Desarrollo de un intérprete de comandos: Scripter

Descripción y consejos

Presentación

- Descripción
- Implementación
- Código inicial
- Entrega

Presentación

- **Descripción**
- Implementación
- Código inicial
- Entrega

Descripción

- Diseñar y codificar en lenguaje C sobre un SO Linux un programa que lea un archivo con comandos y los ejecute:
  - Programa a desarrollar: **scripter.c**
  - Funcionalidad:
    - Detectar las líneas de un fichero de comandos:
      - Leer carácter a carácter hasta detectar el fin de línea.
      - Insertar un '\0' al final del string.
    - Procesar cada línea para identificar:
      - Comandos
      - Argumentos
      - Redirecciones
      - Background
    - Ejecutar cada comando.

Descripción

- **Ejemplo de un fichero de entrada:**
  - La primera línea siempre estará presente.
  - Cada línea contiene un comando o secuencia de comandos.
  - No se permiten líneas vacías.

```
## Script de SSOO
ls -l
cat fich | grep palabra
cat fich > fich2 &
ls -l | sort < fichero
```

Descripción

- **Restricciones:**
  - El scripter debe ejecutar todos los comandos aunque estos fallen
    - Un fallo de un comando no implica un fallo del proceso scripter.
  - El scripter finalizará en los siguientes casos:
    - Cuando se han completado todos los comandos indicados en el fichero.
    - Cuando se produce un error de las llamadas al sistema utilizadas en el código (fork, exec, dup, etc.).
    - Cuando la primera línea del fichero de entrada sea diferente a "**## Script de SSOO**".
    - Se ha encontrado una línea vacía.
  - Mensajes de error:
    - Deben ser mostrados por la salida de error mediante **perror**.

Presentación

- Descripción
- **Implementación**
- Código inicial
- Entrega

Implementación

- **Procesamiento de comandos** → *int procesar_linea(char* linea);*
  - Se encarga de parsear una línea de comandos, organizando información relevante de la siguiente manera:

Incluída en el código inicial: *scripter.c*

  - **char *argvv[]**: matriz similar a "argv" en los programas clásicos, que almacena un comando con sus argumentos.
  - **char *filev[3]:** vector de redirecciones. En caso de detectar una redirección, se referencia al nombre del fichero en la posición que corresponda:
    - filev[0] → STDIN_FILENO
    - filev[1] → STDOUT_FILENO
    - filev[2] → STDERR_FILENO
  - **int background:** indica si hay que ejecutar un comando o secuencia de comandos en foreground (0) o bg (1).

Implementación

- **Lectura de fichero de comandos**
  - Se debe leer el fichero de entrada carácter a carácter hasta encontrar un salto de línea e insertar un '\0' al final del string.
  - Una vez que se tiene una línea, se debe llamar a la función *procesar_linea* para generar las estructuras de datos que el alumno utilizará para ejecutar los comandos mediante el uso de procesos.

Implementación

- **Consideraciones:**
  - Una línea puede ser un comando con o sin argumentos.
  - Una línea puede ser una serie de comandos encadenados (pipes).
  - Una línea puede contener caracteres especiales al final del último comando:
    - **“ < fichero ”**: Redirección de entrada.
    - **“ > fichero ”**: Redirección de salida.
    - **“ !> fichero ”:** Redirección de error.
    - **“ & ”**: Ejecución en background del comando/secuencia.

Implementación

- **Consejos de desarrollo:**
  1. Lectura línea por línea del fichero de entrada.
  2. Ejecución de comandos simples (ls -l, wc, grep).
  3. Ejecución de comandos simples en background (&).
  4. Ejecución de secuencias de comandos con pipes (|).
     - Número de comandos encadenados limitado a 3.
     - Implementaciones que consideren un algoritmo para n comandos tendrán nota complementaria (nota máxima 10)
  5. Ejecución de comandos simples, secuencias con redirecciones (entrada, salida y de error), y en background.
  6. Desarrollo del comando externo ***mygrep***

Implementación

- **Comando externo → *mygrep <ruta_fichero> <cadena_a_buscar>***
  - Busca una cadena de texto en ficheros.
  - En caso de encontrar la cadena, se muestra por pantalla las líneas que la contienen.
  - En caso de no encontrar la cadena, se muestra el mensaje:
    - *" %s not found.\n "*

      Donde **%s** es la cadena a buscar.
  - En caso de error:
    - Se debe mostrar un mensaje de error por la salida de error mediante **perror** (man 3 perror).
    - Se retornará -1.

Implementación

- **Comando externo → *mygrep <ruta_fichero> <cadena_a_buscar>***
  - Este comando debe ser ejecutado como cualquier otro comando del SO en el scripter.
  - La ejecución de este programa se realizará desde archivo de comandos de entrada. Utilizando "exec" con la ruta donde se encuentra el binario (*man 3 exec*).
  - Modificación del *Makefile*:
    - Una vez implementado el programa, se debe agregar una línea de compilación al Makefile para que, al ejecutar "make", "make mygrep" y "make clean", el programa se compile automáticamente o se borren los ficheros generados.
  - Este programa también debe ser considerado en la batería de pruebas de la memoria.

Presentación

- Descripción
- Implementación
- **Código inicial**
- Entrega

Código inicial

- Se proporciona un ZIP inicial con el código fuente de apoyo (*p2_scripter.zip*).
- Al descomprimir se encuentran los siguientes ficheros:
  - **Makefile** → fichero para compilar los programas.
  - **scripter.c** → fichero donde el alumno debe implementar el código del intérprete de comandos. También contiene la función *procesar_linea()*.
  - **mygrep.c** → fichero donde el alumno debe implementar el código del comando externo descrito anteriormente.

Código inicial

- **Compilación**
  - make
  - make scripter
  - make mygrep → debe ser agregado en Makefile por el alumno
  - make clean → debe ser modificado en Makefile por el alumno

- **Ejecución**
  - ./scripter <fichero_de_comandos>

Presentación

- Descripción
- Implementación
- Código inicial
- **Entrega**

Cómo hacer la entrega

- Tras finalizar la implementación y realizar las pruebas:
  - Comprobar que compila y ejecuta en Guernika
  - Si el apartado anterior se cumple:
    - Comprimir los ficheros solicitados con el **formato especificado**: **ssoo_p2_NIA1_NIA2_NIA3**.zip
      - Makefile
      - mygrep.c
      - scripter.c
      - autores.txt
    - Realizar la entrega en Aula Global.
    - Descargar el fichero entregado y **volver a comprobar** que compila y pasa las pruebas.
  - Convertir la memoria a PDF (*ssoo_p2_NIA1_NIA2_NIA3.pdf*) y entregar por Turnitin en Aula Global (**sólo se admite una entrega, no se pueden hacer reenvíos**).

Cómo hacer la entrega

- Plazo de entrega:

03 de abril de 2025 hasta las 23:55

  - Únicamente **un miembro** del equipo sube la práctica a los entregadores habilitados.
  - Entregas tardías recibirán penalizaciones dependiendo del tiempo de retraso.
  - NO se admitirán entregas por correo electrónico.

- Si todo ha ido bien… ¡A descansar!

- ¡MENTIRA! ¡Estáis estudiando!… ¡Ya descansaréis en verano o cuando terminéis!

- Pero al menos habréis hecho las cosas bien…