Índice de contenido

1. Enunciado

Primer Proyecto: ConcuShare
75.59 - Técnicas de Programación Concurrente I

Objetivo

El objetivo de este proyecto es el desarrollo de una aplicación conocida como "ConcuShare". Esta aplicación permitirá a los usuarios enviar y recibir archivos al mismo tiempo, tanto archivos de texto como binarios. Cada usuario ejecutará un programa que le permitirá publicar los archivos que desea compartir con los demás, así como también realizar el intercambio de archivos.

Requerimientos Funcionales

Los requerimientos funcionales se enumeran a continuación:

1. Los usuarios deberán poder compartir archivos entre sí, es decir, habilitar ciertos archivos para que puedan ser descargados por los demás usuarios.

2. Los usuarios deberán poder dejar de compartir uno o más archivos en cualquier momento.

3. Los usuarios podrán consultar qué archivos comparten los demás usuarios y seleccionar uno o más archivos a descargar.

4. Las descargas de archivos deberán realizarse en forma simultánea, por ejemplo, si un usuario desea descargar más de un archivo las descargas no deberán encolarse sino realizarse al mismo tiempo.

5. Si los usuarios se encuentran realizando una o más descargas deberán poder seguir utilizando la aplicación.

6. Deberán finalizarse todas las descargas que están pendientes cuando los usuarios cierran la aplicación.

Requerimientos no Funcionales

Los siguientes son los requerimientos no funcionales de la aplicación:

1. El trabajo práctico deberá ser desarrollado en lenguaje C o C++, siendo este ultimo el lenguaje de preferencia.
2. El programa puede no tener interfaz gráfica y ejecutarse en una o varias consolas de línea de comandos.

3. El proyecto deberá funcionar en ambiente Unix / Linux.

4. La aplicación deberá funcionar en una única computadora.

5. El programa deberá poder ejecutarse en “modo debug”, lo cual dejará registro de la actividad que realiza en uno o más archivos de texto para su revisión posterior.

Tareas a Realizar

A continuación se listan las tareas a realizar para completar el desarrollo del trabajo práctico:

1. Dividir el programa en procesos. El objetivo es lograr que cada programa participante esté conformado por un conjunto de procesos que sean lo más sencillos posible.

2. Una vez obtenida la división en procesos, establecer un esquema de comunicación entre ellos teniendo en cuenta los requerimientos de la aplicación. ¿Qué procesos se comunican entre sí? ¿Qué datos necesitan compartir para poder trabajar?

3. Tratar de mapear la comunicación entre los procesos a los problemas conocidos de concurrencia.

4. Determinar los mecanismos de concurrencia a utilizar para cada una de las comunicaciones entre procesos que fueron detectados en el ítem 2. No se requiere la utilización de algún mecanismo específico, la elección en cada caso queda a cargo del grupo y debe estar debidamente justificada.

5. Realizar la codificación de la aplicación. El código fuente debe estar debidamente documentado.

Informe

Junto con el proyecto se deberá entregar un informe. El informe se deberá presentar en una carpeta o folio con el siguiente contenido:
1. Informe propiamente dicho
2. CD con el código fuente de la aplicación

El informe propiamente dicho deberá contener los siguientes ítems:

1. Breve análisis de problema, incluyendo una especificación de los casos de uso de la aplicación.
2. Detalle de resolución de la lista de tareas anterior. Prestar atención especial al ítem 4 de la lista de tareas, ya que se deberá justificar cada uno de los mecanismos de concurrencia elegidos.
3. Diagramas de clases de un programa.
4. Diagramas de transición de estados de un programa.

