



SISTEMAS OPERATIVOS - 2022

INFORME DEL PROYECTO

UNIVERSIDAD
NACIONAL DEL SUR

DEPARTAMENTO DE CIENCIAS
E INGENIERÍA DE LA COMPUTACIÓN

COMISIÓN 32
BERTI GIROLAMO, NICOLÁS MARTÍN
HERNÁNDEZ, PALOMA

1. Experimentación de Procesos y Threads con los Sistemas Operativos

Punto 1.1

En este punto resolvimos un problema dado en el contexto de una planta que gestiona el reciclado de residuos. La planta cuenta con recolectores que empaquetan y envían lo que recolectan al sector de clasificación de residuos. Luego, se identifica cada paquete y se deriva a los distintos puntos donde los recicladores los procesan.

El problema consiste en modelar el proceso de reciclado de esta planta considerando 4 tipos de residuos (vidrio, cartón, plástico y aluminio), 3 recolectores (quienes empaquetan los residuos), 2 clasificadores (quienes reciben paquetes y envían a los recicladores) y 4 recicladores (uno por tipo de residuo). También tuvimos que tener en cuenta que estos últimos pueden optar por colaborar con otro reciclador en el caso de que no tengan nada que reciclar, o detenerse a tomar mate.

Inciso 1.1-1.a

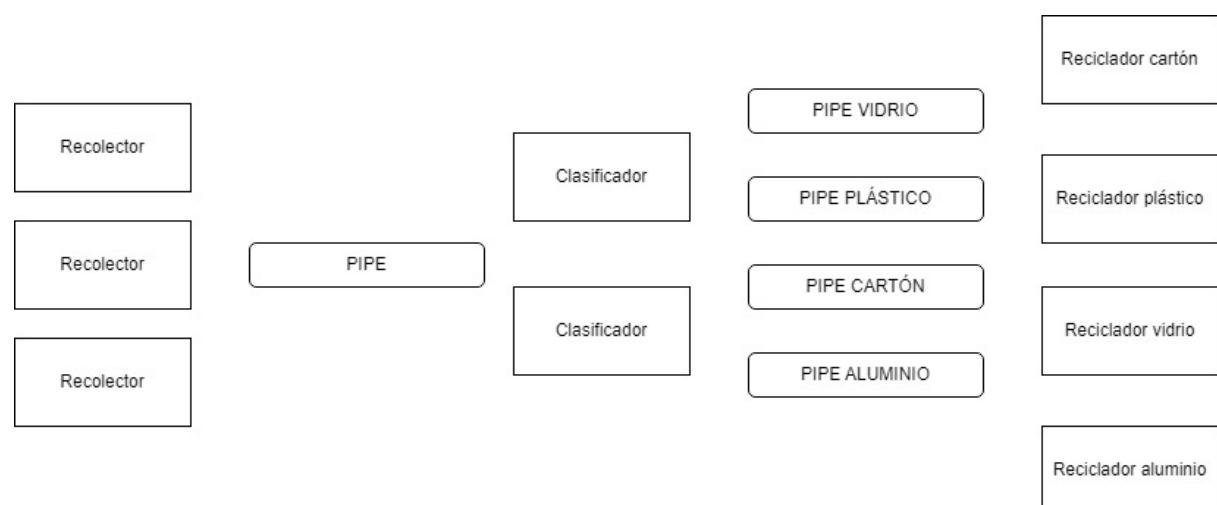
Para resolver el problema de modelar el proceso que lleva a cabo esta planta en este inciso tuvimos que utilizar pipes como forma de comunicación entre los componentes.

Creamos un archivo en el cual se utilizan 5 pipes: uno para modelar la comunicación entre recicladores y clasificadores y los cuatro restantes corresponden cada uno a un tipo de residuo.

En los procesos correspondientes a los recolectores se utiliza una función para generar basura de manera random, la cual se recupera y se envía por el pipe para modelar la comunicación entre recicladores y clasificadores.

En los procesos correspondientes a los clasificadores se lee este pipe en busca de algún residuo para reciclar, el cual una vez obtenido se clasifica según su tipo y se envía por su pipe correspondiente.

En los procesos correspondientes a los recicladores se lee cada pipe según el tipo de residuo al que esté asignado el reciclador y si recibe un residuo, lo recicla. También puede darse el caso de que no esté ocupado con su tipo de residuo y decida ayudar a otro reciclador, en ese caso, recicla otro tipo de residuo que no es el que le corresponde. Si decide no ayudar puede detenerse a tomar mate.



