Examen final de laboratorio SO1819Q1

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 solo 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 (2 puntos)

Crea un programa, llamado **alfa.c** que esté bloqueado en un sigsuspend hasta que suceda una de estas dos cosas:

- O bien reciba un SIGUSR1, y escribirá la letra 'a' (en minúscula) por el canal 8 y vuelve a bloquearse.
- O bien reciba un SIGUSR2 y entonces acabará con exit(n), donde n es el número de caracteres que ha escrito, módulo 255.

Cualquier otro signal programable tiene que estar bloqueado.

Ejercicio 2 (1 punto)

Crea un programa, llamado **omega.c**, que acepte un identificador de proceso como argumento. Este programa ejecuta un bucle leyendo del canal 7 y se debe comportar de la siguiente manera:

- Si lee una 'z' (en minúscula) debe enviar el signal SIGUSR1 al proceso con identificador igual al argumento
- Si lee una 'Z' (en mayúscula) acabará con exit(0).

Cualquier otra letra se ignora.

Ejercicio 3 (4 puntos)

Para este ejercicio necesitas haber completado correctamente los dos anteriores. Adjuntamos el binario solución de cada uno de ellos.

Crea un programa, llamado **gestor.c**, que reciba como argumento el número par de procesos que tiene que crear. Debes guardar los identificadores de cada proceso en un vector para el que tienes que reservar memoria con la función malloc.

> **Escribe en el fichero respuestas.txt.** ¿Cómo puedes averiguar la cantidad de memoria total que ha reservado la llamada a malloc?

Los hijos en posición par del vector deben ejecutar el programa alfa.c, redireccionando las entradas pertinentes en su tabla de canales de tal manera, que todo lo que escriban aparezca por el terminal del padre.

Los hijos en posición impar deben ejecutar el programa omega.c, pasándole como argumento el pid del proceso creado justo antes que él. El proceso padre es el encargado de proporcionar los caracteres a los procesos omega, que están leyendo del canal 7. Para ello se utiliza una única pipe con nombre llamada POPIPE, que debes crear desde el terminal.

> **Escribe en el fichero respuestas.txt** la línea de comandos que has utilizado para crear la pipe.

El proceso padre ejecuta un bucle leyendo de su entrada estándar. Por cada tecla diferente de 'q', escribe el carácter 'z' en POPIPE. Si la tecla es 'q', escribe tantas 'Z's en POPIPE como hijos omega tenga y además envía SIGUSR2 a todos sus hijos alfa.

Como siempre, el padre debe esperar la muerte de sus hijos antes de acabar. Asegúrate también de que el padre libere memoria y entradas de la tabla de canales tan pronto como pueda.

> **Escribe en el fichero respuestas.txt**: mientras el padre espera teclas, averigua, desde Shell y mediante una sola línea de comandos, cuántos procesos omega se están ejecutando. Anota en "respuestas.txt" la línea de comandos Linux que has utilizado.

Ejercicio 4 (2 puntos)

A partir del anterior gestor.c, crea un programa llamado **gestorPro.c**. Ahora, el padre gestionará la muerte de todos sus procesos hijos de forma más elaborada. Cada vez que muera un hijo en posición par del vector de pids, escribirá **al final** del fichero "aes.int" y en formato **entero**, el número de 'a's que ha escrito el proceso hijo que acaba de morir.

> **Escribe en el fichero respuestas.txt**: El fichero "aes.int" lo debes crear desde terminal. Anota en el fichero "respuestas.txt" la línea de comandos que has utilizado para crearlo. Anota también la línea de comandos que has utilizado para averiguar su número de inodo.

Si muere un proceso en posición impar, el proceso padre escribirá por pantalla el pid del proceso que ha muerto.

Qué se tiene que hacer

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

Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas a sistema sea el correcto
- Que 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 en pantalla 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 el código fuente de los programas en C, el Makefile, y las respuestas en respuestas.txt:

```
tar zcvf final.tar.gz Makefile respuestas.txt alfa.c omega.c gestor.c gestorPro.c
```