obstacle */
				if (rv != ' ')                   /* si hi ha alguna cosa */
				{
					comprovar_bloc(f_h, c_pil[in]);
					if (rv == '0')           /* col.lisió amb la paleta? */
					{
						/* XXX: tria la funció que vulgis o implementa'n una
						millor */
						control_impacte();
						vel_c[in] = control_impacte2(c_pil[in], vel_c[in]);
					}
					vel_f[in] = -vel_f[in];       /* canvia sentit velocitat vertical */
					f_h = pos_f[in] + vel_f[in];    /* actualitza posicio hipotetica */
				}
			}
			if (c_h != c_pil[in]) {        /* provar rebot horitzontal */
				rh = win_quincar(f_pil[in], c_h);   /* veure si hi ha algun obstacle */
				if (rh != ' ') {       /* si hi ha algun obstacle */
					comprovar_bloc(f_pil[in], c_h);
					/* TODO?: tractar la col.lisio lateral amb la paleta */
					vel_c[in] = -vel_c[in];      /* canvia sentit vel. horitzontal */
					c_h = pos_c[in] + vel_c[in];    /* actualitza posicio hipotetica */
				}
			}
			if ((f_h != f_pil[in]) && (c_h != c_pil[in])) { /* provar rebot diagonal */
				rd = win_quincar(f_h, c_h);
				if (rd != ' ') {       /* si hi ha obstacle */
					comprovar_bloc(f_h, c_h);
					/* TODO?: tractar la col.lisio amb la paleta */
					vel_f[in] = -vel_f[in];
					vel_c[in]= -vel_c[in];       /* canvia sentit velocitats */
					f_h = pos_f[in] + vel_f[in];
					c_h = pos_c[in] + vel_c[in];        /* actualitza posicio entera */
				}
			}
```

```
                        /* mostrar la pilota a la nova posició */
                        no = win_quincar(f_h, c_h);
                        if (no==' ')
                        {       /* verificar posicio definitiva *//* si no hi ha obstacle */
                                pthread_mutex_lock(&mutex);
                                win_escricar(f_pil[in], c_pil[in], ' ', NO_INV);/* esborra pilota */
                                pthread_mutex_unlock(&mutex);
                                pos_f[in] += vel_f[in];
                                pos_c[in] += vel_c[in];
                                f_pil[in] = f_h;
                                c_pil[in] = c_h;   /* actualitza posicio actual */
                                pthread_mutex_lock(&mutex);
                                if (f_pil[in] != n_fil - 1)     /* si no surt del taulell, */
                                {
                                        win_escricar(f_pil[in], c_pil[in], 48+in, INVERS);
                                        /* imprimeix pilota on caracter que es passa es el codi ascii
                                        de 0+index*/
                                }
                                else{
                                        num_pilotes--;
                                        fi3=1;
                                }
                                pthread_mutex_unlock(&mutex);
                        }
                }
                else
                {       /* posicio hipotetica = a la real: moure */
                        pos_f[in] += vel_f[in];
                        pos_c[in] += vel_c[in];
                }
                pthread_mutex_lock(&mutex);
                fi2 = (nblocs==0 || num_pilotes==0);
                pthread_mutex_unlock(&mutex);
                win_retard(retard);
        } while(!fi1 && !fi2 && !fi3);
        return ((void *) index);
}
```

Función que se encarga del movimiento de la paleta:

```
void * mou_paleta(void * nul)
{
	int tecla, result;
	do{
	/* Bucle paleta */
		result = 0;
		pthread_mutex_lock(&mutex);
		tecla = win_gettec();
		pthread_mutex_unlock(&mutex);

		if (tecla != 0) {
			if ((tecla == TEC_DRETA) && ((c_pal + MIDA_PALETA) < n_col - 1))
			{
				pthread_mutex_lock(&mutex);
				win_escricar(f_pal, c_pal, ' ', NO_INV);   /* esborra primer bloc */
				c_pal++;                                   /* actualitza posicio */
				win_escricar(f_pal, c_pal + MIDA_PALETA - 1, '0', INVERS);
/*esc. ultim bloc */
				pthread_mutex_unlock(&mutex);
			}
			if ((tecla == TEC_ESQUER) && (c_pal > 1)) {
				pthread_mutex_lock(&mutex);
				win_escricar(f_pal, c_pal + MIDA_PALETA - 1, ' ', NO_INV);
/*esborra ultim bloc */
				c_pal--;                                   /* actualitza posicio */
				win_escricar(f_pal, c_pal, '0', INVERS);  /* escriure primer bloc */
				pthread_mutex_unlock(&mutex);
			}
			if (tecla == TEC_RETURN)
				result = 1;                                /* final per pulsacio RETURN */
			pthread_mutex_lock(&mutex);
			dirPaleta = tecla;                    /* per a afectar al moviment de les pilotes*/
			pthread_mutex_unlock(&mutex);
		}
		pthread_mutex_lock(&mutex);
		fi1 = result;
		pthread_mutex_unlock(&mutex);
		win_retard(5);
	} while(!fi1 && !fi2);

	return ((void *) 0);
}
```

Programa principal:

```
int main(int n_args, char *ll_args[])
{
	int i;
	FILE *fit_conf;
	if ((n_args != 2) && (n_args != 3)) {		/* si numero d'arguments incorrecte */
		i = 0;
		do
			fprintf(stderr, "%s", descripcio[i++]);		/* imprimeix descripcio */
		while (descripcio[i][0] != '*');		/* mentre no arribi al final */
		exit(1);
	}
	fit_conf = fopen(ll_args[1], "rt");  /* intenta obrir el fitxer */
	if (!fit_conf) {
		fprintf(stderr, "Error: no s'ha pogut obrir el fitxer \'%s\'\n",
			ll_args[1]);
		exit(2);
	}
	if (carrega_configuracio(fit_conf) != 0)	/* llegir dades del fitxer  */
		exit(3);  /* aborta si hi ha algun problema en el fitxer */

	if (n_args == 3) {		/* si s'ha especificat parametre de retard */
		retard = atoi(ll_args[2]);  /* convertir-lo a enter */
		if (retard < 10)
			retard = 10;	/* ver(sizeof members[0]) );ificar limits */
		if (retard > 1000)
			retard = 1000;
	} else
		retard = 100;	/* altrament, fixar retard per defecte */
	printf("Joc del Mur: prem RETURN per continuar:\n");
	getchar();
	if (inicialitza_joc() != 0)   /* intenta crear el taulell de joc */
		exit(4);  /* aborta si hi ha algun problema amb taulell */
	pthread_create(&tid[id],NULL, &mou_pilota , (intptr_t *) id); /* pilota1*/
	id++;
	pthread_create(&tid[0],NULL, &mou_paleta, (void *) NULL); /*paleta*/

	clock_t inici_temps = clock();		/* variable tipo tiempo para tiempo inicial */
	clock_t t_actual = clock();		/* variable tipo tiempo para tiempo actual */
	int segons = 0, minuts = 0;		/* variable segundos y minutos inicializados a 0 */
	char tiempo[LONGMISS];			/* variable 'String' para tiempo del tamaño
	maximo que se puede imprimir por pantalla */
	do {
		t_actual = clock();
		segons = ((((float) t_actual - (float) inici_temps)/CLOCKS_PER_SEC)*100)-60 *
		minuts;