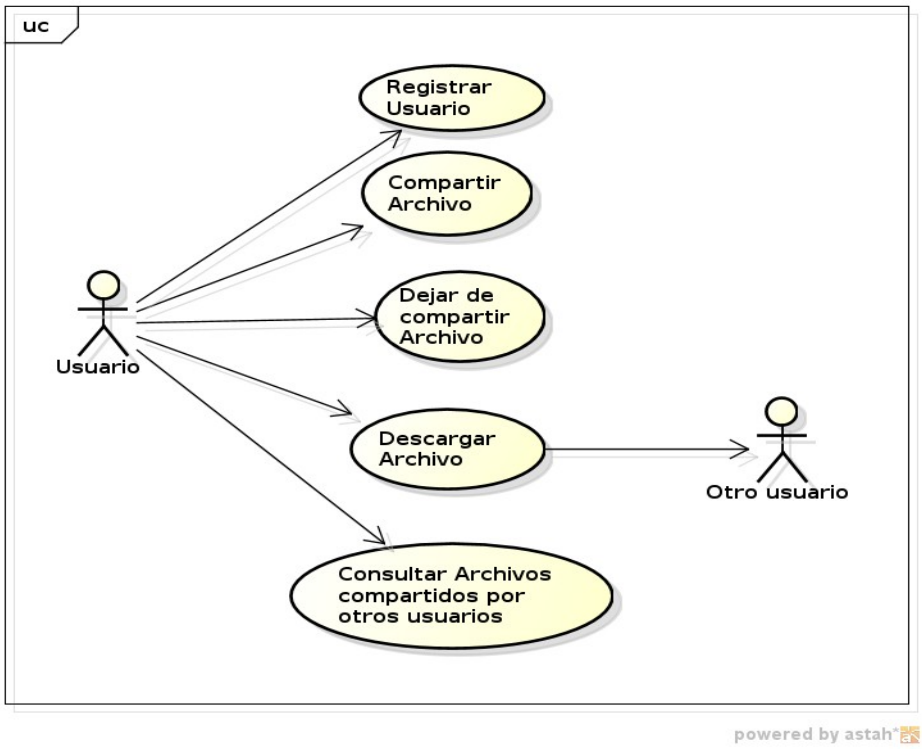
2. Análisis del problema

Se requiere ofrecer un servicio de envío de archivos permitiendo a muchos usuarios compartir archivos entre sí con la posibilidad de seguir trabajando mientras se transfieren los mismos.

2.1 Hipótesis:

- Luego de descargar un archivo, no se comparte el mismo automáticamente.
- Se permite que archivos de distintos usuarios posean el mismo nombre.
- Se decidió crear una carpeta para las descargas de cada usuario; la misma se identifica de la siguiente manera: *"descargas_nombreUsuario_pidUsuario"*
- Se supone que para compartir un archivo debe colocarse el mismo en la capeta en la cual se encuentra el ejecutable.
- Se decidió que al momento de cerrar la aplicación se esperará a que finalicen completamente tanto las descargas que ese usuario este realizando así como también, los envíos de sus archivos a otros usuarios.
- Cuando la aplicación se ejecuta en modo debug se generará un archivo de nombre debug en el directorio en el que se encuentre el ejecutable. En el mismo se deja registro de los envíos, descargas, inicios y finalizaciones de los distintos procesos.

2.2 Casos de uso:



2.2.1 Descripción de actores:

Usuario: representa al usuario de la aplicación que desea compartir archivos y consultar y/o descargar archivos que los demás usuarios comparten.

Otro usuario: representa al usuario que es responsable de iniciar el envío del archivo a descargar.

2.2.2 Descripción de casos de uso:

Registrar Usuario: provee la funcionalidad de registrar al usuario en el sistema para poder iniciar el intercambio de archivos.

Compartir archivo: provee la funcionalidad de registrar en el sistema el archivo que el usuario desea compartir con los demás.

Dejar de compartir archivo: provee la funcionalidad de retirar un archivo que el usuario estaba compartiendo con los demás.

Descargar archivo: provee la funcionalidad descargar un archivo que otro usuario comparte.

Consultar archivos compartidos por otros usuarios: provee la funcionalidad de informar los archivos que los demás usuarios comparten.

2.2.3 Especificación de casos de uso:

Caso de uso: Registrar usuario
Descripción: el usuario se registra en el sistema
Actores participantes: Usuario
Precondiciones:
Flujo
1. El sistema solicita el nombre de usuario
2. El usuario ingresa el nombre de usuario
3. El sistema registra al usuario
Fin caso de uso
Postcondición: queda registrado el usuario en el sistema para que los demás usuarios puedan interactuar con el mismo.

Caso de uso: Compartir archivo
Descripción: el usuario quiere compartir un archivo con los demás
Actores participantes: Usuario
Precondiciones: Se debe haber ejecutado el caso de uso registrar usuario
Flujo
1. El sistema presenta un menú al usuario
2. El usuario selecciona la opción 1: “Ingresar a mis archivos compartidos”
3. El sistema presenta un menú para permitir al usuario compartir un archivo
4. El usuario selecciona la opción 1: “Compartir un archivo”

(Flujo alternativo 4.1 el usuario ingresa opción 3)
5. El sistema solicita la ruta al archivo
6. El usuario ingresa la ruta del archivo
7. El sistema registra el archivo
8. Se vuelve al paso 3
Flujo alternativo
4.1 Se sale del menu y se vuelve al paso 1
Fin caso de uso
Postcondición: queda registrado el archivo en el sistema para que los demás usuarios lo puedan descargar.

Caso de uso: Dejar de compartir archivo
Descripción: el usuario quiere dejar de compartir un archivo
Actores participantes: Usuario
Precondiciones: Se debe haber ejecutado los casos de uso “Registrar usuario” y “Compartir archivo”.
Flujo
1. El sistema presenta un menú al usuario
2. El usuario selecciona la opción 1: “Ingresar a mis archivos compartidos”
3. El sistema presenta un menú para permitir al usuario dejar de compartir un archivo
4. El usuario selecciona la opción 2: “Dejar de compartir un archivo” (Flujo alternativo 4.1 el usuario ingresa opción 3)
5. El sistema presenta una lista de archivos compartidos del usuario asociados a un identificador
6. El sistema solicita el identificador del archivo que desee dejar de compartir
7. El usuario ingresa el identificador
8. El sistema retira el archivo de la lista de archivos compartidos del usuario

9. Se vuelve al paso 3
Flujo alternativo:
4.1 Se sale del menu y se vuelve al paso 1
Fin caso de uso
Postcondición: queda retirado el archivo en el sistema para que los demás usuarios no lo puedan descargar.

