2022

PROYECTO – SISTEMAS OPERATIVOS

COMISIÓN N°23

Gonzalo Martin Perez | Tomas de Giusti

16-11-2022

ÍNDICE

Experimentación de Procesos y Threads con los sistemas operativos

En esta sección se desarrollarán los diferentes aspectos de implementación a lo largo del proyecto, las convenciones adoptadas, planteos de los ejercicios, resultados obtenidos, situaciones improvistas que surgieron, entre otros.

Procesos, Threads y Comunicación

Dentro del desarrollo del proyecto, distinguimos claramente esta sección, ya que es una de las principales, puesto que en ella se realizan las diferentes actividades de implementación y experimentación en lenguaje C con Procesos, Threads y los diferentes mecanismos de comunicación utilizados.

Planta de reciclado

La metodología empleada para el desarrollo de la planta de reciclado fue utilizar una cierta cantidad de procesos, en este caso 9, que realizaban diferentes actividades dentro del funcionamiento de la planta. La distribución de las tareas es la siguiente:

- *3 procesos recolectores*: Son los encargados de generar y empaquetar los ítems de basura.
- *2 procesos clasificadores*: Clasifican cada uno de los ítems empaquetados por los recolectores.
- *4 procesos recicladores*: Son los encargados de recibir los ítems clasificados y los procesan. Des estos últimos procesos, hay uno por tipo de residuo. En caso de que no se dispongan residuos de su tipo, pueden optar por ayudar a los otros procesos recicladores, o bien, tomar mate.

Estrategia con PIPES

Para la planta de reciclado implementada con PIPES, la estrategia empleada fue utilizar en total 5 pipes en total para la comunicación:

- Un proceso principal (o padre) que se encarga de crear los pipes y cada uno de los procesos recolectores, clasificadores y recicladores.
- Un pipe compartido por recolectores-clasificadores, en el cual los procesos recolectores escriben los residuos empaquetados y los procesos recolectores los leen para poder clasificarlos y distribuirlos.
- Cuatro pipes compartidos por clasificadores-recolectores, en los cuales los procesos clasificadores escriben los ítems de basura a los recicladores para que puedan leerlos y procesarlos. Los dos recolectores pueden escribir en cada uno de los pipes al realizar la distribución, mientras que los procesos recicladores tienen un pipe principal asociado como prioridad para leer su tipo de residuo. Es decir, los procesos recicladores solo leen de los pipes de otros recicladores en caso de realizar ayuda.

Estrategia con COLA DE MENSAJES (MQ)

Para la planta de reciclado implementada con MQ (Colas de Mensajes), la estrategia fue utilizar una única cola de mensajes para toda la comunicación de los procesos:

- Un proceso principal (o padre) que se encarga de crear la cola de mensajes y cada uno de los procesos recolectores, clasificadores y recicladores.
- Como existe una única cola de mensajes, se representan las comunicaciones mediante tipos:
    - *Tipo 1*: Representa los ítems de basura recolectados, pendiente de clasificación.
    - *Tipo 2*: Representa los ítems de basura del tipo 'Vidrio.
    - *Tipo 3*: Representa los ítems de basura del tipo 'Cartón'.
    - *Tipo 4*: Representa los ítems de basura del tipo 'Plástico'.
    - *Tipo 5*: Representa los ítems de basura del tipo 'Aluminio'.
- Los procesos recolectores por su parte generar ítems de basura (ahora son mensajes), y los empaquetan con su respectivo nombre (Vidrio, Cartón, Plástico o Aluminio). En cuanto al tipo, los recolectores les asignan tipo 1 por defecto, para que luego sean clasificados.
- Los clasificadores reciben los ítems de tipo 1 enviados por los recicladores y los clasifican en base al nombre del ítem recibido.
    - Si su nombre es "Vidrio" se le asigna tipo 2.
    - Si su nombre es "Cartón" se le asigna tipo 3.
    - Si su nombre es "Plástico" se le asigna tipo 4.
    - Si su nombre es "Aluminio" se le asigna tipo 5.
- Los recicladores procesan los ítems recibidos de un tipo correspondiente y en caso de no disponer ítems de su tipo, ayudan a los otros recicladores reciclando de sus tipos o en su defecto, si no requieren ayuda, toman mate.
- Se modula en diferentes archivos aprovechando las ventajas que ofrece utilizar colas de mensajes y execvp():
    - El archivo principal, en el que se crea la Cola de Mensajes y los procesos recolectores, clasificadores y recicladores.
    - El archivo para recolectores, que tiene los métodos asociados a la recolección
    - El archivo para clasificadores, que tiene los métodos asociados a la clasificación.
    - El archivo para recicladores, que tiene los métodos asociados al reciclado.
    - Tanto recolectores, clasificadores y recicladores utilizan la cola de mensajes creada por el proceso principal.

Implementación de la ayuda

Cuando los procesos recicladores no pueden obtener ítems de basura de su tipo correspondiente, ya sea leyendo de su pipe o recibiendo un mensaje del tipo asociado al ítem deben optar por ayudar a los otros procesos recicladores o en su defecto, tomar mate.

Tanto para la estrategia con PIPES como con MQ, se prioriza siempre ayudar a los otros recicladores. Es decir, el proceso reciclador toma mate como última instancia.

El orden de las acciones genérico de las acciones que realiza un reciclador es el siguiente:

1. Intenta reciclar de su tipo de basura <tipo prioritario>.
2. Si no hay ítems del <tipo prioritario> de Ayuda:
    a. Intenta reciclar de otro tipo de basura i.
    b. Intenta reciclar de otro tipo de basura j.
    c. Intenta reciclar de otro tipo de basura k.
3. Si no hay ítems de ningún tipo para reciclar, se toma mate.

➔ La implementación de tomar mate se opta simplemente por dormir el proceso actual en una cantidad de N segundos. Para ambas implementaciones, se utilizó N = 3.

➔ La idea fundamental de priorizar ayudar a los otros procesos, en lugar de que sea una decisión aleatoria es para favorecer el flujo de trabajo de la planta, ya que hay menos tiempo desperdiciado por parte de los procesos recicladores.

