Práctica 4

Abril 2016

Índice

1. Introducción

El objetivo de esta práctica se centra en el uso del lenguaje C y la utilización de las llamadas a sistema. El objetivo es la implementación de un temporizador digital distribuido entre múltiples procesos. La comunicación entre procesos se realizará mediante señales.

2. Restricción en el uso de funciones C

Para realizar esta práctica es obligatorio utilizar las llamadas a sistema de Unix. No se podrán utilizar funciones C de entrada/salida (*scanf*, *printf*, *fopen*, etc.). En su lugar habrá que utilizar las llamadas a sistema (*open*, *read*, *write*, *close*, ...). Sí que esta permitido utilizar funciones C para formatear y manipular cadenas (*sprintf*, *strlen*, ...). Para ello se incluye un ejemplo de uso. Tampoco está permitido utilizar la función *sleep*, sino que en su lugar habrá que utilizar las funciones *pause* y *alarm* cuando sea necesario.

3. Enunciado

Se desea implementar un temporizador digital mediante 4 procesos independientes que gestionaran los segundos, los minutos y las horas. La funcionalidad de cada proceso es la siguiente; los detalles se describen a lo largo del enunciado.

1. El proceso principal (fichero `principal.c`) gestionará el reloj.
2. El proceso segundos (fichero `segundos.c`) avisará de los segundos a los procesos.
3. El proceso minutos (fichero `minutos.c`) avisará de los minutos a los procesos.
4. El proceso horas (fichero `horas.c`) avisará de las horas a los procesos.

El funcionamiento de los procesos es el siguiente

1. Se iniciarán cada uno de los cuatro procesos en un mismo terminal. Los procesos "principal", "segundos", "minutos" y "horas" guardaran en los ficheros `principal.pid`, `segundos.pid`, `minutos.pid` y `horas.pid`, respectivamente, su identificador de proceso. Para ello se utilizaran las llamadas a sistema *getpid, open*, *read*, *write* y *close*. Si no se puede abrir

el fichero para guardar la información a disco, deberá mostrarse por la salida de error un mensaje de error (no es necesario gestionar que los otros procesos mueran). Una vez los cuatro procesos hayan guardado esta información en un fichero éstos se quedaran esperando, mediante la función *pause,* a recibir una señal para iniciar el contador digital.

2. Cuando el proceso "principal" reciba la señal SIGCONT (que se enviará desde otro terminal mediante el comando `kill`), se iniciará el temporizador digital en este proceso. Para ello el proceso "principal" mostrará un mensaje por la salida estándard y leerá de disco los PIDs del resto de procesos de los ficheros correspondientes. Si no puede abrir alguno de los ficheros, se mostrará un mensaje de error por la salida de error. Una vez leída la información de los PIDs, el proceso "principal" enviará una señal SIGCONT a los procesos "segundos", "minutos" y "horas". Esto hará que se inicien a cero los contadores de segundos, minutos y horas, respectivamente, en cada uno de los procesos.

3. El proceso "segundos" programará de forma iterativa una alarma cada segundo mediante la función *alarm*. Cada vez que el proceso "segundos" reciba una alarma enviará una señal SIGUSR1 al proceso "principal". Cuando hayan pasado 60 segundos el proceso "segundos" enviará una señal SIGCONT al proceso "minutos". Los PIDs de los procesos se podrán obtener de los ficheros correspondientes. Si no se puede abrir alguno de los ficheros, se mostrará un error por la salida de error.

4. Cada vez que el proceso "minutos" reciba la señal SIGCONT de "segundos", enviará una señal SIGURS2 a "principal". Cuando hayan pasado 60 minutos el proceso "minutos" enviará una señal SIGCONT al proceso "horas". Los PIDs de los procesos se podrán obtener de los ficheros correspondientes. Si no se puede abrir alguno de los ficheros, se mostrará un error por la salida de error.

5. Cada vez que el proceso "horas" reciba la señal SIGCONT de "minutos" enviará una señal SIGCONT al proceso "principal". El PID de proceso "principal" se podrá obtener de los ficheros correspondientes. Si no se puede abrir alguno de los ficheros, se mostrará un error por la salida de error.