		if (segons >= 60)
			minuts ++;
```

```
		pthread_mutex_lock(&mutex);
		sprintf(tiempo, "Tiempo: %02d:%02d", minuts, segons);
		win_escristr(tiempo);
		pthread_mutex_unlock(&mutex);
		win_retard(retard);			/* retard del joc */
		} while (!fi1 && !fi2);

		t_actual = clock();
		segons = ((((float) t_actual - (float) inici_temps)/CLOCKS_PER_SEC)*100)-60 *
minuts;
		if (segons >= 60)
		{
			minuts ++;
			segons = 0;
		}
		memset(tiempo, 0, sizeof tiempo);
		if (nblocs == 0)
		{
			sprintf(tiempo, "YOU WIN! Tiempo: %02d:%02d", minuts, segons);
			mostra_final(tiempo);
		}
		else
		{
			sprintf(tiempo, "GAME OVER, Tiempo: %02d:%02d", minuts, segons);
			mostra_final(tiempo);
		}
		win_fi();		/* tanca les curses */

		return (0);			/* retorna sense errors d'execucio */
}
```

1.5. Juego de pruebas

A continuación, se hará el juego de pruebas de la primera fase, el cual consistirá en comprobar el correcto funcionamiento del juego, a diferentes velocidades y con diferente número de pelotas.

Caso	Descripción de la prueba	Salida	¿Correcto?
1	Inicialización del juego.	Se inicializa correctamente, a pesar de diferentes errores de sincronización que se considerarán normales en esta fase.	Sí
2	Inicialización del juego cambiando el retardo.	Se inicializa correctamente, a menor retardo, mayor serà la velocidad de juego.	Sí
3	Inicialización del juego con retardo menor a 1 o mayor a 1000.	Se inicializa correctamente, usando el valor mínimo y màximo por defecto, 1 y 1000 respectivamente.	Sí
4	Inicialización del juego cambiando los tamaños del tablero.	Se inicializa correctamente, aumentando o disminuyendo el tamaño del tablero en función de la entrada.	Sí
5	Inicialización del juego cambiando la velocidad inicial de la pelota.	La pelota cambia correctamente de velocidad, dependiendo del parámetro inicial, cuanto más grande, más rápido irá.	Sí
6	Comprobación del tiempo de juego.	El tiempo de juego avanza correctamente de manera relativa en segundos.	Sí
7	Comprobación del movimiento de la paleta.	La paleta se mueve correctamente.	Sí
8	Comprobación del rebote de la pelota, golpeándola con la esquina derecha de la paleta.	La pelota rebota correctamente hacia la derecha.	Sí
9	Comprobación del rebote de la pelota, golpeándola con la esquina izquierda de la paleta	La pelota rebota correctamente hacia la izquierda.	Sí
10	Comprobación del rebote en	La pelota rebota	Sí

	los bloques tipo ‘####’.	correctamente.	
11	Eliminación de un bloque tipo ‘AAA’.	La pelota elimina correctamente el bloque y rebota en éste.	Sí
12	Eliminación de un bloque tipo ‘BBB’.	La pelota elimina el bloque, rebota y además, se crea una nueva.	Sí
13	Eliminación de varios bloques ‘BBB’.	Se crean varias pelotas, dependiendo del número de bloques ‘BBB’ golpeados, con diferentes velocidades y distinto número para identificarlas.	Sí
14	Comprobación del rebote entre pelotas.	Cuando una pelota choca con otra, se produce un rebote, haciendo que avancen en la dirección opuesta.	Sí
15	Salida de una pelota del campo, habiendo solo una en el juego.	Se finaliza el juego y aparece el mensaje de “GAME OVER”.	Sí
16	Salida de una pelota del campo, habiendo varias más jugando.	El jugador puede seguir usando las pelotas restantes.	Sí
17	Salida de todas las pelotas del campo.	Se finaliza el juego y aparece el mensaje de “GAME OVER”.	Sí
18	Finalización del juego dándole a la tecla “Enter”.	El juego finaliza independientemente del estado de éste.	Sí
19	Finalización del juego por falta de bloques.	Si no quedan más bloques, se considerará que el jugador ha ganado, se finalizará el juego y aparecerá el mensaje de “YOU WIN” en pantalla.	Sí

2. Fase 3 y 4

2.1. Enunciado resumido Fase 3

En esta fase se propone modificar el juego desarrollado en la Fase 2 de forma que se utilicen procesos en vez de hilos. Concretamente, cada pelota será controlada por un proceso hijo.

El código de cada proceso hijo residirá en un fichero ejecutable diferente del perteneciente al programa principal.

El código que deben ejecutar los procesos hijos para controlar las pelotas estará en otro fichero llamado *'pilota3.c'.*

Para acceder a pantalla se dispone de un nuevo grupo de rutinas definidas en *'winsuport2.h'*. Estas están adaptadas al uso de una misma ventana desde diferentes procesos.

La utilización de *winsuport2* debe ser la siguiente:

- Proceso padre (código del main en el mur3.c):
    - invoca a *win_ini()*: devuelve la medida en bytes que se tienen que reservar para almacenar el contenido del campo de juego.
    - crea una zona de memoria compartida de la misma medida a la devuelta por *win_ini()*.
    - incova a win_set(): esta función inicializa el contenido de la ventana de dibujo y le permite el acceso al proceso padre.
    - crea el thread de la paleta como en la fase 2 pero sin usar mutex.
    - crea el primer proceso hijo (primera pelota) pasándole como argumento la referencia de la memoria compartido del campo de juego, además de las dimensiones y otra información necesaria.
    - ejecuta un bucle principal donde periódicamente se actualiza por pantalla el contenido del campo de juego invocando la función *win_update()*.
    - una vez acabada la ejecución del programa incluyendo los procesos, se invoca a la función *win_fi()* y destruye la zona de memoria del campo de juego y los otros recursos compartidos que haya.
- Procesos hijos:
    - mapean la referencia de memoria compartida que contiene el campo de juego usando el identificador que se ha pasado desde el proceso padre.
    - invocan la función *win_set()* con la dirección de la zona de memoria mapeada y las dimensiones que se han pasado por argumento.

- utilizan todas las funciones de escritura y consulta del campo de juego *win_escricar()*, *win_escristr()* y *win_quincar()*.

Los procesos hijos no pueden usar la función *win_gettec()* ya que solamente es accesible por el proceso que ha inicializado el entorno de CURSES.

Para generar los ficheros se utilizarán los comandos siguientes:

