Aquesta obra està subjecta a una llicència Attribution-NonCommercial-ShareAlike 3.0 Spain de Creative Commons. Per veure'n una còpia, visiteu http://creativecommons.org/licenses/by-nc-sa/3.0/es o envieu una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Pràctica 6: Comptabilitat electoral

Sistemes Operatius — iTIC

Sebastià Vila-Marta
Toni Escobet Canal

8 de desembre de 2012

Índex

1 Organització

El fil conductor d'aquesta pràctica és crear una aplicació local que permeti recopilar i visualitzar en temps real els resultats d'unes eleccions. El sistema ha de permetre que un nombre levat d'usuaris estiguin actualitzant les dades mentre que, en temps real, es mostren els darrers resultats actualitzats.

Els objectius inclouen la familiaritzar-se amb el conceptes procés, comunicació per memòria compartida i l'us de semàfors per a la sincronització dels accessos a les estructures compartides. La pràctica requereix consultes freqüents a `man` i l'ús de la toolchain de `C` sobre GNU/Linux amb la que ja teniu experiència.

1.1 Material necessari

Simplement us cal un computador amb sistema operatiu GNU/Linux i connexió a xarxa. Pel que fa a la documentació de referència, les pàgines de `man` han de ser suficients.

1.2 Lliurament

Cal lliurar els exercicis en un tarfile a través d'Atenea en la data 1fixada. En aquest cas també caldrà fer una demostració de les aplicacions en la classe de laboratori que ja s'anunciarà.

1.3 Durada

La durada de la pràctica és de 2 sessions de laboratori.

2 Disseny

La situació és la següent: es fan unes eleccions a la coordinadora d'entitats per la defensa de la llengua a tot el territori de parla catalana. Hi ha uns milers de meses electorals escampades per tot el pais i, a cada mesa, hi ha un responsable de transmetre el resultat del recompte. A tal efecte, es disposa d'un computador central en el que cada responsable hi té un compte d'usuari. Els responsables usen una aplicació en aquest computador central per informar dels resultats de la seva mesa.

L'aplicació es composa de dos programes que col·laboren a través d'una taula de dades que està en una zona de memòria compartida. El primer programa poll, és una aplicació de terminal que utilitzen els responsables per reflectir els vots d'una mesa. En essència el que fa és actualitzar una taula que es troba en una regió de memòria compartida. El segon programa, pollgraph, és l'aplicació destinada a visualitzar les dades en temps real. El seu funcionament es basa en consultar periòdicament la taula que es troba a la memòria compartida i, usant una tercera aplicació, gnuplot, visualitzar un histograma amb la distribució de vots per cada candidat.

La regió de memòria compartida que la crea i la destrueix l'aplicació pollgraph. Per tant aquesta aplicació es pot considerar com "l'aplicació mestra" i cal tenir-la engegada durant tot el procés de recollida de dades.

L'entrada de dades a través de poll és altament concurrent atès que previsiblement hi haurà un pic de meses que acabin el recompte a una hora semblant i per tant caldrà arbitrar l'accés a l'estructura compartida.

La figura 1 mostra un diagrama de blocs del sistema. La taula compartida en memòria es representa de color rosa. Els processos principals que usen aquesta taula es representen de color blau. El procés de color verd correspon a l'aplicació específica de generació de gràfics, que s'usa com a subprocés especialitzat.

