Pràctica 2

Objectiu

L'objectiu d'aquesta pràctica és aprendre conceptes bàsics relacionats amb la programació amb múltiples fils d'execució (*multithreading*).

Grups de pràctiques

Els grups de pràctiques seran de dues o tres persones.

Entorn de treball

Les implementacions s'han de poder executar sobre el servidor *bsd.lab.deim (**bsd**)*.

Lliuraments

El lliurament de la pràctica 2 (fitxers font) cal fer-lo a través de l'aplicació MOODLE. La data límit dels lliuraments en primera i segona convocatòries es poden consultar a l'enllaç 'Distribució de classes' a l'espai *Moodle* de l'assignatura.

Correcció

La correcció de la pràctica 2.1 i 2.2 es realitzarà, en primera convocatòria, durant la sessió del laboratori, mitjançant una entrevista amb un professor i tots els membres del grup de pràctiques. L'entrevista tindrà una durada aproximada de 30 minuts. Les dates d'entrevista s'especificaran a la pàgina Moodle de l'assignatura, així com la manera de concertar hora. A més, **es realitzarà una verificació automàtica de copies entre tots els programes presentats**. Si es detecta alguna còpia, tots els alumnes involucrats (els qui han copiat i els que s'han deixat copiar) tindran la pràctica suspesa i, per extensió, l'assignatura suspesa.

La documentació de la pràctica es la habitual, és a dir: *Especificacions, Disseny, Implementació* i *Joc de proves*. En l'apartat de joc de proves NO s'han d'incloure "fotos de pantalla" del programa, ja que la impressió sobre paper dels resultats no demostra que la implementació presentada realment funcioni. El que cal fer és enumerar (redactar) tots els casos possibles que el programa és capaç de tractar. Les proves presentades i altres afegides pel professor, s'executaran al *bsd* durant l'entrevista. El professor realitzarà preguntes sobre el contingut de l'informe i el funcionament dels programes presentats, també pot preguntar com es farien algunes modificacions sobre la pràctica lliurada. Cada membre del grup tindrà una nota individual, que dependrà del nivell de coneixements demostrat durant l'entrevista.

Enunciat

Es tracta d'implementar un rudimentari joc del "Tron", fent servir un terminal de caràcters ASCII (a través de la llibreria '*curses'*). Sobre un camp de joc rectangular, es mouen uns objectes que anomenarem 'trons' (amb o tancada). En aquesta primera versió del joc, només hi ha un tron que controla l'usuari, i que representarem amb un '0', i un tron que controla l'ordinador, el qual es representarà amb un '1'. Els trons son una espècie de 'motos' que quan es mouen deixen rastre (el caràcter corresponent). L'usuari pot canviar la direcció de moviment del seu tron amb les tecles: 'w' (adalt), 's' (abaix), 'd' (dreta) i 'a' (esquerra). El tron que controla l'ordinador es mourà, canviant de direcció aleatòriament segons un paràmetre del programa (veure Arguments). El joc consisteix en que un tron intentarà 'tancar' a l'altre tron. El primer tron que xoca contra un obstacle (sigui rastre seu o de l'altre tron), esborrarà tot el seu rastre, escriure informació de com ha anat la partida i acabarà, perdent la partida.

Dins el servidor *bsd* es proporciona una versió inicial del programa '`tron0.c`' on l'ordinador només presenta un tron, que anomenarem 'tron oponent', per distingir-lo del tron de l'usuari. Per tal d'aprendre programació *multithreading*, es proposa modificar aquesta versió inicial de forma que l'ordinador pugui moure més d'un tron simultàniament. La idea principal és que la funció que controla el moviment del tron s'adapti per a ser executada com un fil independent o *thread*. Així, des del programa principal només caldrà crear tants fils d'execució com trons oponents es vulguin (fins a 9).

Fases.