6. El proceso "principal" llevará un control del temporizador digital a partir de la información enviada por el resto de procesos. Cada vez que reciba una señal SIGALRM (que se enviará desde otro terminal mediante el comando `kill`) se imprimirá por pantalla el contador digital

en formato "hh:mm:ss". Para formatear la cadena en este formato se recomienda utilizar la función *sprintf*. En concreto, se incluye un ejemplo junto con esta práctica.

7. Cuando el proceso "principal" reciba la señal SIGTERM se parará el temporizador digital: para ello el proceso "principal" enviará primero una señal SIGTERM a el resto procesos: "segundos", "minutos" y "horas". Estos tres procesos finalizaran de forma limpia (por la función *main*) sin emitir mensaje alguno por pantalla. Una vez realizado enviado el SIGTERM, el proceso "principal" imprimirá un mensaje y finalizará también de forma limpia.

4. Entrega

El único proceso que imprimirá mensajes por la pantalla es el proceso "principal". Los procesos "segundos", "minutos" y "horas" no imprimirán ningún mensaje por pantalla a menos que se trate de un mensaje de error, en cuyo caso se imprimirá el mensaje por la salida de error. En caso que se produzca un error no es necesario gestionar que el resto de procesos finalice (ver último punto).

Para la entrega de la práctica se pide entregar lo siguiente

- Entregar el código fuente de los procesos "principal", "segundos", "minutos" y "horas". Los nombres de los ficheros deben ser `principal.c`, `segundos.c`, `minutos.c` y `horas.c`, respectivamente. Observar que todos los códigos fuentes comparten la escritura y lectura de los PIDs de los procesos. Es por ello que esta funcionalidad se incluirá en un fichero aparte, `rw_pid.c`, que permitirá escribir y leer PIDs de disco. Los procesos "principal", "segundos", "minutos" y "horas" deberán utilizar el fichero de cabecera `rw_pid.h` para poder hacer uso de las funciones.

- Entregar un único `Makefile` que permita compilar las cuatro aplicaciones. La compilación deberá hacerse en dos pasos, tal y como se hizo en la primera práctica: primero generar el fichero objeto y después el ejecutable. Al ejecutar `make` únicamente deben compilarse aquellos ficheros fuente C que hayan sido modificados.

- Entregar un script `limpiar.sh` que permita matar de forma automática, mediante el comando kill, los procesos "principal", "segundos", "minutos" y "horas" que se estén ejecutando en ese momento. Ello es debido a que muchas veces se os "colgará" la aplicación. Para realizar

pruebas fiables, por lo tanto, es necesario asegurar que no haya ningún otro proceso de la práctica ejecutándose. Para ello se supone que los procesos a matar se llaman "principal", "segundos", "minutos" y "horas".

- Entregar un script llamado `iniciar.sh` que permita ejecutar los 4 procesos. Antes de ejecutar los procesos se deben borrar los ficheros "*.pid" correspondientes. El script `limpiar.sh` se ejecutará antes de ejecutar los 4 procesos.
- Entregar un script llamado `iniciar_contador.sh` que permita iniciar el contador digital. El contador se inicia enviando una señal SIGCONT al proceso "principal" con el comando kill. Se supone que hay un único proceso "principal" ejecutándose en el ordenador.
- Entregar un script llamado `mostrar_contador.sh` que haga que el proceso principal muestre por pantalla el valor del contador. Para ello habrá que enviar una señal SIGALRM al proceso "principal". Se supone que hay un único proceso "principal" ejecutándose en el ordenador.
- Entregar un script llamado `finalizar_contador.sh` que permita finalizar el contador digital. El contador digital se finaliza enviando una señal SIGTERM al proceso "principal" con el comando kill. Se supone que hay un único proceso "principal" ejecutándose en el ordenador.
- Se valorará positivamente los comentarios y la correcta justificación del código.

El peso de la entrega es de un 80 %, mientras que el test tendrá un peso del 20 %.