Caso de uso: Consultar archivos compartidos por otros usuarios
Descripción: el usuario quiere visualizar los archivos que los demás usuarios desean compartir
Actores participantes: Usuario
Precondiciones: Se debe haber ejecutado el caso de uso Registrar Usuario y Compartir Archivo de otro usuario.
Flujo
1. El sistema presenta un menú al usuario
2. El usuario selecciona la opción 2: “Buscar un archivo”
3. El sistema busca los archivos que los demás usuarios comparten y los muestra por pantalla
4. El sistema le solicita al usuario si desea descargar un archivo o no.
5. El usuario selecciona la opción “n”
Fin caso de uso
Postcondición: queda presentado en pantalla la lista de archivos compartidos por los demás usuarios

Caso de uso: Descargar archivo
Descripción: el usuario se comunica con la aplicación para descargar un archivo compartido por otro usuario.
Actores participantes: Usuario, Gestor de descargas de otro usuario
Precondiciones: Se deben haber ejecutado los casos de uso Registrar Usuario y Compartir

archivo de otro usuario
Flujo
1. El sistema presenta un menú al usuario
2. El usuario selecciona la opción 2: “Buscar un archivo”
3. El sistema busca los archivos que los demás usuarios comparten y los muestra por pantalla. Asociando un identificador al otro usuario y un identificador a cada archivo que comparte cada usuario.
4. El sistema le solicita al usuario si desea descargar un archivo o no.
5. El usuario selecciona la opción “s”
6. El sistema solicita el identificador de usuario del cual se desea descargar un archivo
7. El usuario ingresa el identificador del otro usuario
8. El sistema solicita el identificador del archivo que se desea descargar.
9. El sistema se comunica con el servidor de archivos del otro usuario y se procede a la descarga del archivo.
Fin caso de uso
Postcondición: se crea un nuevo archivo en la carpeta creada especialmente para el usuario formada por su nombre y process id.

3. Detalle de resolución de problemas

3.1 División de procesos

El programa está dividido en varios procesos, iniciando con un proceso de usuario y otro de escucha de peticiones de envío.

Además de estos, se suma un proceso por cada transferencia de archivo. Entendiéndose como transferencia tanto el envío como la descarga de un archivo. Estos procesos se crean dinámicamente ante la petición de la transferencia de un archivo.

3.1.1 Descripción de procesos:

- El proceso Usuario se encarga de realizar la interacción con el usuario. Este presenta menús por consola y lee de la misma para analizar las peticiones del usuario.

- El proceso Escucha se encarga de responder las peticiones de los demás usuarios que desean descargar un archivo compartido por el usuario. Lee constantemente un FIFO donde le llegan las peticiones y por cada petición crea un proceso nuevo que se encargará de enviar el archivo.

- El proceso Descarga se encarga de realizar la peticion al proceso Escucha del otro usuario y recibir el archivo por un FIFO. Este FIFO se crea con los process id de los procesos que están implicados en la transferencia del archivo.

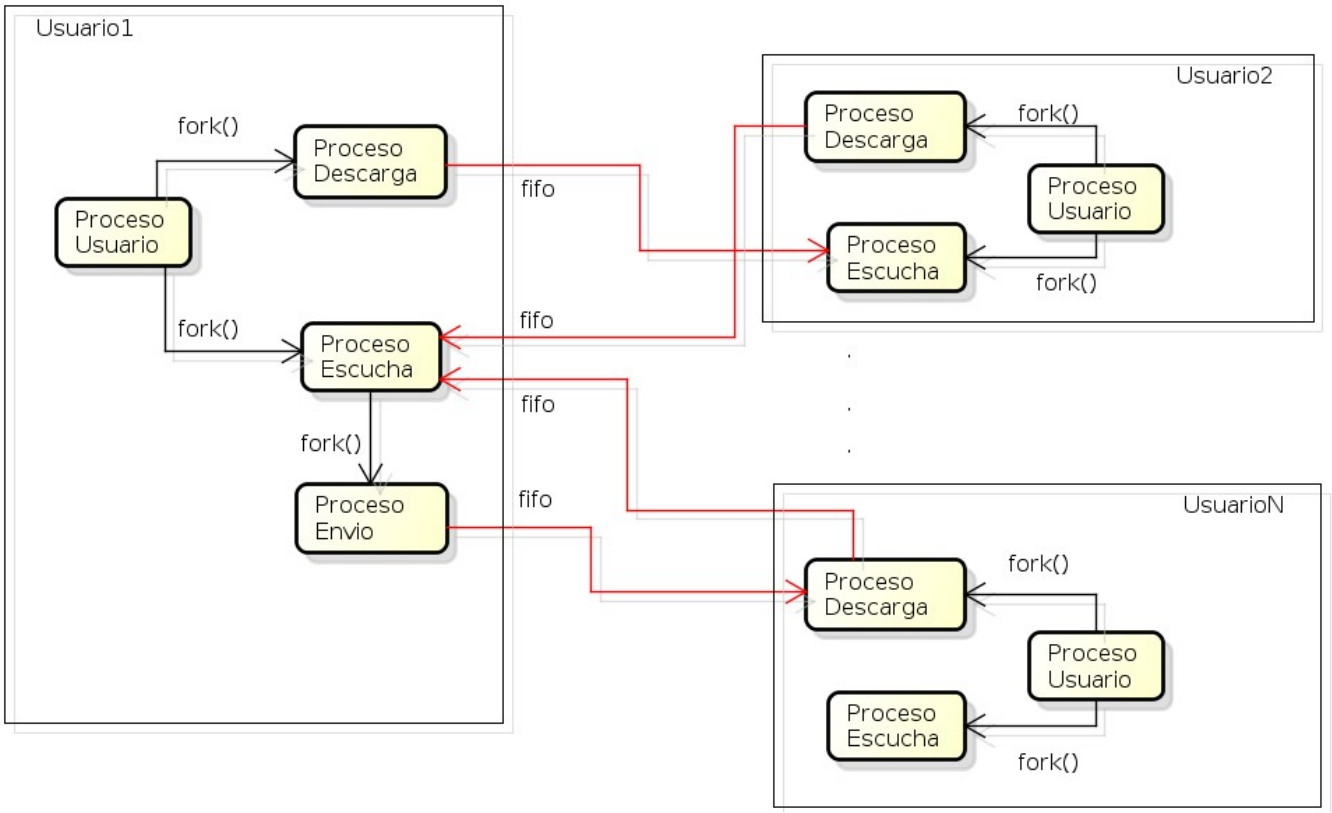
- El proceso Envío es el que se encarga de enviar el archivo a través del FIFO.