Per tal de facilitar el disseny de la pràctica es proposen 3 fases:

0. Programa seqüencial (`'tron0.c'`):

Versió inicial amb un sol tron oponent. Aquesta versió es proporciona dins del següent directori del servidor *bsd*:`/usr/local/assignatures/fso/practica2`

1. Creació de threads (`'tron1.c'`):

Partint de l'exemple anterior, modificar les funcions que controlen el moviment dels trons (usuari i oponent) per a què puguin actuar com a fils d'execució independents. Així el programa principal crearà un fil pel jugador i varis fils per als trons oponents.

2. Sincronització de threads (`'tron2.c'`):

Completar la fase anterior de manera que es sincronitzi l'execució dels fils a l'hora d'accedir als recursos compartits (per exemple, l'entorn de dibuix). D'aquesta manera s'han de solucionar els problemes de visualització que presenta la versió anterior, els quals es fan més evidents quan s'intenta executar el programa sobre el servidor *bsd*.

Fase 0.

Dins del directori "`/usr/local/assignatures/fso/practica2.1`" del servidor *sol* estan disponibles els següents fitxers: `'tron0.c', 'winsuport.c' i 'winsuport.h'. A més al Moodle disposareu d'aquests fitxers. Els podreu trobar en el paquet 'practica2.1.tar'.`

El primer que cal fer és copiar aquests fitxers dins d'un directori propi al servidor *bsd* (atenció, cal substituir la paraula *login* pel vostre identificador d'usuari):

```
$   ssh bsd –l login
$   mkdir Practica2FSO
$   cp /usr/local/assignatures/fso/practica2.1/* Practica2FSO
$   exit
```

Per tal de generar la visualització del joc es proporcionen una sèrie de rutines dins del fitxer '`winsuport.c`', que utilitzen una llibreria d'UNIX anomenada '*curses*'. Aquesta llibreria permet escriure caràcters ASCII en posicions específiques de la pantalla (fila, columna). El fitxer '`winsuport.h`' inclou les definicions i les capçaleres de les rutines implementades, així com la descripció del seu funcionament:

```
int  win_ini(int *fil, int *col, char creq, unsigned int inv);
void win_fi();
void win_escricar(int f, int c, char car, unsigned int invers);
char win_quincar(int f, int c);
int  win_quinatri(int f, int c);
void win_escristr(char *str);
int  win_gettec(void);
void win_retard(int ms);
```

Cal invocar la funció `win_ini` abans d'utilitzar qualsevol de les altres funcions. El número de files i columnes de la finestra de dibuix, es passaran per referència i, en el cas de tenir com a valor 0, la funció els actualitzarà amb els valors màxims de la terminal. L'última fila de la finestra de dibuix no formarà part del camp de joc, ja que es reserva per escriure missatges. La resta de files es reserva per a tauler de joc. Es dibuixa un requadre a tot el tauler de joc, format per caràcters creq, que es mostren invertits si el paràmetre inv es diferent de zero.

El codi ASCII servirà per identificar el tipus d'element del camp de joc (espai lliure, paret, rastre tron oponent, rastre usuari). La funció `win_quincar` permet consultar el codi ASCII d'una posició determinada, mentre que `win_quinatri` serà l'encarregada de dir-nos si el caràcter es o no INVERS. La funció `win_escristr` serveix per escriure un missatge en l'última línia de la finestra de dibuix. Per llegir una tecla es pot invocar la funció `win_gettec`.

Es recomana utilitzar les tecles definides en el fitxer de capçalera '`winsuport.h`' per especificar el moviment del jugador. La funció `win_retard` permet aturar l'execució d'un procés o d'un fil durant el número de mil·lisegons especificat per paràmetre.