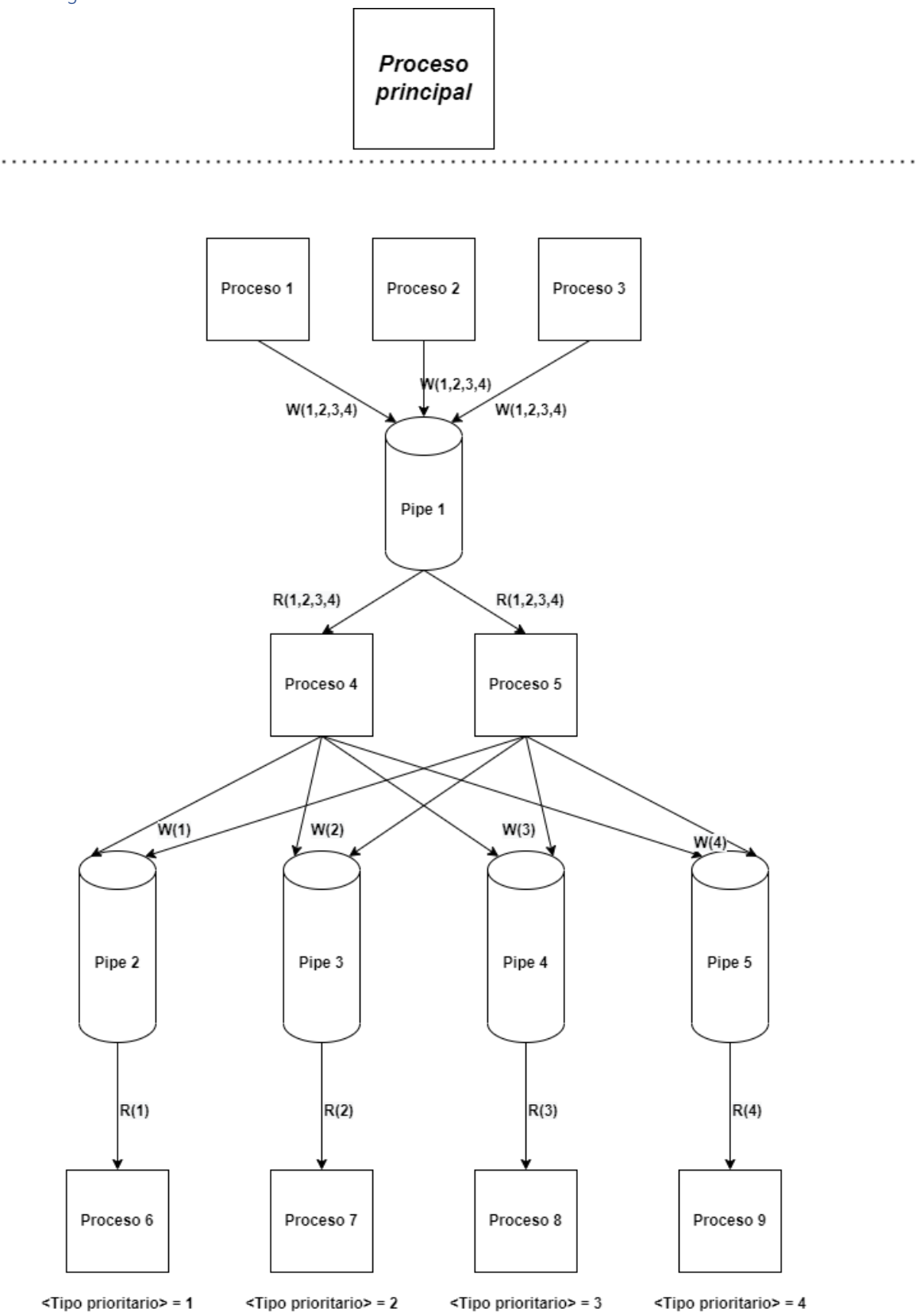
Estrategias gráficamente

A continuación, se abstrae las ideas de ambas estrategias en una presentación gráfica:

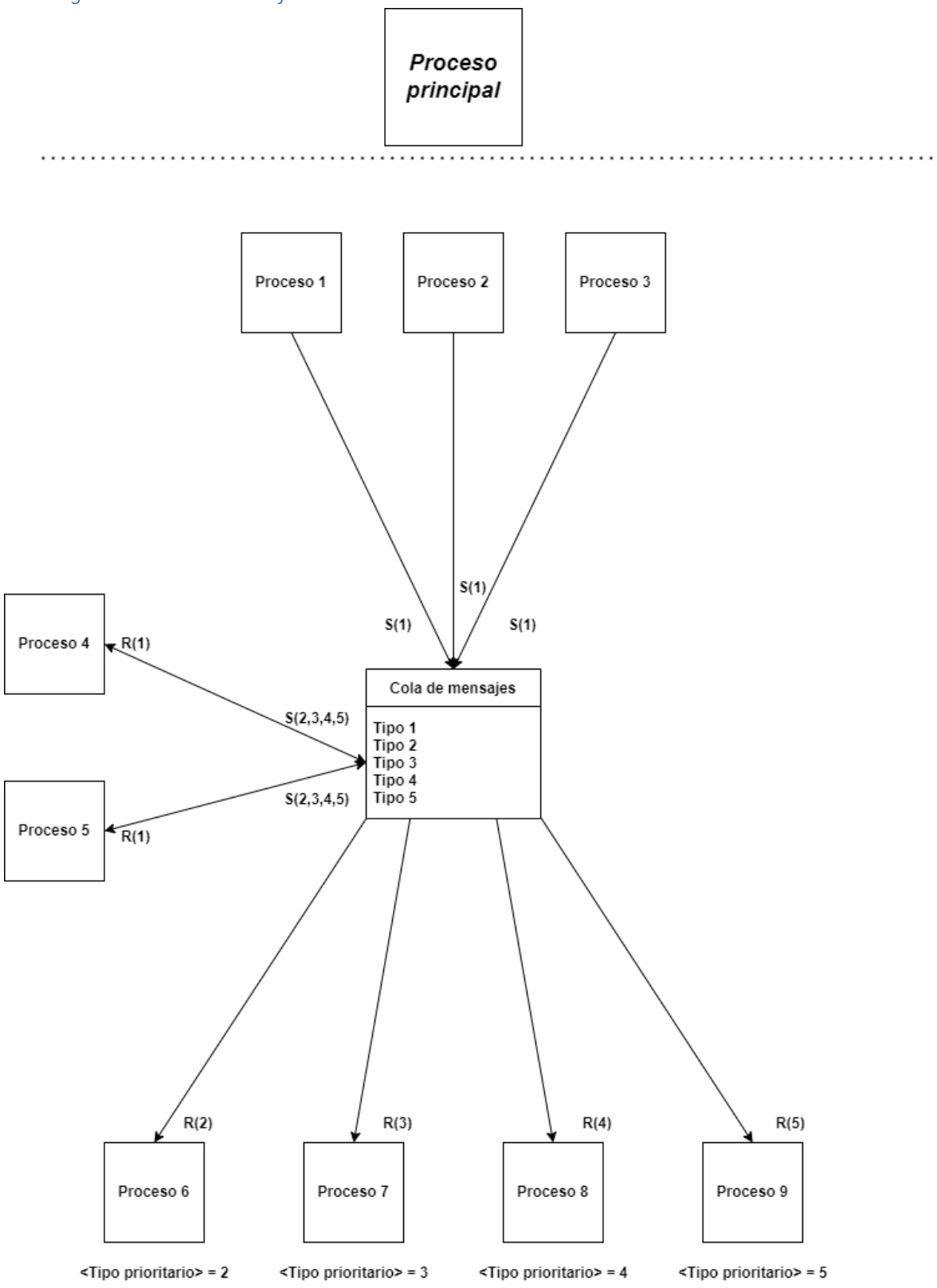
- En el caso de PIPES, W(X) representa que el proceso escribe datos del tipo X.
- En el caso de PIPES, R(X) representa que el proceso lee datos del tipo X.
- En el caso de MQ, S(X) representa que el proceso envía mensajes del tipo X.
- En el caso de MQ, R(X) representa que el proceso recibe mensajes del tipo X.

➔ Para las lecturas y mensajes recibidos R(X), se detalla el tipo principal en X, pero cabe destacar que los procesos pueden leer de otros tipos R(i), R(j) y R(k) cuando realizan la ayuda a otros procesos recicladores.

