Pràctica 2.1

Objectiu.

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

Grups de pràctiques.

Els grups de pràctiques seran de dos o tres alumnes.

Entorn de treball.

Les implementacions s'han d'executar sobre el servidor *bsd* i en la distribució Linux dels laboratoris de l'assignatura.

Lliuraments.

Es faran dos lliuraments de la pràctica al moodle. El primer correspondrà a les fases 1 i 2, i el segon a les fases 3 i 4. Heu de seguir estrictament totes les instruccions donades especificades a les tasques del lliurament del moodle. La data límit de lliurament en primera convocatòria és la mateixa per a tots els grups de laboratori (veure l'entrada Distribució de classes dins l'espai Moodle de l'assignatura).

Correcció.

La revisió i correcció de la pràctica 2 es realitzarà mitjançant una entrevista entre el professor i tots els membres del grup de pràctiques. L'entrevista final tindrà una durada aproximada de 20 minuts. Les dates d'entrevista s'especificaran a l'espai moodle de l'assignatura, així com la manera de concertar hora.

El dia de l'entrevista cal portar un informe escrit on es documenti la pràctica de la manera habitual, és a dir: *Especificacions, Disseny, Implementació* i *Joc de proves*. En l'apartat de joc de proves NO s'han d'incloure "pantallazos" del programa, ja que la impressió sobre paper dels resultats no demostra que la implementació 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 sobre un ordinador com els dels laboratoris durant l'entrevista.

El professor realitzarà preguntes sobre el contingut de l'informe, del codi que presenteu i del funcionament dels programes. Cada membre del grup tindrà una nota individual, que dependrà del nivell de coneixements demostrat durant l'entrevista.

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.

Enunciat.

Es tracta d'implementar un rudimentari joc de tennis amb caràcters ASCII: es dibuixa un camp de joc rectangular amb una "porteria" a cada costat, una paleta per a l'usuari, una paleta per a l'ordinador i una pilota que va rebotant sobre els elements del camp de joc. L'usuari disposa de dues tecles per controlar la seva paleta, mentre que l'ordinador mou la seva paleta automàticament (amunt i avall). Evidentment, es tracta d'intentar fer sortir la pilota per la porteria de l'ordinador (porteria de la dreta), abans que la pilota surti per la porteria de l'usuari (porteria de l'esquerra).

Per tal d'aprendre programació *multithreading*, es proposa modificar el joc bàsic de forma que l'ordinador pugui moure més d'una paleta simultàniament. La idea principal és que caldrà programar una rutina pel control automàtic d'una paleta d'ordinador, la qual s'executarà com un fil independent o *thread*. Així, des del programa principal només caldrà crear tants fils d'execució com paletes de l'ordinador es vulguin.

Fases.

Per tal de facilitar el disseny de la pràctica es proposen 5 fases:

0. Programa seqüencial (`tennis0.c`):

Es tracta d'implementar el joc bàsic, és a dir, una única paleta per l'ordinador. Aquesta versió es proporciona com a exemple de programa escrit en llenguatge C, dins el paquet `exemples_P2.tar` que hi ha a l'espai moodle de l'assignatura.

1. Creació de threads (`tennis1.c`):

Partint de l'exemple anterior, modificar les funcions que controlen el moviment de la pilota i de les paletes per a que puguin actuar com a fils d'execució independents. Així el programa principal ha de crear un fil per a la pilota, un fil per a la paleta de l'usuari i varis fils per a les paletes de l'ordinador.

2. Sincronització de threads (`tennis2.c`):

Completar la fase anterior de manera que l'execució dels fils a l'hora d'accedir als recursos compartits (entorn de dibuix, variables globals, etc.) es sincronitzi. D'aquesta manera s'han d'evitar els problemes d'accés concurrent al mateix recurs per part de diversos fils d'execució. També afegirem una finalització del joc després d'un nombre concret de moviments de les paletes.

3. Creació de processos (`tennis3.c` / `pal_ord3.c`):

Partint del codi de la fase anterior, realitzar les modificacions necessàries per convertir els *threads* que controlen les paletes de l'ordinador en processos independents. El control de la pilota i de la paleta de l'usuari continuarà com a

threads del procés pare. Així, el programa principal crearà un *thread* per a la pilota, un *thread* per a la paleta de l'usuari i varis processos per a les paletes de l'ordinador. En aquesta fase no es demana cap mena de sincronització ni comunicació entre els processos que intervenen en el joc (la sincronització entre *threads* no funciona entre processos).