El programa d'exemple '`tron0.c`' utilitza les funcions anteriors per dibuixar el camp de joc i controlar el moviment del tron usuari i del tron oponent. Per generar l'executable primer cal compilar el fitxer '`winsuport.c`', per tal d'obtenir un fitxer objecte '`winsuport.o`' que contindrà les instruccions en codi màquina de les rutines de dibuix:

```
$  gcc -c winsuport.c -o winsuport.o
```

La comanda anterior només cal invocar-la una vegada per cada màquina on s'hagin d'executar les rutines de dibuix. És a dir, cal generar un fitxer '`winsuport.o`' per al LINUX i un altre per al *bsd*. Després ja podem generar el fitxer executable '`tron0`' corresponent al joc del tron bàsic. La següent comanda s'encarrega de fer això a partir del fitxer font '`tron0.c`', el fitxer objecte anterior i de la llibreria '`curses`':

```
$  gcc tron0.c winsuport.o -o tron0 -lcurses
```

Ara ja es pot executar el programa. Abans però, s'han de conèixer els arguments que cal passar-li. El primer argument serveix per a controlar la variabilitat del canvi de direcció del tron oponent: s'ha de proporcionar com a primer argument un número del 0 al 3, el qual indicarà si els canvis s'han de produir molt freqüentment (3 es el màxim) o poc freqüent, on 0 indicarà que només canviarà per esquivar les parets. Com a segon argument es passarà el nom d'un fitxer on anirem guardant informació de la partida realitzada. A més, opcionalment, es podrà afegir un segon argument per a indicar, el temps que trigarà tant del tron usuari com dels trons oponents controlats per l'ordinador en realitzar cada moviment (en ms). El valor per defecte d'aquest paràmetre es de 100 (1 dècima de segon).

```
$  ./tron0 variabilitat [retard]
```

Així, la comanda per executar el programa amb un tron oponent que variï la seva direcció només per a evitar les parets i 200 mil·lisegons de retard de moviment és la següent:

```
$  ./tron0 0 fitxer 200
```

Fase 1.

En aquesta fase cal modificar les funcions anteriors de moviment del tron usuari i del tron oponent per tal que puguin funcionar com a fils d'execució independents. Les funcions bàsiques de creació (i unió) de *threads* s'expliquen a la sessió L5 dels laboratoris d'FSO. A més, cal fer una sèrie de canvis addicionals per a què tot funcioni correctament. A continuació es proposen una sèrie de passos que divideixen la feina a fer en petites tasques. Abans de fer cap modificació, cal fer una copia del programa original (i canviar els comentaris de la capçalera):

```
$   cp tron0.c tron1.c
```

Pas 1.1: La primera lectura del programa.

Abans que res, cal donar-li una ullada general a la versió inicial del programa `'tron0.c'`, per tal d'entendre la seva estructura.

Primer cal observar les estructures de dades i variables globals més importants, és a dir, les que defineixen els objectes de moviment:

```
typedef struct {        /* per un objecte (usuari o oponent) */
      int f;                  /* posició actual: fila */
      int c;                  /* posició actual: columna */
      int d;                  /* direcció actual: [0..3] */
} objecte;

objecte usu;            /* informació del usuari */
objecte opo;            /* informació del oponent */
```

Després observem que el programa està organitzat en les següents funcions:

```
void esborrar_posicions(pos *p_pos[], int n_pos);
void inicialitza_joc(void);
int mou_oponent(void);
int mou_usuari(void);
```

Finalment, ens fixarem en l'estructura bàsica del programa principal, el qual utilitza les funcions anteriors per inicialitzar la finestra de dibuix i, si tot ha funcionat bé, executar el bucle principal, el qual activa periòdicament les funcions per moure el tron usuari i moure el tron oponent. El programa acaba quan s'ha premut la tecla `RETURN` o quan un dels dos trons xoca contra un objecte. (`'fi1'` o `'fi2'` diferents de zero):

```
int main(int n_args, char *ll_args[])
{
  retwin = win_ini(&n_fil,&n_col,'+',INVERS);  /* intenta crear taulell */
  if (retwin > 0)      /* si aconsegueix accedir a l'entorn CURSES */
  {
    inicialitza_joc();
    fi1 = 0; fi2 = 0;
    do                 /********** bucle principal del joc **********/
    {
      fi1 = mou_usuari();
      fi2 = mou_oponent();
      win_retard(retard);
    } while (!fi1 && !fi2);
    win_fi();
  }
}
```

Pas 1.2: Modificar la funció de moviment dels trons oponents.

