Sistemas Operativos 2/2024
Laboratorio 2

Profesores:
Cristóbal Acosta (cristobal.acosta@usach.cl)
Fernando Rannou (fernando.rannou@usach.cl)
Miguel Cárcamo (miguel.carcamo@usach.cl)
Ayudantes:
Ricardo Hasbun (ricardo.hasbun@usach.cl)
Daniel Calderón (daniel.calderon.r@usach.cl)

November 21, 2024

1 Objetivos Generales

Este laboratorio tiene como objetivo aplicar técnicas de programación imperativa mediante lenguaje C, como la recepción de parámetros mediante `getopt` y compilación mediante `Makefile` sobre sistema operativo Linux.

Con los tres programas que se construyeron en el trabajo anterior (laboratorio 1), ahora se debe desarrollar un cuarto programa, el que tendrá como finalidad llamar y coordinar a los otros tres.

2 Objetivos Específicos

1. Conocer y usar las funcionalidades de `getopt()` como método de recepción de parámetros de entradas.
2. Crear procesos utilizando la función `fork()`.
3. Utilizar `pipe()` para la comunicación entre procesos.
4. Hacer uso de la familia `exec()` para ejecutar procesos.
5. Duplicar descriptores con `dup()` o `dup2()`.
6. Validar los parámetros recibidos con `getopt()`.
7. Practicar técnicas de documentación de programas.
8. Conocer y practicar uso de `Makefile` para compilación por partes de programas.

3 Enunciado

3.1 Funciones UNIX

Este tipo de funciones son componentes clave de los sistemas operativos basados en UNIX, como Devian, Linux o macOS, entre otros similares. Se encuentran diseñadas para realizar tareas esenciales en el sistema operativo y proporcionar una capa de interacción eficiente entre el usuario y el hardware o entre diferentes programas. Estas funciones pueden ser tanto aquellas que vienen integradas con el sistema operativo, como las que realizan llamadas al sistema para poder interactuar con el núcleo (kernel) del SO. Algunos usos de estas funciones son:

- Gestión de archivos y directorios: `ls`, `cd`, `cp`, `mv`, `rm`.
- Manipulación de procesos: `ps`, `kill`, `top`.
- Interacción con el hardware: `mount`, `df`, `lpstat`, `ifconfig`.
- Manipulación de texto: `cat`, `tail`, `awk`, `cut`, `echo`, `paste`.

La importancia de estos comandos radica en su simplicidad, flexibilidad y robustez, en el que cada uno de ellos está hecho para cumplir una tarea en específico, y que sea realizado de la mejor forma posible. Esto permite que sean combinables entre sí para poder resolver problemas complejos.

4 El programa

Para esta experiencia de laboratorio, se trabajará con un archivo `.csv`. Este tipo de archivos consisten en datos organizados en filas y columnas, donde cada fila representa un registro, y cada columna contiene un valor. Las columnas se encuentran delimitadas por comas (,), punto y comas (;), 1 o más espacios en blanco ( ) y tabulaciones (\t). A continuación, se presenta un ejemplo de archivo `.csv`:

```
Nombre;Edad;Ciudad;Profesión
Juan Pérez;30;Madrid;Ingeniero
María Gómez;25;Barcelona;Doctora
Carlos Sánchez;40;Valencia;Abogado
Ana López;35;Madrid;Arquitecta
```

Por otro lado, para este laboratorio deberán realizar algunas funciones simplificadas de las que provienen de UNIX, en el que realizarán distintas manipulaciones de texto sobre los archivos `.csv` que se les dispondrán. Cada una de estas funciones será un archivo .c distinto, el cual realizará el símil de la función UNIX respectiva. Finalmente, estos archivos se ejecutarán por línea de comando.

4.1 Cut

El programa `cut` se usa para extraer información sobre nuestro archivo de entrada `.csv` o, en caso de no especificar un archivo de entrada, sobre un `stdin`, con el objetivo de obtener ciertas columnas de interés. Al momento de usar la instrucción `cut`, esta extraerá la o las columnas seleccionada(s) del

archivo. La selección de la o las columnas se realiza en la misma línea de comando al momento de dar la instrucción cut. No se admite el uso de funciones de la librería `string.h` (como el `strtok`) o similar que permita separar la información mediante un delimitador. Finalmente se deberá guardar la información extraída en un archivo de salida o, en caso de no especificar un archivo de salida, la información resultante será mostrada por `stdout`. En el caso de que se extraigan 2 o más columnas, estas deben ir separadas por el mismo delimitador.

