Final Laboratorio SO2020-2021Q2

Indicaciones para hacer el Examen de Laboratorio

En este examen habrá dos entregas. Una a los 60 minutos del inicio del examen (Entrega-MITAD), en la que tendrás que entregar todo lo que hayas hecho hasta ese instante, y otra al final del examen (Entrega-FINAL), que se corresponde a la entrega final del examen.

Después del último ejercicio hay indicaciones para saber cómo comprobar la correcta ejecución de los programas. Para cada uno de ellos, os damos un ejecutable que hemos creado nosotros para que podáis validar que la ejecución de vuestro código es la esperada. De todas formas, si no sois capaces de hacer alguno de los ejercicios, pasad al siguiente y usad el ejecutable que nosotros os ofrecemos para evitar las dependencias entre códigos.

NOTA: Se considera entrega completa sólo si se han hecho las dos entregas.

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().

- Para todos los ejercicios: Cuando haya un error en los parámetros el programa terminará con un exit(1), cuando sea un error de llamada a sistema acabará con un exit(2)

(2 puntos) espera_sig.c

Haz un programa que llamaremos **espera_sig.c** que espera un signal de los siguientes: SIGINT, SIGUSR1, SIGUSR2

- El programa no debe tratar ningún otro signal.
- El programa debe esperar los signals sin consumir CPU.
- El programa tendrá los signals bloqueados (desde tan pronto como sea posible) excepto durante la espera pasiva de los signals.
- Si se recibe un SIGINT, SIGUSR1 o SIGUSR2 el programa escribirá un mensaje por su salida de error std indicando el signal recibido y terminará su ejecución con un estado 1, 2 o 3 respectivamente.
- Pruébalo enviando diferentes signals a mano desde un terminal.
- Apunta en el fichero de texto **respuestas.txt** la línea de comando que has utilizado para enviar los signals y copia la salida del programa.

Final Laboratorio SO2020-2021Q2

(3.5 puntos) n_espera_sig.c

Haz un programa, llamado **n_espera_sig.c**, que recibe un número (N) como parámetro.

El programa crea N procesos concurrentes. Cada uno de ellos escribirá por su salida std su PID y mutarán al programa anterior (espera_sig). El proceso inicial esperará a que terminen sus hijos y escribirá, para cada uno de ellos, su PID y su estado de finalización. Estos datos se escribirán en un fichero que llamaremos "exit_status.int". El programa debe crear el fichero si no existe y truncar su contenido si existe. El fichero tendrá permisos de lectura y escritura para el propietario, lectura para grupo y ninguno para los demás.

Los datos estarán en formato int (no texto). Este fichero contendrá los datos de los procesos que hayan acabado (2 ints para cada uno, para guardar el PID y el estado de finalización). Guardaremos estos datos sólo en caso que el proceso haya terminado por un exit. Además, han de estar en la posición correspondiente al orden de creación. Es decir, el primer proceso creado, independientemente de cuando termine, usará los bytes del primer y segundo int, el segundo a continuación etc. Para ello, debes utilizar los accesos directos (lseek) que hemos trabajado.

Los resultados se han de ir escribiendo a medida que acaban los procesos, no podéis esperarlos en orden de creación.

Haz una gestión de memoria dinámica para almacenar la información necesaria (excepto en caso que no sepas, asume un máximo de 20 procesos). Se valorará realizar tanto la reserva de memoria dinámica como la liberación de la misma antes de finalizar. Utiliza las funciones de la librería de C para gestionar la memoria.

(3.5 puntos) Controlando los procesos: lee_signal.c y envia_signal.c

Vamos a hacer dos programas sencillos que se comunicarán mediante una pipe con nombre.

(2.25 puntos) lee_signals.c: El programa lee_signals recibe un nombre de fichero como parámetro. Este fichero contiene pares de enteros en formato int. Estas parejas de números corresponden con pares PID-SIGNAL. El programa hará lo siguiente para cada par de números: leerá los dos números, los enviará por una pipe con nombre que llamaremos “mis_eventos”. Esta pipe se ha de crear por línea de comandos.

- Escribe en el fichero **respuestas.txt** la línea de comando para crear la pipe

Para cada par de números, el programa esperará 5 segundos (espera bloqueante) y entonces leerá el fichero “exit_status.int” y escribirá su contenido, en texto, por la salida std. Como sabemos que el contenido son los estados de los procesos que han ido terminando, escribiremos, para cada proceso, una frase del estilo “El proceso 0 con PID XXX ha terminado con estado YYY”. (XXX y YYY son los números que leerás del fichero). Se valorará no utilizar sleep para hacer la espera.

Para que tenga sentido, el fichero debe contener los PIDs correspondientes a los procesos que creará el programa del ejercicio anterior (n_espera_sig). Para generar este fichero os damos el programa *guarda_pares* que añade pares de números, que recibirá mediante parámetros de entrada, al fichero “datos.int”. En cada ejecución sólo acepta una dupla, que se añade al final del fichero. Por tanto, hay que introducir tantas líneas de comando como pares de números queréis que tenga el fichero. De

hecho, este fichero es el que luego usaréis como parámetro de entrada de lee_signals. Como el nombre del fichero es fijo recuerda borrarlo si cambian los PIDs.

- Para ver el número de los signals, puedes ejecutar “kill -l”

(1.25 puntos) envia_signals.c: El segundo programa leerá de la pipe “mis_eventos” los dos números y enviará un signal al proceso con PID = primer número y el signal indicado en el segundo número. Este programa no lee sólo dos números, sino que leerá todos los datos que haya en la pipe e irá enviando los signals correspondientes. Cada vez que envíe un signal escribirá por su salida std un mensaje indicando a quien envía el signal y qué signal es (el número es suficiente). Terminará su ejecución cuando no queden datos en la pipe.

Cómo validar la ejecución

- En una terminal ejecutáis el programa n_espera_sig para que cree varios procesos.
- En otra terminal, asegúrate de borrar el fichero datos.int. A continuación, usando el programa guarda_pares, generáis varios pares PID-SIGNAL en el fichero datos.int. Para ello, ejecuta tantas líneas de comandos como procesos hayas creado en n_espera_sig. En cada línea de comandos pon uno de los PIDs que muestra n_espera_sig y alguno de los signals SIGINT, SIGUSR1 y SIGUSR2.
- Ejecutáis envia_signals en otra terminal.
- Finalmente, en otra terminal lee_signals pasándole como parámetro el fichero datos.int. La salida para cada pareja de enteros que hayáis puesto en datos.int, debería corresponder con la información que hay en el fichero exit_status.int y que veréis por pantalla en texto. También podéis usar la línea de comandos “xxd exit_status.int” para leer el contenido de este fichero mostrando los valores en hexadecimal.

Qué hay que hacer

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

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

Final Laboratorio SO2020-2021Q2

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_lab.tar.gz Makefile *.c respuestas.txt
```