```
$ gcc Wall mur3.c winsuport2.o o mur3 lcurses lpthread
$ gcc Wall pilota3.c winsuport2.o o pilota3 lcurses
```

2.2. Enunciado resumido Fase 4

En esta fase se deben establecer secciones críticas que eviten los problemas de concurrencia de la fase anterior. Además, hace falta implementar la funcionalidad de comunicación entre los procesos hijos y padre.
Para establecer secciones críticas se necesitan semáforos. La comunicación entre procesos se realizará mediante buzones de mensajes.

En este punto, se añadirán dos características al juego:

1. Cuando la pelota salga de la portería comunique a las otras pelotas su velocidad y coordenada.
   El resto de procesos pelota recibirán esta velocidad y aplicarán el siguiente algoritmo:
   - si la velocidad propia es menor a la recibida, cogerá la velocidad recibida incrementada un 25%.
   - si la velocidad propia es mayor se coge la recibida.

2. Cuando una pelota toque los bloques A (cambiad el símbolo de los bloques superiores del tablero de juego por una T) además de que desaparezca, debe iniciar un temporizador en 5 segundos. Durante el tiempo que dure el temporizador, las pelotas se deben dibujar en modo invertido y cuando toquen los bloques ### reboten borrando el carácter con el que ha chocado destruyendo de forma parcial la muralla.
   Mientras dure el temporizador si se vuelve a tocar un bloque T, se incrementará el tiempo que quedaba en 5 segundos más.

3. En la parte inferior de la pantalla se debe ir decrementando el temporizador para saber el tiempo que queda disponible.

2.3. Diseño

División del código en procesos:

Primero para comenzar la división del código en diferentes procesos se ha copiado el contenido de las funciones **mou_pilota()**, **comprovar_bloc()**, **control_impacte()** y **control_impacte2()** a un fichero nuevo llamado 'pilota3.c'/. Este será el fichero fuente que contendrá los procesos hijos que controlan las pelotas.
Todo el resto de código se ha dejado en el fichero 'mur3.c'.

mur3.c

Se le suprimen las funciones **mou_pilota()**, **comprovar_bloc()**, **control_impacte()** y **control_impacte2().**

pilota3.c

Se le volverá a definir todas las librerías, constantes y variables globales que se utilizan en el fichero 'mur3.c'. Esto permitirá el uso y modificación de estas en caso de que se traten de variables del proceso.

Habilitación del código a la ejecución en procesos:

mur3.c

Se cambian diferentes variables a variables en memoria compartida. Estas serán:

- num_pilotes: número de pelotas existente en el tablero. Debe ser de memoria compartida para que todos los procesos usen la variable actualizada.
- nblocs: número de bloques 'AAA' restantes en el tablero. Debe ser de memoria compartida para que todos los procesos usen la variable actualizada.
- c_pal: columna en la que se encuentra la paleta. Debe ser de memoria compartida para que los procesos de las pelotas tengan constancia de la actualización de la posición de la paleta.
- f_pal: fila en la que se encuentra la paleta. Debe ser de memoria compartida para que los procesos de las pelotas tengan constancia de la actualización de la posición de la paleta.
- dirPaleta: dirección que tiene la paleta y será tomada por la primera pelota que choque contra ella. Debe ser de memoria compartida para que los procesos de las pelotas tengan constancia de su valor y actualización y puedan usar estos datos.
- fi1: valor de condición de fin de juego, se da si se ha pulsado la tecla 'ENTER' del teclado.

- addr_tauler: dirección del tablero. Debe ser de memoria compartida para que todos los procesos tengan constancia del tablero.

La ejecución de la paleta sigue realizándose en un thread aparte.
En cuanto a la creación de procesos, se ha decidido que el programa principal del mur3.c solamente creará el proceso de la primera pelota. Para crearlos, se les pasa las variables de memoria compartida definidas anteriormente y las siguientes:

- n_fil : número de filas totales del campo de juego.
- n_col: número de columnas totales del campo de juego.
- vel_f: velocidad en el eje de las filas que toma la pelota al ser creada.
- vel_c: velocidad en el eje de las columnas que toma la pelota al ser creada.
- pos_f: posición en float de la fila en la que se crea la nueva pelota.
- pos_c: posición en float de la columna en la que se crea la nueva pelota.
- f_pil: número de fila en el que se debe crear la pelota nueva.
- c_pil: número de columna en el que se debe crear la pelota nueva.
- retard: retardo del programa introducido por el usuario.

Estos procesos se crean usando la función: **execlp()**.
Además de esto, se ha decidido añadir una línea que imprime en la consola de comandos en la que se ha ejecutado el programa si no ha sido posible crearse el proceso hijo.

Una vez se finaliza el bucle principal del juego, se ha añadido que este debe esperar a que todos los procesos hijos que hayan sido creados se finalicen por lo que se utiliza la función **wait_pid()**.

pilota3.c

Para este fichero nuevo, el main será el equivalente a la función **mou_pilota()**.

Se ha decidido que cada proceso de pelota creará otro proceso pelota si se choca con un bloque de tipo ‘BBB’ por lo que cada uno de estos debe ser capaz de crear procesos nuevos.
Para poder realizar esta función se ha añadido a la comprobación de bloque que si se choca un bloque de tipo ‘BBB’ se crea un proceso pasándole los mismos parámetros que al crearse un proceso desde el mur3.c

En cuanto a lo que realiza un proceso pelota al ser creado, lo primero es coger de la lista de parámetros recibidos lo que equivale a cada uno de ellos.
Una vez tiene todas las variables realizará, con todas las variables de memoria compartida, una función map_mem() pasándole la variable de tipo *int* recibida al ser creada que indica la posición de memoria de la variable.
Una vez realizados estos cambios se debe utilizar todas las variables de memoria compartida como punteros a la memoria por lo que siempre irán precedidas del carácter *****.

Cuando el bucle principal del proceso finalice este debe quedarse a la espera de la finalización de sus procesos hijos. En caso de que no tenga, no esperará y finalizará permitiendo que su proceso padre pueda finalizar a su vez.

Sincronización del código para la ejecución en procesos:

Para una correcta sincronización de procesos, se ha añadido antes y después de la actualización de cada variable de memoria compartida un waitS() y signalS() respectivamente. Con esto lograremos la sincronización entre los procesos entre sí, el thread de la paleta y el programa principal.
El semáforo será inicializado a 1 para permitir que simplemente un proceso pueda acceder a la variable y modificarla.

Con estos cambios se han actualizado los nombres de los ficheros a **'mur4.c'** y **'pilota4.c'**.

Resolución de la comunicación de la velocidad en el eje vertical entre pelotas :

pilota4.c

Para realizar la comunicación necesaria entre procesos de tipo pelota al salirse una de ellas por el espacio de la portería se ha utilizado buzones y pasos de mensajes.
En el momento en el que una pelota sale de la portería, se envía tantos mensajes como pelotas existentes haya en el tablero informando de la velocidad vertical que tenía esta. Además, se hace un signalS() para garantizar la exclusión mútua. Para asegurarnos de que cada pelota existente solamente coge un mensaje, se ha hecho que al leer un mensaje se haga un waitS() y además, que se quede en un bucle que no acaba hasta que todas las pelotas hayan leído un mensaje.

Para no permitir que otros procesos que no sean pelotas consuman los waitS() pertenecientes a estos se debe hacer ante cada waitS() un bucle que controle si el valor del semáforo es mayor a 1 o no. En caso positivo se esperará hasta que todas las pelotas hayan leído y consumido su valor en el semáforo.

Resolución temporizador:

Para ejecutar el temporizador y que funcionara de manera adecuada se ha hecho el conteo en un thread nuevo en el que se hace un sleep de 1 segundo. El contador se inicia a 5 cuando una pelota choca un bloque de tipo 'AAA'. y se va decrementando cada segundo.
Este variable será de memoria compartida para permitir su modificación en caso de chocar con un bloque de tipo 'TTT'.
Para aprovechar esta mecánica se ha usado el thread para contar además el tiempo del juego.
Esta variable compartida también se debe pasar por parámetro a cada proceso creado para que puedan añadir 5 segundos al tiempo o reiniciar el temporizador.

2.4. Implementación

A continuación se muestra de color azul los cambios realizados en el código, en las fases tres y cuatro.

Librerías añadidas:

```
#include <stdio.h>       /* incloure definicions de funcions estandard */
#include <stdint.h>      /* intptr_t for 64bits machines */
#include <stdlib.h>
#include <string.h>
#include <pthread.h>    /* incloure threads */
#include <unistd.h>      /* fork */
#include <time.h>
#include "winsuport2.h"    /* incloure definicions de funcions propies */
#include "memoria.h"
#include "semafor.h"
#include "missatge.h"
#include <sys/wait.h>
```

Declaración de variables globales:

```
pthread_t tid[MAX_THREADS];    /* taula d'identificadors de threads */
pid_t tpid[MAX_THREADS]; /* taula d'identificadors  de processos */
intptr_t id = 0;
int *num_pilotes, n_p;
int n_fil, n_col;            /* numero de files i columnes del taulell */
int m_por;                   /* mida de la porteria (en caracters) */
int *f_pal, *c_pal, f_p, c_p;      /* posicio del primer caracter de la paleta */
int f_pil, c_pil;            /* posicio de la pilota, en valor enter */
float pos_f, pos_c;          /* posicio de la pilota, en valor real */
float vel_f, vel_c;          /* velocitat de la pilota, en valor real */
int *identificador, id_pilota;
int *nblocs, n_b;
int *dirPaleta, dir_p;
int retard;                  /* valor del retard de moviment, en mil.lisegons */
int *fi1, fi_1, fi2;                  /* valor de condicions finals */
char strin[LONGMISS];        /* variable per a generar missatges de text */
int id_mem_tauler, *addr_tauler;
int id_sem, id_bustia, id_sem_pilotes;
int *temp=0, temporitzador, start_t, segons;
clock_t *start_time;
```

Función encargada de inicializar el juego:

```
int inicialitza_joc(void)
{
   int i, retwin;
   int i_port, f_port;                    /* inici i final de porteria */
   int c, nb, offset;

   retwin = win_ini(&n_fil, &n_col, '+', INVERS);    /* intenta crear taulell */

   if (retwin < 0)
   {                          /* si no pot crear l'entorn de joc amb les curses */
         fprintf(stderr, "Error en la creacio del taulell de joc:\t");
         switch (retwin)
         {
               case -1:
                     fprintf(stderr, "camp de joc ja creat!\n");
                     break;
               case -2:
                     fprintf(stderr,
                           "no s'ha pogut inicialitzar l'entorn de curses!\n");
                     break;
               case -3:
                     fprintf(stderr,
                     "les mides del camp demanades son massa grans!\n");
                     break;
               case -4:
                     fprintf(stderr, "no s'ha pogut crear la finestra!\n");
                     break;
         }
         return (retwin);
   }

   id_mem_tauler = ini_mem(retwin);
   addr_tauler = map_mem(id_mem_tauler);
   win_set(addr_tauler, n_fil, n_col);

   if (m_por > n_col - 2)
         m_por = n_col - 2;               /* limita valor de la porteria */
   if (m_por == 0)
         m_por = 3 * (n_col - 2) / 4;     /* valor porteria per defecte */

   i_port = n_col / 2 - m_por / 2 - 1;    /* crea el forat de la porteria */
   f_port = i_port + m_por - 1;
   for (i = i_port; i <= f_port; i++)
         win_escricar(n_fil - 2, i, ' ', NO_INV);
```

```
n_fil = n_fil - 1;              /* descompta la fila de missatges */
*f_pal = n_fil - 2;             /* posicio inicial de la paleta per defecte */
*c_pal = (n_col-MIDA_PALETA) / 2;                   /* a baix i centrada */
for (i = 0; i < MIDA_PALETA; i++)           /* dibuixar paleta inicial */
        win_escricar(*f_pal, *c_pal + i, '0', INVERS);

/* generar la pilota */
for (i = 1; i <= *num_pilotes; i++)
/* for per si volem iniciaŕ els valors de    més d'una pilota en params.txt */
{
        if (pos_f > n_fil - 1)
                pos_f = n_fil - 1;    /* limita posicio inicial de la pilota */
        if (pos_c > n_col - 1)
                pos_c = n_col - 1;
        f_pil = pos_f;
        c_pil = pos_c;                      /* dibuixar la pilota inicialment */
        win_escricar(f_pil, c_pil, 48+i, INVERS);
}

/* generar els blocs */
nb = 0;
*nblocs = n_col / (BLKSIZE + BLKGAP) - 1;
offset = (n_col - (*nblocs) * (BLKSIZE + BLKGAP) + BLKGAP) / 2;
 /* offset de columna inicial */
for (i = 0; i < *nblocs; i++)
{
        for (c = 0; c < BLKSIZE; c++)
        {
                win_escricar(3, offset + c, TEMPCHAR, INVERS);
                nb++;
                win_escricar(4, offset + c, BLKCHAR, NO_INV);
                nb++;
                win_escricar(5, offset + c, FRNTCHAR, INVERS);
                nb++;
        }
        offset += BLKSIZE + BLKGAP;
}
*nblocs = nb / BLKSIZE;
/* generar les defenses */
nb = n_col / (BLKSIZE + 2 * BLKGAP) - 2;
offset = (n_col - nb * (BLKSIZE + 2 * BLKGAP) + BLKGAP) / 2;
/* offset de columna inicial */
for (i = 0; i < nb; i++)
{
        for (c = 0; c < BLKSIZE + BLKGAP; c++)
                win_escricar(6, offset + c, WLLCHAR, NO_INV);
```

```
            offset += BLKSIZE + 2 * BLKGAP;
    }
    sprintf(strin, "Tecles: \'%c\'-> Esquerra, \'%c\'-> Dreta, RETURN->
    sortir\n", TEC_AMUNT, TEC_AVALL);
     win_escristr(strin);
     return (0);
}
```

Función nueva encargada del tiempo:

```
void * temporitza(void * nul)
{
    do{
            waitS(id_sem);
            if((*temp)!=0)
            {
                    (*temp)--;
            }
            segons++;
            signalS(id_sem);
            sleep(1);
    } while(!(*fi1) && !fi2);
    return ((void *) 0);
}
```

Función de mover la paleta:

```
void * mou_paleta(void * nul)
{
    int tecla, result;
    do
    {       /* Bucle paleta */
            result = 0;
            while(sem_value(id_sem)>1);
            waitS(id_sem);
            tecla = win_gettec();
            signalS(id_sem);

            if (tecla != 0) {
                if ((tecla == TEC_DRETA)&&((*c_pal+MIDA_PALETA)<n_col - 1))
                    {
                            while(sem_value(id_sem)>1);
```

