Sistemas Operativos

2025

Práctico 1: Comandos, redirección, pipes…

Ejercicios:

1. Realizar los siguiente comandos en el shell (de comandos) de su sistema (familia UNIX).
    a. Listar los procesos que están en estado de ejecución (ver el comando `ps`, hacer `man ps` para leer el *manual* del comando).
    b. Listar los usuarios conectados (logged) al sistema con el comando `who` y ver la salida del comando `who am i`.
    c. Determinar cuál es el *directorio (carpeta) corriente*.
    d. Determinar cuál es su directorio de trabajo (*home*) (ver comandos `cd`).
    e. Listar los archivos (con sus detalles) de la carpeta corriente y de la carpeta raíz sin cambiarse de directorio.
    f. Listar el contenido de algún archivo. También hacerlo por páginas (o pantallas).
    g. Listar y analizar las *variables de ambiente* de su sesión (ver comando `env`).
    h. Crear un archivo con al menos tres comandos diferentes.
    i. Crear un *enlace simbólico* o *alias* a un archivo o carpeta (ver el comando *ln*).
    j. Eliminar el *enlace simbólico* creado. Determinar la diferencia entre *symbolic links* y *hard links*.

2. Comandos de shell: Redirección, secuencia y concurrencia.
    a. Dados dos archivos de texto el cual contiene una lista de direcciones de correo electrónico (una por línea), generar un archivo con la lista ordenada y sin duplicados. Ayuda: ver `sort` y `uniq`. Los archivos temporales deberían crearse en `/tmp`.
    b. Idem al anterior pero hacerlo en un único comando usando el operador secuencial.
    c. Escribir un comando que filtre y cuente las direcciones terminadas en *google.com*. Ver `grep` y `wc`.

3. Repetir lo del primer ítem del ejercicio anterior pero sin usar archivos temporales (ver *pipes*).

4. Escribir un *shell script* con los comandos del último ítem del ejercicio anterior. Probar su ejecución mediante el comando `sh` y luego agregando la línea de intérprete. Dar permisos de ejecución.

5. Escribir un programa C que retorne como *exit status* el valor ingresado en la línea de comandos. Ejemplo de invocación: `./myprog 1`.
    a. Escribir un comando para ejecutar el programa y determinar su valor de salida. Ayuda: La variable $? toma el valor de salida del último comando ejecutado.
    b. Escribir un comando que ejecute otro si `myprog` finaliza con éxito (0).
    c. Escribir un comando que ejecute otro si `myprog` no finaliza con éxito.

6. La biblioteca estándar de C ofrece la función `int system(const char* string)`, la cual crea un subproceso (*child*) generado por el comando pasado en su argumento. Definir `system()` en términos de las llamadas al sistema `fork()`, `exec()` y `wait()` (ver páginas del manual).

7. Implementar un programa en C que comunique al proceso padre con un proceso hijo por medio de un *pipe* (ver `pipe()` sy scall). El proceso padre deberá enviarle un string y el hijo deberá responderle con el string invertido. El proceso padre deberá mostrar el identificador de proceso (pid) del hijo y la cadena recibida.

8. Idem al ejercicio anterior pero haciendo que los programas que se comuniquen por medio de un FIFO (named pipe). Ayuda: ver el comando `mkfifo` para crearlo.

9. Escribir un programas C `minish.c` que reciba hasta 3 argumentos en la línea de comandos y emule los siguientes comandos de shell:
    a. `cmd1 &`
    b. `cmd1 < input_file`
    c. `cmd > output_file`
    d. `cmd >> output_file`
    e. `cmd1 ; cmd2`
    f. `cmd1 || cmd2`
    g. `cmd1 && cmd2`
    h. `cmd1 | cmd2`

10. Hacer un programa C que, estando en un ciclo, capture la señal `SIGINT` (generada por la terminal con Ctrl-C). El manejador de la señal deberá contar el nro señales recibidas. Terminar el proceso enviando la señal al proceso usando el comando `kill`.

11. Hacer un programa C que dado un arreglo de valores enteros, de dimensión *N*, cree *p* procesos hijos, los cuales deberán calcular el mínimo valor de la porción que le corresponda (*N/p* elementos) y enviarlo al padre a través de un pipe. El padre deberá esperar los resultados y computar el mínimo global.

12. Escribir un programa que lea de un FIFO (named pipe). Si no recibe contenidos luego de N segundos (dado como argumento en la línea de comandos), debería finalizar. Ayuda: Ver la función `alarm()`.