Bibliotecas SOP

Generado por Doxygen 1.9.3

Capítulo 1

Bibliotecas SOP

BibliotecasSOP es un sistema de gestión bibliotecario para Solicitar, Renovar o Devolver libros. Se compone de un único Servidor al cual múltiples Clientes pueden conectarse y procesas solicitudes

1.1. Compilación

Ejecutables Para compilar el programa sólo hace falta utilizar el comando 'make' en la carpeta padre 'bibliotecassop'

```
make
```

Los ejecutables se encontrarán en la carpeta 'bin' y todos los archivos de prueba se copiarán a ese mismo directorio.

Documentación Para crear la documentación de Doxygen puede correr el siguiente comando desde la carpeta padre:

```
make docs
```

Los archivos HTML y LATEX se encuentran en la carpeta './docs/doxy', la página HTML puede verse abriendo el archivo 'Documentacion.html' dentro de la carpeta padre 'bibliotecassop'

Limpiar Para eliminar todos los archivos generados por make (incluye binarios y documentación), puede correr el siguiente comando desde la carpeta padre:

```
make clean
```

1.2. Uso

1.2.1. Servidor

El Servidor se encarga de leer y manipular la Base de Datos (BD) de los libros, los operaciones a realizar en la BD están dadas por las peticiones que hagan los Clientes al Servidor (véase ¿Cómo se envían información entre Cliente y Servidor?), debe crear el Servidor antes que cualquier Cliente de la siguiente manera:

Uso: ./server -p pipeServidor -f baseDeDatos

el flag -f se utiliza para específicar el archivo de texto donde se almacena la base de datos de todos los libros (véase Base de datos)

1.2.2. Cliente

El Cliente se encargará de recibir las peticiones a realizar y se las enviará al Servidor (véase Servidor).
Antes de que crear cualquier Cliente, debe haber un Servidor actualmente en ejecución y el nombre de su pipe (Cliente->Servidor) debe pasarse por parámetro al Cliente

Uso: ./client [-i Archivo] -p pipeServidor
[-i archivo] es opcional!

Las peticiones pueden realizarse mediante un archivo de texto con el flag -i, si no se utiliza este flag se mostrará un menú (véase Archivo de peticiones)

Sólo se puede tener un único servidor pero múltiples clientes conectados al mismo.

1.3. Archivos de texto

1.3.1. Base de datos

Este archivo será leído por el Servidor, un ejemplo puede encontrarse en (./archivo_prueba/BD.txt)

Formato:

Nombre del Libro,ISBN,ejemplares
NumeroDeEjemplar, estado,fecha(dd/mm/yy)

1.3.1.0.1. Estado Caractér que indica el estado

- D: Disponible
- P: Prestado

1.3.2. Archivo de peticiones

Este archivo será leído por el Cliente y es opcional, si no se utiliza archivo el cliente mostrará un menú (véase Cliente), un ejemplo de este archivo puede ser encontrado en (./archivo_prueba/PS.txt) y (./archivo_prueba/PS1.txt)

Formato:

peticion,Nombre del libro,ISBN

1.3.2.0.1. Peticion Caractér que indica el tipo de petición

- P: Prestar
- D: Devolver
- R: Renovar

1.4. ¿Cómo se envía información entre Cliente y Servidor?

1.4.1. Pipes

La comunicación entre Clientes y Servidor se da mediante pipes nominales de la librería POSIX

- El Servidor tiene un único pipe de lectura mediante el cual todos los clientes envían sus peticiones (Cliente-> Servidor), el nombre de este pipe se da mediante parámetros en la creación del Servidor
- Cada cliente tiene su propio pipe de lectura mediante el cual el Servidor envía información al Cliente (Servidor-> Cliente) y el nombre se asigna de acuerdo al PID del proceso cliente "pipeCliente_PID"

Según lo anterior, existen múltiples pipes (Servidor-> Cliente) pero sólo un pipe(Cliente-> Servidor) y será el creador del pipe el encargado de borrarlo al finalizar (véase Protocolo de comunicación)

1.4.2. Paquetes

Para evitar problemas en la escritura y lectura de información en el pipe, tanto Clientes como Servidor escriben y reciben datos de tipo <*data_t* > , esta estructura es lo único que se puede leer y escribir de los pipes y usualmente nos referimos a ella como 'paquete', este paquete contiene el PID del cliente quien manda la petición, un indicador del tipo de paquete (véase Tipo de Paquete), y una unión a la información del paquete