Por ejemplo, tomando en consideración el archivo `csv` anterior, si se quiere extraer la columna 3 y 4 (correspondiente a la Ciudad y Profesión, respectivamente), el archivo de salida debería de contemplar lo siguiente:

```
Ciudad;Profesión
Madrid;Ingeniero
Barcelona;Doctora
Valencia;Abogado
Madrid;Arquitecta
```

4.2 Srep

El programa `srep` (String Replace) realiza un reemplazo del "string objetivo" (el que se encuentra dentro del archivo) por el "string nuevo" (el que ocupará el nuevo lugar dentro del archivo). Este cambio debe realizarse para todas las instancias en el que se encuentre el "string objetivo". Finalmente se deberá guardar la información reemplazada en un archivo de salida.

Un ejemplo de ello, utilizando nuevamente el `.csv` anterior, es que necesiten cambiar la ciudad *Madrid* por *Santiago*. Luego, el archivo de salida sería del modo:

```
Nombre;Edad;Ciudad;Profesión
Juan Pérez;30;Santiago;Ingeniero
María Gómez;25;Barcelona;Doctora
Carlos Sánchez;40;Valencia;Abogado
Ana López;35;Santiago;Arquitecta
```

4.3 Count

El programa `count` se usa para obtener el total de líneas, palabras o bytes contenidos en el archivo csv, siendo en el caso de indicar un archivo de entrada; en caso contrario, la entrada será por `stdin`. Notar que para cualquier carácter, se sumarán los casos en que sean mayúsculas o minúsculas, y en el caso de una vocal, también se contabilizarán cuando posea alguna tilde (recordar que los saltos de línea, tabulaciones, espacios, entre otros caracteres delimitadores, también se consideran un carácter).

Un ejemplo de lo anterior, usando nuevamente el `.csv` de prueba, es que se obtener la cantidad total de caracteres y, además, obtener la cantidad de lineas en el archivo, obteniéndose la siguiente salida (por `stdout`):

```
5 180
```

Teniendo en cuenta que siempre tendrá prioridad la cantidad de líneas por sobre la cantidad de caracteres.

4.4 Línea de comando

Tomando como ejemplo el siguiente archivo `input.txt`:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:102:105::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:103:106:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
```

La ejecución del programa tendrá los siguientes parámetros que deben ser procesados por `getopt()`:

4.4.1 Cut

Para el programa `cut`, las flag son las siguientes:

- `-d`: indica el carácter separador de las columnas. Por ejemplo, podría ser: coma (,), punto y coma (;), dos puntos (:), 1 espacio en blanco (b), 2 o más espacios en blanco (bbbbb igual a 5 espacios en blanco) o una tabulación (t). En caso de no indicar el carácter, el utilizado por defecto será una tabulación.
- `-c`: indicar la o las columnas. Por ejemplo: 2,3,4 especifica las columnas 2, 3 y 4, separadas por el delimitador especificado. También es posible indicar sólo una columna. En caso de no especificar qué columnas extraer, se considerarán todas las columnas.
- `-i`: nombre del archivo de entrada. En caso de no especificar, deberá asumir que la entrada es por `stdin`.
- `-o`: nombre del archivo de salida. En caso de no especificar, deberá asumir que la salida es por `stdout`.

Por ejemplo:

```
$ ./cut -i input.txt -o output.txt  -d : -c 2,4
```

Obteniéndose el archivo `output.txt` con la siguiente información:

```
x:0
x:1
x:2
x:3
x:65534
x:60
x:12
x:7
x:8
x:9
x:10
x:13
x:33
x:34
x:38
x:39
x:41
x:65534
x:102
x:103
x:105
x:106
```

4.4.2 Srep

Para el programa `srep`, las flag son las siguientes:

- `-s`: para indicar el string objetivo.
- `-S`: para indicar el string nuevo.
- `-i`: nombre del archivo de entrada. En caso de no especificar, deberá asumir que la entrada es por `stdin`.
- `-o`: nombre del archivo de salida. En caso de no especificar, deberá asumir que la salida es por `stdout`.

Por ejemplo:

```
$ ./srep -i input.txt -o output.txt -s / -S \
```

Obteniéndose el archivo `output.txt` con la siguiente información:

```
root:x:0:0:root:\root:\bin\bash
daemon:x:1:1:daemon:\usr\sbin:\usr\sbin\nologin
bin:x:2:2:bin:\bin:\usr\sbin\nologin
sys:x:3:3:sys:\dev:\usr\sbin\nologin
sync:x:4:65534:sync:\bin:\bin\sync
games:x:5:60:games:\usr\games:\usr\sbin\nologin
man:x:6:12:man:\var\cache\man:\usr\sbin\nologin
lp:x:7:7:lp:\var\spool\lpd:\usr\sbin\nologin
mail:x:8:8:mail:\var\mail:\usr\sbin\nologin
news:x:9:9:news:\var\spool\news:\usr\sbin\nologin
uucp:x:10:10:uucp:\var\spool\uucp:\usr\sbin\nologin
proxy:x:13:13:proxy:\bin:\usr\sbin\nologin
www-data:x:33:33:www-data:\var\www:\usr\sbin\nologin
backup:x:34:34:backup:\var\backups:\usr\sbin\nologin
list:x:38:38:Mailing List Manager:\var\list:\usr\sbin\nologin
irc:x:39:39:ircd:\run\ircd:\usr\sbin\nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):\var\lib\gnats:\usr\sbin\nologin
nobody:x:65534:65534:nobody:\nonexistent:\usr\sbin\nologin
systemd-network:x:100:102:systemd Network Management,,,:\run\systemd:\usr\sbin\nologin
systemd-resolve:x:101:103:systemd Resolver,,,:\run\systemd:\usr\sbin\nologin
messagebus:x:102:105::\nonexistent:\usr\sbin\nologin
systemd-timesync:x:103:106:systemd Time Synchronization,,,:\run\systemd:\usr\sbin\nologin
```

4.4.3 Count

Para el programa `count`, las flag son las siguientes:

- `-C`: cuenta el número de caracteres.
- `-L`: cuenta el número de lineas.
- `-i`: nombre del archivo de entrada. En caso de no especificar, deberá asumir que la entrada es por `stdin`.

A continuación, se presentan diversos escenarios respecto al programa `count`:

```
$ ./count -i input.txt -C
1180
$ ./count -i input.txt -L
22
$ ./count -i input.txt -L -C
22 1180
$ ./count -i input.txt -C -L
22 1180
```

4.5 Lógica de la solución

Anteriormente, se describieron los programas `cut, count y srep`, los que debieron ser construidos e implementados en el laboratorio 1. Ahora se deberá desarrollar un cuarto programa, el que tendrá la misión de llamar y coordinar a los programas mencionados anteriormente (`cut, count y srep`). Este cuarto programa debe ser llamado `lab2.c` el cual recibirá como argumento de línea de comando un string que describe la ejecución de uno, dos o tres de los programas (`cut, count y srep`) conectados por pipes. Por ejemplo,

```
$ ./lab2 ./srep -i input.txt -s / -S \ | cut -d : -c 2,4 -o output.txt
```

La idea del lab2 es simular las operaciones que realiza la bash, cuando se ejecuta este comando. Es decir, la ejecución del lab2 del ejemplo anterior debiera producir los mismo resultados que:

```
$ ./srep -i input.txt -s / -S \ | cut -d : -c 2,4 -o output.txt
```

Entonces, el proceso `lab2` es el encargado de crear a los procesos `srep` y `cut` y conectarlos por pipes, como lo vimos en clases. En el ejemplo anterior, `lab2` debe:

1. Leer sus argumentos de línea de comandos
2. *Parsear* estos argumentos y determinar, los procesos que deben ejecutarse, los pipes que deben crearse, y los argumentos de línea de comando para cada programa
3. Duplicar `stdin` y/o `stdout` con los pipes según corresponda
4. Crear tantos hijos como procesos ejecutará
5. Finalmente, los hijos ejecutar los programas con sus respectivos argumentos

Los programas del lab1 deben modificarse, de tal forma que ahora se pueda distinguir cuándo la entrada de un proceso es por archivo (opción `-i`) o por `stdin`, y cuándo la salida es por archivo (opción `-o`) o por `stdout`. En el ejemplo, el proceso `srep` debe leer su entrada desde archivo, pero su salida debe ser por `stdout`. En cambio, el proceso `cut` debe leer por `stdin`, y escribir en el archivo de salida. Sin embargo, en el siguiente ejemplo:

```
$ ./lab2 ./srep -i input.txt -s / -S \ | cut -d : -c 2,4
```

`cut` debe leer por `stdin` y escribir por `stdout`.

Luego, cada programa debe tener la capacidad de determinar si su operación de I/O, ya sea de lectura o escritura, es por archivo o por `stdin`, y `stdout`. También, el proceso `lab2` debe saber cuándo duplicar las entradas y salidas estándar con los pipes. Por ejemplo, en la siguiente ejecución