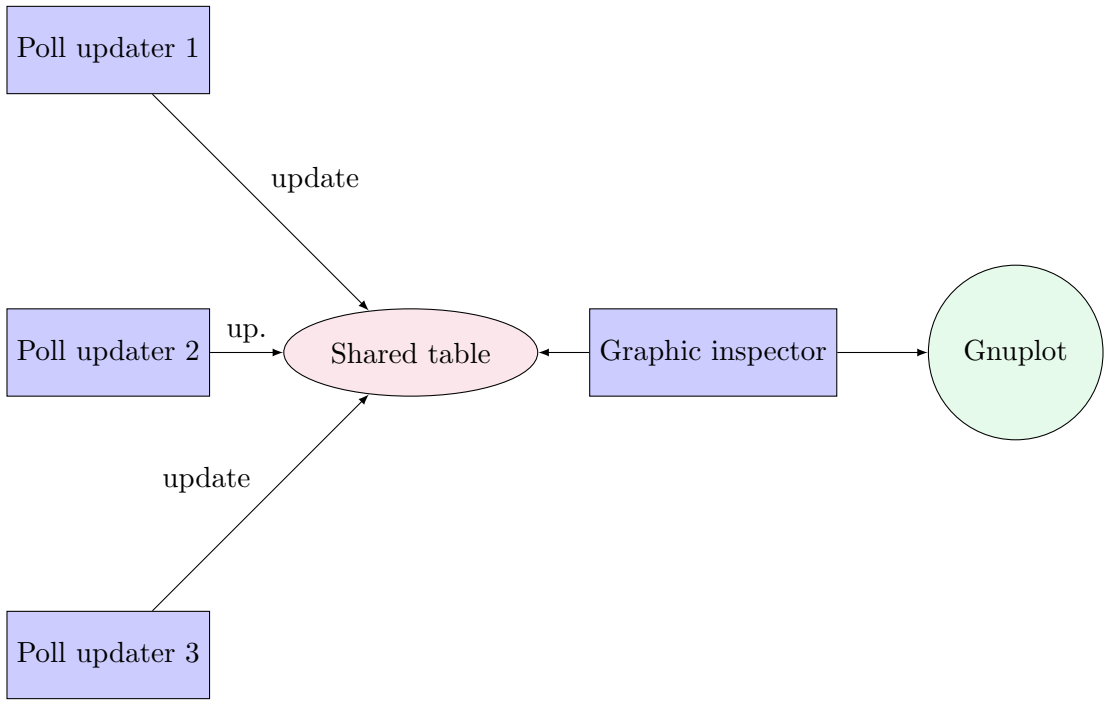

Figura 1: Diagrama de blocs del sistema

3 Desenvolupament

El desenvolupament es proposa fer-lo de manera incremental, és construirà un primer programa que s'aproxima lleugerament al que es demana i s'anirà modificant successivament fins arribar al resultat desitjat.

Exercici 3.1 L'objectiu del primer pas és familiaritzar-se amb gnuplot. gnuplot és una aplicació especialitzada en generar gràfics de dades sobre una gran varietat de suports, entre d'altres sobre la mateixa pantalla. Els gràfics es descriuren usant un llenguatge especialitzat que podeu trobar especificat a la seva documentació [2]. També us pot ser interessant aquest site, [1], per la quantitat d'exemples que concentra.

Com a repte cal crear un script de gnuplot que permeti crear un histograma com el de la figura 2. L'histograma s'ha de fer a partir d'unes dades en dues columnes en les que la primera columna correspon al nom del candidat i la segona columna als vots que ha obtingut fins el moment. La mateixa figura 2 mostra la taula de dades que correspon al gràfic.

Quan sigueu capaços de generar un histograma com l'indicat, cal que estudieu l'ordre replot i com aquesta es pot usar per refrescar el gràfic amb unes noves dades.

```
"Petra" 20
"Pol" 3
"Paula" 7
"Pep" 7
```

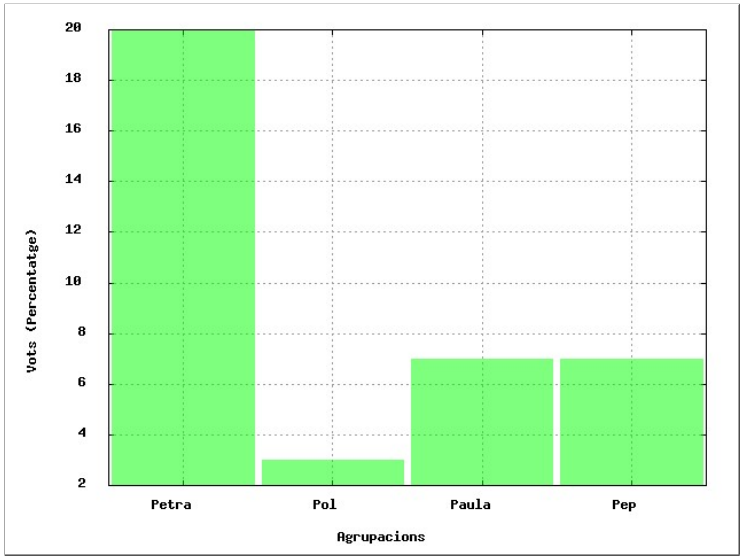

Figura 2: Taula de dades i histograma associat.