1.4.2.0.1. Tipo de paquete Existen tres tipos de paquetes que pueden ser enviados: SIGNAL, BOOK y ERR

SIGNAL Este tipo de paquete indica que se está enviando una señal (Usualmente el Servidor manda una señal al Cliente de que la operación fue exitosa o que el libro no existe) (data_t.data.signal)

Listado de señales: *Señales de peticiones:*

Señal	Codigo	Descripción
PET_ERROR	-3	Error de lectura de un archivo
SOLICITUD	3	Solicitud exitosa
RENOVACION	4	Renovación exitosa
DEVOLUCION	5	Devolución exitosa

Señales de confirmación de comunicación:

Señal	Codigo	Descripción
START_COM	1	Señal para empezar comunicación
STOP_COM	-1	Señal para detener confirmación
SUCCEED_COM	2	Señal de confirmación de comunicación
FAILED_COM	-2	Señal de fallo en la comunicación (TERMINACIÓN)

BOOK Este tipo de paquete contiene la información de un libro, usualmente el Cliente envía este tipo de paquete al Servidor para solicitar, renovar o devolver un libro, (data_t.data.libro)

Cada libro tiene un tipo de petición: SOLICITAR, RENOVAR, DEVOLVER y BUSCAR que el Servidor puede leer

ERR Este tipo de dato no está asociado a ninguna estructura, se usa para indicar un error genérico como respuesta

1.5. Protocolo de comunicación

- Sólo existe un pipe (Cliente->Servidor) por el cual todos los Cliente se comunican con el servidor, este pipe lo crea y destruye el Servidor
- Existe un pipe por cada cliente (Servidor->Cliente), este pipe lo crea y destruye el cliente dueño
- El servidor tiene una lista interna con los pid de todos los cliente actualmente conectados

1.5.1. Apertura de la comunicación

Cuando el Servidor inicia comunicación:

- Servidor debe iniciar comunicación antes que cualquier cliente y sólo lo hace una vez al ejecutarse

1. Servidor crea el pipe (Cliente->Servidor)
2. Servidor abre el pipe (Cliente->Servidor) para LECTURA

Cuando el Cliente inicia comunicación:

- Cualquier cliente tiene que iniciar comunicación después que el servidor, de no existir el pipe (Cliente->Servidor) el proceso finalizará
- el Cliente intenta establecer conexión con el servidor un número determinado de intentos [INTENTOS_↩ ESCRITURA] y esperará una respuesta del servidor durante [TIMEOUT_COMUNICACION] segundos

1. Cliente abre el pipe (Cliente->Servidor) para ESCRITURA
2. Cliente crea un pipe (Servidor->Cliente)
3. Cliente envía a Servidor el nombre del pipe (Servidor->Cliente) mediante una señal [START_COM]
4. Cliente abre el pipe (Servidor->Cliente) para LECTURA
5. Servidor abre el pipe (Servidor->Cliente) para ESCRITURA
6. Servidor guarda la información de Cliente con su respectivo pipe de comunicación
7. Servidor envía una señal de confirmación a Cliente
8. Cliente espera una señal de Servidor [SUCCEED_COM]

1.5.2. Cierre de la comunicación

Cuando el Cliente termina comunicación:

- Cuando un Cliente pierda la comunicación debe terminar el proceso Cliente
- el Cliente que quiera finalizar la comunicación debe eliminar el pipe (Servidor->Cliente) asociado
- Cuando no hayan Clientes conectados al Servidor, éste también debe finalizar

1. Cliente manda una petición de terminación de comunicación al Servidor
2. Servidor cierra la escritura del pipe (Servidor->Cliente)
3. Servidor actualiza la lista de clientes
4. Cliente espera a que se cierre el pipe
5. Cliente cierra la lectura del pipe (Servidor->Cliente)
6. Cliente elimina el pipe (Servidor->Cliente)
7. Cliente cierra la escritura del pipe (Cliente->Servidor)
8. El proceso Cliente finaliza

1.6. Diagrama de secuencia

Representación gráfica del proceso de comunicación (véase Protocolo de comunicación)

1.7. Lista de códigos de error

Si existe un error en la ejecución de alguno de los procesos o solicitudes, el programa puede retornar alguno de los siguientes códigos de error:

Errores genéricos:

Error	Codigo	Descripción
SUCCESS_GENERIC	0	Exitoso
FAILURE_GENERIC	-1	Falló
ERROR_FATAL	1	Error irrecuperable