```
                waitS(id_sem);
                win_escricar(*f_pal,*c_pal, ',NO_INV);
                /* esborra primer bloc */
                (*c_pal)++;             /* actualitza posicio */
                win_escricar(*f_pal,*c_pal+MIDA_PALETA-1, '0', INVERS);
                /*esc. ultim bloc */
                signalS(id_sem);
            }
            if ((tecla == TEC_ESQUER) && (*c_pal > 1)) {
                while(sem_value(id_sem)>1);
                waitS(id_sem);
                win_escricar(*f_pal, *c_pal+MIDA_PALETA-1,' ', NO_INV);
                /*esborra ultim bloc */
                (*c_pal)--;                     /* actualitza posicio */
                win_escricar(*f_pal, *c_pal, '0', INVERS);
                /* escriure primer bloc */
                signalS(id_sem);
            }
            if (tecla == TEC_RETURN)
                result = 1;     /* final per pulsacio RETURN */
            while(sem_value(id_sem)>1);
            waitS(id_sem);
            *dirPaleta = tecla; /* per a afectar al moviment de les pilotes*/
            signalS(id_sem);
        }
        while(sem_value(id_sem)>1);
        waitS(id_sem);
        *fi1 = result;
        fi2 = ((*nblocs==0) || (*num_pilotes==0));
        signalS(id_sem);
        win_retard(5);
    } while(!(*fi1) && !fi2);
    return ((void *) 0);
}
```

Programa principal:

```
int main(int n_args, char *ll_args[])
{
/* inicialitzacio variable compartida num_pilotes*/
n_p = ini_mem(sizeof(int));/* crear zona mem. compartida */
num_pilotes = map_mem(n_p);/* obtenir adreça mem. compartida */
*num_pilotes = 1;/* inicialitza variable compartida */

/* inicialitzacio variable compartida nblocs*/
n_b = ini_mem(sizeof(int));
nblocs = map_mem(n_b);
*nblocs = 0;

/* inicialitzacio variable compartida c_pal*/
c_p = ini_mem(sizeof(int));
c_pal = map_mem(c_p);

/* inicialitzacio variable compartida f_pal*/
f_p = ini_mem(sizeof(int));
f_pal = map_mem(f_p);

/* inicialitzacio variable compartida dirPaleta*/
dir_p = ini_mem(sizeof(int));
dirPaleta = map_mem(dir_p);
*dirPaleta = 0;

/* inicialitzacio variable compartida fi1*/
fi_1 = ini_mem(sizeof(int));
fi1 = map_mem(fi_1);
*fi1 = 0;

/* inicialitzacio variable compartida temp*/
temporitzador = ini_mem(sizeof(int));
temp = map_mem(temporitzador);
*temp = 0;

/* inicialitzacio variable compartida identificador*/
id_pilota = ini_mem(sizeof(int));
identificador = map_mem(id_pilota);
*identificador = 1;

/* inicialitzacio variable compartida start_time*/
start_t = ini_mem(sizeof(int));
start_time = map_mem(start_t);
*start_time=0;
```

```
int i;
FILE *fit_conf;

id_sem = ini_sem(1);              /* creacio semafor */
if ((n_args != 2) && (n_args != 3)) /* si numero d'arguments incorrecte */
{
        i = 0;
        do
                fprintf(stderr, "%s", descripcio[i++]); /* imprimeix descripcio */
         while (descripcio[i][0] != '*');    /* mentre no arribi al final */
         exit(1);
}
fit_conf = fopen(ll_args[1], "rt");    /* intenta obrir el fitxer */
if (!fit_conf)
{
        fprintf(stderr, "Error: no s'ha pogut obrir el fitxer \'%s\'\n", ll_args[1]);
        exit(2);
}
if (carrega_configuracio(fit_conf) != 0)    /* llegir dades del fitxer  */
        exit(3);    /* aborta si hi ha algun problema en el fitxer */
if (n_args == 3) /* si s'ha especificat parametre de retard */
{
    retard = atoi(ll_args[2]);    /* convertir-lo a enter */
    if (retard < 10)
          retard = 10;    /* ver(sizeof members[0])) ; identificar limits */
    if (retard > 1000)
          retard = 1000;
} else
     retard = 100;    /* altrament, fixar retard per defecte */

printf("Joc del Mur: prem RETURN per continuar:\n");
getchar();

if (inicialitza_joc() != 0)    /* intenta crear el taulell de joc */
     exit(4);    /* aborta si hi ha algun problema amb taulell */

char id_str[SIZE_ARRAY], fil_str[SIZE_ARRAY], col_str[SIZE_ARRAY];
char id_mem_tauler_str[SIZE_ARRAY], vel_f_str[SIZE_ARRAY],
char vel_c_str[SIZE_ARRAY];
char f_pil_str[SIZE_ARRAY], c_pil_str[SIZE_ARRAY];
char pos_f_str[SIZE_ARRAY], pos_c_str[SIZE_ARRAY],
char id_pilota_str[SIZE_ARRAY];
char nblocs_str[SIZE_ARRAY], npils_str[SIZE_ARRAY],
char retard_str[SIZE_ARRAY];
char c_pal_str[SIZE_ARRAY], f_pal_str[SIZE_ARRAY],
```

```
char dirPaleta_str[SIZE_ARRAY], fi1_str[SIZE_ARRAY];
char id_sem_str[SIZE_ARRAY], id_bustia_str[SIZE_ARRAY],
char temp_str[SIZE_ARRAY], start_str[SIZE_ARRAY];

id_bustia = ini_mis();              /* creacio bustia IPC */
segons=0;

pthread_create(&tid[0],NULL, &mou_paleta, (void *) NULL);
pthread_create(&tid[1],NULL, &temporitza, (void *) NULL);

tpid[0] = fork();
if (tpid[0] == (pid_t)0)   /* Es tracta del proces fill */
{
        sprintf(id_str, "%d", 1);
        sprintf(id_mem_tauler_str, "%d", id_mem_tauler);
        sprintf(fil_str, "%d", n_fil);
        sprintf(col_str, "%d", n_col);
        sprintf(vel_f_str, "%f", vel_f);
        sprintf(vel_c_str, "%f", vel_c);
        sprintf(pos_f_str, "%f", pos_f);
        sprintf(pos_c_str, "%f", pos_c);
        sprintf(f_pil_str, "%d", f_pil);
        sprintf(c_pil_str, "%d", c_pil);
        sprintf(nblocs_str, "%d", n_b);
        sprintf(npils_str, "%d", n_p);
        sprintf(retard_str, "%d", retard);
        sprintf(c_pal_str, "%d", c_p);
        sprintf(f_pal_str, "%d", f_p);
        sprintf(dirPaleta_str, "%d", dir_p);
        sprintf(fi1_str, "%d", fi_1);
        sprintf(id_sem_str, "%d", id_sem);
        sprintf(id_bustia_str, "%d", id_bustia);
        sprintf(temp_str, "%d", temporitzador);
        sprintf(start_str, "%d", start_t);
        sprintf(id_pilota_str, "%d", id_pilota);

        execlp("./pilota4", "pilota4", id_str, id_mem_tauler_str, fil_str,
                col_str, vel_f_str, vel_c_str, pos_f_str, pos_c_str, f_pil_str,
                c_pil_str, nblocs_str, npils_str, retard_str, c_pal_str, f_pal_str,
                dirPaleta_str, fi1_str, id_sem_str, id_bustia_str, temp_str,
                start_str, id_pilota_str, (char *)0);
                fprintf(stderr,"Error: No puc executar el proces fill\'pilota3\'\n");
                exit(1);  /* Retornem error */
} else if (tpid[0] <  0 )  /* ERROR*/
        {
                fprintf(stderr, "Hi ha hagut un error en la creacio del proces");
```

