SISTEMAS OPERATIVOS 1. Enxeñería Informática. Curso 2007-2008

Práctica 1: Filedaemon.

Implementar un demonio que periódicamente inspeccione los ficheros que han sido modificados recientemente a partir de un directorio pasado como argumento. En la primera iteración mostrará los modificados en el día actual o en los días previos indicados como argumento de entrada. En las siguientes iteraciones mostrará los modificados a partir de la fecha del chequeo previo. Para tales ficheros se debe mostrar la misma información que muestra el comando *ls -lisa*. El formato de visualización de esa información es libre, aunque debe reflejarse claramente en qué directorio se encuentra cada fichero mostrado.

La sintaxis del ejecutable será:

$*filedaemon dir [-d dias][-s segundos]*

dir es el directorio inicial a partir del cual se chequea (de modo recursivo en subdirectorios) cuáles ficheros han sido modificados en los días previos.

-d dias especifica el número de días previos que se consideran para el primer chequeo. Es opcional, por lo que el valor por defecto, de no especificarse, será 0 días, es decir, se muestran los modificados en el día actual. Si el valor es 1 se mostrarán los modificados hoy o ayer, si es 2 ...

-s segundos especifica el intervalo en segundos en el que el demonio realiza el chequeo. Su valor por defecto será de 60 segundos.

Ejemplo (los comentarios y formato de visualización son libres):

$ *filedaemon /home/juan -d 2 -s 120&*

Starting filedaemon test in /home/juan at 18:00:00 on 15-10-2007

Found file /home/juan/SO1/apuntes_SO1 modified at 16:34:13 on 14-10-2007
File type: symbolic link to /home/juan/apuntes/apuntes_SO1.pdf
Owner: juan
Inode: 5548
Size: 34 bytes
…..

filedaemon has ended the test

Starting filedaemon test in /home/juan at 18:02:00

Found file /home/juan/SO1/practicas/p1.c modified at 18:01:23 on 15-10-2007
File type: regular
Owner: juan
Inode: 2342
Size: 1345 bytes
…..

filedaemon has ended the test

Comentarios

- La información de cada fichero se obtiene con *stat* o *lstat*.
- Para obtener el listado de los ficheros de un directorio deben usarse las funciones *opendir, readdir y closedir,* del cabecera "*dirent.h*".
- Los nombres de los usuarios, a partir del *login* pueden obtenerse con *getpwent ó getpwuid* y los del grupo con *getgrent ó getgruid*.
- Los permisos deben indicarse en la forma rwxrwxrwx.
- El lugar apuntado por un enlace simbólico puede obtenerse con *realpath*. No se procederá a mostrar recursivamente el contenido del directorio al que apunte un link simbólico.
- La fecha puede convertirse a formato cadena con *ctime.* Se proporcionan utilidades para comparativas temporales.
- Para implementar un temporizador usar las llamadas definidas en *signal.h*: *alarm*, *pause*, *sigaction*. Se proporciona un código inicial con el esquema del temporizador.

Modo de entrega

Fecha de entrega límite: Viernes 23 de Noviembre 2007

La práctica será presentada "in situ" al profesor de prácticas, quien realizará una inspección minuciosa de su funcionamiento, al mismo tiempo que podrá requerir a los integrantes del grupo las explicaciones pertinentes o la realización de cambios en el código.

Detalles a recordar

- Si se entregan todas las prácticas, éstas suman un 1.5 en la nota final -el examen de teoría será sobre un total de 8.5-.
- No todas las prácticas puntúan igual. Las dos primeras prácticas puntúan 0.6, mientras la tercera puntuará 0.3. Sin embargo, los profesores se guardan el derecho de cambiar estas puntuaciones *on-line* durante el cuatrimestre.

Código temporizador

Comentarios iniciales:

Una señal es el análogo en software a una interrupción hardware, un evento que se produce en cualquier momento mientras se ejecuta un proceso. Esta naturaleza impredecible significa que las señales son asíncronas. Las señales se pueden utilizar para comunicar y sincronizar procesos, pero también las usa el núcleo para comunicar información a los procesos. Por ejemplo, una referencia ilegal a memoria genera una señal que el núcleo envía al proceso correspondiente. Cuando un proceso recibe una señal, puede realizar unas de estas tres posibilidades:

- Puede ignorarla.
- Puede "atraparla", lo que hace que se ejecute una porción de código denominada "manejador (*handler*) de la señal".
- Puede permitir que se produzca la acción predeterminada de la señal.