Apertura de arhivos

Error	Codigo	Descripción
ERROR_APERTURA_ARCHIVO	2	Error en la apertura de un archivo
ERROR_CIERRE_ARCHIVO	3	Error en el cierre de un archivo

Error de pipes:

Error	Codigo	Descripción
ERROR_PIPE_SER_CTE	4	Error en el pipe (Servidor->Cliente)
ERROR_PIPE_CTE_SER	5	Error en el pipe (Cliente->Servidor)
ERROR_COMUNICACION	6	Error de comunicación

Lectura / Escritura:

Error	Codigo	Descripción
ERROR_LECTURA	7	Error de lectura de un archivo
ERROR_ESCRITURA	8	Error de escritura de un archivo

Otros errores:

Error	Codigo	Descripción
ERROR_ARG_NOVAL	9	Error en argumentos
ERROR_MEMORY	10	Error de alojamiento de memoria
ERROR_PID_NOT_EXIST	11	PID del cliente no existe
ERROR_SOLICITUD	12	Solicitud inválida

1.8. Créditos

Proyecto para la materia de Sistemas Operativos
Pontificia Universidad Javeriana, Facultad de Ingeniería 2021

- Ángel David Talero
- Juan Esteban Urquijo
- Humberto Rueda Cataño

Capítulo 2

Índice de estructura de datos

2.1. Estructura de datos

Lista de estructuras con una breve descripción:

Capítulo 3

Indice de archivos

3.1. Lista de archivos

Lista de todos los archivos documentados y con descripciones breves:

Capítulo 4

Documentación de las estructuras de datos

4.1. Referencia de la Estructura client_list

Arreglo dinámico de clientes conectados.

`#include <server.h>`

Campos de datos

- int nClients
- client_t * clientArray

4.1.1. Descripción detallada

Arreglo dinámico de clientes conectados.

4.1.2. Documentación de los campos

4.1.2.1. clientArray

`client_t* client_list::clientArray`

Arreglo de clientes conectados

4.1.2.2. nClients

```
int client_list::nClients
```

Número de clientes en el arreglo

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/server.h

4.2. Referencia de la Estructura client_t

Estructura con la información de un clinete.

```
#include <server.h>
```

Campos de datos

- int pipe
- pid_t clientPID
- char pipeNom [TAM_STRING]

4.2.1. Descripción detallada

Estructura con la información de un clinete.

4.2.2. Documentación de los campos

4.2.2.1. clientPID

```
pid_t client_t::clientPID
```

PID del cliente

4.2.2.2. pipe

```
int client_t::pipe
```

File descriptor del pipe (Servidor->Cliente) asociado

4.2.2.3. pipeNom

```
char client_t::pipeNom[TAM_STRING]
```

Nombre del pipe (Servidor->Cliente)

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/server.h

4.3. Referencia de la Estructura data_t

Dato que será enviado a través de los pipes, también conocido como Paquete o Mensaje.

```
#include <data.h>
```

Campos de datos

- pid_t client
- enum TYPE_T type
- union IN_DATA_T data

4.3.1. Descripción detallada

Dato que será enviado a través de los pipes, también conocido como Paquete o Mensaje.

4.3.2. Documentación de los campos

4.3.2.1. client

```
pid_t data_t::client
```

PID del cliente que envió o recibe el paquete

4.3.2.2. data

```
union IN_DATA_T data_t::data
```

Contenido del paquete

4.3.2.3. type

```
enum TYPE_T data_t::type
```

Tipo del paquete

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/data.h

4.4. Referencia de la Estructura ejemplar

Información de cada uno de los libros.

```
#include <libro.h>
```

Campos de datos

- enum PETICION petition
- int isbn
- int num_ejemplar
- char nombre [TAM_STRING]
- struct libro libroEjem

4.4.1. Descripción detallada

Información de cada uno de los libros.

4.4.2. Documentación de los campos

4.4.2.1. isbn

```
int ejemplar::isbn
```

ISBN del libro

4.4.2.2. libroEjem

```
struct libro ejemplar::libroEjem
```

Información adicional de ejemplares

4.4.2.3. nombre

```
char ejemplar::nombre[TAM_STRING]
```

Nombre del libro

4.4.2.4. num_ejemplar

```
int ejemplar::num_ejemplar
```

Cantidad de ejemplares

4.4.2.5. petition

```
enum PETICION ejemplar::petition
```

Tipo de petición que se solicita

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/libro.h

4.5. Referencia de la Unión IN_DATA_T

Datos del mensaje transmitido por data_t, puede ser de tipo señal o de tipo libro.

```
#include <data.h>
```

Campos de datos

- struct SIGNAL_T signal
- struct ejemplar libro

4.5.1. Descripción detallada

Datos del mensaje transmitido por data_t, puede ser de tipo señal o de tipo libro.

