Control Laboratori 2 SO1718q2

Makefile (0,5 puntos)

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

E/S (4 puntos)

Vamos a desarrollar un desencriptador de mensajes. El mensaje original ha sido encriptado aplicando una transformación a los caracteres en posiciones pares y otra transformación a los caracteres en posiciones impares. Para recuperar el mensaje original utilizaremos diversos programas auxiliares:

A) **[1 punto]** Implementad un programa (llamadlo **mux.c**) que concentre los caracteres que lea por los canales 20 y 21 a su salida estándar. Se escribirá por la salida estándar el primer carácter leído por el canal 20, a continuación el primer carácter leído por el canal 21, a continuación el segundo carácter leído por el canal 20, y así sucesivamente. En el momento que se detecte final de fichero (sea por el canal 20 o por el 21), el programa dejará de leer más caracteres y podrá finalizar.

   Observaciones: Haced las lecturas/escrituras **carácter a carácter**. Disponéis del ejecutable `mux_ok` que se comporta como debería hacerlo vuestro programa. Podéis probar vuestro programa con la orden:

```
./mux 20< test_fd20 21< test_fd21
```

B) **[3 puntos]** Implementad un programa (llamadlo **mux_buff.c**) con la misma funcionalidad que **mux.c** pero que haga las lecturas utilizando dos buffers de tamaño `N` bytes y las escrituras utilizando un buffer de tamaño `2·N` bytes, donde el valor `N` se debe recibir como parámetro por la línea de comandos. Los buffers para hacer las lecturas se crearán dinámicamente utilizando `malloc` y el buffer para hacer las escrituras se creará dinámicamente utilizando `sbrk`. Debéis liberar la memoria al acabar la ejecución del programa.

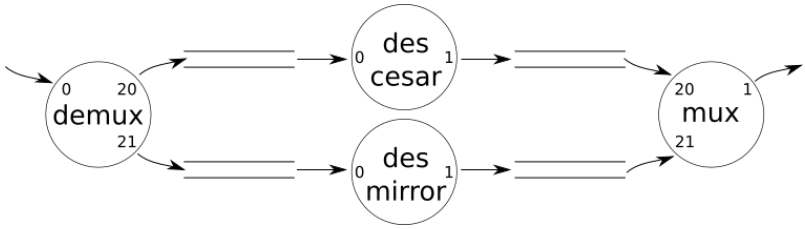
   Verificad con la utilidad `strace` que el ejecutable correspondiente a **mux_buff.c** hace las lecturas/escrituras mediante buffers. Adjuntad en **respuestas.txt** fragmentos de la salida de `strace` que lo muestren.

   Observación: suponiendo que vuestro código itere leyendo en cada iteración del canal 20 y a continuación del canal 21, podéis asumir que en cada iteración el número de caracteres que se leerán por el canal 21 serán los mismos o uno menos que los que se leerán por el canal 20.

Control Laboratori 2 SO1718q2

Pipes (4 puntos)

C) **[4 puntos]** Escribid un programa (**principal_np.c**) que se encargue de crear los procesos, named pipes y de realizar las redirecciones siguientes:



Para ello os subministramos diversos ficheros (no debéis modificarlos):

- El programa **des.c**. Espera un parámetro (el nombre del algoritmo de desencriptación, *cesar* o *mirror*). El programa aplica el algoritmo a todos los caracteres que se lean por la entrada estándar y escribe el resultado por la salida estándar. Podéis probarlo mediante las órdenes `./des cesar < test_cesar` o `./des mirror < test_mirror`
- El ejecutable `demux_ok`, complementario a `mux`, distribuye el contenido de un canal entre dos canales. Utilitzadlo en **principal_np.c**
- Los ejecutables `mux_ok` (una solución correcta al ejercicio A)) y `des_ok` (`des.c`. compilado). Utilizadlos en **principal_np.c**
- Para que podáis hacer pruebas del funcionamiento, el script `des_doble.sh` se encarga de invocar **secuencialmente** todos los ejecutables involucrados. Podéis examinarlo con cualquier editor de texto y probarlo mediante la orden: `./des_doble.sh < test_doble1`

Respecto a los parámetros de **principal_np.c**:

- Si no recibe ninguno, los datos encriptados se leerán por stdin y los datos desencriptados se escribirán por stdout.
- Si se recibe uno, se interpretará como el nombre del fichero con los datos encriptados; los datos desencriptados se escribirán por stdout.
- Si se reciben dos, el primero se interpretará como el nombre del fichero con los datos encriptados, el segundo se interpretará como el nombre del fichero donde almacenar los datos desencriptados. Si el fichero destino no existiera, deberá crearse con permisos `rw-r-----`; si ya existiera, su contenido previo debe ser descartado.
- El programa deberá rechazar cualquier otra parametrización.

A diferencia del script, los procesos creados por **principal_np.c** se deben ejecutar **concurrentemente** y las comunicaciones entre ellos deben realizarse mediante **named pipes** (en vez de ficheros temporales); si las named pipes no existen, deben crearse con protecciones `rw-------`.

Control Laboratori 2 SO1718q2

Ficheros (1,5 puntos, a responder en respuestas.txt)

D) **[0,5 puntos]** Indicad cómo averiguaríais cuántos inodes libres hay en la partición donde se almacena vuestro directorio actual.

E) **[1 punto]** Escribid una rutina `int write_directo(int fd, char c, int pos)` tal que, dado un canal correspondiente a un fichero ordinario, escriba el carácter `c` en la posición `pos`. Al finalizar la ejecución de la rutina, el puntero de lectura/escritura debe apuntar a la misma posición a la que apuntaba al iniciarse la ejecución de la rutina. En caso de error, la rutina devolverá -1; en cualquier otro caso, retornará 0.

Qué se tiene que hacer

- El Makefile
- El código de los programas en C con la función Usage() en cada programa
- El fichero respuestas.txt con las respuestas a las preguntas

Qué se valora

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

Qué hay que entregar

Un único fichero tar.gz con vuestros ficheros fuentes, Makefile, y las respuestas en respuestas.txt:

```
tar zcvf clab2.tar.gz Makefile respuestas.txt *.c
```