Final Laboratorio SO1819q2

Makefile (0.5 puntos)

Crea un Makefile que permita generar todos los programas del enunciado a la vez y cada uno por separado. Añade una regla (clean) para borrar todos los binarios y/o ficheros objeto y dejar sólo los ficheros fuente. Los programas deben generarse si, y sólo si, ha habido cambios en los ficheros fuente.

Control de errores (0.5 puntos)

Para todos los programas deben comprobarse los errores de TODAS las llamadas a sistema, excepto el write por pantalla, controlar los argumentos de entrada y definir la función Usage()

Ejercicios (9 puntos)

Los ejercicios 1, 2 y 3 son independientes entre ellos y los podéis resolver en cualquier orden. Los ejercicios tienen también algunas preguntas que deben contestarse en el fichero respuestas.txt

Ejercicio 1 (3.5 puntos)

Se pide un programa prog1.c que recibe un único argumento **n**, que es el número de hijos a crear (se ejecutarán de manera concurrente).

Cada hijo escribe por la salida estándar su número de hijo (entre 0 y **n**-1) y su PID, y a continuación pasa a esperar la recepción de un SIGUSR1. La espera debe ser bloqueante. El resto de signals deben ignorarse. Una vez recibido el SIGUSR1, el programa debe escribir el mensaje "PID: USR1 recibido" y acabar.

El programa padre, cuando ha creado todos los hijos lo indica por la salida estándar y espera que se pulse la tecla RETURN. A continuación envía un SIGUSR1 a todos los hijos y muere después de liberar todos los hijos.

Importante: la gestión de los PID de los hijos debe realizarse obligatoriamente con memoria dinámica usando las funciones de la librería de C.

Pregunta 1 para respuestas.txt: ¿cómo puedo comprobar desde otro terminal que se han creado los procesos que deberían haberse creado?

Ejercicio 2. (3 puntos)

Este ejercicio utiliza el ejecutable **aux**, que os incluimos entre los ficheros del enunciado. Este programa recibe un único argumento **n**, y hace dos cosas: escribe por la salida estándar su PID en formato interno de la máquina, y a continuación se bloquea **n** segundos. Por ejemplo, si ejecutáis: `_$ ./aux 3`

Por la pantalla saldría el PID (en formato interno, por lo que aparece el ASCII del hexadecimal del PID, por ejemplo el PID=29549 sacaría por pantalla **ms**) y aux esperaría 3 segundos para finalizar.

Se pide un programa prog2.c que recibe dos argumentos: el nombre de una NAMED PIPE y el número de hijos a crear. Por ejemplo: `_$ ./prog2 mypipe 5`

- El proceso padre crea el número de hijos solicitado que se ejecutarán de manera concurrente.
- Cada hijo mutará a ejecutar **aux**. El argumento para ejecutar cada mutación es el número de hijo. Así, si hay 5 hijos el primero ejecutará ./aux 1, el segundo ./aux 2, hasta el quinto que ejecutará ./aux 5. Además, hemos de garantizar que la salida de **aux** vaya a la entrada de la NAMED PIPE.

- Una vez creados los hijos, el padre leerá lo que llegue por la NAMED PIPE (el PID de sus hijos) y los escribirá por la salida estándar en formato texto. Una vez ha acabado de leer toda la información comprueba la muerte de todos sus hijos, escribiendo por la salida estándar el PID de cada hijo que comprueba que ha muerto.
- Si la named pipe no existe, el padre debe crearla con permisos de lectura y escritura para el propietario, el grupo y el resto de usuarios.

Pregunta 2 para respuestas.txt: la información recibida por el padre por medio de la NAMED PIPE (el PID de sus hijos) puede llegar en un orden que no sea el de creación de los hijos. Justifica tu respuesta.

Ejercicio 3 (2.5 puntos)

Se pide un programa prog3.c que recibe como argumentos el nombre de un fichero y dos números. Por ejemplo: `_$ ./prog3 myfile 3 7`

El programa abre el fichero pasado como argumento para lectura y lee los caracteres en las posiciones entre los dos números que son los otros parámetros. Los caracteres leídos deben escribirse por la salida estándar. En el ejemplo, y suponiendo que myfile es un fichero de caracteres que contiene "ABCDEFGHIJKLMNOPQRSTUVWXYZ" (es decir, 26 caracteres) el padre leería los caracteres entre la posición 3 y la 7, y los escribiría por la salida estándar. Es decir, escribiría "DEFGH".

Imprescindible: que se compruebe, además de los errores habituales, que los argumentos que indican las posiciones del fichero a leer sean correctas.

Disponéis del fichero myfile como el descrito para realizar pruebas.

Pregunta 3 para respuestas.txt: si el argumento que se le pasa a prog3 en lugar de un fichero fuera una NAMED PIPE, ¿cómo afectaría al problema?

Qué hay que hacer

- El Makefile
- Los códigos de los programas en C
- La función Usage() de cada programa
- El fichero respuestas.txt con las respuestas de los ejercicios

Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas a sistema sea correcto
- Que se comprueben los errores de **todas** las llamadas a sistema
- Que el código sea claro y correctamente indentado
- Que el Makefile tenga bien definidas las dependencias y los objetivos
- Que la función Usage() muestre por pantalla cómo debe invocarse correctamente el programa en caso de que los argumentos recibidos no sean los adecuados
- El fichero respuestas.txt

Qué hay que entregar

Un único fichero tar.gz con el código de todos los programas, el Makefile y las respuestas en respuestas.txt

```
_$ tar zcvf final.tar.gz Makefile respuestas.txt *.c
```