Control Laboratorio SO2122q1

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 sólo los ficheros fuente. Los programas deben generarse si, y solo si, ha habido cambios en los ficheros fuente.

Control de errores (0,5 puntos)

Para todos los programas que se piden a continuación 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().

Ejercicio 1 (1,5 puntos)

Escribe un programa (jerarquia1.c) que reciba una secuencia de entre 1 y 10 nombres de fichero como argumento. El programa debe crear de forma **concurrente** tantos procesos hijo como nombres de fichero haya recibido. Cada proceso hijo se encarga de tratar uno de los ficheros. En este primer apartado el tratamiento consiste simplemente en mostrar por pantalla un mensaje con el nombre del fichero que le ha tocado y acabar la ejecución pasando como parámetro de finalización el número de orden de su creación.

Por su parte, el proceso padre sólo tiene que esperar a que acaben todos los procesos hijos. La espera la tiene que hacer en orden de creación (hasta que no libere el PCB del primer proceso creado no puede pasar a liberar el PCB del segundo). Para cada hijo muerto, deberá mostrar en pantalla un mensaje con el pid del proceso hijo y con el parámetro que el hijo ha pasado al exit.

Ejercicio 2 (2,5 puntos)

Haz una copia de jerarquia1.c y llámala jerarquia2.c. Modifica el código de jerarquia2.c para que cada hijo haga ahora un tratamiento diferente. **Cada hijo**, después de mostrar el mensaje por pantalla con fichero que tiene que tratar, creará un grupo de tres procesos que se ejecutarán de manera **secuencial. Cada grupo de 3 "hermanos"** mutará de manera que ejecutarán la siguiente secuencia de comandos (siendo nombre_fichero el nombre del fichero que le toca tratar y nombre_nuevo se crea concatenando nombre_fichero y el sufijo "sin_repes")**:**

1) wc -l nombre_fichero

   Muestra por pantalla el número de líneas de "nombre_fichero"

2) sort -u nombre_fichero -o nombre_nuevo

   Guarda en el fichero "nombre_nuevo" el contenido ordenado de "nombre_fichero" y sin líneas repetidas

3) wc -l nombre_nuevo

   Muestra por pantalla el número de líneas de "nombre_nuevo"

[0,5 puntos] ¿En qué orden se van a escribir los mensajes de todos los wc ejecutados? ¿Podemos garantizar que será siempre el mismo? Justifica tu respuesta en un fichero de texto llamado "respuestas.txt".

Ejercicio 3 (5 puntos)

Haz una nueva copia de jerarquía1.c y llámala signals.c. Modifica el código de signals.c permitiendo que sea el usuario el que decida cuándo empieza cada hijo con el tratamiento de su fichero mediante ctrl-C. La implementación que debéis hacer es la siguiente.

1) Código de cada hijo: empezará la ejecución con una **espera activa** que acabará al recibir el signal SIGUSR1. Al salir de la espera activa será cuando ejecute el tratamiento del primer ejercicio (mostrar el mensaje con el nombre del fichero y acabar).

2) Código del padre:

   - Una vez creados todos los hijos, el padre ejecutará un bucle para esperar tantos SIGINTs como hijos tenga. La espera tiene que ser pasiva (por bloqueo). Al salir de cada espera pasiva, se mostrará un mensaje por pantalla indicando que se ha salido de la espera y se enviará un signal de tipo SIGUSR1 al hijo que tenga el turno (el siguiente en orden de creación). **Simplificación:** podéis suponer que el usuario esperará a ver el mensaje de fin de espera antes de mandar el siguiente SIGINT.

   - Las esperas pasivas estarán limitadas a un máximo de 2 segundos: si en dos segundos no se ha recibido un SIGINT se enviará a todos los hijos un signal de tipo SIGKILL y se saldrá del bucle de esperar SIGINTS. **Simplificación**: podéis enviar un SIGKILL a todos los hijos independientemente de que hayan muerto ya o no.

   - Una vez fuera del bucle de espera de SIGINTS y antes de acabar el padre tratará **sin ningún orden** la muerte de sus hijos. Este tratamiento tiene que mostrar por pantalla un mensaje indicando si han muerto por signal o por exit. En caso de muerte por signal deben indicar el número de signal que lo ha matado y en caso de muerte por exit el parámetro de exit que habían utilizado.

NOTA: las llamadas a sistema relacionadas con signals y temporizadores que podéis usar son sólo sigaction, sigsuspend, alarm, sigprocmask y kill

Ejercicio extra (1 punto, la nota final será min (10, nota calculada))

Haz una copia de signals.c y llámala signals_extra.c. Modifica el código de signals_extra.c para que el proceso padre, cuanto antes, detecte el fin de cada hijo y realice el tratamiento correspondiente, sin esperar al final del programa. Haz además que, en caso de saltar el temporizador, el padre envíe SIGKILL sólo a los hijos que sigan vivos en ese momento.

Qué hay que hacer

- El Makefile
- Los códigos de los programas en C

- La función Usage() para cada programa
- El fichero respuestas.txt con todas las respuestas a las preguntas

Qué se valora

- Que sigas las especificaciones del enunciado
- Que el uso de las llamadas al sistema sea el correcto
- Que se comprueben los errores de **todas** las llamadas al sistema
- Que el código sea claro y correctamente indentado
- Que el Makefile tenga bien definidas las dependencias y objetivos
- Que la función Usage() muestre por pantalla como debe invocarse correctamente el programa en el caso 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 clab.tar.gz Makefile respuestas.txt *.c
```