3.2 Esquema de comunicación de procesos

La arquitectura del programa es similar a la de una red peer-to-peer, dado que el programa funciona simultáneamente como cliente y servidor.

Al principio se consideró una arquitectura cliente-servidor, en la que el servidor trabajaba de una forma casi serial al comunicar los clientes. Luego de esto, los mismos continuaban con la transferencia de archivos por su cuenta. Es por este motivo, que se decidió desarrollar la aplicación con una arquitectura peer-to-peer, de la cual se detalla la forma en la que interactúan un cliente con otro en la siguiente sección.

3.3 Diagrama de comunicación de procesos:



3.3.1 Secuencia de comunicación:

3.3.1.1 Secuencia de descarga de un archivo

1. El usuario ingresó el archivo que desea descargar de otro usuario
2. El proceso usuario crea un nuevo proceso Descarga para resolver la petición.
3. El proceso Descarga se comunica a través de un FIFO con el proceso Escucha del otro usuario.
4. El proceso Escucha del otro usuario recibe la petición y crea un nuevo proceso Envío para enviar el archivo al proceso Descarga.
5. Los procesos Envío y Descarga se comunican por un nuevo FIFO creado solo para el propósito de la transferencia.
6. El proceso Descarga lee el FIFO por el cual se va a pasar el archivo al mismo tiempo que el proceso Envio va escribiendo el archivo en el FIFO.
7. Al finalizar la transferencia del archivo, ambos procesos terminan su ejecución.

3.3.1.2 Secuencia de envío de un archivo

1. Un proceso Descarga de otro usuario se comunica al proceso Escucha del usuario que comparte el archivo a través de un FIFO.
2. El proceso Escucha recibe la petición y crea un nuevo proceso Envio para resolver la petición.
3. Los procesos Envío y Descarga se comunican por un nuevo FIFO creado solo para el propósito de la transferencia.
4. El proceso Envío comenzará a leer el archivo y colocarlo en el FIFO al mismo tiempo que el proceso Descarga lee el FIFO.
5. Al finalizar la transferencia del archivo, ambos procesos terminan su ejecución.

3.3.2 Los datos necesarios para la comunicación son:

1. El FIFO de escucha de cada usuario, creado cuando el usuario se registra en la aplicación.
2. Los process id de los procesos que están transfiriendo un archivo, llamados proceso Envío y Descarga.
3. El FIFO creado por cada transferencia. Su nombre se compone de los process id de los procesos Envío y Descarga.
4. El nombre de archivo que se desea descargar.
5. Un archivo de usuarios compartido que indica qué usuarios están disponibles y que archivos comparten dichos usuarios.

4. Mapeo de sincronización y comunicación a problemas conocidos de concurrencia.

En la aplicación se ven las siguientes instancias de comunicación y sincronización entre procesos:

- La transferencia del archivo mismo, la cual se puede interpretar como el problema de productor-consumidor en el cual el proceso de descarga es el consumidor y el proceso de envío es el productor.
- Las peticiones de descarga de archivos entre dos usuarios también es un problema de productor-consumidor en el cual el proceso escucha es el que “consume” las peticiones de los otros usuarios y el proceso descarga es el que “produce” la petición para descargarse un archivo que comparte ese usuario. La diferencia con el anterior es que en este caso hay muchos productores y un solo consumidor; en este caso, se debe sincronizar la escritura de peticiones.
- El acceso al archivo de usuarios para saber que usuarios están activos es un problema de exclusión mutua entre procesos, ya que cualquier proceso puede leer o escribir el mismo.
- El acceso al archivo de debug para realizar la escritura por parte de los usuarios que se

ejecuten en modo debug es un problema de exclusión mutua entre procesos.