Estrategia con PIPES



Estrategia con Cola de mensajes



Muestra de ejecución: Planta de reciclado con PIPES.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.1 - Planta de Reciclado/Inciso a $ ./PlantaRecicladoPIPES
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4376
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4378
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4377
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4376
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4378
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4377
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4380
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4376
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4378
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4377
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Plastico (3) | Proceso = 4381
CLASIFICADOR | Item clasificado = Carton (2) | Proceso = 4380
RECICLADOR ALUMINIO | AYUDANDO (No hay aluminio para reciclar) | Item reciclado = Carton (2) | Proceso = 4384
RECICLADOR CARTON | AYUDANDO (No hay carton para reciclar) | Item reciclado = Plastico (3) | Proceso = 4382
RECICLADOR PLASTICO | Item reciclado = Plastico (3) | Proceso = 4383
RECOLECTOR | Item recolectado = Aluminio (4) | Proceso = 4376
RECOLECTOR | Item recolectado = Aluminio (4) | Proceso = 4378
CLASIFICADOR | Item clasificado = Carton (2) | Proceso = 4379
RECOLECTOR | Item recolectado = Aluminio (4) | Proceso = 4377
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Carton (2) | Proceso = 4381
CLASIFICADOR | Item clasificado = Carton (2) | Proceso = 4380
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Carton (2) | Proceso = 4383
RECICLADOR ALUMINIO | Estoy tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4384
RECICLADOR CARTON | Tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4382
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4378
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4376
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4377
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4380
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Plastico (3) | Proceso = 4381
RECICLADOR PLASTICO | Item reciclado = Plastico (3) | Proceso = 4383
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4376
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4378
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4377
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Plastico (3) | Proceso = 4381
CLASIFICADOR | Item clasificado = Aluminio (4) | Proceso = 4380
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Aluminio (4) | Proceso = 4383
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4378
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4376
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4377
CLASIFICADOR | Item clasificado = Aluminio (4) | Proceso = 4379
CLASIFICADOR | Item clasificado = Aluminio (4) | Proceso = 4380
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Aluminio (4) | Proceso = 4381
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Aluminio (4) | Proceso = 4383
RECICLADOR ALUMINIO | Estoy tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4384
RECICLADOR CARTON | Tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4382
```

Muestra de ejecución: Planta de reciclado con Cola de Mensajes.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.1 - Planta de Reciclado/Inciso b $ ./PlantaRecicladoMQ
RECOLECTOR | Item recolectado = Carton (1) | Proceso = 4426
RECOLECTOR | Item recolectado = Carton (1) | Proceso = 4428
RECOLECTOR | Item recolectado = Carton (1) | Proceso = 4427
RECOLECTOR | Item recolectado = Plastico (1) | Proceso = 4428
RECOLECTOR | Item recolectado = Plastico (1) | Proceso = 4426
RECOLECTOR | Item recolectado = Plastico (1) | Proceso = 4427
CLASIFICADOR | Item clasificado = Carton (3) | Proceso = 4430
CLASIFICADOR | Item clasificado = Carton (3) | Proceso = 4429
RECOLECTOR | Item recolectado = Aluminio (1) | Proceso = 4428
RECOLECTOR | Item recolectado = Aluminio (1) | Proceso = 4426
CLASIFICADOR | Item clasificado = Carton (3) | Proceso = 4429
RECOLECTOR | Item recolectado = Aluminio (1) | Proceso = 4427
CLASIFICADOR | Item clasificado = Plastico (4) | Proceso = 4430
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Carton (3) | Proceso = 4431
RECICLADOR ALUMINIO | AYUDANDO (No hay aluminio para reciclar) | Item reciclado = Carton (3) | Proceso = 4434
RECICLADOR CARTON | Item reciclado = Carton (3) | Proceso = 4432
RECICLADOR PLASTICO | Item reciclado = Plastico (4) | Proceso = 4433
RECOLECTOR | Item recolectado = Carton (1) | Proceso = 4427
CLASIFICADOR | Item clasificado = Plastico (4) | Proceso = 4430
RECOLECTOR | Item recolectado = Carton (1) | Proceso = 4428
RECOLECTOR | Item recolectado = Carton (1) | Proceso = 4426
CLASIFICADOR | Item clasificado = Plastico (4) | Proceso = 4429
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Plastico (4) | Proceso = 4431
RECICLADOR PLASTICO | Item reciclado = Plastico (4) | Proceso = 4433
RECICLADOR CARTON | Tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4432
RECICLADOR ALUMINIO | Tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4434
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4427
CLASIFICADOR | Item clasificado = Aluminio (5) | Proceso = 4430
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Aluminio (5) | Proceso = 4433
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4428
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4426
CLASIFICADOR | Item clasificado = Aluminio (5) | Proceso = 4429
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Aluminio (5) | Proceso = 4431
RECOLECTOR | Item recolectado = Aluminio (1) | Proceso = 4427
CLASIFICADOR | Item clasificado = Aluminio (5) | Proceso = 4430
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Aluminio (5) | Proceso = 4433
RECOLECTOR | Item recolectado = Aluminio (1) | Proceso = 4428
RECOLECTOR | Item recolectado = Aluminio (1) | Proceso = 4426
CLASIFICADOR | Item clasificado = Carton (3) | Proceso = 4429
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Carton (3) | Proceso = 4431
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4427
CLASIFICADOR | Item clasificado = Carton (3) | Proceso = 4430
RECICLADOR PLASTICO | Tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4433
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Carton (3) | Proceso = 4431
CLASIFICADOR | Item clasificado = Carton (3) | Proceso = 4429
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4428
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4426
RECOLECTOR | Item recolectado = Plastico (1) | Proceso = 4427
CLASIFICADOR | Item clasificado = Vidrio (2) | Proceso = 4430
```

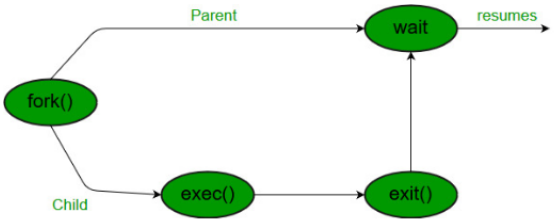
Mini Shell

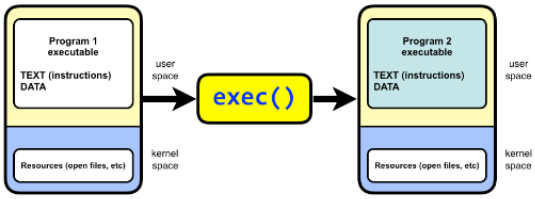
El método de construcción utilizado para la Mini Shell fue utilizar un proceso principal que se encarga del parseo de la instrucción ingresada. Es decir, recibe la entrada del usuario, divide los argumentos, crea un proceso hijo y realiza una llamada execvp(), para que dicho proceso hijo realice la ejecución del comando. La lectura de instrucciones se realiza dinámicamente hasta que el comando ingresado por el usuario sea un 'exit´.

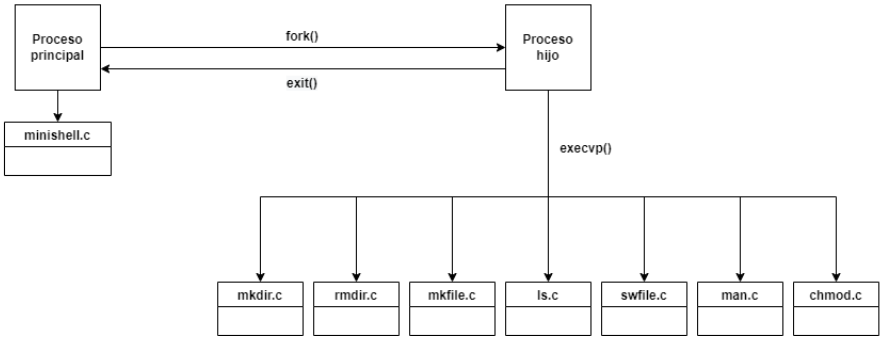
Los comandos brindados por la mini Shell son los siguientes:

- ***mkdir***: Crear un directorio.
- ***rmdir***: Eliminar un directorio.
- ***mkfile***: Crear un archivo.
- ***ls***: Listar el contenido de un directorio.
- ***swfile***: Mostrar el contenido de un archivo.
- ***man***: Mostrar una ayuda con los comandos posibles.
- ***chmod***: Modificar los permisos de un archivo. Los permisos son de lectura, escritura y ejecución.
- ➔ ***exit***: Podría ser considerado un comando / instrucción que permite cerrar la mini shell.

Gráficamente, obtenemos el siguiente planteo:







- ➔ Con el comando execvp(), el proceso hijo creado no tiene que ejecutar el mismo programa que el proceso padre. Esta llamada al sistema permite que un proceso ejecute cualquier archivo de programa, que incluye un ejecutable binario o un script.
- ➔ En nuestro caso, la llamada a execvp() es utilizada para que el proceso hijo ejecute un comando en particular y termine.
- ➔ La ejecución de la mini Shell se realiza dinámicamente hasta que el usuario ingresa 'exit'. Esta instrucción en particular, permite finalizar la ejecución del proceso principal (o proceso padre).

Muestra de ejecución: Comando mkdir.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> mkdir SO
>> mkdir(): Directorio SO creado con éxito.
>> mkdir SO
>> mkdir(): Error al crear el directorio SO.
>> mkdir
>> mkdir(): Debe ingresar el nombre del archivo que desea crear.
>> mkdir SO HOLA BUEN DIA
>> mkdir(): Error de ingreso de argumentos.
>> exit
```

Muestra de ejecución: Comando rmdir.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> rmdir SO
>> rmdir(): Directorio SO removido con éxito.
>> rmdir SO
>> rmdir(): Error al remover el directorio SO.
>> rmdir
>> rmdir(): Debe ingresar el nombre del archivo que desea eliminar.
>> rmdir SO HOLA BUEN DIA
>> rmdir(): Error de ingreso de argumentos.
>> exit
```

Muestra de ejecución: Comando mkfile.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> mkfile
>> mkfile(): Debe ingresar un nombre para el archivo.
>> mkfile Gonza.txt
>> mkfile(): El archivo Gonza.txt fue creado exitosamente o ya existe.
>> mkfile Gonza.txt SO
>> mkfile(): La cantidad de argumentos ingresada es incorrecta.
>> exit
```

Muestra de ejecución: Comando ls.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> ls
minishell
rmdir
mkfile.c
colores.h
man
swfile
makefile
chmod.c
mkdir.c
rmdir.c
man.c
chmod
ls.c
ls
mkfile
mkdir
minishell.c
colores.c
Gonza.txt
swfile.c
>> ls(): Listado de directorio /home/gonzalo/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell realizado exitosamente.
>> ls /home/gonzalo/Desktop
multiplicacionMatrices.c
Ejercicio4.c
mutex.c
Proyecto 2022 - SO
MS
Practica - SO
MemoriaCompartida.c
Comandos utiles
Minishell
Actividad 5 - SO
>> ls(): Listado de directorio /home/gonzalo/Desktop realizado exitosamente.
>> ls HOLA
>> ls(): Error al listar el contenido del directorio HOLA.
>> ls HOLA BUEN DIA
>> ls(): La cantidad de argumentos ingresada no es valida.
>> exit
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> touch
execvp(): Error al ejecutar el comando ./touch en el proceso 5190.
>> exit
```