La forma de ejecución es desde la consola `./main`

Inciso 1.1-1.b

En este ejercicio se pide que se resuelva el problema pero usando colas de mensajes. Como en las colas de mensajes no es necesario que los procesos que usen las colas estén vinculados a los otros (relación padre-hijo) hicimos diferentes archivos .c que representan a los recolectores, clasificados y recicladores.

Entonces, mediante una llave (la cual es igual para todos) se conectan a la cola de mensajes.

En la cola de mensajes se diferencian 8 tipos de mensajes.

Tipos de basura

- 1: Plástico sin clasificar
- 2: Vidrio sin clasificar
- 3: Aluminio sin clasificar
- 4: Carton sin clasificar
- 5: Plástico clasificado
- 6: Vidrio clasificado
- 7: Aluminio clasificado
- 8: Cartón clasificado

El mecanismo de resolución es identico al del punto 1.1-1.a. Se usan los mismos algoritmos para recolectar, clasificar y reciclar (ya sea de su tipo, o cuando ayuda (siempre y cuando no se haya ido a tomar mate))

La forma de ejecución es desde la consola **./main**

Inciso 1.1-2

En este punto se pide una shell con mínimo 6 comandos, los comandos que implementamos son,

- Crear un directorio: mkdir [ubicación] [nombre]
- Remover un directorio: rmdir [ubicación] [nombre]
- Crear un archivo: mkfile [ubicación] [nombre.formato]
- Listar el contenido de un directorio: ldir [ubicación]
- Listar el contenido de un archivo: lfile [ubicación] [nombre.formato]
- Mostrar una ayuda con los comandos posibles: help

OBS: Es importante que la ubicación termine con / o en su defecto, el nombre empiece con / ya que lo que se hace es concatenar los strings “ubicación” y “nombre”

La forma que se utilizó para implementarlos es la misma que la anterior, se implementan en .c separados, así la shell queda con la posibilidad de agregar nuevos comandos sin la necesidad de compilarla nuevamente (solo se tienen que compilar los nuevos comandos)

Los comandos se ejecutan mediante la llamada al sistema “execvp” y los comandos se ejecutan en procesos nuevos, mientras que el proceso principal (la shell) se queda esperando a que terminen para pedir otro comando.

La forma de ejecución es desde la consola **./shell**

Punto 1.2

Inciso 1.2-1.a

En este ejercicio se implementa la secuencia pedida en el enunciado mediante hilos y semáforos. No hay mucho que agregar sobre la implementación, es análogo a los ejemplos vistos en la teoría/clases de consulta.

La forma de ejecución es desde la consola **./shell**

Inciso 1.2-1.b

En este ejercicio se implementa la secuencia anterior pero con el uso de pipes. Esto es posible ya que el pipe simula un semáforo puesto que es “bloqueante” si no hay nada dentro de él.

La forma de ejecución es desde la consola **./shell**

Inciso 1.2-2.I

En este ejercicio se implementa el problema de cruzar el puente de una sola dirección. Para esto, los vehículos tienen un sentido (norte ó sur). Se implementan 3 semáforos y 2 mutex.

- puenteDisponible: es un semáforo que pueden usar ambos vehículos, existe una sola instancia de dicho semáforo y se utiliza para cruzar el puente cuando no hay nadie sobre él. Es decir, solo los vehículos de una dirección pueden tenerlo.
- autosNorte: es un semáforo de conteo que indica cuantos vehículos hay cruzando el puente del sentido norte. Se utiliza para que un vehículo del norte ingrese al puente en el caso que haya un vehículo de este sentido cruzando. (*Los vehículos que circulan en la misma dirección que el que se encuentra cruzando, pueden acceder al puente al mismo tiempo*)
- autosSur: lo mismo que autosNorte pero para los autos del sur.
- mutexNorte: con este mutex se cumple que solo se pueda verificar si hay vehículos cruzando el puente sobre el sentido norte de uno a la vez ó, si hay vehículos cuando terminó de cruzar un vehículo el puente para liberar el puente en caso de ser el último.
- mutexSur: lo mismo que mutexNorte pero para los autos del sur.