4.1 Mecanismos de concurrencia utilizados

Para establecer la comunicación de los procesos para los problemas de productor consumidor se consideraron las siguientes opciones:

- Crear una memoria compartida sincronizada con semáforos.
- Utilizar un FIFO.
- Utilizar Pipes
- Señales

La ventaja de la memoria compartida en comparación con FIFOS es en cuestiones de eficiencia, ya que no involucra al sistema operativo para acceder a la misma, solamente se involucra cuando se accede a los semáforos para sincronizar pero esto no es comparable con los accesos de escritura y lectura. Además es necesario tener un control de la inicialización de semáforos que podría traer problemas en los mismos.

La ventaja de los FIFOS con respecto a la memoria compartida es que el sistema operativo ya tiene implementada la sincronización de los procesos en cuanto al acceso de escritura y lectura. Otra ventaja es que en la inicialización de los FIFOS no hay que tener un control robusto en comparación con los semáforos. Tampoco es necesario crear una clave única para crear el FIFO, este permite identificarlo de alguna manera como un “archivo” en el filesystem de Linux. Además para establecer la comunicación entre dos procesos solamente necesitan abrir el FIFO, conociendo su nombre en el filesystem, y no tener que compartir un archivo para generar una clave específica como en la utilización de memoria compartida y semáforos.

Los Pipes sólamente se pueden utilizar con procesos que estén relacionados, y los procesos ejecutados por distintos usuarios no se encuentran relacionados, por una decisión de diseño.

Debido a estas consideraciones se optó por utilizar FIFOS en la comunicación de los procesos ya que nos llevó a una implementación mas sencilla de la aplicación.

Las señales sirven para informar eventos y se utilizan cuando se desea finalizar el programa. Al seleccionar la opción del menú de Salir del programa, se envía una señal *SIGINT(-2)* al proceso de Escucha, que se había creado al iniciarse la aplicación. Esta señal es capturada y manejada de forma tal de que este proceso finalice su ejecución correctamente, cerrando y liberando los recursos que el mismo estuviese utilizando.

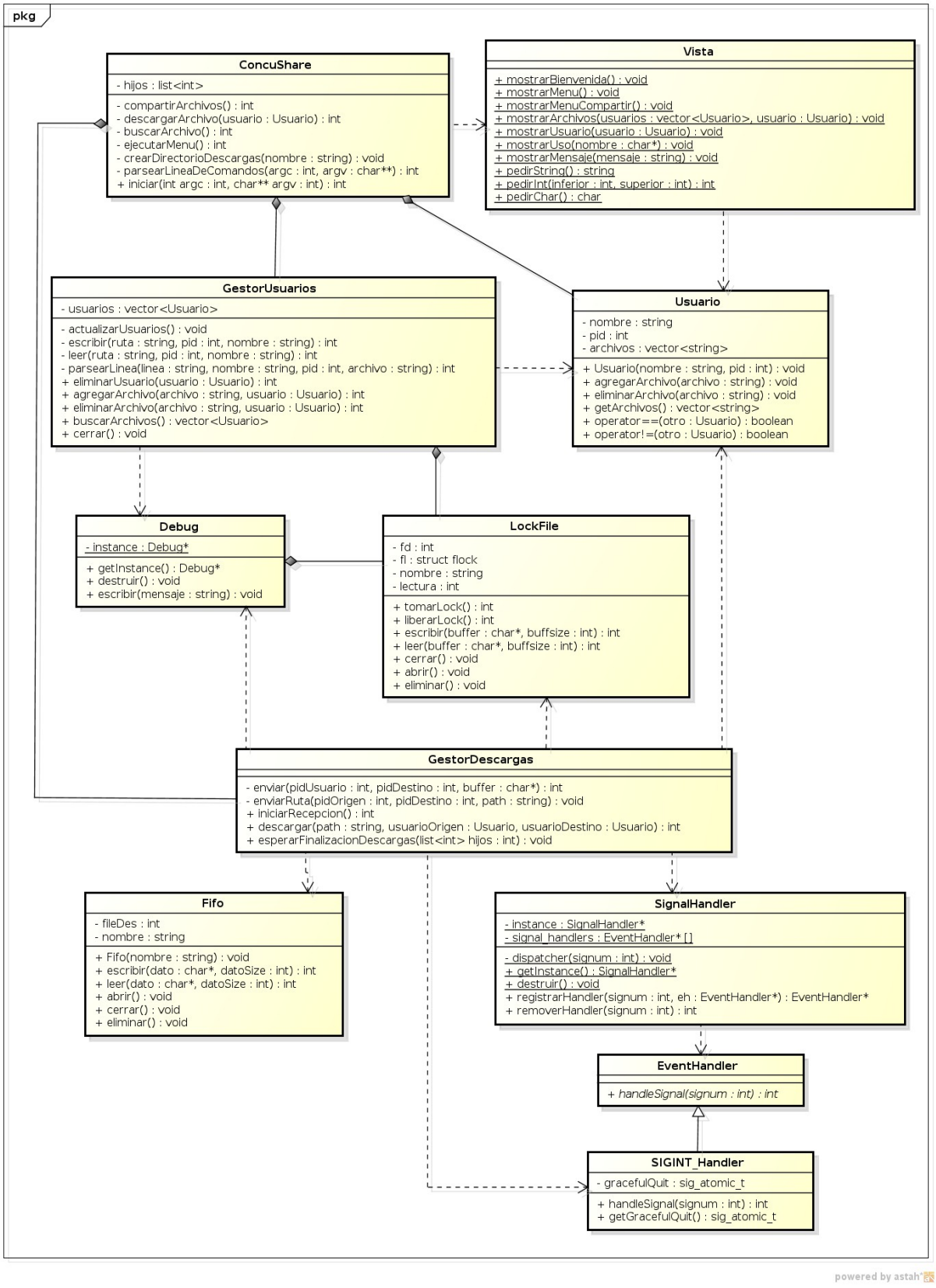
Además de los FIFOS se utilizaron LockFiles para la sincronización de procesos en el acceso de archivos:

- Cuando se requiere acceder al archivo de usuarios se aplica el lock para acceder al mismo

garantizando exclusión mutua.

- Cuando se requiere acceder a un FIFO de escucha de otro usuario, los procesos que envían las peticiones se sincronizan con un LockFile para garantizar que uno solo escribe a la vez y prevenir el conflicto entre los mensajes.
- Como todos los procesos escriben en el mismo archivo de debug es necesario sincronizar también la escritura con un lock.

5. Diagrama de clases:



5.1 Especificación de clases:

ConcuShare: tiene la responsabilidad de interactuar con el usuario mediante la Vista y responder a las peticiones del mismo.

Atributos:

- gestorDescargas: es un objeto de tipo GestorDescargas utilizado para realizar las descargas y los envíos.
- GestorUsuarios: es un objeto de tipo GestorUsuarios utilizado para realizar el alta, baja y modificación de usuarios y sus correspondientes archivos.
- Usuario: es un objeto de tipo Usuario encapsulando los datos del usuario que ejecuta la aplicación.
- Hijos: es una lista de enteros que almacena los identificadores de los procesos de descargas.

Métodos:

- compartirArchivos: muestra el menú Compartir Archivos y responde a la petición del usuario en cuanto a sus archivos compartidos.
- DescargarArchivo: muestra los archivos del usuario elegido para determinar el archivo a descargar. Incia la descarga.
- BuscarArchivos: obtiene del gestorDescarga los archivos compartidos por otros usuarios.
- EjecutarMenu: muestra el menú principal y atiende la petición del usuario.
- Iniciar: inicia sesión y comienza la ejecución de la aplicación.

Vista: tiene la responsabilidad de realizar la interacción con el usuario.

Métodos:

- mostrarBienvenida: imprime en consola una bienvenida al usuario.
- mostrarMenu: imprime en consola el menú con opciones para el usuario.
- mostrarMenuCompartir: imprime en cosola el menú de opciones de compartir archivos.
- mostrarArchivos: imprime por pantalla los archivos que un determinado usuario comparte.
- mostrarUsuario: imprime por pantalla el nombre de usuario y los archivos que el mismo comparte.
- mostrarUso: imprime como se debe utilizar la aplicación.
- mostrarMensaje: imprime un mensaje por pantalla pasado por parámetro.
- pedirString: lee de la entrada estándar una palabra y lo retorna.

- pedirInt: lee un número de la entráda estándar y valida que esté entre dos numeros pasados por parámetro.
- pedirChar: lee un caracter de entrada estándar y lo retorna.

Gestor Usuarios: tiene la responsabilidad de agregar y remover usuarios y archivos compartidos por esos usuarios en el archivo de usuario.

Atributos:

usuarios: es una lista de tipos Usuario conteniendo a los usuarios disponibles.

lock: es un objeto de tipo LockFile utilizado para sincronizar el acceso al archivo de usuarios

Métodos:

- eliminarUsuario: remueve un usuario del archivo de usuarios.
- agregarArchivo: agrega un archivo a la lista de archivos compartidos de un usuario.
- eliminarArchivo: remueve un archivo de la lista de archivos compartidos de un usuario.
- actualizarUsuarios: actualiza los datos de la lista de usuarios con el archivo de usuarios.
- buscarArchivos: actualiza a los usuarios y luego devuelve la lista de los mismos.
- cerrar: libera el lock del archivo y lo cierra.
- escribir: escribe una línea en el archivo de usuarios compuesta por el nombre de usuario, el process id del usuario y la ruta al fifo de escucha del mismo.
- leer: lee una línea del archivo de usuarios y obtiene el nombre del usuario, el process id y la ruta al fifo de escucha.
- parsearLinea: obtiene de un string el nombre de usuario, el process id y la ruta al fifo de escucha.

Usuario: tiene la responsabilidad de agregar, remover y consultar los archivos disponibles de un usuario en particular.

Atributos:

- nombre: es un string conteniendo el nombre del usuario.
- pid: es un entero conteniendo el process id de la instancia del usuario.
- archivos: es un vector de strings conteniendo los archivos compartidos del usuario.

Métodos:

- agregarArchivo: agrega un archivo al vector de archivos compartidos.
- eliminarArchivo: remueve un nombre de archivo del vector, el nombre es pasado por parámetro.