Muestra de ejecución: Comando swfile.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> swfile Gonza.txt
HOLA BUEN DIA
ESTA ES UNA PRUEBA DEL COMANDO swfile
PROYECTO SO - 2022

>> swfile(): Se mostro el archivo Gonza.txt exitosamente.
>> swfile HOLA
>> swfile(): Error al abrir el archivo HOLA
>> swfile HOLA BUEN DIA
>> swfile(): La cantidad de argumentos ingresados es incorrecta.
>> exit
```

Muestra de ejecución: Comando man.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> man
>> man(): No se ingresaron argumentos. La sintaxis es: 'man <comando>'
>> man mkdir
mkdir(): Crea un directorio en la ubicación de compilacion de la minishell.
Formato del comando: mkdir [nombre_directorio]
[nombre_directorio] es el nombre del directorio que se crea.
Valores de retorno:
        -> En caso de exito: 0 | Se crea el directorio y se muestra un mensaje de éxito.
        -> En caso contrario: -1 | Se muestra un mensaje de error.
>> man HOLA BUEN DIA
>> man(): La cantidad de argumentos ingresada es incorrecta.
>> man touch
>> man(): El comando ingresado no esta implementado en la minishell.
>> exit
```

Muestra de ejecución: Comando chmod.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> chmod
>> chmod(): Debe ingresar el path del archivo.
>> chmod Gonza.txt
>> chmod(): Debe ingresar un argumento entero para identificar los permisos del archivo.
>> chmod Gonza.txt 777
>> chmod(): Se cambiaron exitosamente los permisos del archivo Gonza.txt.
>> chmod /home/gonzalo/Desktop/Prueba.txt 666
>> chmod(): Se cambiaron exitosamente los permisos del archivo /home/gonzalo/Desktop/Prueba.txt.
>> chmod ARCHIVO_NO_EXISTENTE.txt 000
>> chmod(): Ocurrio un error al cambiar los permisos del archivo ARCHIVO_NO_EXISTENTE.txt.
>> chmod Gonza.txt 0
>> chmod(): Los permisos ingresados no son validos.
>> exit
```

Robustez

En cuanto al aspecto de robustez, se trató de abarcar la mayor cantidad de casos posibles en cada comando y en las entradas posibles, buscando casos de éxito y no éxito. Entre los que se destacan los siguientes tipos de instrucciones:

- Instrucciones con comando y argumentos correctos.
- Instrucciones comando correcto y argumentos incorrectos.
- Instrucciones con comando no existente.
- Instrucciones con excesiva cantidad de argumentos.
- Instrucciones vacías. (Presionar ‘Enter’ solamente)

Secuencia

La metodología utilizada para la implementación de la secuencia, fue utilizar una cierta cantidad de hilos o procesos para la ejecución de un método asociado a la impresión de cada letra y para la sincronización semáforos y pipes respectivamente.

Para la construcción fue necesario probar diferentes alternativas, en primera instancia, ubicando los semáforos cuidadosamente para poder realizar la sincronización. Esto es fundamental, ya que existen soluciones alternativas en las que se puede ahorrar una cierta cantidad de semáforos pero que podrían fallar dependiendo de la configuración de hardware utilizada. Es decir, la sincronización debe mantenerse independientemente de la computadora en la que se esté ejecutando.

Esto se hizo presente en una primera solución obtenida con 5 semáforos, en la que se realizaba primero un printf() y luego se enviaba un signal() a un semáforo. Es decir, había una parte que dependía de algo que era secuencial y asumía que las dos instrucciones eran atómicas, lo cual no es correcto. Otras alternativas era duplicar el código en alguno de los métodos de una letra, cuando era necesario imprimirla dos veces en un patrón y demás.

Es por esto, que se optimizo la solución exhaustivamente hasta lograr una solución que reúna las mejores condiciones posibles para evitar repetir el código y no depender de partes secuenciales. Para esto fueron necesarios 6 semáforos diferentes:

- 2 semáforos para la letra A.
- 1 semáforo para la letra B.
- 1 semáforo para la letra C.
- 1 semáforo para las letras D y E.
- 1 semáforo para la letra F.

➔ La clave para resolver el problema fue subdividir el semáforo de A en dos semáforos, ya que dicha letra aparece en la secuencia como un patrón en el cual puede estar seguida después de la aparición de varias letras. Claramente, acompañado de una correcta inicialización de los semáforos al iniciar la ejecución.

Pasaje de Hilos y Semáforos a Procesos y Pipes

Una vez resuelta la sincronización de la secuencia con Hilos y Semáforos, el pasaje a Procesos y Pipes es directo, ya que se utiliza un pipe por cada semáforo utilizado y se reemplazan los hilos por procesos.

- Para simular la operación sem_wait() provista por semáforos se utiliza la instrucción read() del pipe que es bloqueante. Es decir, la lectura se realiza solo cuando
- Para simular la operación sem_post() provista por semáforos se utiliza la instrucción write() del pipe.

➔ Claramente, las operaciones read() y write() del pipe deben realizarse cuidadosamente sobre el file descriptor (fd[]) indicado.

➔ Además de esto, es necesario cerrar los file descriptor no utilizados en cada uno de los métodos, ya que pueden generarse problemas.

Muestra de ejecución: Secuencia con Hilos y Semáforos.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.1 - Secuencia/Inciso a $ make
gcc SecuenciaHS.c -o SecuenciaHS -pthread
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.1 - Secuencia/Inciso a $ ./SecuenciaHS
ABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABEACDFABEACDFABEACDFABEACDFABEACDF
```

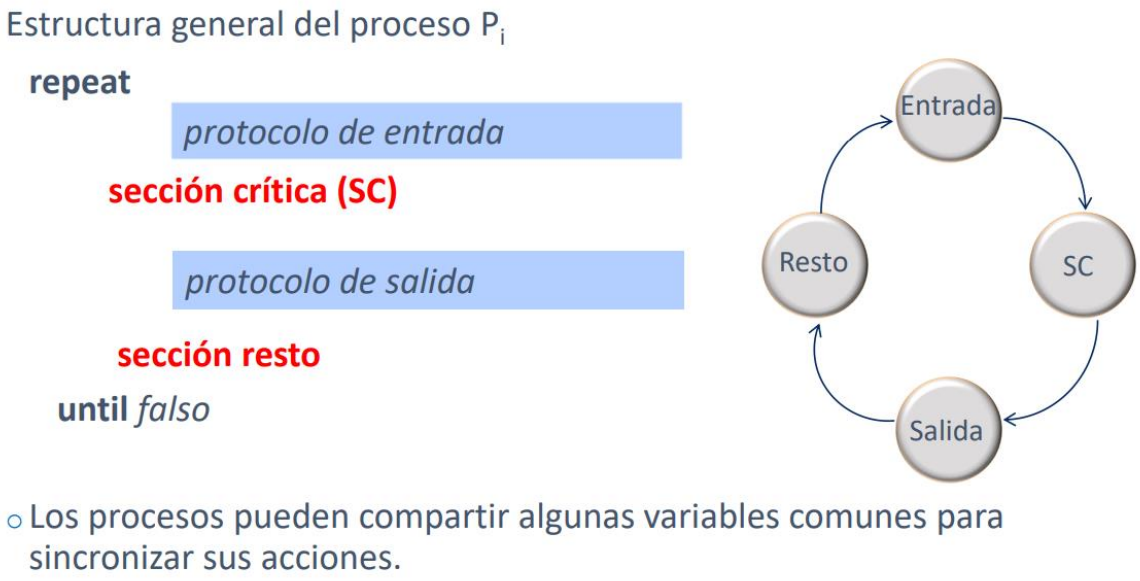
Muestra de ejecución: Secuencia con Procesos y Pipes.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.1 - Secuencia/Inciso b $ make
gcc SecuenciaPP.c -o SecuenciaPP
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.1 - Secuencia/Inciso b $ ./SecuenciaPP
ABDACEFABDACEFABDACEFABDACDFABEACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEF
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.1 - Secuencia/Inciso b $ ./SecuenciaPP
ABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABDACEFABEACEFABDACDFABDACEFABDACEFABDACEFABDACEFABDACEFABEACDF
```

➔ En ambas muestras de ejecución, se puede observar las 4 posibles combinaciones del patrón de la secuencia.

Puente de una sola mano

Para el desarrollo del ejercicio “Puente de una sola mano”, fue necesario un desarrollo progresivo en el que se buscaron diferentes alternativas y planteos, hasta obtener una solución completa, correcta y ordenada, que respete la estructura general brindada por la cátedra para una buena solución:



Para simplificar la explicación, se brinda el siguiente pseudo código que corresponde a la solución inicial para el inciso i):