4.5.2. Documentación de los campos

4.5.2.1. libro

```
struct ejemplar IN_DATA_T::libro
```

Datos del libro

4.5.2.2. signal

```
struct SIGNAL_T IN_DATA_T::signal
```

Datos de la señal

La documentación para esta unión fue generada a partir del siguiente fichero:

- src/data.h

4.6. Referencia de la Estructura libro

Describe la información adicional de cada uno de los ejemplares que componen un libro.

```
#include <libro.h>
```

Campos de datos

- int numero
- char estado
- char fecha [TAM_STRING]

4.6.1. Descripción detallada

Describe la información adicional de cada uno de los ejemplares que componen un libro.

4.6.2. Documentación de los campos

4.6.2.1. estado

```
char libro::estado
```

Estado (D o P)

4.6.2.2. fecha

```
char libro::fecha[TAM_STRING]
```

Fecha de préstamo (dd-mm-YY)

4.6.2.3. numero

```
int libro::numero
```

Número de ejemplar

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/libro.h

4.7. Referencia de la Estructura peticion_t

Estructura en la cual se almacenan las diferentes peticiones que se leen por archivo en el proceso solicitante (Cliente)

```
#include <client.h>
```

Campos de datos

- char peticion
- char nombre [TAM_STRING]
- int isbn

4.7.1. Descripción detallada

Estructura en la cual se almacenan las diferentes peticiones que se leen por archivo en el proceso solicitante (Cliente)

4.7.2. Documentación de los campos

4.7.2.1. isbn

```
int peticion_t::isbn
```

ISBN del libro

4.7.2.2. nombre

```
char peticion_t::nombre[TAM_STRING]
```

Nombre del libro

4.7.2.3. peticion

```
char peticion_t::peticion
```

Tipo de peticion a realizar

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/client.h

4.8. Referencia de la Estructura SIGNAL_T

Estructura que compone una señal.

```
#include <data.h>
```

Campos de datos

- int code
- char buffer [TAM_STRING]

4.8.1. Descripción detallada

Estructura que compone una señal.

4.8.2. Documentación de los campos

4.8.2.1. buffer

```
char SIGNAL_T::buffer[TAM_STRING]
```

Buffer opcional

4.8.2.2. code

```
int SIGNAL_T::code
```

Código de la señal

La documentación para esta estructura fue generada a partir del siguiente fichero:

- src/data.h

Capítulo 5

Documentación de archivos

5.1. Referencia del Archivo src/client.h

Proceso solicitante.

```
#include <stdbool.h>
#include "data.h"
```

Estructuras de datos

- struct peticion_t

  Estructura en la cual se almacenan las diferentes peticiones que se leen por archivo en el proceso solicitante (Cliente)

defines

- #define PIPE_NOM_CTE "pipeCliente_"

Funciones

- void **mostrarUso** (void)

  Mostrar el uso correcto del ejecutable con sus flags.
- static bool manejarArgumentos (int argc, char ∗argv[ ], char ∗pipeNom, char ∗fileNom)

  Verificar y manipular los argumentos recibidos.
- static void iniciarComunicacion (const char ∗pipeCTE_SER, char ∗pipeSER_CTE, int ∗pipe)

  Iniciar la comunicación con el servidor Descripción del proceso en README.md.
- static void detenerComunicacion (int ∗pipe, char ∗pipeSER_CTE)

  Detener la comunicación con el servidor.
- data_t generarSenal (pid_t src, int code, char ∗buffer)

  Generar un paquete de tipo Señal.

- int prestarLibro (int ∗pipes, const char ∗nombreLibro, int ISBN)

  Función que se encarga de pedir prestado un libro al servidor.
- int devolverLibro (int ∗pipes, const char ∗nombreLibro, int ISBN, int ejemplar)

  Función que se encarga de pedir devolver un libro al servidor.
- int renovarLibro (int ∗pipes, const char ∗nombreLibro, int ISBN, int ejemplar)

  Función que se encarga de pedir renovar un libro al servidor.
- struct ejemplar buscarLibro (int ∗pipes, const char ∗nombre, int ISBN)

  Pedirle al servidor la información de un libro en específico.

5.1.1. Descripción detallada

Proceso solicitante.