4. Sincronització de processos (`tennis4.c' /pal_ord4.c'`):

Completar la fase anterior de manera que l'execució dels processos es sincronitzi 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 fase anterior, els quals es fan patents quan s'intenta executar el programa sobre el servidor remot *bsd*. En aquesta fase també s'ha d'implementar la interacció entre la pilota i les paletes d'ordinador que provoqui el desplaçament horitzontal de les mateixes.

Fase 0.

Dins del directori "`exemples_P2`" estan disponibles (entre d'altres) els següents fitxers:

`camp1.txt, camp2.txt, camp3.txt, tennis0.c, winsuport.c i winsuport.h`

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 n_fil, int n_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);
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, indicant el número de files i columnes de la finestra de dibuix, així com el caràcter per emmarcar el camp de joc. L'última fila de la finestra de dibuix no formarà part del camp de joc, ja que es reserva per escriure missatges.
- En acabar el programa, cal invocar la funció `win_fi`.
- Per escriure un caràcter en una fila i columna determinades cal utilitzar la funció `win_escricar`. El paràmetre `invers` permet ressaltar el caràcter amb els atributs de vídeo invertits (definicions `INVERS` o `NOINV`). El codi ASCII servirà per identificar el tipus d'element del camp de joc (espai lliure `' '`, paret `'+'`, pilota `'.'`, paleta usuari `'0'`, primera paleta ordinador `'1'`).
- La funció `win_quincar` permet consultar el codi ASCII d'una posició determinada.
- 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`, utilitzant les tecles definides en el fitxer de capçalera '`winsuport.h`' per especificar el moviment de la paleta de l'usuari.
- Finalment, la rutina `win_retard` permet aturar l'execució d'un procés o d'un fil d'execució durant el número de mil·lisegons especificat per paràmetre.

El programa d'exemple `tennis0.c` utilitza les funcions anteriors per dibuixar el camp de joc i controlar el moviment de les paletes i de la pilota. 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 obtenir el fitxer '`winsuport.o`' corresponent a la màquina on s'han d'executar les rutines de dibuix. Després ja podem generar el fitxer executable '`tennis0`' corresponent al joc del tennis bàsic. La següent comanda s'encarrega de fer això a partir del fitxer font '`tennis0.c`', el fitxer objecte anterior i de la llibreria '`curses`':

```
$   gcc tennis0.c winsuport.o –o tennis0 –lcurses
```

Ara ja es pot executar el programa. Abans però, s'han de conèixer els arguments que cal passar-li. El primer argument consisteix en el nom d'un fitxer de text que conté tots els paràmetres de configuració del joc. El format d'aquest fitxer ha de ser el següent:

```
n_fil n_col m_por l_pal
pil_pf pil_pc pil_vf pil_vc
ipo_pf ipo_pc po_vf
```

on `n_fil`, `n_col` defineixen les dimensions del taulell de joc, `m_por` és la mida de les dues porteries, `l_pal` és la longitud de les dues paletes; `pil_pf`, `pil_pc` defineixen la posició inicial (fila,columna) i `pil_vf`, `pil_vc` defineixen les velocitats inicials (vertical i horitzontal) de la pilota; i finalment, `ipo_pf`, `ipo_pc` indiquen la posició del primer caràcter de la paleta de l'ordinador, mentre que la seva velocitat vertical ve determinada pel paràmetre `po_fv`.

Un exemple de fitxer de configuració és el '`camp1.txt`':

```
$   cat camp1.txt
20 50 9 4
10 35 0.5 -1.0
8 46 0.6
```

Opcionalment, es pot especificar un segon argument per indicar el retard de moviment de la pilota i la paleta de l'ordinador (en mil·lisegons). Si no s'especifica res, el valor per defecte d'aquest argument és 100 (una dècima de segon).

La comanda per executar el programa amb el fitxer de configuració '`camp1.txt`' i 50 mil·lisegons de retard de moviment és la següent:

```
$   ./tennis0 camp1.txt 50
```

L'estructura bàsica del programa principal és la següent:

```
int main(int n_args, char *ll_args[])
{
  n_fil = win_ini(n_fil,n_col,'+',INVERS);     /* intenta crear taulell */
  if (n_fil > 0)        /* si aconsegueix accedir a l'entorn CURSES */
  {
    inicialitza_joc();
    do                  /********** bucle principal del joc **********/
    {   tec = win_gettec();
        if (tec != 0) mou_paleta_usuari(tec);
        mou_paleta_ordinador();
        cont = moure_pilota();
        win_retard(retard);
    } while ((tec != TEC_RETURN) && (cont==-1));
    win_fi();
}
```

Bàsicament el programa inicialitza la finestra de dibuix i executa el bucle principal, el qual activa periòdicament les funcions per moure les paletes de l'usuari i de l'ordinador, així com la funció per moure la pilota. El programa acaba quan s'ha premut la tecla `RETURN` o quan la pilota ha sortit del camp de joc per alguna de les porteries.