```
<Direccion> <- NORTE o SUR

# PROTOCOLO DE ENTRADA
bloquearDireccion()
si (hayAutosDireccionCruzando())
    entonces: noBloquearPuente()
    de lo contrario: bloquearPuente() // Si es el primer auto en cruzar bloquea el puente
agregarAutoCruzando()
desbloquearDireccion()

# SECCION CRITICA
pasarPuente()

# PROTOCOLO DE SALIDA
bloquearDireccion()
liberarAutoCruzando()
si (hayAutosDireccionCruzando())
    entonces: noLiberarPuente()
    de lo contrario: liberarPuente() // Si es el último auto en cruzar libera el puente
desbloquearDireccion()

# SECCION RESTO
```

- ***Protocolo de entrada***: Se bloquea la dirección del auto que esta pasando, ya sea Norte o Sur, para evitar que varios autos puedan ejecutar simultáneamente los chequeos posteriores y evitar inconsistencias. Luego, se revisa si hay autos de la dirección (Norte o Sur) cruzando actualmente, si los hay, no es necesario bloquear el puente y puede pasar con normalidad, pero si no los hay, es decir, es el primero en pasar, debe bloquear el puente (si esta disponible) o esperar para poder obtener el bloqueo del mismo. Finalmente, se desbloquea la dirección y la ejecución continúa.
- ***Sección critica***: La sección crítica es básicamente, pasar el puente, que se puede implementar de diversas maneras según lo que se desee realizar, pero basta con mostrar el auto que esta pasando actualmente y su dirección correspondiente.
- ***Protocolo de salida***: Se bloquea nuevamente la dirección del auto que esta pasando, ya sea Norte o Sur, para evitar que varios autos puedan ejecutar simultáneamente los chequeos posteriores, se libera el auto correspondiente y se chequea si hay autos de la misma dirección cruzando el puente, si los hay, no es posible liberar el puente, pero si no los hay, es decir, el auto actual es último en pasar, debe liberar el puente. Luego, se desbloquea la dirección y la ejecución continúa.
- ***Sección resto***: Para este caso en particular, no fue necesario realizar acciones en la sección resto, ya que, la parte fundamental es cruzar el puente que forma parte de la sección crítica. Podría mostrarse que se completo exitosamente la salida del puente, pero por convención adoptada, no se muestra nada, ya que consideramos que no es necesario.

Veamos a modo de ilustración, que el pasaje del algoritmo en pseudo código al código fuente es directo:

```c
/*
 * Realiza las acciones de los vehiculos provenientes del NORTE al cruzar el puente.
 **/
void vehiculoNorte(){
    while(1) {
        // PROTOCOLO DE ENTRADA
        pthread_mutex_lock(&mutexN); // Si llegan dos autos del NORTE, solo el primero obtiene el mutex.
        if(sem_trywait(&pasandoN) == 0){ // Si hay autos del NORTE pasando actualmente.
            sem_post(&pasandoN);
        }else{ // Si NO hay autos del NORTE pasando actualmente.
            sem_wait(&puente); // Bloquea el puente o espera para poder obtener el bloqueo.
        }
        sem_post(&pasandoN);
        pthread_mutex_unlock(&mutexN);

        // SECCION CRITICA
        printf("%sNORTE: Auto del norte pasando el puente. PATENTE: %i\n %s", blue(), gettid(), reset());
        sleep(1);

        // PROTOCOLO DE SALIDA
        pthread_mutex_lock(&mutexN);
        sem_wait(&pasandoN);
        if(sem_trywait(&pasandoN) == 0){ // Si quedan autos del NORTE pasando todavía.
            sem_post(&pasandoN);
        }else{ // Si es el ultimo auto del NORTE en pasar, libera el bloqueo del puente.
            sem_post(&puente);
        }
        pthread_mutex_unlock(&mutexN);

        // SECCION RESTO
    }

    pthread_exit(EXIT_SUCCESS);
}
```

➔ La implementación para vehículos del Norte y del Sur es análoga, únicamente difiere en los nombres de algunos de los semáforos.

Las únicas diferencias que podemos notar son las siguientes:

- Se utiliza un loop infinito, para representar la llegada de diferentes autos en una situación mas cercana a lo que ocurriría en la vida real. Es decir, se reutilizan los hilos para representar lo que seria el trafico del puente. Esto es básicamente, para no tener que usar una cantidad excesiva de recursos y aprovechar los existentes.
- Al realizar el chequeo para saber si hay autos de la dirección cruzando el puente, es necesario utilizar try_wait() y en caso exitoso, restablecer el token obtenido, ya que try_wait() decrementa el semáforo y necesitamos que se mantenga el valor actual, ya que es simplemente una consulta.

Muestra de ejecución: Puente de una sola mano con Hilos y Semáforos.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.2 - Puente de una sola mano/Inciso i $ make
gcc PuenteHS.c -o PuenteHS -pthread
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.2 - Puente de una sola mano/Inciso i $ ./PuenteHS
NORTE: Auto del norte pasando el puente. PATENTE: 8801
NORTE: Auto del norte pasando el puente. PATENTE: 8807
NORTE: Auto del norte pasando el puente. PATENTE: 8804
NORTE: Auto del norte pasando el puente. PATENTE: 8805
NORTE: Auto del norte pasando el puente. PATENTE: 8806
NORTE: Auto del norte pasando el puente. PATENTE: 8809
NORTE: Auto del norte pasando el puente. PATENTE: 8810
NORTE: Auto del norte pasando el puente. PATENTE: 8814
NORTE: Auto del norte pasando el puente. PATENTE: 8815
NORTE: Auto del norte pasando el puente. PATENTE: 8817
NORTE: Auto del norte pasando el puente. PATENTE: 8818
NORTE: Auto del norte pasando el puente. PATENTE: 8819
NORTE: Auto del norte pasando el puente. PATENTE: 8802
NORTE: Auto del norte pasando el puente. PATENTE: 8808
NORTE: Auto del norte pasando el puente. PATENTE: 8811
NORTE: Auto del norte pasando el puente. PATENTE: 8803
NORTE: Auto del norte pasando el puente. PATENTE: 8822
NORTE: Auto del norte pasando el puente. PATENTE: 8812
NORTE: Auto del norte pasando el puente. PATENTE: 8813
NORTE: Auto del norte pasando el puente. PATENTE: 8816
NORTE: Auto del norte pasando el puente. PATENTE: 8820
NORTE: Auto del norte pasando el puente. PATENTE: 8821
NORTE: Auto del norte pasando el puente. PATENTE: 8823
NORTE: Auto del norte pasando el puente. PATENTE: 8824
NORTE: Auto del norte pasando el puente. PATENTE: 8825
NORTE: Auto del norte pasando el puente. PATENTE: 8801
NORTE: Auto del norte pasando el puente. PATENTE: 8807
NORTE: Auto del norte pasando el puente. PATENTE: 8804
NORTE: Auto del norte pasando el puente. PATENTE: 8805
NORTE: Auto del norte pasando el puente. PATENTE: 8806
NORTE: Auto del norte pasando el puente. PATENTE: 8809
NORTE: Auto del norte pasando el puente. PATENTE: 8810
NORTE: Auto del norte pasando el puente. PATENTE: 8814
NORTE: Auto del norte pasando el puente. PATENTE: 8815
NORTE: Auto del norte pasando el puente. PATENTE: 8817
NORTE: Auto del norte pasando el puente. PATENTE: 8818
NORTE: Auto del norte pasando el puente. PATENTE: 8819
NORTE: Auto del norte pasando el puente. PATENTE: 8802
NORTE: Auto del norte pasando el puente. PATENTE: 8808
NORTE: Auto del norte pasando el puente. PATENTE: 8811
NORTE: Auto del norte pasando el puente. PATENTE: 8803
NORTE: Auto del norte pasando el puente. PATENTE: 8822
NORTE: Auto del norte pasando el puente. PATENTE: 8812
NORTE: Auto del norte pasando el puente. PATENTE: 8813
NORTE: Auto del norte pasando el puente. PATENTE: 8816
NORTE: Auto del norte pasando el puente. PATENTE: 8820
NORTE: Auto del norte pasando el puente. PATENTE: 8821
NORTE: Auto del norte pasando el puente. PATENTE: 8823
```

Problemas de la solución propuesta

Si bien la solución propuesta es correcta y satisface los criterios de una buena solución, no incluye ningún mecanismo para controlar el paso de las direcciones, por lo que alguna de las direcciones, ya sea Norte o Sur, puede tener inanición y eventualmente nunca cruzar el puente. Esto se ve claramente reflejado en la muestra de ejecución presentada anteriormente.