```
$ ./lab2 cut -i input.txt -d : -c 2,4  | ./srep  -s / -S \ | count -C
```

- La salida estándar de `cut` se debe duplicar con el lado de escritura del primer pipe
- La entrada estándar de `srep` se debe duplicar con el lado de lectura del segundo pipe
- La salida estándar de `srep` se debe duplicar con el lado de escritura del segundo pipe

- La entrada estándar de `count` se debe duplicar con el lado de lectura del segundo pipe.

Además, en el caso anterior, `cut` debe imprimir sus resultado por `stdout`

Otro ejemplo:

```
$ ./lab2 ./cut -i input.txt -d : -c 2,4 | srep -s / -S \ | count -o output.txt -C -L
```

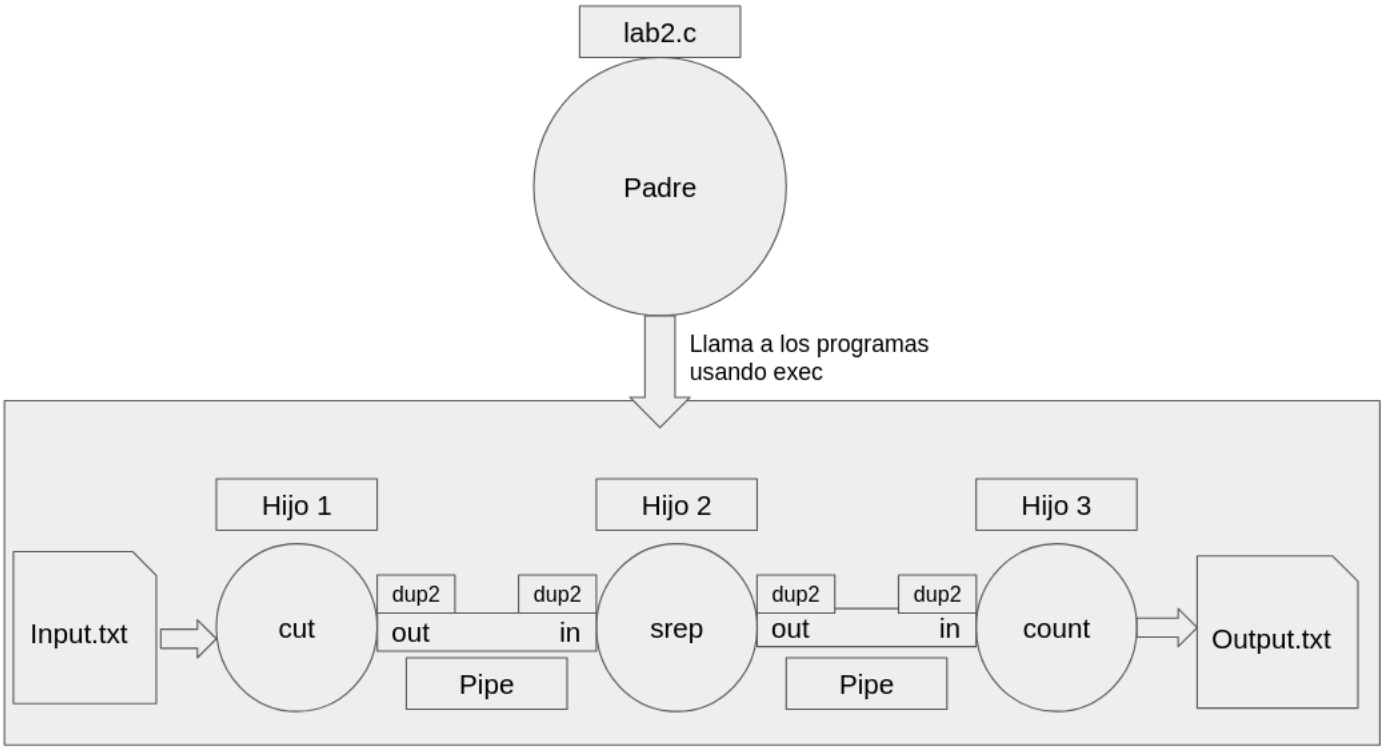

Figure 1: El programa lab2.c es quien debe coordinar a los otros tres programas. Para después, llamarlos usando alguna función de la familia `exec` según la instrucción ingresada.

Luego, se muestra el diagrama secuencial entre programas según linea de comando $3 que se muestra en la figura N°1. Sin embargo, no necesariamente deben de seguir el orden que se muestra. En este ejemplo, el primer hijo es el encargado de ejecutar a `cut`, para luego enviar el resultado a través del `stdout` del pipe, con el objetivo de que `srep`, siendo ejecutado por el hijo 2, lea la información entregada por el pipe por `stdin`. Finalmente el hijo 3, que ejecuta a `count`, generará el archivo de salida.

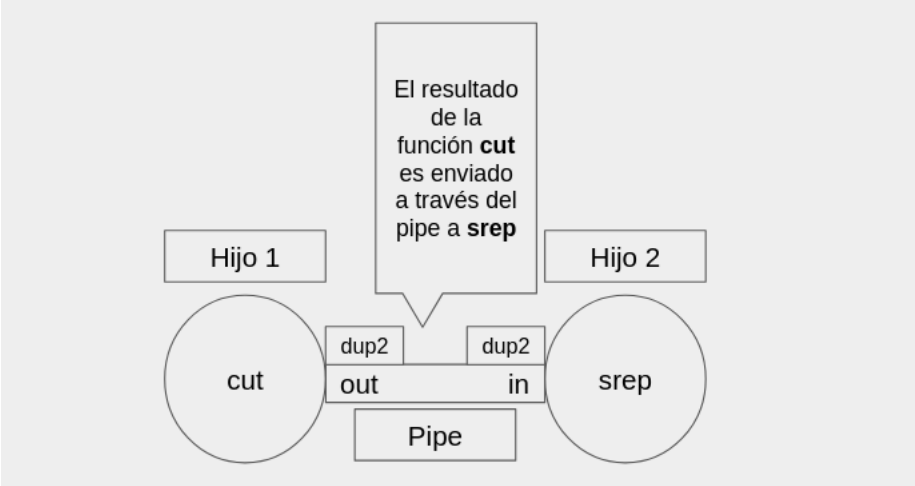

Figure 2: En este caso, la función `cut` envía sus resultados por `stdout`, a través de un pipe. Qué luego, la función `srep` recibe por `stdin`.

4.6 Requerimientos

Como requerimientos no funcionales, se exige lo siguiente:

- Debe funcionar en sistemas operativos con kernel Linux.
- Debe ser implementado en lenguaje de programación `C`.
- Se debe utilizar un archivo `Makefile` para compilar los distintos targets.
- Realizar el programa utilizando buenas prácticas, dado que este laboratorio no contiene manual de usuario ni informe, es necesario que todo esté debidamente comentado.