Com que hem d'adaptar la funció **`mou_oponent`** per a que es pugui executar com un fil d'execució independent, el primer que cal fer és canviar la capçalera:

```
void * mou_oponent(void * index)
```

El valor que es passa pel paràmetre `index` serà un enter que indicarà l'ordre de creació del tron (0 -> primer tron, 1 -> segon tron, etc.). Aquest paràmetre servirà per accedir a una taula global d'informació dels oponents, així com per imprimir el caràcter corresponent ('1' pel primer tron, '2' pel segon, etc.). De moment, però, NO utilitzarem el paràmetre, ja que farem una primera versió de la funció per manegar un únic tron oponent com un fil d'execució independent.

Al contrari que en la fase 0, la nova funció de moviment s'ha d'executar de manera independent al bucle principal del programa. Això implica que cal implementar el seu propi bucle per generar el moviment dels objectes. Per finalitzar l'execució d'aquest bucle de moviment, es pot consultar les variables `'fi1'` i `'fi2'`, que indiquen alguna condició de final de joc (usuari mort, oponent mort, tecla `RETURN`). Per tant, aquestes variables ara han de ser globals (no locals al `main`), i s'actualitzaran directament des de dins de les funcions de moviment del tron oponent i del tron usuari.

Pas 1.3: Modificar la funció de moviment del tron usuari.

Igual que en el cas anterior, s'ha d'adaptar la funció **`mou_usuari`** per a que es pugui executar com un fil d'execució independent. La nova capçalera serà la següent, on el paràmetre `nulo`

no conté cap informació:

```
void * mou_usuari(void * nulo)
```

Una altra vegada, caldrà crear un bucle independent de moviment per al tron usuari, amb les mateixes condicions d'acabament generals.

Pas 1.4: Modificar la funció principal del programa.

Després cal modificar la funció **`main`**, creant els fils d'execució pertinents un cop s'hagi inicialitzat correctament l'entorn del joc. El seu bucle principal ara NO ha d'invocar a les funcions de moviment de l'usuari i oponent, donat que ja s'executen de manera concurrent, sinó que ha d'executar una tasca independent fins que es doni alguna condició de finalització. De moment, però, el bucle principal NO farà res; simplement es passarà a esperar a què tots els fils creats acabin la seva execució, per després destruir l'entorn de dibuix.

Arribats a aquest punt ja podem provar els canvis efectuats, encara que de moment només es mourà un sol oponent. El fitxer executable s'anomenarà `'tron1'`. Per a generar-lo, s'ha d'invocar la següent comanda, la qual crida al compilador de C passant-li la referència de la llibreria que conté el codi de les funcions per treballar amb multithreading (`'pthread'`):

```
$  gcc tron1.c winsuport.o -o tron1 -lcurses -lpthread
```

Pas 1.5: Modificar les variables globals.

Per controlar els múltiples oponents, s'ha de convertir la variable global `'opo'` en un vector o taula de màxim 9 posicions (es recomana fer servir una constant per fixar aquest límit). En aquesta, cada tron oponent podrà disposar de la seva informació. Serà la mateixa que indiquem en el punt 1.2

Pas 1.6: Modificar la inicialització del joc. ***

Després cal modificar la funció **`inicialitza_joc`** per a què col·loqui tant al tron usuari com tots els trons oponents a les posicions inicials a més de mostrar-los per primera vegada en pantalla. La posició inicial de tron usuari serà la mateixa que en la `'tron0.c'`, mentre que els trons oponents mantindran la columna però es repartiran, de forma equidistant per les diferents files d'aquesta, es a dir: ((n_fil-1)/ (num_oponents+1). Finalment, la direcció inicial de moviment serà aleatòria.