Extensión de la solución inicial

Para solucionar el problema de la inanición, se opto por utilizar un contador con una cantidad N de tokens, que permite controlar el paso de las distintas direcciones por el puente. Esto es:

- Permite que pase una de las direcciones N veces, hasta que el contador queda en 0.
- El último auto en pasar (dentro de los N que fueron habilitados por el contador) es el encargado es el encargado de liberar el puente y habilitar la dirección opuesta, nuevamente con N tokens para que puedan pasar por el puente.
- La dirección que paso inicialmente quedará esperando a que la dirección opuesta libere el puente y envié nuevamente los tokens.
- Esto se repite sucesivamente para que eventualmente todos los autos que están esperando, puedan pasar.

A continuación, el pseudo código con la extensión mencionada:

```
<Direccion> <- NORTE o SUR

# PROTOCOLO DE ENTRADA
esperarContadorDireccion()
// Si el contador no es vacio, el auto avanza.
bloquearDireccion()
si (hayAutosDireccionCruzando())
    entonces: noBloquearPuente()
    de lo contrario: bloquearPuente() // Si es el primer auto en cruzar bloquea el puente
agregarAutoCruzando()
desbloquearDireccion()

# SECCION CRITICA
pasarPuente()

# PROTOCOLO DE SALIDA
bloquearDireccion()
liberarAutoCruzando()
si (hayAutosDireccionCruzando())
    entonces: noLiberarPuente()
    de lo contrario: liberarPuente() // Si es el último auto en cruzar libera el puente
                     habilitarDireccionOpuesta()
                    // El ultimo auto en cruzar de los permitidos por el contador, habilita el puente para la otra direccion.
desbloquearDireccion()

# SECCION RESTO
```

Pasaje de Hilos y Semáforos a Procesos y Cola de Mensajes

Para la extensión del inciso iii) que soluciona el problema de la inanición, además de modificar el algoritmo fue necesario cambiar las entidades del problema de Hilos a Procesos y para la sincronización, en lugar de utilizar Semáforos se utilizaron Colas de Mensajes.

- Se reemplaza cada Hilo por un Proceso para representar a un auto de una dirección.
- En la cola de mensajes se asocia un tipo de mensaje a cada uno de los Semáforos.
- Se agregan 2 tipos de mensajes nuevos (2 semáforos de conteo) para realizar el contador.
- Se modula en diferentes archivos, aprovechando las ventajas que provee utilizar Colas de mensajes y execvp():
    - Se utiliza un archivo principal en el que se crea la cola de mensajes y los procesos asociados a los autos.
    - Se utiliza un archivo para los autos del norte, que tiene todos los métodos de los autos del norte.

    - Se utiliza un archivo para los autos del sur, que tiene todos los métodos de los autos del sur.
    - Tanto los autos del norte como los del sur, utilizan la cola de mensajes creada por el proceso principal.
- Se ofrece un archivo para el manejo de los mensajes, encapsulando todo lo referido a los mismos, con ciertas macros para identificar los tipos (como si fueran nombres semáforos). Esto facilita la identificación de los mensajes que se deben enviar o recibir, ya que no es necesario utilizar los números explícitamente.
- Se reemplazan las instrucciones sem_wait() por msgrcv() y las instrucciones sem_post() por msgsnd() a partir de la primera solución obtenida.

Nuevamente, el pasaje del pseudo código al código fuente es análogo:

```c
/*
 * Realiza las acciones de los vehiculos provenientes del NORTE al cruzar el puente.
 **/
void vehiculoNorte() {
    while(1) {
        MENSAJE S1, S2, S3, S4, S5, S6, S7;

        // PROTOCOLO DE ENTRADA
        coordinarNorte();
        msgrcv(mq, &S1, SIZE_MENSAJE, MutexN, 0);
        if (msgrcv(mq, &S2, SIZE_MENSAJE, PasandoN, IPC_NOWAIT) != -1) {
            S2 = crearMensaje(PasandoN);
            msgsnd(mq, &S2, SIZE_MENSAJE, 0);
        } else {
            msgrcv(mq, &S3, SIZE_MENSAJE, Puente, 0);
        }
        S4 = crearMensaje(PasandoN);
        msgsnd(mq, &S4, SIZE_MENSAJE, 0);
        S1 = crearMensaje(MutexN);
        msgsnd(mq, &S1, SIZE_MENSAJE, 0);

        // SECCION CRITICA
        printf("%sNORTE: Auto del norte pasando el puente. PATENTE: %d %s\n", blue(), getpid(), reset());
        sleep(1);

        // PROTOCOLO DE SALIDA
        msgrcv(mq, &S5, SIZE_MENSAJE, MutexN, 0);
        msgrcv(mq, &S6, SIZE_MENSAJE, PasandoN, 0);
        if (msgrcv(mq, &S7, SIZE_MENSAJE, PasandoN, IPC_NOWAIT) != -1) {
            S7 = crearMensaje(PasandoN);
            msgsnd(mq, &S7, SIZE_MENSAJE, 0);
        } else {
            S3 = crearMensaje(Puente);
            msgsnd(mq, &S3, SIZE_MENSAJE, 0);
            coordinarSur();
        }
        S5 = crearMensaje(MutexN);
        msgsnd(mq, &S5, SIZE_MENSAJE, 0);

        // SECCION RESTO
    }
}
```

```c
/*
 * Chequea si el contador de los autos del NORTE posee un token.
 * Si hay tokens disponibles del NORTE, habilita el paso del vehiculo.
 * De lo contrario, bloquea el paso del vehiculo hasta que hayan tokens disponibles.
 */
void coordinarNorte() {
    MENSAJE CN;
    msgrcv(mq, &CN, SIZE_MENSAJE, CoordinadorN, 0);
}

/*
 * Si es el ultimo auto del NORTE en pasar despues de utilizar todos los tokens disponibles.
 * Es decir, cuando el contador llega a 0, habilita el paso de los autos del SUR con una
 * cantidad N de tokens.
 */
void coordinarSur() {
    for (int i = 0; i < CONTADOR_PUENTE; i ++) {
        MENSAJE CS = crearMensaje(CoordinadorS);
        msgsnd(mq, &CS, SIZE_MENSAJE, 0);
    }
}
```

Como se mencionó anteriormente, se ofrece un archivo ‘mensajes.c’ para el manejo de los mensajes utilizados en la Cola de Mensajes:

```c
#define SIZE_MENSAJE sizeof(MENSAJE) - sizeof(long)
#define CONTADOR_PUENTE 5
#define Puente 1
#define PasandoN 2
#define PasandoS 3
#define MutexN 4
#define MutexS 5
#define CoordinadorN 6
#define CoordinadorS 7

/*
 * struct utilizado para la representacion de las señales del puente.
 * Son utilizados como Mensajes en las operaciones de las MQs.
 **/
typedef struct mensaje {
    long tipo;
    char msg[20];
} MENSAJE;

MENSAJE crearMensaje(long tipo) {
    MENSAJE m;
    m.tipo = tipo;

    switch(tipo) {
        case Puente: strcpy(m.msg, "Puente"); break;
        case PasandoN: strcpy(m.msg, "NortePuente"); break;
        case PasandoS: strcpy(m.msg, "SurPuente"); break;
        case MutexN: strcpy(m.msg, "MutexN"); break;
        case MutexS: strcpy(m.msg, "MutexS"); break;
        case CoordinadorN: strcpy(m.msg, "CoordinadorN"); break;
        case CoordinadorS: strcpy(m.msg, "CoordinadorS"); break;
    }

    return m;
}
```

- ➔ Notemos que cada tipo de mensaje se corresponde con una macro definida (con nombres similares a los utilizados para los semáforos en la primera solución), lo que facilita enviar y recibir mensajes, ya que no hace falta hacer explicito el tipo como un número.
- ➔ Se le asigna un string o arreglo de caracteres al mensaje creado solamente por completitud, el cual indica el nombre del mensaje o semáforo que se representa. Esto se realiza porque, si el mensaje es solo el tipo y no contiene nada más, se producen diferentes errores propios de la cola de mensajes. Es decir, no tiene ninguna funcionalidad, pero es indispensable de que el mensaje contenga algo más que el tipo.