- Los programas se encuentren desacoplados, es decir, que se desarrollen las funciones correspondientes en otro archivo .c para mayor entendimiento de la ejecución.

5 Entregables

El laboratorio es en parejas. Si se elije una pareja, esta no podrá ser cambiada durante el semestre. Se descontará 1 punto (de nota) por día de atraso con un máximo de tres días, a contar del cuarto se evaluará con nota mínima. Debe subir en un archivo comprimido ZIP (una carpeta) a USACH virtual con, al menos, los siguientes entregables:

- `Makefile`: Archivo para compilar los programas.
- `lab2.c`: archivo con el desarrollo del proceso padre, el cual se encargará de crear a los hijos, los que luego ejecutarán cada programa (`cut, srep o count`) respectivamente.
- `cut.c`: archivo con el desarrollo del programa `cut`.
- `srep.c`: archivo con el desarrollo del programa `srep`.

- `count.c`: archivo con el desarrollo del programa `count`.

Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje. Se deben comentar todas las funciones de la siguiente forma:

```
// Entradas: explicar qué se recibe
// Salidas: explicar qué se retorna
// Descripción: explicar qué hace
```

El archivo comprimido (al igual que la carpeta) debe llamarse:
RUTESTUDIANTE1_RUTESTUDIANTE2.zip
Ejemplo 1: 19689333k_186593220.zip

- **NOTA 1**: El archivo debe ser subido a uvirtual en el apartado "Entrega Laboratorio N°2".
- **NOTA 2**: Cualquier diferencia en el formato del laboratorio que es entregado en este documento, significará un descuento de puntos.
- **NOTA 3**: SOLO UN ESTUDIANTE DEBE SUBIR EL LABORATORIO.
- **NOTA 4**: En caso de solicitar corrección del laboratorio, esta será en los computadores del Diinf, es decir, si funciona en esos computadores no hay problema.
- **NOTA 5**: Cualquier comprimido que no siga el ejemplo 1, significará un descuento de 1 punto de nota.

6 Fecha de entrega

JUEVES 05 DE DICIEMBRE, 2024.