Debug: tiene la responsabilidad de escribir los mensajes en el archivo de depuración. Esta clase es un Singleton.

Atributos:

- instance: es un atributo de clase de tipo puntero a un objeto Debug. Contiene una única instancia de objeto debug.
- lock: es un objeto de tipo LockFile utilizado para sincronizar el acceso al archivo de debug.

Métodos:

- getInstance: devuelve la única instancia posible de esta clase. Si no existe la crea.
- destruir: elimina de memoria el objeto instancia creado.
- escribir: escribe un mensaje en el archivo de debug.

Gestor Descargas: tiene la responsabilidad de crear los procesos de descarga y envío de los archivos y realizar la transferencia de los mismos.

Métodos:

- iniciarRecepcion: crea el proceso que envia el archivo y establecer el canal de comunicación.
- enviar: lee el archivo y lo envía por el fifo.
- enviarRuta: envía una ruta del fifo de transferencia a través del fifo de escucha.
- descargar: lee el el archivo del fifo de transferencia y lo escribe en el archivo.

Signal Handler: tiene la función de administrar las señales que son manejadas por la aplicación. Esta clase es un Singleton.

Atributos:

- instance: es un atributo de clase de tipo puntero a un objeto SignalHandler. Contiene una

única instancia de objetos SignalHandler.

- signal_handlers: es un array de tipo puntero a EventHandler que contiene los manejadores de las disitintas señales.

Métodos:

- getInstance: devuelve la única instancia posible de esta clase. Si no existe la crea.
- destruir: elimina el objeto instancia creado.
- registrarHandler: almacena en el vector de manejadores un puntero al objeto pasado por parámetro y registra la función dispatcher que manejará dicha señal.
- dispatcher: método destinado a manejar todas las señales recibidas que luego transfiere el manejo al manejador correspondiente de la señal.
- removerHandler: saca del vector de manejadores el manejador específico de esa señal.

Even Handler: tiene la responsabilidad de definir una interfaz necesaria para implementar el manejo de una señal en particular.

Métodos:

- handleSignal: es un método virtual abstracto llamado cuando se desea manejar la señal, se deberá implementar en una clase derivada.

SIGINT_Handler: tiene la responsabilidad de manejar la señal SIGINT.

Atributos:

- gracefulQuit: es de tipo sig_atomic_t el cual permite que su modificación sea atómica cuando se está manejando una señal.

Métodos:

- handleSignal: es una implementación de handleSignal para manejar la señal SIGINT.

Fifo: tiene la responsabilidad de administrar un recurso FIFO del sistema operativo para facilitar su utilización.

Atributos:

- nombre: es un string conteniendo el nombre del fifo en el filesystem.
- fileDes: es un entero el cual contiene el descriptor del fifo cuando es abierto.

Métodos:

- escribir: envía los bytes de un puntero pasado por parámetro por el fifo.
- leer: lee del fifo una cantidad de bytes pasada por parámetro y lo coloca en el puntero recibido.
- cerrar: cierra el descriptor del fifo.
- abrir: crea el fifo si no existe.
- eliminar: cierra el fifo y lo elimina del filesystem.

LockFile: tiene la responsabilidad de administrar un recurso Lock del sistema operativo para facilitar su utilización.

Atributos:

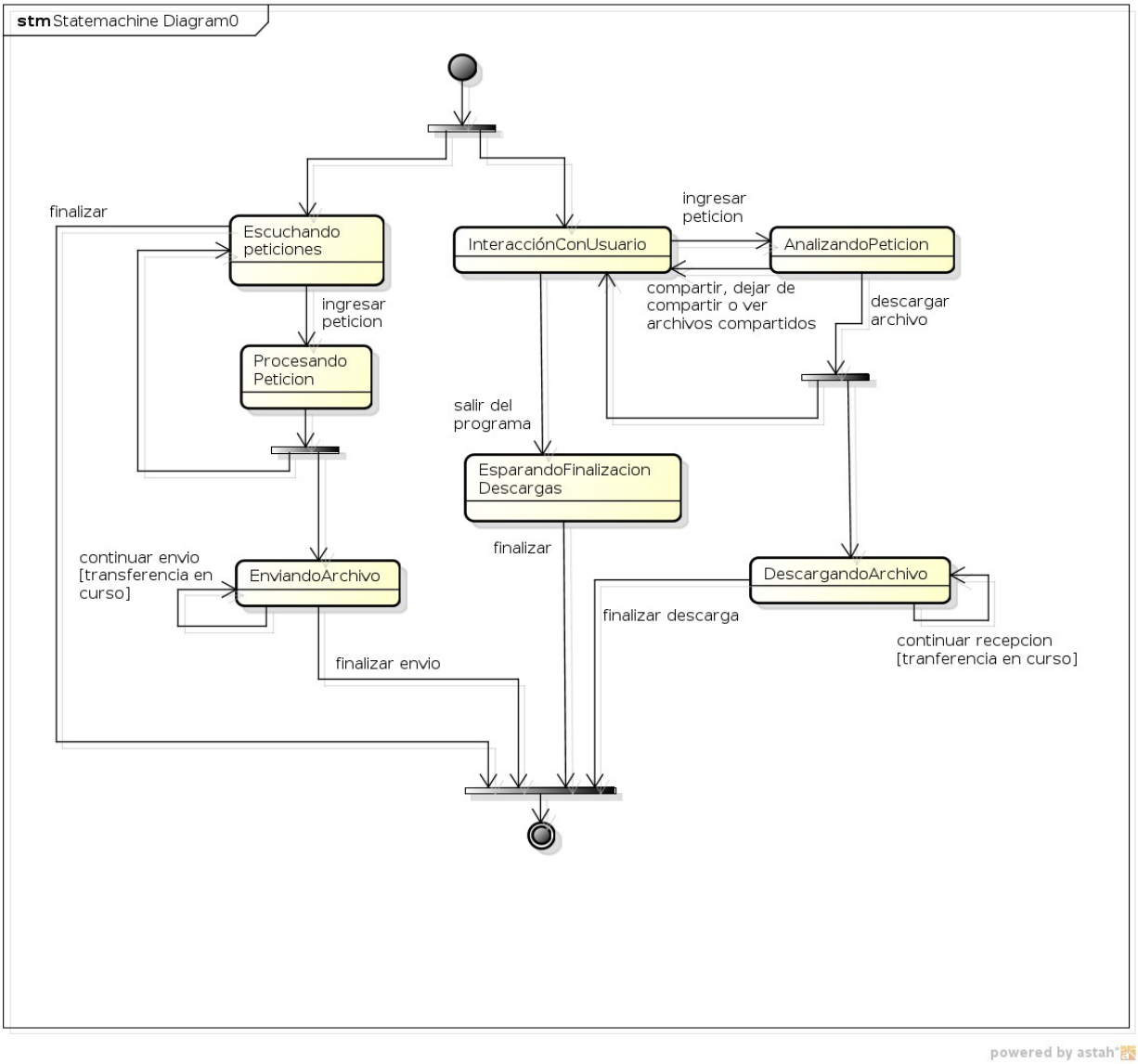
- fl: es una estructura de tipo flock la cual es necesaria para aplicar un lock al archivo.
- fd: es un entero el cual contiene el descriptor del archivo cuando es abierto.
- nombre: contiene la ruta al archivo que se desea utilizar para lockear.
- lectura: es un entero indicando la posición de lectura del archivo.

Métodos:

- tomarLock: aplica el lock de tipo escritura al archivo completo.
- liberarLock: libera el lock aplicado con el método anterior.
- escribir: agrega al final del archivo los datos pasados por parámetro.
- leer: lee desde la posición de lectura anterior una cantidad de bytes indicada y los coloca en un puntero pasado por parámetro.
- abrir: abre el archivo como lectura y escritura, si no existe lo crea.
- cerrar: cierra el descriptor del archivo.
- eliminar: cierra el archivo y lo elimina del filesystem.

6. Diagrama de estados:

A continuación se desarrolla el diagrama de estados del programa a nivel global:



6.1 Descripción de estados:

Interacción con usuario: el programa interactúa a través de la consola con el usuario recibiendo las peticiones del mismo.

Analizando Petición: el programa analiza la petición del usuario:

- Si es una petición que no involucra a otro usuario se procesa la misma y se vuelve al estado Interacción con usuario.
- Si es una petición de descarga de un archivo, se crea un nuevo proceso que pasa al estado Descarga archivo y el proceso creador vuelve al estado Interacción con usuario

Descargando archivo: el programa va leyendo de a fragmentos el archivo enviado a través del FIFO y lo va escribiendo en el archivo. Mientras no finalice el envio del archivo el proceso sigue en este estado.

Esperando finalización descargas: el proceso principal espera a que los demás procesos de transferencia finalicen.

Esperando petición: el proceso de escucha se queda leyendo el FIFO para recibir una petición de otro usuario.

Procesando petición: el proceso de escucha obtiene cual es el archivo que hay que enviar, e inicializa un nuevo proceso para que se encargue de enviar el archivo.

Enviando archivo: el programa va leyendo de a fragmentos el archivo y lo va escribiendo en el FIFO. Mientras no finalice el archivo el proceso sigue en este estado.

7. Instalación, compilación y modo de uso:

Para compilar/instalar la aplicación se provee un archivo de Makefile junto con el código fuente de la misma.

Los pasos a realizar para la compilación son los siguientes:

1. Copiar el contenido de la carpeta Trabajo Practico 1 del CD al directorio en que se desee ejecutar el programa.

2. Posicionarse en el directorio en el cuál fue copiado el programa.

3. Ejecutar el comando: ~$: make

Luego de estos pasos, verificar si la compilación fue exitosa. En caso de serlo, se contará con un ejecutable llamado **tp.**

Para ejecutar la aplicacion es necesario posicionarse en el directorio en cuestion y desde la linea de comandos ejecutar:

~$:./tp Este comando ejecutará la aplicación en modo normal

~$:./tp -d Este comando ejecutará la aplicación en modo debug, dejando un archivo de debug para analizar el funcionamiento de la misma.

~$:./tp -h Este comando ejecutrá la aplicación de forma tal de que se muestre la ayuda (help) de la misma; permitiendo al usuario observar la forma correcta de utilizarla.