Muestra de ejecución: Puente de una sola mano con Procesos y Colas de Mensajes.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.2 - Puente de una sola mano/Inciso iii $ make
gcc PuentePMQ.c -o PuentePMQ
gcc VehiculoNORTE.c -o VehiculoNORTE
gcc VehiculoSUR.c -o VehiculoSUR
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.2.2 - Puente de una sola mano/Inciso iii $ ./PuentePMQ
NORTE: Auto del norte pasando el puente. PATENTE: 8929
NORTE: Auto del norte pasando el puente. PATENTE: 8939
NORTE: Auto del norte pasando el puente. PATENTE: 8937
NORTE: Auto del norte pasando el puente. PATENTE: 8935
NORTE: Auto del norte pasando el puente. PATENTE: 8932
SUR: Auto del sur pasando el puente. PATENTE: 8955
SUR: Auto del sur pasando el puente. PATENTE: 8954
SUR: Auto del sur pasando el puente. PATENTE: 8957
SUR: Auto del sur pasando el puente. PATENTE: 8961
SUR: Auto del sur pasando el puente. PATENTE: 8959
NORTE: Auto del norte pasando el puente. PATENTE: 8941
NORTE: Auto del norte pasando el puente. PATENTE: 8933
NORTE: Auto del norte pasando el puente. PATENTE: 8943
NORTE: Auto del norte pasando el puente. PATENTE: 8942
NORTE: Auto del norte pasando el puente. PATENTE: 8940
SUR: Auto del sur pasando el puente. PATENTE: 8956
SUR: Auto del sur pasando el puente. PATENTE: 8963
SUR: Auto del sur pasando el puente. PATENTE: 8958
SUR: Auto del sur pasando el puente. PATENTE: 8965
SUR: Auto del sur pasando el puente. PATENTE: 8960
NORTE: Auto del norte pasando el puente. PATENTE: 8945
NORTE: Auto del norte pasando el puente. PATENTE: 8944
NORTE: Auto del norte pasando el puente. PATENTE: 8947
NORTE: Auto del norte pasando el puente. PATENTE: 8946
NORTE: Auto del norte pasando el puente. PATENTE: 8948
SUR: Auto del sur pasando el puente. PATENTE: 8962
SUR: Auto del sur pasando el puente. PATENTE: 8968
SUR: Auto del sur pasando el puente. PATENTE: 8964
SUR: Auto del sur pasando el puente. PATENTE: 8970
SUR: Auto del sur pasando el puente. PATENTE: 8972
```

➔ Ahora vemos que es posible que pasen autos de ambas direcciones de una forma controlada. Con N = 5, pasan 5 autos de una dirección y 5 autos de la otra dirección, alternándose, por lo que eventualmente todos los autos que quieran pasar el puente, en algún momento lo harán.

Extras

Compilación

Archivos make

Para facilitar la compilación de los códigos fuentes asociados a los distintos ejercicios, se ofrecen archivos make o makefiles en su defecto, que facilitan la compilación y ejecución de los mismos. Además, cada uno de los makefile ofrece una regla phony para la limpieza post ejecución de los archivos de extensión '.o' generados al compilar.

Para cada uno de los ejercicios, se encuentra un archivo de nombre "makefile" dentro de la carpeta correspondiente al ejercicio, que contiene todas las instrucciones de compilación de cada uno de los archivos del ejercicio. Por ejemplo, el ejercicio de implementación de la Mini Shell contiene en total 8 archivos diferentes que deben compilarse por separado para un correcto funcionamiento unitario. Con el makefile de Mini Shell, se realiza la compilación del archivo principal y de cada uno de los comandos:

```
# makefile para compilar la minishell

COMANDOS = mkdir rmdir mkfile ls swfile man chmod

# Si se ejecuta solo 'make' 'se ejecuta el objetivo all, que incluye todo.
all: minishell $(COMANDOS)

# Compilacion del archivo principal.
minishell:
    gcc minishell.c -o minishell

# Compilacion del comando: Crear un directorio.
mkdir:
    gcc mkdir.c -o mkdir

# Compilacion del comando: Eliminar un directorio.
rmdir:
    gcc rmdir.c -o rmdir

# Compilacion del comando: Crear un archivo.
mkfile:
    gcc mkfile.c -o mkfile

# Compilacion del comando: Listar el contenido de un directorio.
ls:
    gcc ls.c -o ls

# Compilacion del comando: Mostrar el contenido de un archivo.
swfile:
    gcc swfile.c -o swfile

# Compilacion del comando: Mostrar una ayuda con los comandos posibles.
man:
    gcc man.c -o man

# Compilacion del comando: Modificar los permisos de un archivo.
chmod:
    gcc chmod.c -o chmod

# Regla phony para la eliminacion de archivos .o. (Limpieza post-ejecucion)
clean:
    rm -f minishell $(COMANDOS) *.o
```

- Notar la regla phony clean que puede ser utilizada en caso de ser necesaria una limpieza post ejecución.

Pasos de compilación y ejecución

➔ Si bien cada uno de los archivos makefile, puede ofrecer una o más reglas de compilación, se construyeron de tal manera que solo sea necesario utilizar la instrucción 'make' al compilar, ya que make, por defecto utiliza la primera regla que contiene todas las instrucciones necesarias para la compilación unitaria.

A continuación, una serie de pasos para realizar una correcta compilación y ejecución de los diferentes ejercicios, y de ser necesaria, también una limpieza de los archivos ejecutables generados:

1. Abrir una terminal.
2. Ubicarse dentro del directorio donde se encuentra los archivos fuente con el comando cd.
3. Realizar una llamada al comando '***make'***. Luego de esto, se muestran cada una de las instrucciones de compilación realizadas.
4. Ejecutar el primer archivo generado por make con el nombre adecuado. Esto se realiza por convención, ya que, en el orden de compilación, siempre el archivo principal es el primero en compilar. Es decir, si la compilación consiste de dos o más archivos (dependiendo del ejercicio), los demás archivos son utilizados por el principal y no es necesario una ejecución independiente de los mismos.
5. De ser necesaria una limpieza de los archivos '.o' generados por make. Realizar la instrucción '***make clean'***. Esto se debe, a que la regla clean especificada dentro de make esta construida especialmente para esta función.

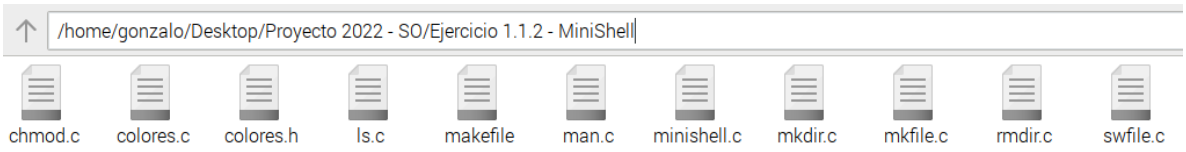
Ejemplo de compilación y ejecución: makefile de Mini Shell.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ make
gcc minishell.c -o minishell
gcc mkdir.c -o mkdir
gcc rmdir.c -o rmdir
gcc mkfile.c -o mkfile
gcc ls.c -o ls
gcc swfile.c -o swfile
gcc man.c -o man
gcc chmod.c -o chmod
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ ./minishell
>> mkdir SO
>> mkdir(): Directorio SO creado con éxito.
>> rmdir SO
>> rmdir(): Directorio SO removido con éxito.
>> exit
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.2 - MiniShell $ make clean
rm -f minishell mkdir rmdir mkfile ls swfile man chmod *.o
```

➔ Al realizar la llamada 'make', se construyen los archivos ".o" ejecutables. Pero como se detallo en el paso 5 de compilación y ejecución, solo es necesario ejecutar el primero de todos los archivos ejecutables que se generan. Por ejemplo, en este caso la instrucción para ejecutar seria: "./minishell", ya que es el primer archivo en compilar y por ende, el archivo principal (convención adoptada).