Fase 1.

En aquesta fase cal modificar les funcions anteriors de moviment de la pilota i de les paletes per a que puguin funcionar com a fils d'execució independents. Les funcions bàsiques de creació (i unió) de *threads* s'expliquen a la sessió 5 dels laboratoris. Les capçaleres de les noves funcions de moviment han de ser les següents:

```
void * moure_pilota(void * cap);
void * mou_paleta_usuari(void * cap);
void * mou_paleta_ordinador(void * index);
```

En el cas de `mou_paleta_ordinador`, el valor que es passa pel paràmetre `index` serà un enter que indicarà l'ordre de creació de la paleta (0 -> primera paleta, 1 -> segona paleta, etc.). Cada paleta s'escriurà com un caràcter invertit amb un codi ASCII diferent ('1' per la primera, '2' per la segona, etc.).

Per a les funcions `moure_pilota`, `mou_paleta_usuari` el paràmetre `cap` no contindrà cap informació.

Al contrari que en la fase 0, les tres funcions de moviment han d'executar-se de manera independent al bucle principal del programa (*main thread*). Això implica que cada funció ha d'implementar el seu propi bucle per generar el moviment dels objectes. Per finalitzar l'execució dels fils de moviment, es poden fer servir variables globals per indicar si la pilota ha sortit per alguna de les porteries o si s'ha premut la tecla `RETURN`.

Per controlar les múltiples paletes de l'ordinador, s'han de convertir les variables globals de posició i de velocitat de la paleta en un vector. Lògicament, s'ha de modificar la funció de càrrega dels paràmetres de configuració del joc per a que pugui inicialitzar fins a 9 paletes de l'ordinador. Els paràmetres de cada paleta s'especificaran en línies de text consecutives. El nou format del fitxer de configuració és el següent (compatible amb el format anterior):

```
n_fil n_col m_por l_pal
pil_pf pil_pc pil_vf pil_vc
ipo_pf1 ipo_pc1 po_vf1
ipo_pf2 ipo_pc2 po_vf2
...
ipo_pfn ipo_pcn po_vfn
```

Com a exemples de fitxers de configuració es proporcionen '`camp2.txt`' i '`camp3.txt`':

```
$ cat camp3.txt
24 78 15 5
10 35 -0.5 1.0
8 75 0.6
16 73 0.8
2 68 -0.4
```

També s'ha de modificar la funció d'inicialitzar el joc per dibuixar la posició inicial de totes les paletes carregades. El fil d'execució principal del programa s'ha d'encarregar de crear tots els fils de moviment i passar a executar una tasca independent fins que es doni alguna condició de finalització.

En el nostre cas, la tasca independent consistirà en controlar el temps de joc (`minuts:segons`) i mostrar-lo per la línia de missatges (última línia del camp de joc). Quan la pilota surti per alguna porteria o es premi la tecla `RETURN`, el fil d'execució principal sortirà del bucle i passarà a esperar a què tots els fils finalitzin la seva execució abans de destruir l'entorn de dibuix. L'última acció del fil principal serà la d'escriure el temps total de joc per la sortida estàndard, juntament amb els missatges de finalització proposats anteriorment.