Exercici 3.2 Creeu un programa amb C que crea un subprocés gnuplot amb el que es comunica a través d'una pipe. A tal efecte useu la primitiva popen(). Noteu que popen() només crea una pipe que connecta el procés pare amb l'entrada estàndart del procés fill: no us cal cap més pipe.

A quin nivell de llibreria pertany la primitiva popen()?

Feu que aquest programa generi un histograma enviant a gnuplot les ordres apropiades i que, cada 4 s actualitzi el gràfic usant replot amb unes noves dades fictícies. Per temporitzar l'enviament de dades useu la primitiva sleep(). Tingueu en compte que després de cada línia que s'envia a gnuplot cal forçar l'enviament del buffer usant la primitiva fflush().

Exercici 3.3 L'objectiu d'aquest exercici és implementar el modul shrtbl. Aquest mòdul és el responsable de gestionar la taula de dades compartida entre tots els processos que formen el sistema i arbitrar el seu accés de forma que es garanteixi la correctesa de les dades.

El mòdul té operacions per crear, destruir i accedir a una regió de memòria compartida que hostatja:

1. Una taula dades semidinàmica que conté el nombre de vots de cada entitat.
2. Els semàfors necessaris per arbitrar l'accés a la taula de dades.

Comenceu per dissenyar com ha de ser aquesta àrea. Definiu el tipus de dades que correspon a la taula semidinàmica. Estudieu com es defineixen els semàfors POSIX i quines són les operacions disponibles per a aquests semàfors.

Usant els coneixements que ja teniu de memporia compartida i sabent com són les dades que cal emmagatzemar-hi definiu les següents funcions públiques del mòdul:

- OK, ERR

  Constants que exporta el mòdul per definir les condicions d'error de les seves operacions.

- **int** create_shared_table(**void**)

  Crea un objecte de memòria compartida per tal de poder encabir les dades comunes dels processos d'aquesta aplicació. Els permisos han de ser tals que qualsevol usuari del computador pugui llegir i escriure-hi. Únicament un sol procés pot crear una taula, altrament es produeix un conflicte. Retorna OK si tot va bé o ERR en cas contrari.

- **int** remove_shared_table(**void**)

  Esborra l'objecte de memòria compartida que crea la funció anterior. Les dades desapareixen de la memòria. Retorna OK si tot va bé o ERR en cas contrari.

- **int** bind_shared_table(**void**)

  Assumint que existeix un objecte de memòria compartida creat anteriorment, l'obre i el mapeja en memòria deixant-lo a punt per a ser utilitzat. Cada procés que vol accedir a les dades en memòria compartida ha de cridar aquesta funció abans que cap funció de manipulació de la taula. Retorna OK si tot va bé o ERR en cas contrari.

Una vegada s'ha creat una zona de memòria compartida i s'ha fet el corresponent binding, la taula es pot usar per gestionar les dades. Les següents funcions públiques del mòdul permeten gestionar aquestes dades:

- **void** init_table(**void**)

  Inicialitza la taula de dades a buida. Inicialitza els mecanismes de sincronització per la que la resta d'operacions es puguin fer en exclusió mútua si cal.

- **int** add_party(**const char** id[])

  Afegeix una entitat (per analogia aquí l'anomenem partit) amb identificador id a la taula. Retorna OK si tot va bé i ERR en cas contrari.

- **void** del_party(**const char** id[])

  Esborra el partit de nom id i les seves dades. En cas que el partit id no existeixi deixa la taula com estava.

- **void** inc_votes(**const char** party[], **int** votes)

  Incrementa en votes els vots del partit id. En cas que el partit id no existís, no fa res.

- **int** get_votes(**const char** party[])

  Obté els vots acumulats del partit id. En cas que id no consti a la taula torna ERR.

  COMPTE: aquest us d'ERR delimita els valors que pot prendre la constant ERR.

- **int** get_nparties(**void**)

  Retorna el nombre de partits que consten a la taula.