```
        }

int minuts = 0;             /* variable segundos y minutos inicializados a 0 */
char tiempo[LONGMISS]; /* variable 'String' para tiempo del tamaño maximo que se puede
imprimir por pantalla */
do
{
   if (segons >= 60)
   {
         waitS(id_sem);
         minuts ++;
         segons=0;
         signalS(id_sem);
    }
    while(sem_value(id_sem)>1);
    waitS(id_sem);
    if ((*temp)>0)
    {
         sprintf(tiempo,"Tiempo:%02d:%02d\t\tTemp:%02d",minuts,segons,
        (*temp));
    }
   //memset(tiempo, 0, sizeof tiempo);
    else
          sprintf(tiempo, "Tiempo: %02d:%02d", minuts, segons);
          win_escristr(tiempo);
          //pthread_mutex_unlock(&mutex);
          win_update();
          signalS(id_sem);
          win_retard(100);        /* retard del joc */

} while (!(*fi1) && !((*nblocs) == 0 || (*num_pilotes) == 0));
int stat;
win_update();
pthread_join(tid[0], 0);
pthread_join(tid[1], 0);
waitpid(tpid[0], &stat, 0);


 if ( *nblocs == 0)
 {
     sprintf(tiempo, "YOU WIN! Tiempo: %02d:%02d", minuts, segons);
     mostra_final(tiempo);
 }else
 {
     sprintf(tiempo, "GAME OVER, Tiempo: %02d:%02d", minuts, segons);
     mostra_final(tiempo);
```

```
  }
  win_update();

  elim_mem(id_mem_tauler);
  elim_mem(n_p);
  elim_mem(n_b);
  elim_mem(c_p);
  elim_mem(f_p);
  elim_mem(dir_p);
  elim_mem(temporitzador);
  elim_mem(start_t);
  elim_mem(fi_1);
  elim_mem(id_pilota);
  elim_sem(id_sem);
  elim_mis(id_bustia);
  win_fi();                    /* tanca les curses */
  return (0);       /* retorna sense errors d'execucio */
}
```

A continuación se verán los cambios realizados en un archivo.c nuevo, “pilota4.c”. Cuyas funciones són **comprovar_bloc()**, **control_impacte()**, **control_impacte2(), recep_miss()** y el main de éste.

Función comprobar_bloque:

```
void comprovar_bloc(int f, int c)
{
	int col;
	char quin = win_quincar(f, c);

	if (quin == WLLCHAR && (*temp) !=0)
	{
		win_escricar(f, c, ' ', NO_INV);
	}

	if (quin == BLKCHAR || quin == FRNTCHAR || quin == TEMPCHAR)
	{
		col = c;
		while (win_quincar(f, col) != ' ')
		{
			win_escricar(f, col, ' ', NO_INV);
			col++;
		}
		col = c - 1;
		while (win_quincar(f, col) != ' ')
		{
			win_escricar(f, col, ' ', NO_INV);
			col--;
		}

		if (quin == FRNTCHAR)
			(*temp) = 5;

		if (quin == TEMPCHAR)
		{
			if ((*temp)!=0)
				(*temp) = (*temp) + 5;
		}

		if (quin == BLKCHAR)
		{
			char id_str[SIZE_ARRAY], fil_str[SIZE_ARRAY],
			col_str[SIZE_ARRAY];
			char id_mem_tauler_str[SIZE_ARRAY], vel_f_str[SIZE_ARRAY],
			vel_c_str[SIZE_ARRAY];
```

```
char f_pil_str[SIZE_ARRAY], c_pil_str[SIZE_ARRAY];
char pos_f_str[SIZE_ARRAY], pos_c_str[SIZE_ARRAY],
id_pilota_str[SIZE_ARRAY];
char nblocs_str[SIZE_ARRAY], npils_str[SIZE_ARRAY],
retard_str[SIZE_ARRAY];
char c_pal_str[SIZE_ARRAY], f_pal_str[SIZE_ARRAY],
dirPaleta_str[SIZE_ARRAY], fi1_str[SIZE_ARRAY];
char id_sem_str[SIZE_ARRAY], id_bustia_str[SIZE_ARRAY],
temp_str[SIZE_ARRAY], start_str[SIZE_ARRAY];

(*id)++;
tpid[num_fills] = fork();

if (tpid[num_fills] == 0)   /* Es tracta del proces fill */
{
   sprintf(id_str, "%d", (*id));
   sprintf(id_mem_tauler_str, "%d", id_mem_tauler);
   sprintf(fil_str, "%d", n_fil);
   sprintf(col_str, "%d", n_col);
   sprintf(vel_f_str, "%f", (float)rand()/(float)(RAND_MAX/2)-1);
   sprintf(vel_c_str, "%f", (float)rand()/(float)(RAND_MAX/2)-1);
   sprintf(pos_f_str, "%d", f);
   sprintf(pos_c_str, "%d", c);
   sprintf(f_pil_str, "%d", f);
   sprintf(c_pil_str, "%d", c);
   sprintf(nblocs_str, "%d", n_b);
   sprintf(npils_str, "%d", n_p);
   sprintf(retard_str, "%d", retard);
   sprintf(c_pal_str, "%d", c_p);
   sprintf(f_pal_str, "%d", f_p);
   sprintf(dirPaleta_str, "%d", dir_p);
   sprintf(fi1_str, "%d", fi_1);
   sprintf(id_sem_str, "%d", id_sem);
   sprintf(id_bustia_str, "%d", id_bustia);
   sprintf(temp_str, "%d", temporitzador);
   sprintf(start_str, "%d", start_t);
   sprintf(id_pilota_str, "%d", id_pilota);

    execlp("./pilota4", "pilota4", id_str, id_mem_tauler_str, fil_str,
    col_str, vel_f_str, vel_c_str, pos_f_str, pos_c_str, f_pil_str,
    c_pil_str, nblocs_str, npils_str, retard_str, c_pal_str, f_pal_str,
    dirPaleta_str, fi1_str,  id_sem_str, id_bustia_str, temp_str,
    start_str, id_pilota_str, (char *)0);
    fprintf(stderr, "Error: No puc executar el proces fill \'pilota4\' \n");
    exit(1);  /* Retornem error */
}
```