El fitxer executable s'anomenarà `tennis1'` . 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 tennis1.c winsuport.o –o tennis1 –lcurses –lpthread
$  ./tennis1 camp2.txt 50
```

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 fitxer de configuració (fins a 9 fils).

Fase 2.

En aquesta fase cal establir seccions crítiques que evitin problemes de concurrència entre els múltiples fils d'execució. Per fer això caldrà incloure semàfors per tal d'establir seccions crítiques en l'accés a l'entorn de dibuix i algunes variables globals compartides. Les funcions bàsiques de manipulació de semàfors per sincronitzar els *threads* s'expliquen a la sessió 6 dels laboratoris.

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.

També es vol afegir el comptatge total dels moviments de les paletes de l'ordinador i de l'usuari. És a dir, cada vegada que una paleta de l'ordinador canviï la seva posició entera (x, y) dins el taulell de joc, una variable global s'ha d'incrementar. Quan el número total de moviments superi un cert valor màxim, el programa ha d'acabar la seva execució. En la línia de missatges del taulell de joc s'ha de visualitzar el número de moviments realitzats i el nombre de moviments que falten per a què s'acabi el joc.

El fitxer executable s'anomenarà '`tennis2`'. El número màxim de moviments es passarà com a segon argument de línia de comandes (després del nom de fitxer de configuració del taulell). El tercer argument, opcional, serà el valor del retard (si no s'especifica, retard=100). Per

exemple, per carregar el fitxer 'camp3.txt' amb un màxim de 100 moviments i un retard de 50 mil·lisegons, hauriem d'invocar el programa de la següent forma:

```
$   ./tennis2 camp3.txt 100 50
```

Per validar el seu funcionament, caldrà executar-lo sobre el *bsd*. Donat que el retard de xarxa evidencia més els problemes de sincronisme.

Per tal d'agilitzar el procés de desenvolupament de la pràctica 2, es recomana la utilització d'un fitxer de text anomenat 'Makefile', el qual ha d'estar ubicat en el mateix directori que els fitxers font a compilar.

Per permetre la compilació de la nova fase, es poden afegir les següents linies:

```
tennis2 : tennis2.c winsuport.o winsuport.h
      gcc tennis2.c winsuport.o –o tennis2 –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 'tennis2' 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 de sortida. Aquest control el realitza la comanda *make*. Per exemple, podem realitzar les següents peticions:

```
$ make winsuport.o
$ make tennis2
$ 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.

Fase 3.

L'objectiu d'aquesta fase és aprendre conceptes bàsics relacionats amb la programació amb múltiples processos (multiprocés), per tant, es proposa modificar el joc desenvolupat a la fase 2 de forma que ara s'utilitzin processos en lloc de *threads*. Concretament, cada paleta de l'ordinador serà controlada per un procés fill. El codi d'aquests processos fill haurà de residir en un fitxer executable diferent del fitxer exectuable del programa principal (també anomenat programa pare).

Les funcions bàsiques de creació de processos, més la gestió de zones de memòria compartida, s'expliquen a la sessió 7 dels laboratoris d'FSO.

A més del fitxer font principal 'tennis3.c' (procés pare), caldrà crear un altre fitxer font 'pal_ord3.c' que contindrà el codi que han d'executar els processos fill per controlar les paletes de l'ordinador.

Per accedir a pantalla es disposa d'un nou grup rutines definides a 'winsuport2.h'. Aquestes rutines estan adaptades a l'ús d'una mateixa finestra des de diferents processos. El seu funcionament és diferent de l'anterior versió de winsuport per a *threads*. Es disposa d'un exemple d'utilització als fitxers 'multiproc2.c' i 'mp_car2.c'.

A grans trets, la utilització de `winsuport2` ha de ser el següent:

- **Procés pare**:
  - invoca a `win_ini()`: la nova implementació de la funció retorna la mida en bytes que cal reservar per emmagatzemar el contingut del camp de joc.
  - crea una zona de memòria compartida de mida igual a la retornada per `win_ini()`.
  - invoca a `win_set()`: aquesta funció inicialitza el contingut de la finestra de dibuix i permet l'accés al procés pare.
  - crea els processos fill passant-los com argument la referència (identificador IPC) de la memòria compartida del camp de joc, a més de les dimensions (files, columnes) i altres informacions necessàries.
  - executa un bucle principal on, periòdicament (p. ex. cada 100 mil·lisegons) actualitza per pantalla el contingut del camp de joc, invocant la funció `win_update()`.
  - un cop acabada l'execució del programa (inclosos tots els processos), invoca la funció `win_fi()` i destrueix la zona de memòria del camp de joc.
- **Processos fill**:
  - mapejen la referència de memòria compartida que conté el camp de joc, utilitzant la referència (identificador) que s'ha passat per argument des del procés pare.
  - invoquen la funció `win_set()` amb l'adreça de la zona de memòria mapejada anteriorment i les dimensions (files, columnes) que s'han passat per argument.
  - utilitzen totes les funcions d'escriptura i consulta del camp de joc `win_escricar()`, `win_escristr()` i `win_quincar()`.

  Els processos fill, però, no poden fer servir la funció `win_gettec()`, la qual només és accessible al procés que ha inicialitzat l'entorn de CURSES (el procés pare).

El fitxer executable principal s'anomenarà `tennis3'` i el de les paletes d'ordinador s'anomenarà `pal_ord3'`. Per generar-los, s'han d'utilitzar les següents comandes:

```
$ gcc tennis3.c winsuport2.o –o tennis3 –lcurses –lpthread
$ gcc pal_ord3.c winsuport2.o –o pal_ord3 –lcurses
```

L'execució es farà de manera similar a la pràctica anterior, invocant l'executable `tennis3'` amb els paràmetres corresponents al fitxer de configuració del joc (una fila per les dimensions més una fila per la pilota més una fila per cada paleta de l'ordinador), el nombre màxim de moviments de les paletes i un valor opcional de retard.

Fase 4.

En aquesta fase cal establir seccions crítiques que evitin els problemes de concurrència de la fase anterior. També cal implementar la funcionalitat de comunicació entre els diversos processos fill (i potser també amb el procés pare), per practicar amb el mecanisme de comunicació entre processos. Per establir seccions crítiques cal utilitzar semàfors. La comunicació entre processos es realitzarà mitjançant bústies de missatges. Les funcions bàsiques de manipulació de semàfors i de bústies per sincronitzar i comunicar els processos s'expliquen a la sessió 8 dels laboratoris de FSO.

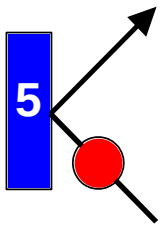
En aquesta última fase, cal implementar una nova funcionalitat de desplaçament horitzontal de les paletes de l'ordinador. En el nombre de moviments totals de la paleta, s'han de tenir en compte els moviments horitzontals d'aquestes.

A partir d'ara cal controlar les següents situacions:

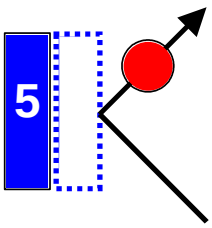
- quan la pilota reboti amb una paleta de l'ordinador, la paleta s'hauria de desplaçar una columna cap a l'esquerra o la dreta, segons per on s'hagi produït l'impacte de la pilota; si no hi ha cap obstacle, la paleta afectada es desplaça a la columna corresponent,
- si hi ha algun obstacle que impedeixi el canvi de columna (pared del camp de joc, una altra paleta), la paleta original es mantindrà a la seva columna, però indicarà a la paleta (o paletes) que l'estigui obstaculitzant que es desplacin una columna en el sentit del rebot original (transmissió de moviment entre paletes d'ordinador),
- a la vegada, si aquesta/es segona/es paleta/es trobessin un altre "obstacle", li transmetrien el moviment,
- si una paleta s'ha de desplaçar fins l'última columna del camp de joc (on es troba la porteria de l'ordinador), la paleta s'eliminarà del joc.

Evidentment, la pilota i les paletes de l'ordinador s'hauran d'enviar missatges per indicar-se els rebots que es produeixin. A continuació es mostren alguns exemples d'interaccions entre pilota i paletes d'ordinador i quin és el comportament esperat:

- Rebot simple (la paleta 5 es desplaça una columna a l'esquerra):

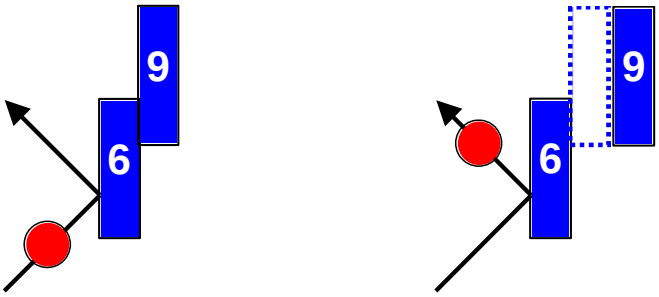


(abans de l'impacte)



(després de l'impacte)

- Rebot amb propagació de moviment (la paleta 6 traspassa el moviment a la 9):



- Rebot múltiple amb propagació de moviment i eliminació de paleta (3):

El fitxer executable principal (pare) s'anomenarà `tennis4'` i el fitxer executable que controla les paletes d'ordinador (fills) s'anomenarà `pal_ord4'` . Per validar el seu funcionament, caldrà executar-lo sobre l'entorn del servidor *bsd*. La utilització d'un `Makefile'` permet facilitar el cicle de desenvolupament.