Pas 1.7: Acabar la funció de moviment dels oponents.

Un cop ja tenim creada la taula amb la informació de cada oponent, cal completar la funció **`mou_oponent`** per a que pugui gestionar les dades del tron oponent que se li assignarà a través del paràmetre `index`.

Pas 1.8: Acabar la funció principal del programa.

Finalment, cal completar la funció main per arrancar els múltiples fils corresponents a tots els oponents, a més del fil corresponent a l'usuari. El nombre de trons oponents que haurà d'executar el nostre programa, serà indicat com a argument en la crida a aquest. Els valors aniran de 1 a 9:

```
$   ./tron1 num_oponents fitxer variabilitat [retard]
```

A més, cal modificar el bucle principal del programa per tal que realitzi una tasca independent, que en el nostre cas consistirà en controlar el temps de joc (`minuts:segons`) i mostrar-lo per la línia de missatges. L'última acció del programa principal serà la d'escriure el temps total de joc per la sortida estàndard, juntament amb els missatges de finalització proposats en la fase anterior.

ATENCIÓ: la implementació de la fase 1 no realitza cap mena de sincronisme entre els fils. Per tant, és possible que apareguin errors ocasionals a causa de l'execució concurrent i descontrolada d'aquests fils. Això no obstant, el propòsit d'aquesta fase és veure que s'executen tots els fils requerits en el pas per paràmetre (fins a 9).

Fase 2.

En aquesta fase cal establir seccions crítiques que evitin els problemes de concurrència de la fase anterior. Per fer això caldrà incloure semàfors per fer l'exclusió mútua entre els fils. Les funcions bàsiques de manipulació de mutex per sincronitzar els *threads* s'expliquen a la L6 dels laboratoris de FSO.

Les seccions crítiques han de servir per impedir l'accés concurrent a determinats recursos per part dels diferents fils d'execució. En el nostre cas, cal establir seccions crítiques per accedir a l'entorn de dibuix, apart d'alguna variable global que s'hagués de protegir contra els accessos concurrents desincronitzats.

Cal tenir en compte que quan un tron, sigui usuari o oponent, xoqui haurà d'eliminar la seva traça però durant aquest procés, la resta de trons continuaran la seva execució, contràriament a com passava en la versió 'tron0.c'. Caldrà contemplar la possibilitat que mentre un tron (usuari o oponent) retira la seva traça, si un altre tron (usuari o oponent) xoca també morirà i, per tant, caldrà contemplar la possibilitat d'empat si escau.

El fitxer executable s'anomenarà `'tron2'`. Per validar el seu funcionament, caldrà executar-lo sobre el *bsd*.

Per tal d'agilitar el procés de desenvolupament de la pràctica 2 (i també de la 3), es recomana la creació d'un fitxer de text anomenat `'Makefile'`, el qual ha d'estar ubicat en el mateix directori que els fitxers font a compilar. El contingut d'aquest fitxer hauria de ser similar al següent:

```
$ vi Makefile
tron2 : tron2.c winsuport.o winsuport.h
	gcc tron2.c winsuport.o -o tron2 -lcurses -lpthread
winsuport.o : winsuport.c winsuport.h
	gcc -c winsuport.c -o winsuport.o
```

D'aquesta manera s'estableixen les dependències entre els fitxers de sortida (en aquest cas `'tron2'` i `'winsuport.o'`) i els fitxers font dels quals depenen, a més d'especificar la comanda que s'ha d'invocar en cas que la data de modificació d'algun dels fitxers font sigui posterior a la data de l'última modificació del fitxer a generar. Aquest control el realitza la comanda `make`. Per exemple, podem realitzar les següents peticions:

```
$   make winsuport.o
$   make tron2
$   make
```

Si no s'especifica cap paràmetre, `make` verificarà les dependències del primer fitxer de sortida que trobi dins del fitxer `'Makefile'` del directori de treball actual.