- **void** traverse(travapp ∗**const** f, **void** ∗**const** data)

  Aquesta funció és un iterador: recorre la taula de dades i per a cada parella partit-vots crida a la funció f que s'ha indicat en el paràmetre. La funció f respon al següent tipus:

  typedef void travapp(**const char** ∗**const** id, **int** votes, **void** ∗**const** data)|

  Cada vegada que l'iterador crida a la funció f, li passa pel paràmetre id el nom del partit, pel paràmetre votes els vots que té acumulats i pel paràmetre data l'apuntador a les dades suplementàries que s'han passat també a l'invocar traverse.

Noteu que entre les funcions hi ha un iterador. És una estructura freqüent quan en un mòdul es vol oferir a l'usuari la possibilitat de recórrer un estructura sense haver de revelar com s'implementa. El seu ús és molt còmode. Imagineu que voleu escriure les dades de la taula. Només heu d'escriure una funció que escrigui una de les seves files i cridar-la per cada fila usant l'iterador. En aquest cas no són necessàres dades suplementàries i per tant el paràmetre data pot ser NULL. Aquest seria el resultat:

```
static void printentry(const char *const id, int votes, void *const data)
{
  printf("%16s %4d\n", id, votes);
}

...
traverse(printentry, NULL);
...
```

Per arbitrar l'accés a les regions crítiques que suposen aquestes funcions useu un únic semàfor.

EXERCICI 3.4 Implementeu un programa per tal que els representants de cada mesa puguin afegir a les dades els resultats de les seves taules. El programa cal que pugui executar-se com una ordre de la terminal amb la sintaxi següent:

```
poll {new <party> | add <party> <votes>}
```

Així doncs l'ordre ha de permetre donar d'alta un nou partit i també incrementar el nombre de vots d'un partit. Per implementar l'ordre useu la funcionalitat del mòdul shrtbl.

EXERCICI 3.5 Modifiqueu el programa pollgraph per tal que, en comptes de generar un graph amb dades aleatòries, reflecteixi l'estat de la taula de dades compartida. Més exactament el que cal que faci és el següent:

1. Crear una àrea de memòria compartida.
2. Fer un bind a aquesta àrea.
3. Inicialitzar la taula de dades.
4. Cada 4 s reconsultar les dades i actualitzar el gràfic
5. Esborrar l'àrea de memòria compartida.

En aquest procés hi ha un detall interesant d'aclarir. Noteu que el procés té una durada indefinida ja que itera actualitzant el gràfic *ad libitum*. Per aturar el procés cal enviar un signal al procés per que s'aturi. Ho podem fer simplement prement `Ctrl-C`, però si fe això el procés no acaba normalment i per tant no executa la darrera part del programa que esborra l'àrea de memòria compartida.

La solució rau en "capturar" el signal de forma que quan l'usuari faci un `Ctrl-C` s'executi una funció específica, una mena de "rutina d'interrupció". Aquesta rutina simplement canvia el valor d'una variable global que és la variable de la que depen el bucle infinit que controla el refresc. L'esquema és el següent:

```
static volatile bool must_end = false;

...

static void handler(int sig) {
  must_end = true;
}

...

refresh(out);
while (!must_end) {
  sleep(5);
  refresh(out);
}
```

Per instal·lar el handler del signal cal usar la funció signal() i associar-la al signal SIGINT, que correspon al `Ctrl-C`. Addicionalmet també és interessant associar-la al signal SIGHUP.

EXERCICI 3.6 Aquest és un exercici optatiu. Com haureu notat, algunes de les operacions del mòdul shrtbl accedeixen a les dades en mode lectura i altres en mode escriptura. Una millora del projecte consistiria a implementar un sistema d'accés que tingués en compte aquest detall i permetés que diversos processos estiguin llegint simultàniament. A tal efecte cal implementar un esquema de sincronització lectors/escriptors amb prioritat pels lectors.

Referències

[1] *Demo scripts for gnuplot version 4.6*. 2012. URL: `http://gnuplot.sourceforge.net/demo` (vis. 05-12-2012).

[2] Thomas Williams, Colin Kelley i d'altres. *Gnuplot Manual*. Ang. Ver. 4.6. 2012. 238 pàgs. URL: `http://www.gnuplot.info/docs_4.6/gnuplot.pdf`.