PRÀCTICA 2
El pagès i el seu hortal

PROSO

Q1 2008

Josep Martí Pascual

Felip Moll Marquès

YO16

- Practica 2 -

Introducció:

Aquesta pràctica consisteix en implementar dos mòduls per al kernel de Linux. El primer mòdul s'encarregarà de recollir estadístiques sobre crides al sistema, i el segon serà un dispositiu que ens permetrà accedir a la informació d'aquest primer. Els hem anomenat per una part Mihuerto.c, que serà el primer mòdul i representarà que anar fent estadístiques és com fer que les verdures d'un hort germinin i creixin , i per altra banda ElPayes.c, que és qui recull aquests fruits.

MiHuerto.c:

El funcionament d'aquest mòdul no té complicació conceptual; només consistirà en interrompre les crides al sistema que volguem monitoritzar, i que en aquest cas són:

sys_open, sys_write, sys_lseek i sys_clone.

Voldrem que per cada crida que faixi un procés, es guardin dins algun camp del procés el nombre de crides fetes en total, el nombre de fallides, de correctes, i el temps total invertit en totes les crides.

A més per cada crida i per cada procés guardar tota aquesta informació individualment, per exemple:

El pid N ha fet 20 crides:

Temps total: X

Satisfactories: Y

Errònies: Z

10 opens : 9 satisfactoris, 1 errròni, i per cada una el seu temps

10 writes....

A més de tot això també hem volgut guardar, per cada crida individual, el nombre de vegades que s'ha executat correctament, el temps mig de la crida, i altres estadístiques. La diferència amb l'anterior és que l'anterior estava enfocat a procés i aquest a crida.

Estructures necessàries:

Quad: Hem definit un nou tipus quad que correspon a un unsigned long long

```
typedef unsigned long long quad;
```

Struct th_info_est: Es situarà al final del thread_info que és la informació de procés i es pot obtenir mitjançant la funció del kernel get_thread_info. Per aconseguir guardar-la allà ho farem amb una struct que englobi els elements que volem, i aquesta és la th_info_est

```
struct th_info_est
{
  struct thread_info info_th;
  struct pid_stats estadistiques[N_CRIDES_A_MONITORITZAR + 1];
  int pid;
};
```

Struct pid_stats: Contindrà tota la informació que volem guardar per cada procés

```
struct pid_stats

{
  int num_entrades;
  int num_sortides_ok;
  int num_sortides_error;
  quad durada_total;

};
```

Struct sysc_stats: Contindrà tota la informació que volem guardar per cada crida, i també tindrem la sysc_info_table que contindrà aquesta informació per totes les crides.

```
struct sysc_stats
{
  int num_crides;
  int num_fallides;
  int num_satisfactories;
  quad temps_execucio;
} sysc_info_table[N_CRIDES_A_MONITORITZAR];
```

Funcionament:

El funcionament del mòdul és senzill, com ve explicat a l'enunciat l'únic que farem serà definir les operacions locals sys_open, sys_close, sys_write, sys_lseek i sys_clone que seran les substitutes de les crides reals del sistema. Per fer això farem que quan es cridi a un sys_open, aquest vengui a parar a la nostra rutina i dins la nostra rutina agafarem el temps actual, farem la crida real, tornarem agafar el temps i llavors incrementarem els comptadors que faixin falta segons la crida que sigui i segons el procés que hagi fet la crida.

El procés que farà la crida el sabrem en tot moment amb la funció current, com havíem vist a Zeos. Només haurem d'obtenir el seu thread_info, que es podrà fer de varies formes, i llavors fer un casting a l'struct que conté les estadístiques (també es podria fer amb un sizeof i accedir a la posició necessària).

Per fer que s'interceptin les crides reals, només haurem de modificar la taula de crides del sistema, sys_call_table, i fer que les originals apuntin a les nostres tal com està explicat a l'enunciat. Això ho farem a l'init del mòdul. A l'exit del mòdul desfarem aquesta acció, i a activar_monit. o desactivar_monit. modificarem també la intercepció aconseguint interceptar unes o altres crides.

Hem exportat també les funcions i estructures que eren necessàries. Es pot veure al final del fitxer include de mihuerto.

Per fer més llegible el codi havíem pensat en fer una macro per no haver d'escriure el mateix codi a cada operació, ja que l'únic que canvia és la crida general que es fa. El problema ha estat que no hem tingut prou temps.

ELPAYES.C

Aquest mòdul serveix per recollir les estadístiques que va guardant el primer mòdul i per controlar alguns paràmetres i obtenir informació específica en temps real de les crides i processos que vulguem monitoritzar. El funcionament bàsic serà el de carregar el mòdul i registrar un nou dispositiu que posteriorment crearem amb mknod. El modul farà de driver del dispositiu definint les operacions necessàries que trobem típicament a l'estructura del kernel file_operations, un cop carregat aquest i creat el dispositiu que s'enllaça amb el mòdul, podrem actuar sobre aquest per obtenir, controlar, activar, desactivar, etc. l'obtenció d'estadístiques des de el primer mòdul.