El algoritmo funciona de la siguiente manera

El vehículo se fija si hay vehículos de su mismo sentido cruzando el puente.

Si los hay, cruza con ellos al mismo tiempo (semáforo autosNorte)

Si no, espera a que el puente esté libre. (espera al semáforo puenteDisponible)

Cuando termina de cruzar, verifica si lo siguieron (i.e hay vehículos aún en el puente) para liberar el puente.

Para hacer las pruebas utilizamos la constante “CANTIDAD_AUTOS 6”. Esta constante puede modificarse por un número muy grande para simular un while(1).

La forma de ejecución es desde la consola **./shell**

Inciso 1.2-2.II

A raíz de la implementación anterior, nos dimos cuenta que ocurre el problema de **inanición** puesto que, si un vehículo de cierto sentido está cruzando el puente, a medida que lleguen vehículos, como pueden cruzar al mismo tiempo, siempre van a cruzar si llega uno puesto que va haber vehículos cruzando el puente (ya que se tarda, no es que se cruza de inmediato). Por lo que, el otro sentido nunca llegaría a cruzar el puente ya que el que espera por el token “puenteDisponible” nunca lo obtendrá.

Inciso 1.2-2.III

En este ejercicio se usan procesos y colas de mensajes para solucionar el problema presentado en la implementación anterior.

Para solucionar este problema, se implementan dos colas de mensajes.

- **colaEspera:** cuando un vehículo de cierto sentido llega al puente, se inserta en esta cola. En esta cola se puede diferenciar un mensaje especial, que es el mensaje de tipo **1** ya que se simula un “mutex” y el resto de los mensajes son de tipo **pid+2** del vehículo, ya que cada vehículo es un proceso diferente.
- **colaPasar:** cuando un vehículo se inserta en esta cola, es que está habilitado a cruzar el puente. Para esta cola también se simula un mutex con un tipo **1** y el resto de los mensajes son del tipo **pid+2** del vehículo.

Los mensajes contienen el pid (tipo) y el sentido (en el caso de ser el mutex, es un sentido neutro, no es relevante)

El algoritmo comienza cuando un vehículo de cierto sentido llega al puente. En este momento, el vehículo se inserta en la **colaEspera** ya que se encuentra a la espera de cruzar el puente.

Mientras que en el proceso principal, hay un “intermediario” que es el que habilita a los autos a cruzar el sentido.

El intermediario (operario, como se quiera llamar) lo que hace es ver los vehículos que llegaron al puente (los quita de la cola **colaEspera**, como las colas son FIFO -> lo que sucede es que va sacando al primero que llegó y así..) e ir insertando a los vehículos en la **colaPasar**.

Esto es, suponga que llegan en este orden

Norte, Norte, Norte, Sur, Sur, Norte, Sur

El operario insertará 3 del norte y se quedará esperando a que crucen el puente. Cuando terminen de cruzar, insertará a los 2 del sur, esperará a que pasen, luego al del norte, espera a que pase, luego al del sur, espera a que pase, y así...

Los vehículos lo que hacen es esperar que su pid+2 esté en la **colaPasar** para cruzar el puente. Al final, cuando terminan de cruzar verifican de a 1 (esto se puede hacer por los mensajes que simulan un mutex) si fueron los últimos en cruzar el puente (*acá se hace uso de la colaPasar ya que cuando se quita de la cola el vehículo se inserta nuevamente para indicar a sus compañeros que está cruzando el puente, cuando termina de cruzar, se quita*) para darle la señal al operario que ya terminaron de cruzar todos y que pueda cambiar de sentido (esta señal, es con el mutex pero de la otra cola)

(Se agrega un script adicional que se llama clear que lo que hace es borrar la cola de mensajes en el caso de que se aborte la ejecución del programa principal, ya que la cola de mensajes sigue estando en memoria)

Gracias a esta resolución, se puede resolver el problema de inanición respetando la concurrencia.

2. Problemas

2.1. Lectura

En este inciso, a partir de la lectura de algunos artículos, debíamos generar alguna propuesta multimedia para "promover"/"vender"/"presentar" la idea que se desarrollaba en uno de ellos.

Nuestra elección fue el artículo "Allied Telesis AlliedWare Plus Operating System" donde se expone el sistema operativo AlliedWare Plus y realizamos un Podcast para presentarlo.

El podcast se encuentra en la entrega final en formato .mp3