El archivo cabecera *signal.h* declara la lista de posibles señales que se pueden enviar a los procesos en un sistema. La señal *SIGALRM* es la que define el fin de una temporización.

- La función *sigemptyset (sigset_t *set)* inicia un conjunto de señales.
- El servicio para enviar señales entre procesos es:

  kill(pid_t pid, int sig), que envía la señal *sig* al proceso o grupo de procesos especificado por *pid* (no se necesita en esta práctica).
- El armado de una señal se define con:

  *int sigaction (int sig, struct sigaction *act, struct sigaction *oact);*

  El primer parámetro establece el número de señal para la que se quiere establecer el manejador, el segundo es un puntero a una estructura de tipo *sigaction* para establecer el nuevo manejador, y el último parámetro es un puntero a una estructura del mismo tipo que almacena información sobre el manejador establecido anteriormente.
- La estructura *sigaction*, definida en *signal.h*, tiene los campos:

```
struct sigaction {
    void (*sa_handler) ();   /* manejador para la señal */
    sigset_s sa_mask;        /* señales bloqueadas durante la ejecución del manejador */
    int sa_flags;            /* opciones especiales */
    }
```

  El primer campo indica la acción a ejecutar cuando se reciba la señal. Su valor puede ser:

  SIG_DFL: se lleva a cabo la acción por defecto.
  SIG_IGN: la señal será ignorada cuando se reciba.
  Una función que devuelve un *void* y que acepta como parámetro un entero.

- *int pause (void)*. Cuando se quiere esperar la recepción de una señal se utiliza este servicio, que bloquea al proceso que lo invoca hasta que llega una señal.

- *unsigned int alarm (unsigned int seconds)*. Esta función establece un "cronómetro" y envía al proceso la señal *SIGALRM* después de pasados el número de segundos especificados en el parámetro *seconds*.

Código:

El siguiente código corresponde a un temporizador típico implementado con servicios POSIX. El código se incluye en las direcciones web:

http://www.dc.fi.udc.es/ai/people/santos/so1/so1_07_08.html

http://vios.dc.fi.udc.es/so

```
void startDaemon(char *path, int days, int sec) {
struct sigaction sact;

//prepares options for sigaction (in this case, just to change what to do on SIGALRM signal)

sigemptyset( &sact.sa_mask );  //signals that should be avoided during the handling process
//sigaddset(&sact.sa_mask, SIGINT);    // to ignore SIGINT signal while the handler is
                                       // executing (not necessary in this practice)
sact.sa_flags = 0; // a set of flags which modify the behaviour of the signal handling process
sact.sa_handler = catcher;

//used to change the action taken by a process on receipt of a specific signal ("SIGALARM")
sigaction( SIGALRM, &sact, NULL );


while (1) {
        //arranges for a SIGALRM signal to be delivered to the process in "sec" seconds.
        //that is, timer will pop in sec seconds.
        alarm(sec);
        pause();  //sleep until a signal is received
        }
}

void catcher( int sig ) {
        //printf( "Signal catcher called for signal %d\n", sig );
        ...
        processDir(…);  // 2nd and next calls – practice main code
        …
}
```

Utilidades para comparativas temporales

Algunas funciones que se pueden utilizar, de modo optativo, para la comparación de fechas de ficheros. El código se incluye en los ficheros tiempo.c y tiempo.h en las mismas urls.

```
// Obtains current time value in seconds since 1970.
time_t getCurrentTime(){
        time_t t;
        time(&t);
        return t ;
        }

// Obtains the time value for today at 00:00h
time_t getToday() {
        struct tm * ltime;
        time_t t1;
        time(&t1);
        ltime = localtime (&t1);
        t1 -= (ltime->tm_hour*3600 + ltime->tm_min*60 + ltime->tm_sec);
        return t1;
        }

// Gets the time value of "days" days before today at 00:00h
// Returns:
//    time value corresponding to today at 00:00h if days ==0 --> the same as getToday()
//    time value corresponding to yesterday at 00:00h if days ==1
//    time value corresponding to x days before at 00:00h if days ==x
time_t getStartTimeDaysBefore(int days) {
        time_t t;
        t = getToday();
        t -= (days * SECONDS_IN_A_DAY);
        return t;
        }

// Calls to ctime, skipping the last '\n' char.
// Therefore, it just returs a formatted char * value corresponding to parameter t.
char *getTimeAsString(time_t t) {
        char *str; int len;
        char buff[1000];
        ctime_r (&t,buff);
        len = strlen(buff)-1;
        str = (char *) malloc (sizeof(char) * (len+1));

        strncpy (str,buff,len);
        return(str);                    }
```