Autores

Ángel David Talero Juan Esteban Urquijo Humberto Rueda Cataño

Copyright

2021 Pontificia Universidad Javeriana Facultad de Ingeniería Bogotá D.C - Colombia

5.1.2. Documentación de los 'defines'

5.1.2.1. PIPE_NOM_CTE

```
#define PIPE_NOM_CTE "pipeCliente_"
```

Nombre con el cual crear los pipes de cliente

5.1.3. Documentación de las funciones

5.1.3.1. buscarLibro()

```
struct ejemplar buscarLibro (
            int * pipes,
            const char * nombre,
            int ISBN )
```

Pedirle al servidor la información de un libro en específico.

Parámetros

pipes	Pipes de comunicación
nombre	Nombre del libro
ISBN	ISBN del libro

Devuelve

struct ejemplar Estructura que contiene al libro, en caso de error la petición del libro es BUSCAR, cualquier otro tipo de petición es éxito

5.1.3.2. detenerComunicacion()

```
static void detenerComunicacion (
             int * pipe,
             char * pipeSER_CTE )  [static]
```

Detener la comunicación con el servidor.

Parámetros

pipe	Pipes de lectura/escritura
pipeSER_CTE	Nombre del pipe creado por el Cliente

5.1.3.3. devolverLibro()

```
int devolverLibro (
             int * pipes,
             const char * nombreLibro,
             int ISBN,
             int ejemplar )
```

Función que se encarga de pedir devolver un libro al servidor.

Parámetros

pipes	Arreglo con los pipes
nombreLibro	Nombre del libro
ISBN	ISBN del libro
ejemplar	Número de ejemplar

Devuelve

int Código de error o SUCCESS_GENERIC (0) si éxito

5.1.3.4. generarSenal()

```
data_t generarSenal (
            pid_t src,
            int code,
            char * buffer )
```

Generar un paquete de tipo Señal.

Parámetros

src	PID del cliente quien envía
code	Código de la señal
buffer	Buffer [OPCIONAL], NULL si no se necesita

Devuelve

data_t Nuevo paquete con la señal

5.1.3.5. iniciarComunicacion()

```
static void iniciarComunicacion (
            const char * pipeCTE_SER,
            char * pipeSER_CTE,
            int * pipe )  [static]
```

Iniciar la comunicación con el servidor Descripción del proceso en README.md.

Parámetros

pipeCTE_SER	Nombre del pipe (Cliente->Servidor)
pipeSER_CTE	RETORNA: nombre escogido para el pipe (Servidor->Cliente)
pipe	pipe[0] tiene el pipe del pipe (Cliente -> Servidor) y pipe[1] tiene el pipe del pipe (Servidor -> Cliente) Use las macros WRITE y READ con pipe EJ: para escribir en el pipe se utilizar pipe[WRITE]

5.1.3.6. manejarArgumentos()

```
static bool manejarArgumentos (
            int argc,
            char * argv[],
            char * pipeNom,
            char * fileNom )  [static]
```

Verificar y manipular los argumentos recibidos.

Parámetros

argc	Numero de argumentos
argv	Vector con los argumentos
pipeNom	RETORNA: nombre del pipe
fileNom	RETORNA: nombre del archivo

Devuelve

true Se utilizó un archivo

false No se utilizó un archivo, por lo tanto ignorar el contenido de fileNom

5.1.3.7. prestarLibro()

```
int prestarLibro (
            int * pipes,
            const char * nombreLibro,
            int ISBN )
```

Función que se encarga de pedir prestado un libro al servidor.

Parámetros

pipes	Arreglo con los pipes
nombreLibro	Nombre del libro
ISBN	ISBN del libro

Devuelve

int Código de error o SUCCESS_GENERIC (0) si éxito

5.1.3.8. renovarLibro()

```
int renovarLibro (
            int * pipes,
            const char * nombreLibro,
            int ISBN,
            int ejemplar )
```

Función que se encarga de pedir renovar un libro al servidor.