Al realizar la instrucción ‘make’ obtenemos los archivos ejecutables con su respectivo nombre:

Al realizar la instrucción ‘make clean’ se eliminan los archivos de extensión “.o” generados durante la compilación:



Impresión por pantalla

Colores

Para realizar ciertas distinciones en algunos puntos de las trazas realizadas en los ejercicios, se decidió optar por crear dos archivos: colores.c y colores.h.

Estos archivos, simplemente ofrecen ciertas facilidades para las distintas implementaciones de los ejercicios al realizar impresión por pantalla (en terminal).

Esto toma cierto valor al realizar por ejemplo una distinción en la impresión de un proceso o un hilo con una tarea en específico. Por ejemplo, en el caso del ejercicio de la planta de reciclado, una manera muy sencilla de trazar el funcionamiento es realizando las correspondientes impresiones por pantalla de cada una de las etapas de la planta, es decir, que cada proceso imprima la acción que está realizando actualmente, pero si a esto le sumamos, que los recolectores impriman con un determinado color, los recolectores con otro y los recicladores con otro, la distinción se hace notable y facilita el trazado del funcionamiento.

```
gonzalo@raspberry:~/Desktop/Proyecto 2022 - SO/Ejercicio 1.1.1 - Planta de Reciclado/Inciso a $ ./PlantaRecicladoPIPES
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4376
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4378
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4377
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4376
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4378
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4377
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4380
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4376
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4378
RECOLECTOR | Item recolectado = Plastico (3) | Proceso = 4377
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Plastico (3) | Proceso = 4381
CLASIFICADOR | Item clasificado = Carton (2) | Proceso = 4380
RECICLADOR ALUMINIO | AYUDANDO (No hay aluminio para reciclar) | Item reciclado = Carton (2) | Proceso = 4384
RECICLADOR CARTON | AYUDANDO (No hay carton para reciclar) | Item reciclado = Plastico (3) | Proceso = 4382
RECICLADOR PLASTICO | Item reciclado = Plastico (3) | Proceso = 4383
RECOLECTOR | Item recolectado = Aluminio (4) | Proceso = 4376
RECOLECTOR | Item recolectado = Aluminio (4) | Proceso = 4378
CLASIFICADOR | Item clasificado = Carton (2) | Proceso = 4379
RECOLECTOR | Item recolectado = Aluminio (4) | Proceso = 4377
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Carton (2) | Proceso = 4381
CLASIFICADOR | Item clasificado = Carton (2) | Proceso = 4380
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Carton (2) | Proceso = 4383
RECICLADOR ALUMINIO | Estoy tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4384
RECICLADOR CARTON | Tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4382
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4378
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4376
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4377
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4380
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Plastico (3) | Proceso = 4381
RECICLADOR PLASTICO | Item reciclado = Plastico (3) | Proceso = 4383
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4376
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4378
CLASIFICADOR | Item clasificado = Plastico (3) | Proceso = 4379
RECOLECTOR | Item recolectado = Vidrio (1) | Proceso = 4377
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Plastico (3) | Proceso = 4381
CLASIFICADOR | Item clasificado = Aluminio (4) | Proceso = 4380
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Aluminio (4) | Proceso = 4383
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4378
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4376
RECOLECTOR | Item recolectado = Carton (2) | Proceso = 4377
CLASIFICADOR | Item clasificado = Aluminio (4) | Proceso = 4379
CLASIFICADOR | Item clasificado = Aluminio (4) | Proceso = 4380
RECICLADOR VIDRIO | AYUDANDO (No hay vidrio para reciclar) | Item reciclado = Aluminio (4) | Proceso = 4381
RECICLADOR PLASTICO | AYUDANDO (No hay plastico para reciclar) | Item reciclado = Aluminio (4) | Proceso = 4383
RECICLADOR ALUMINIO | Estoy tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4384
RECICLADOR CARTON | Tomando mate (Los otros recicladores no requieren ayuda) | Proceso = 4382
```

➔ Los recolectores imprimen con color rojo, los clasificadores con amarillo y los recicladores con verde respectivamente.

Archivos colores.c y colores.h

Los archivos colores.c y colores.h son simplemente el archivo de implementación de las funciones de colores de impresión y su correspondiente header.

- Las funciones de impresión se construyen como punteros a char, ya que son simplemente macros que se escriben como un string (%s) dentro de las funciones printf().
- Para utilizar las funciones, basta con realizar un printf() en el que se incluye la llamada a la función del color que se desea imprimir (previo al texto) y luego una llamada la función reset(), que permite restablecer el color de impresión original de la terminal.

```c
#include <stdlib.h>
#include <stdio.h>

#define BLACK "\033[1;30m"
#define RED "\033[1;31m"
#define GREEN "\033[1;32m"
#define YELLOW "\033[1;33m"
#define BLUE "\033[1;34m"
#define PURPLE "\033[1;35m"
#define CYAN "\033[1;36m"
#define WHITE "\033[1;37m"
#define RESET "\033[0m"
```

➔ Las macros definidas corresponden a los colores en tipo bold.

```c
/*
 * colores.c: Funciones para impresion de colores por pantalla.
 * -> Cada una de las funciones establece un color especifico para imprimir por pantalla.
 * -> La función reset() permite restablecer el color original de impresión de la consola.
 */

char * black() {
    return BLACK;
}

char * red() {
    return RED;
}

char * green() {
    return GREEN;
}

char * yellow() {
    return YELLOW;
}

char * blue() {
    return BLUE;
}

char * purple() {
    return PURPLE;
}

char * cyan() {
    return CYAN;
}

char * white() {
    return WHITE;
}

char * reset() {
    return RESET;
}
```

Por su parte, el archivo colores.h como se mencionó, es el header asociado a colores.c, que permite facilitar la importación del código fuente desde otros archivos, ya que permite abstraerse de la implementación de colores.c definiendo únicamente las signaturas:

```c
/*
 * colores.h: Header asociado a colores.c.
 * -> Facilidad de importacion de las funciones de impresión por pantalla.
 */

char * black();

char * red();

char * green();

char * yellow();

char * blue();

char * purple();

char * cyan();

char * white();

char * reset();
```

La convención de que adopta de utilizar punteros a char, en lugar de realizar procedimientos que realicen directamente el printf(), es que al utilizar muchos procesos/hilos en general en los ejercicios, se pueden ocasionar problemas con la concurrencia en los colores de impresión, ya que no se puede asumir que se ejecutan dos o tres llamadas a la función printf() seguidas, es decir, debe estar todo contenido dentro una única instrucción para que cuando se ejecute, se realicen sus operaciones de forma unitaria.

- ➔ En primera instancia se había realizado la construcción de estos archivos de esta manera, es decir, como procedimientos que realizaban los printf() y ya establecían el color de impresión por pantalla, pero al trabajar con concurrencia entre hilos/procesos, se descubrió en el ejercicio del 'Puente de una sola mano' los vehículos del norte o del sur imprimían a veces con su color correspondiente y a veces, con el color de la otra dirección, por lo que la forma correcta de realizarlo era retornando los punteros a char y utilizar las macros dentro de una única instrucción printf(). Luego de realizar los cambios necesarios y adaptar el código nuevamente, todos estos problemas se solucionaron.

Ejemplo de impresión: Mini Shell.

```c
printf("%s>> %s", yellow(), reset());
```

Problemas

En esta sección se desarrollarán los diferentes aspectos del desarrollo de los problemas propuestos del proyecto.

Lectura

El artículo elegido por la comisión fue “Virtualization Overview”, que trata acerca del tema de Virtualización y la propuesta realizada es un flyer que consta de 2 páginas y abstrae los aspectos que consideramos más importantes.

- ➔ El flyer producido se encuentra en formato PDF en la carpeta “2.1. Lectura” dentro del entregable de la comisión.

Problemas conceptuales

Para la realización de los problemas conceptuales se opto por realizarla en archivos PDF por separado para evitar obstruir la lectura del presente informe, cada uno de los puntos se encuentra en su carpeta correspondiente con su resolución. Ambas carpetas pueden encontrarse dentro del archivo entregable presentado por la comisión.