Makefile (0,5 puntos)

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

Control de Errores y Usage (0,5 puntos)

Todos los programas deben incluir un control adecuado de los argumentos usando la función Usage() y deben controlar los errores en todas las llamadas al sistema.

Ejercicio 1 (1 punto)

Edita el shell script, llamado **client.sh**, adjunto con este enunciado. Añade una línea de comandos antes de `cat >> ./MIPIPE` para crear la pipe llamada **MIPIPE**. Dale al shell script permisos de ejecución para el propietario, el grupo del propietario y el resto de los usuarios.

Ejercicio 2 (2 puntos)

Escribe un programa en C llamado **slave.c**, que implemente el código necesario para ejecutar los siguientes pasos: (1) Abrir un fichero en modo lectura. El nombre del fichero se pasará como primer argumento del programa. (2) Descubrir el tamaño de ese fichero SIN hacer ningún acceso a disco. (3) Reservar una zona de memoria dinámica de ese tamaño, utilizando `sbrk(…)`, y asignar la dirección de esa zona a una variable llamada `buffer`. El objetivo es crear un buffer que pueda almacenar todo el contenido del fichero de entrada. (4) Leer en `buffer` el contenido del fichero. (5) Finalmente, escribir en salida estándar el contenido de buffer, liberar la memoria dinámica reservada y cerrar el fichero. La función `main` debe finalizar ejecutando `exit(0)`.

Ejercicio 3 (2 puntos)

Escribe un programa en C llamado **server1.c**. Lo primero que tiene que hacer este código es crear una pipe con nombre en el directorio actual de trabajo que se llama “MIPIPE”, si la pipe ya existe el programa debe continuar. A continuación, el programa debe implementar los siguientes pasos: (1) Abrir la pipe “MIPIPE” y leer de ella carácter a carácter. Si el carácter leído es ‘#’ (código 35 en la tabla ASCII), eso marca el inicio de un string (secuencia de caracteres que acaba con el carácter ‘\0’), que será el parámetro a pasar al ejecutable **slave.** (2) Cada vez que lea un string, el proceso principal debe crear un hijo. (3) El hijo mutará (i.e. reemplazará su imagen actual por una nueva) al programa que implementa **slave.c**. El proceso padre esperará hasta que el hijo acabe y entonces acabará. NOTA: puedes usar el Shell script “client.sh” para probar este ejercicio.

Ejercicio 4 (1,5 puntos)

Copia el fichero server1.c en **server2.c** y modifica server2.c de la siguiente manera. Ahora el programa principal recibe un argumento que representa el número máximo de procesos que el servidor puede crear (guárdalo en una variable global llamada `max`). En esta versión del servidor, la función main entra en un bucle infinito, que lee de la pipe con nombre. Cada vez que lea el carácter ‘#’ y un string, creará un nuevo hijo (a no ser que se haya llegado al máximo número de hijos, en cuyo caso no debe hacer nada). Como en el apartado anterior, el hijo de mutar al ejecutable slave. Los hijos deben ejecutarse concurrentemente con el padre y cada vez que un hijo muera el padre debe liberar su PCBs (*Process Control Block*) y continuar con su trabajo.

Ejercicio 5 (1,5 puntos)

Copia el fichero server2.c en **server3.c** y modifica server3.c de la siguiente manera. Ahora el proceso padre debe capturar el signal SIGHUP. Si el padre recibe un SIGHUP escribe en salida de errores estándar un mensaje con el número de hijos vivos que tiene el padre en ese momento. Además, modifica **server3.c** para que la salida de todos los hijos se envíe a un fichero llamado **logger** sin modificar el código de escritura de los hijos que aparece en slave.c. El servidor crea el fichero **logger** con permisos de lectura para todos los usuarios. Si el fichero existe, trúncalo.

Ejercicio 6 (1 punto)

Responde en el fichero **respuestas.txt** a las siguientes preguntas de manera justificada:

A) [0.25 puntos] Indica en qué región de memoria se guarda la variable `buffer` (programa slave.c) y `max` (programa server2.c). Indica qué línea(s) de código y/o comando(s) puedes utilizar para confirmar tu respuesta.

B) [0.25 puntos] En el ejercicio 4 has tenido que rellenar el campo `sa_flags` de una estructura `sigaction`. ¿Cómo afecta tu elección al resto del código de **server2.c**?

C) [0.25 puntos] ¿Qué línea de comandos tienes que ejecutar para enviar un SIGHUP al proceso server3 desde el terminal?

D) [0.25 points] ¿Qué línea de comandos tienes que ejecutar para averiguar cuántas llamadas a Sistema `read` ejecuta el programa `server1`? Ejecuta la línea de comandos y escribe también el descriptor de fichero (canal) que utiliza `server1` para sus lecturas.

Se tendrá en cuenta

Que se sigan las especificaciones del enunciado.

Que se cierren todos los canales abiertos una vez que ya no son necesarios.

Uso correcto de las llamadas a sistema.

Código bien indentado y legible.

Que las reglas del Makefile tengan los objetivos y las dependencias necesarias.

Que la función Usage() function muestre la línea de comandos correcta para invocar el programa cuando los parámetros de entrada usados no son los correctos.

A entregar

Un único fichero tar con los códigos Fuente, el Makefile y el fichero respuestas.txt

tar –cvzf final.tar.gz respuestas.txt client.sh Makefile *.c