Parámetros

pipes	Arreglo con los pipes
nombreLibro	Nombre del libro
ISBN	ISBN del libro
ejemplar	Número de ejemplar

Devuelve

int Código de error o SUCCESS_GENERIC (0) si éxito

5.2. Referencia del Archivo src/common.h

Macros y Typedefs compartidos entre Cliente y Servidor.

```
#include <stdint.h>
```

defines

- #define TAM_STRING 100
- #define WEEK_SEC 604800
- #define PERMISOS_PIPE S_IRWXU
- #define INTENTOS_ESCRITURA 5
- #define TIMEOUT_COMUNICACION 10
- #define WRITE 0
- #define READ 1
- #define PET_ERROR-3
- #define SOLICITUD 3
- #define RENOVACION 4
- #define DEVOLUCION 5
- #define START_COM 1
- #define STOP_COM-1
- #define SUCCEED_COM 2
- #define FAILED_COM-2
- #define SUCCESS_GENERIC 0
- #define FAILURE_GENERIC-1

- #define ERROR_FATAL 1
- #define ERROR_APERTURA_ARCHIVO 2
- #define ERROR_CIERRE_ARCHIVO 3
- #define ERROR_PIPE_SER_CTE 4
- #define ERROR_PIPE_CTE_SER 5
- #define ERROR_COMUNICACION 6
- #define ERROR_LECTURA 7
- #define ERROR_ESCRITURA 8
- #define ERROR_ARG_NOVAL 9
- #define ERROR_MEMORY 10
- #define ERROR_PID_NOT_EXIST 11
- #define ERROR_SOLICITUD 12

5.2.1. Descripción detallada

Macros y Typedefs compartidos entre Cliente y Servidor.

Autores

Ángel David Talero Juan Esteban Urquijo Humberto Rueda Cataño

Copyright

5.2.2. Documentación de los 'defines'

5.2.2.1. DEVOLUCION

```
#define DEVOLUCION 5
```

Devolución exitosa

5.2.2.2. ERROR_APERTURA_ARCHIVO

```
#define ERROR_APERTURA_ARCHIVO 2
```

Error en la apertura de un archivo

5.2.2.3. ERROR_ARG_NOVAL

```
#define ERROR_ARG_NOVAL 9
```

Error en argumentos

5.2.2.4. ERROR_CIERRE_ARCHIVO

```
#define ERROR_CIERRE_ARCHIVO 3
```

Error en el cierre de un archivo Error de pipes

5.2.2.5. ERROR_COMUNICACION

```
#define ERROR_COMUNICACION 6
```

Error de comunicación Lectura / Escritura

5.2.2.6. ERROR_ESCRITURA

```
#define ERROR_ESCRITURA 8
```

Error de escritura de un archivo Otros errores

5.2.2.7. ERROR_FATAL

```
#define ERROR_FATAL 1
```

Error irrecuperable Apertura de arhivos

5.2.2.8. ERROR_LECTURA

```
#define ERROR_LECTURA 7
```

Error de lectura de un archivo

5.2.2.9. ERROR_MEMORY

```
#define ERROR_MEMORY 10
```

Error de alojamiento de memoria

5.2.2.10. ERROR_PID_NOT_EXIST

```
#define ERROR_PID_NOT_EXIST 11
```

PID del cliente no existe

5.2.2.11. ERROR_PIPE_CTE_SER

```
#define ERROR_PIPE_CTE_SER 5
```

Error en el pipe (Cliente->Servidor)

5.2.2.12. ERROR_PIPE_SER_CTE

```
#define ERROR_PIPE_SER_CTE 4
```

Error en el pipe (Servidor->Cliente)

5.2.2.13. ERROR_SOLICITUD

```
#define ERROR_SOLICITUD 12
```

Solicitud inválida **COMMON_H**

5.2.2.14. FAILED_COM

```
#define FAILED_COM`2
```

Señal de fallo en la comunicación (TERMINACION)