Estructures necessàries

No hem tingut la necessitat de introduir noves estructures de dades, ja que l'únic que hem fet és exportar els símbols del mòdul 1 que necessitàvem aquí i fer-los servir normalment. El que si hem fet és crear noves variables globals, concretament:

lock : Ens dirà si el dispositiu està bloquejat, si ho està no serà possible fer un altre open
proces_monitoritzat: És el pid del procés del que volem obtenir informació mitjançant el dispositiu. No l'hem de confondre amb el pid_inicial que estava declarat al primer mòdul, només s'assembla a n'aquell en que si no especifiquem proces_monitoritzat aquest valor agafa el del pid_inicial.

sys_call_monitoritzat: Per defecte és la OPEN, i serà la crida de la que volem obtenir informació.

Aquests valors es poden controlar mitjançant funcions del propi dispositiu, les explicarem a continuació.

Funcionament

Començarem amb la funció d'inici del mòdul:

ir_al_huerto_init():

Aquesta funció inicialitza el proces monitoritzat al mateix pid_inicial que té el mòdul 1, i la syscall monitoritzada a OPEN.

Després d'aquesta operació passam a crear un nou major i minor dintre d'una estructura dev_t, ho farem amb la macro MKDEV.

```
maj_min = MKDEV(MAJ,MIN);
```

Aquest pas l'haviem realitzat amb la funció alloc_chrdev_region, que ens dona un major i un minor qualsevol i ja el registra. El problema que teniem era que des de fora del mòdul no podiem saber quin se'ns havia assignat i per tant era més complicat fer el joc de proves: No podiem crear el dispositiu amb mknod. Per tant hem decidit deixar un major i un minor fixes, encara que no sigui una acció correcte del tot, i treballar amb aquests valors. Estan definits dins elpayes.h.

Un cop tenim el major i el minor dintre de l'estructura dev_t, passem a registrar el major i el minor amb nom payes amb la funció register_chrdev_region, i a registrar el dispositiu dintre del kernel amb cdev_alloc. El dispositiu que ens retorna cdev_alloc serà el que li donarem com a propietats de ops el file_ops que ens haurem definit a elpayes.h. Finalment registrarem el dispositiu juntament amb el major i el minor.

salir_del_huerto_exit()

Simplement haurem de deixar lliure el major i el minor i eliminar el dispositiu del kernel:

unregister_chrdev_region i cdev_del.

A continuació les funcions que introduïm dins el file_operations del dispositiu:

pages_read_dev:

L'objectiu d'aquesta funció és obtenir les estadístiques del proces monitoritzat (i a una futura ampliació també de syscall monitoritzada) i enviar-les a l'usuari. Ens entren com a paràmetre una direcció de memòria a la que hi podem posar informació i en la que hi posarem l'equivalent a una estructura pid_stats. Si l'usuari no ens deixa prou espai per posar-hi la informació, l'hi posarem fins que hi capi i ell serà el responsable de tractar amb la informació.

Per obtenir la informació del pid monitoritzat ho farem amb la funció obtenir_estadistiques del mòdul 1 i passant-li com a paràmetre un pid_stats temporal. Llavors amb copy_to_user copiarem els bytes que ens hagi indicat l'usuari de pid_stats temporal a la @ de memòria que ell ens ha donat.

pages_ioctl_dev:

L'objectiu d'aquesta funció és la de controlar el dispositiu i, més concretament, quin és el procés que monitoritzem, quina crida monitoritzem, i si tenim activada o desactivada la motorització.

Segons el paràmetre arg1 haurem de fer una cosa o altra.

Si arg1 val 0, significarà que arg2 conté el nou pid de procés a monitoritzar del que voldrem obtenir informació. Atenció, això no desactiva la monitorització dels altres processos, només fa que la informació que rebem sigui d'aquest procés.

Si arg1 val 1, significarà que arg2 conté el nou nombre de crida a monitoritzar, pot anar de 0 a 4.

Si arg1 val 2, significarà que hem de fer un reset de tots els valors del procés que actualment estem monitoritzant.

Si arg1 val 3, significarà que farem un reset de tots els valors de les estadístiques de tots els processos del sistema.

Si arg1 val 4, significarà que volem activar la monitorització per una crida determinada, el seu nombre vindrà especificat per el paràmetre arg2. Si arg2 val -1 activarem totes les crides.

Si arg1 val 5, significarà que volem desactivar la monitorització d'una crida determinada, el seu nombre vé determinat per el paràmetre arg2. Si arg2 val -1 desactivarem totes les crides.

pages_open_dev:

Aquesta crida controlarà que qui fa un open del dispositiu sigui l'usuari root, mirant l'uid amb el que s'està executant el procés actual. També mirarà que no és puguin fer més de dos opens per el mateix dispositiu incrementant el lock.

pages_release_dev:

És l'invers a l'open, no haurà de controlar que qui intenti fer un release sigui el pare perque se suposa que primer ha d'haver obert el dispositiu per poder tancar-lo. Només farà un decrement a lock.

reset_valors i reset_tots_valors:

Aquestes dues funcions criden per el pid que corerspongui a reset_info del primer modul, provocant que les estadístiques d'aquests processos quedin totes a 0.

activar_sys_call i desactivar_sys_call:

Criden a la funció del primer mòdul que ja està explicada a la primera part.