```
                else if (tpid[num_fills] <  0 )     /* ERROR*/
                {
                        fprintf(stderr, "Hi ha hagut un error en la creacio del proces\n");
                }
                        (*num_pilotes)++;
                        num_fills++;
                }
        (*nblocs)--;
        }
}
```

Función de control de impacto:

```
void control_impacte(void)
{
        int i;
        for(i = 1; i <=  *num_pilotes; i++)
        {
                if (*dirPaleta == TEC_DRETA)
                {
                        if (vel_c <= 0.0)                /* pilota cap a l'esquerra */
                                vel_c = -vel_c - 0.2;    /* xoc: canvi de sentit i reduir velocitat */
                        else
                        {                                /* a favor: incrementar velocitat */
                                if (vel_c <= 0.8)
                                        vel_c += 0.2;
                        }
                }
                else
                {
                        if (*dirPaleta == TEC_ESQUER)
                        {
                                if (vel_c >= 0.0)                /* pilota cap a la dreta */
                                        vel_c = -vel_c + 0.2;    /* xoc: canvi de sentit i reduir la
velocitat */
                                else
                                {                                /* a favor: incrementar velocitat */
                                        if (vel_c >= -0.8)
                                                vel_c -= 0.2;
                                }
                        }
                        else
                        {                                        /* XXX trucs no documentats */
                                if (*dirPaleta == TEC_AMUNT)
                                        vel_c = 0.0;                            /* vertical */
                                else
                                {
```

```
                                        if (*dirPaleta == TEC_AVALL)
                                                if (vel_f <= 1.0)
                                                        vel_f -= 0.2;           /* frenar */
                                }
                        }
                }
                waitS(id_sem);
                (*dirPaleta)=0;                         /* reset perque ja hem aplicat l'efecte */
                signalS(id_sem);
        }
}
```

Función de control de impacto2:

```
float control_impacte2(int c_pil, float velc0)
{
        int distApal;
        float vel_c;

        distApal = c_pil - (intptr_t) *c_pal;
        if (distApal >= 2*MIDA_PALETA/3)                        /* costat dreta */
                vel_c = 0.5;
        else if (distApal <= MIDA_PALETA/3)                     /* costat esquerra */
                vel_c = -0.5;
        else if (distApal == MIDA_PALETA/2)                             /* al centre */
                vel_c = 0.0;
        else                                                    /*: rebot normal */
                vel_c = velc0;
        return vel_c;
}
```

Función que se encarga de recibir los mensajes por las pelotas:

```
void recep_missatges()
{
        float vel_f_n;

        waitS(id_sem);
        receiveM(id_bustia, &vel_f_n);
        if (vel_f_n > vel_f)
        {
                vel_f = vel_f_n * 1.25;
                if (vel_f>1)
                        vel_f=1;
                if (vel_f < -1)
                        vel_f=-1;
```

```
        }
        else if (vel_f_n < vel_f)
                vel_f = vel_f_n;

                while(sem_value(id_sem)>1);
}
```

Función que se encarga de recibir los mensajes por las pelotas:

```
int main(int n_args, char *ll_args[])
//void * mou_pilota(void * ind)
{
        num_fills = 0;
        int fi2 = 0, fi3 = 0;
        /* Parsing arguments */
        ind = atoi(ll_args[1]);
        id_mem_tauler = atoi(ll_args[2]);
        n_fil = atoi(ll_args[3]);
        n_col = atoi(ll_args[4]);
        vel_f = atof(ll_args[5]);
        vel_c = atof(ll_args[6]);
        pos_f = atof(ll_args[7]);
        pos_c = atof(ll_args[8]);
        f_pil = atoi(ll_args[9]);
        c_pil = atoi(ll_args[10]);
        n_b = atoi(ll_args[11]);
        n_p =  atoi(ll_args[12]);
        retard = atoi(ll_args[13]);
        c_p = atoi(ll_args[14]);
        f_p = atoi(ll_args[15]);
        dir_p = atoi(ll_args[16]);
        fi_1 = atoi(ll_args[17]);
        id_sem = atoi(ll_args[18]);
        id_bustia = atoi(ll_args[19]);
        temporitzador = atoi(ll_args[20]);
        start_t = atoi(ll_args[21]);
        id_pilota = atoi(ll_args[22]);

        void * addr_tauler = map_mem(id_mem_tauler);
        win_set(addr_tauler, n_fil, n_col);

        num_pilotes = map_mem(n_p);
        nblocs = map_mem(n_b);
        c_pal = map_mem(c_p);
        f_pal = map_mem(f_p);
        dirPaleta = map_mem(dir_p);
```

```
fi1 = map_mem(fi_1);
temp = map_mem(temporitzador);
start_time = map_mem(start_t);
id = map_mem(id_pilota);

int f_h, c_h;
float vel_f_n;
char rh, rv, rd, no;

do                                                          /* Bucle pelota */
{

        f_h = pos_f + vel_f;            /* posicio hipotetica de la pilota (entera) */
        c_h = pos_c + vel_c;
        rh = rv = rd = ' ';
        if ((f_h != f_pil) || (c_h != c_pil))
        {
        /* si posicio hipotetica no coincideix amb la posicio actual */
                if (f_h != f_pil)                               /* provar rebot vertical */
                {
                        if(sem_value(id_sem)>1)
                                recep_missatges();

                        waitS(id_sem);
                        rv = win_quincar(f_h, c_pil);   /* veure si hi ha algun obstacle */
                        signalS(id_sem);
                        if (rv != ' ')                  /* si hi ha alguna cosa */
                        {
                                if(sem_value(id_sem)>1)
                                        recep_missatges();
                                waitS(id_sem);
                                comprovar_bloc(f_h, c_pil);
                                signalS(id_sem);
                                if (rv == '0')          /* col.lisió amb la paleta? */
                                {
                        /* XXX: tria la funció que vulgis o implementa'n una millor */
                                        control_impacte();
                                        vel_c = control_impacte2(c_pil, vel_c);
                                }
                                vel_f = -vel_f;         /* canvia sentit velocitat vertical */
                                f_h = pos_f + vel_f;    /* actualitza posicio hipotetica */
                        }
                }

                if (c_h != c_pil) /* provar rebot horitzontal */
                {
```

```
        if(sem_value(id_sem)>1)
                recep_missatges();
        waitS(id_sem);
        rh = win_quincar(f_pil, c_h);   /* veure si hi ha algun obstacle */
        signalS(id_sem);

        if (rh != ' ') /* si hi ha algun obstacle */
        {
                if(sem_value(id_sem)>1)
                        recep_missatges();
                waitS(id_sem);
                comprovar_bloc(f_pil, c_h);
                signalS(id_sem);
                /* TODO?: tractar la col.lisio lateral amb la paleta */
                vel_c = -vel_c;/* canvia sentit vel. horitzontal */
                c_h = pos_c + vel_c;  /* actualitza posicio hipotetica */
        }
}

if ((f_h != f_pil) && (c_h != c_pil)) /* provar rebot diagonal */
{
        if(sem_value(id_sem)>1)
                recep_missatges();
        waitS(id_sem);
        rd = win_quincar(f_h, c_h);
        signalS(id_sem);
        if (rd != ' ') /* si hi ha obstacle */
        {
                if(sem_value(id_sem)>1)
                        recep_missatges();
                waitS(id_sem);
                comprovar_bloc(f_h, c_h);
                signalS(id_sem);
                /* TODO?: tractar la col.lisio amb la paleta */
                vel_f = -vel_f;
                vel_c= -vel_c;  /* canvia sentit velocitats */
                f_h = pos_f + vel_f;
                c_h = pos_c + vel_c;  /* actualitza posicio entera */
        }
}