5.2.2.15. FAILURE_GENERIC

```
#define FAILURE_GENERIC`1
```

Falló

5.2.2.16. INTENTOS_ESCRITURA

```
#define INTENTOS_ESCRITURA 5
```

Intentos de escritura en caso de falla

5.2.2.17. PERMISOS_PIPE

```
#define PERMISOS_PIPE S_IRWXU
```

Permiso para Leer, Escribir, Ejecutar

5.2.2.18. PET_ERROR

```
#define PET_ERROR`3
```

Error de petición

5.2.2.19. READ

```
#define READ 1
```

Definiciones para el vector fd

5.2.2.20. RENOVACION

```
#define RENOVACION 4
```

Renovación exitosa

5.2.2.21. SOLICITUD

```
#define SOLICITUD 3
```

Solicitud exitosa

5.2.2.22. START_COM

```
#define START_COM 1
```

Señal para empezar comunicación

5.2.2.23. STOP_COM

```
#define STOP_COM`1
```

Señal para detener confirmación

5.2.2.24. SUCCEED_COM

```
#define SUCCEED_COM 2
```

Señal de confirmación de comunicación

5.2.2.25. SUCCESS_GENERIC

```
#define SUCCESS_GENERIC 0
```

< Errores genéricos Exitoso

5.2.2.26. TAM_STRING

```
#define TAM_STRING 100
```

Tamaño de los Strings

5.2.2.27. TIMEOUT_COMUNICACION

```
#define TIMEOUT_COMUNICACION 10
```

Tiempo límite (s) para establecer comunicación

5.2.2.28. WEEK_SEC

```
#define WEEK_SEC 604800
```

Una semana en segundos

5.2.2.29. WRITE

```
#define WRITE 0
```

Definiciones para el vector fd

5.3. Referencia del Archivo src/data.h

Estructuras que se movilizan a través del pipe.

```
#include <sys/types.h>
#include "common.h"
#include "libro.h"
```

Estructuras de datos

- struct SIGNAL_T

  Estructura que compone una señal.
- union IN_DATA_T

  Datos del mensaje transmitido por data_t, puede ser de tipo señal o de tipo libro.
- struct data_t

  Dato que será enviado a través de los pipes, también conocido como Paquete o Mensaje.

Enumeraciones

- enum TYPE_T { SIGNAL , BOOK , ERR }

  Tipos de datos que pueden ser enviados con el paquete.

5.3.1. Descripción detallada

Estructuras que se movilizan a través del pipe.

Autores

Ángel David Talero, Juan Esteban Urquijo, Humberto Rueda Cataño,

Copyright

5.3.2. Documentación de las enumeraciones

5.3.2.1. TYPE_T

```
enum TYPE_T
```

Tipos de datos que pueden ser enviados con el paquete.

Valores de enumeraciones

SIGNAL	asociado struct SIGNAL_T
BOOK	asociado struct ejemplar
ERR	NO TIENE TIPO DE DATO ASOCIADO (Sólo señalar errores)

5.4. Referencia del Archivo src/libro.h

Estrucuras que representan los libros.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include "common.h"
```

Estructuras de datos

- struct libro

  Describe la información adicional de cada uno de los ejemplares que componen un libro.
- struct ejemplar

  Información de cada uno de los libros.

defines

- #define MAX_CANT_LIBROS 100

Enumeraciones

- enum PETICION { **SOLICITAR** , **RENOVAR** , **DEVOLVER** , **BUSCAR** }

  Describe los diferentes tipos de peticiones para libros que puede enviar el Cliente al Servidor.

Variables

- struct ejemplar **ejemplar** [MAX_CANT_LIBROS]

5.4.1. Descripción detallada

Estrucuras que representan los libros.

Autores

Ángel David Talero Juan Esteban Urquijo Humberto Rueda Cataño

Copyright

5.4.2. Documentación de los 'defines'

5.4.2.1. MAX_CANT_LIBROS

```
#define MAX_CANT_LIBROS 100
```

Máxima cantidad de libros en el arreglo

5.5. Referencia del Archivo src/server.h

Proceso receptor de peticiones.

```
#include <stdbool.h>
#include "common.h"
#include "data.h"
```

Estructuras de datos

- struct client_t

  Estructura con la información de un clinete.
- struct client_list

  Arreglo dinámico de clientes conectados.

Funciones

- void **mostrarUso** (void)

  Mostrar el uso correcto del ejecutable con sus flags.
- static void manejarArgumentos (int argc, char ∗argv[ ], char ∗pipeNom, char ∗fileIn, char ∗fileOut)

  Verificar y manipular los argumentos recibidos.
- static int iniciarComunicacion (const char ∗pipeCTE_SER)

  Iniciar la comunicación, permitir a los cliente conectarse Descripción del proceso en README.md.
- int conectarCliente (struct client_list ∗clients, data_t package)

  Conectar un cliente a la lista.
- int retirarCliente (struct client_list ∗clients, data_t package)

  Desconectar un cliente de la lista.
- int interpretarSenal (struct client_list ∗clients, data_t package)

  Interpretar una señal.
- data_t generarRespuesta (pid_t dest, int code, char ∗buffer)

  Generar una señal como respuesta a un Cliente.
- client_t crearCliente (int pipefd, pid_t clientpid, char ∗pipenom)

  Crear un Cliente.

- int guardarCliente (struct client_list *clients, client_t client)

  Guardar un cliente en el arreglo.
- int removerCliente (struct client_list *clients, pid_t clientToRemove)

  Remover un cliente dado su pid.
- int buscarCliente (struct client_list *clients, pid_t client)

  Buscar un cliente dado su PID.
- int manejarLibros (struct client_list *clients, data_t package, struct ejemplar ejemplar[ ])

  Manejar una solicitud de libro.

5.5.1. Descripción detallada

Proceso receptor de peticiones.

Autores

Ángel David Talero Juan Esteban Urquijo Humberto Rueda Cataño

Copyright

2021 Pontificia Universidad Javeriana Facultad de Ingeniería Bogotá D.C - Colombia

5.5.2. Documentación de las funciones

5.5.2.1. buscarCliente()