/* mostrar la pilota a la nova posició */
if(sem_value(id_sem)>1)
        recep_missatges();
waitS(id_sem);
no = win_quincar(f_h, c_h);
```

```
                signalS(id_sem);
                if (no == ' ')
                {       /* verificar posicio definitiva *//* si no hi ha obstacle */
                        if(sem_value(id_sem)>1)
                                recep_missatges();
                        waitS(id_sem);
                        win_escricar(f_pil, c_pil, ' ', NO_INV);          /* esborra pilota */
                        signalS(id_sem);
                        pos_f += vel_f;
                        pos_c += vel_c;
                        f_pil = f_h;
                        c_pil = c_h;    /* actualitza posicio actual */
                        if(sem_value(id_sem)>1)
                                recep_missatges();
                        waitS(id_sem);
                        if (f_pil != n_fil - 1)      /* si no surt del taulell, */
                        {
                                if ((*temp) == 0)
                                        win_escricar(f_pil, c_pil, 48+ind, INVERS);

                /* imprimeix pilota on caracter que es passa es el codi ascii de 0+ind*/
                                else
                                        win_escricar(f_pil, c_pil, 48+ind, NO_INV);
                /* imprimeix pilota invertida*/

                        }
                        else
                        {
                                (*num_pilotes)--;
                                fi3 = 1;
                        }
                        signalS(id_sem);
                }
        }
        else
        {       /* posicio hipotetica = a la real: moure */
                pos_f += vel_f;
                pos_c += vel_c;
        }
        if(sem_value(id_sem)>1)
                recep_missatges();
        waitS(id_sem);
        fi2 = ((*nblocs) == 0 || (*num_pilotes) == 0);
        signalS(id_sem);
        win_retard(retard);

} while(!fi3 && !fi2 && !*(fi1));
```

```
        /* fer bucle fins que la pilota surti de la porteria i llavors acabar el proces */
        int i;
        int stat;
        vel_f_n = vel_f;

        for (i=0; i<=*num_pilotes-1; i++)
        {
                signalS(id_sem);
                sendM(id_bustia, &vel_f_n, sizeof vel_f_n);
        }

        for (i=0; i <= num_fills; i++){
                waitpid(tpid[i], &stat, 0);
        }

        return (fi3);
}
```

2.5. Juego de pruebas

Caso	Descripción de la prueba	Salida	¿Correcto?
1	Inicialización del juego con diferentes parámetros.	Se inicializa correctamente.	Sí
2	Inicialización del juego cambiando la velocidad del *params.txt*	Se inicializa correctamente cambiando la velocidad.	Sí
3	Inicialización del juego cambiando el retardo.	Se inicializa correctamente, a menor retardo, mayor serà la velocidad de juego.	Sí
4	Inicialización del juego con retardo menor a 1 o mayor a 1000.	Se inicializa correctamente, usando el valor mínimo y màximo por defecto, 1 y 1000 respectivamente.	Sí
5	Comprobación de la sincronización general pelota-paleta.	Tanto la paleta como la pelota se mueven simultáneamente.	Sí
6	Comprobación del movimiento de la paleta.	La paleta se mueve correctamente.	Sí
7	Comprobación del rebote de la pelota.	La pelota rebota correctamente cambiando de dirección dependiendo de la posición del choque con la paleta.	Sí
8	Eliminación de los bloques ‘AAA’.	Cuando la pelota choca con dicho bloque, se elimina y provoca un rebote en la pelota, además se incrementa el contador para poder eliminar los bloques tipo ‘###’.	Sí
9	Eliminación de los bloques ‘BBB’.	Se elimina el bloque y se crea una nueva pelota, independiente de la anterior.	Sí
10	Eliminación de los bloques ‘TTT’.	Se elimina el bloque, provoca un rebote en la pelota, además se incrementa el contador en 5 seg. para poder eliminar los	Sí

		bloques tipo ‘###’.	
11	El contador empieza en 5s al chocar con un bloque tipo ‘AAA’.	Correcto.	Sí
12	El contador suma 5s al chocar con un bloque tipo ‘TTT si esta el temporizador activado.	Correcto.	Sí
13	Cambio de color en la pelota.	Cuando tenemos el contador activado por haber chocado contra un bloque tipo ‘AAA’, el color de la pelota se invierte.	Sí
14	Cambio de color en todas las pelotas del juego.	Cuando el contador está activado, se invierten todos los colores las pelotas.	Sí
15	Choque con bloque ‘###’ sin haber activado nunca el contador.	La pelota rebota.	Sí
16	Choque con bloque ‘###’ con el contador activado.	La pelota rebota y el carácter desaparece.	Sí
17	Choque con un bloque ‘###’ después de pararse el contador.	La pelota rebota.	Sí
18	Choque con los laterales del tablero cuando el contador está activado.	No elimina nada.	Sí
19	Choque con la paleta cuando el contador está activado.	La pelota rebota dependiendo de donde haya chocado.	Sí
20	Choque entre pelotas cuando el contador está activado.	Las pelotas chocan y rebotan entre ellas.	Sí
21	Choque entre dos pelotas.	Provoca un rebote y las pelotas cambian de dirección.	Sí
22	Eliminación de la pelota cuando se sale fuera del tablero.	El juego finaliza si ya no quedan más pelotas en el tablero y se muestra el mensaje de ‘GAME OVER’.	Sí

23	Eliminación de la pelota cuando se sale fuera del tablero y hay más dentro del juego.	Se puede seguir jugando con normalidad.	Sí
24	Eliminación de la pelota cuando está el contador activado.	El juego se finaliza.	Sí
25	Eliminación de la pelota cuando está el contador activado y hay más pelotas creadas.	Se puede seguir jugando con normalidad.	Sí
26	Creación de pelotas después de que se haya salido una.	Las siguientes pelotas creadas, tendrán su número sin importar la pelota fuera de juego.	Sí
27	Cambio en la velocidad de la pelota cuando una se ha salido del juego.	Las pelotas cambian dependiendo de su velocidad y la velocidad de la pelota que ha salido del juego.	Sí
28	Parar el juego a mitad de partida.	El juego finaliza, mostrando por pantalla el tiempo total de juego	Sí
29	Cuando no quedan más pelotas en partida.	El juego finaliza, mostrando por pantalla ‘Game Over’ y el tiempo total de juego.	Sí
30	Finalización del juego cuando ya no quedan más bloques.	El juego finaliza, mostrando por pantalla un mensaje de “YOU WIN” más el tiempo de la partida.	Sí
31	Finalización del juego cuando ya no quedan bloques normales pero sí del tipo ‘###’.	El juego finaliza, mostrando por pantalla un mensaje de “YOU WIN” más el tiempo de la partida.	Sí