```
int buscarCliente (
            struct client_list * clients,
            pid_t client )
```

Buscar un cliente dado su PID.

Parámetros

client_list	Lista con los clientes
client	PID del cliente

Devuelve

int Retorna el FD del pipe (-1 si no existe)

5.5.2.2. conectarCliente()

```
int conectarCliente (
            struct client_list * clients,
            data_t package )
```

Conectar un cliente a la lista.

Parámetros

clients	Apuntador a la lista de clientes
package	Copia del paquete recibido por el pipe

Devuelve

int Exit error code or SUCCESS_GENERIC

5.5.2.3. crearCliente()

```
client_t crearCliente (
            int pipefd,
            pid_t clientpid,
            char * pipenom )
```

Crear un Cliente.

Parámetros

pipefd	File Descriptor del pipe (Servidor->Cliente)
clientpid	PID del cliente
pipenom	Nombre del pipe del cliente

Devuelve

client_t Estructura con el cliente

5.5.2.4. generarRespuesta()

```
data_t generarRespuesta (
            pid_t dest,
            int code,
            char * buffer )
```

Generar una señal como respuesta a un Cliente.

Parámetros

dest	PID del cliente destino
code	Código de la señal
buffer	Buffer [OPCIONAL], NULL de no necesitarse

Devuelve

data_t Nuevo paquete a enviar

5.5.2.5. guardarCliente()

```
int guardarCliente (
            struct client_list * clients,
            client_t client )
```

Guardar un cliente en el arreglo.

Parámetros

clients	Apuntador a arreglo de clientes
client	Cliente a guardar

Devuelve

SUCCESS_GENERIC si éxito, cualquier otro valor de lo contrario

5.5.2.6. iniciarComunicacion()

```
static int iniciarComunicacion (
            const char * pipeCTE_SER )  [static]
```

Iniciar la comunicación, permitir a los cliente conectarse Descripción del proceso en README.md.

Parámetros

pipeCTE_SER	Nombre del pipe (Cliente->Servidor)

Devuelve

fd del Pipe (Cliente-Servidor)

5.5.2.7. interpretarSenal()

```
int interpretarSenal (
            struct client_list * clients,
            data_t package )
```

Interpretar una señal.

Parámetros

package	Paquete con la señal

Devuelve

int (-1) si hay algún error

5.5.2.8. manejarArgumentos()

```
static void manejarArgumentos (
            int argc,
            char * argv[],
            char * pipeNom,
            char * fileIn,
            char * fileOut )  [static]
```

Verificar y manipular los argumentos recibidos.

Parámetros

argc	Numero de argumentos
argv	Vector con los argumentos
pipeNom	RETORNA: nombre del pipe
fileIn	RETORNA: Nombre del archivo de entrada
fileOut	RETORNA: Nombre del archivo de salida

5.5.2.9. manejarLibros()

```
int manejarLibros (
            struct client_list * clients,
            data_t package,
            struct ejemplar ejemplar[] )
```

Manejar una solicitud de libro.

Parámetros

clients	Lista de los clientes
package	Paquete recibido
ejemplar	Arreglo con los libros de la BD

Devuelve

SUCCESS_GENERIC si éxito, cualquier otro valor de lo contrario

5.5.2.10. removerCliente()

```
int removerCliente (
            struct client_list * clients,
            pid_t clientToRemove )
```

Remover un cliente dado su pid.

Parámetros

clients	Apuntador a arreglo de clientes
clientToRemove	Cliente a guardar

Devuelve

SUCCESS_GENERIC si éxito, cualquier otro valor de lo contrario

5.5.2.11. retirarCliente()

```
int retirarCliente (
            struct client_list * clients,
            data_t package )
```

Desconectar un cliente de la lista.

Parámetros

clients	Apuntador a la lista de clientes
package	Copia del paquete recibido por el pipe

Devuelve

int Exit error code or SUCCESS_GENERIC