Apellido y Nombres	Legajo	# de Hojas

Parte práctica. Enunciado.

Se necesita implementar un sistema compuesto por tres archivos ejecutables diferentes que interactuarán entre sí por medio de diferentes mecanismos de intercomunicación de procesos vistos durante el curso. Se representa en la Fig.1.

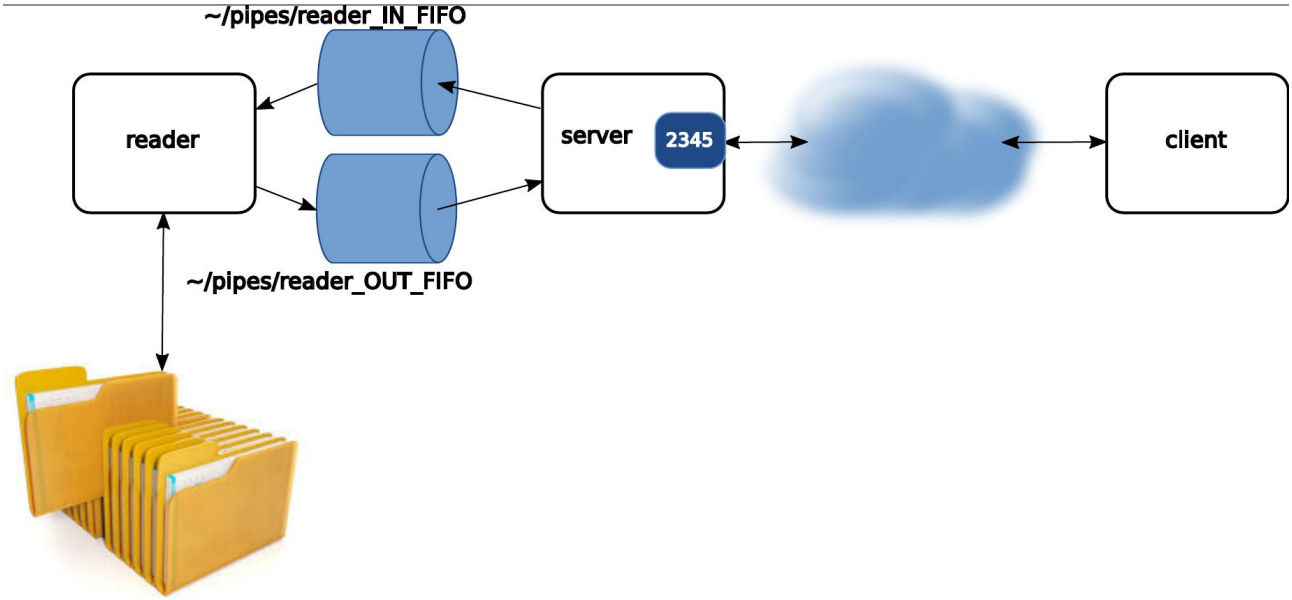

Fig.1 Diagrama del sistema a desarrollar

El Sistema deberá procesar los datos contenidos en un archivo, correspondientes a registro de valores de una distribución de campo eléctrico en el espacio.
Para ello se dispone de archivos con esta información compuestos de valores en punto flotante de precisión simple y un encabezado en texto que indica el ancho y largo del plano de distribución así como su altura

reader:
Lee un *Named FIFO*, ubicado en **~/pipes/reader_IN_FIFO**, por donde recibe la ruta completa del archivo de datos que debe leer. Abre el archivo desde donde carga los datos del archivo en una estructura como la que se indica a continuación y luego transmite en ASCII los valores por el *Named FIFO* ubicado en **~/pipes/reader_OUT_FIFO**. Finalizada la escritura vuelve a leer *Named FIFO*.

```
struct efd
{
        char * unidades;
        int x; // cantidad de puntos de ancho de la distribución en el plano
        int y; // cantidad de puntos de largo de la distribución en el plano
        int z; // cantidad de puntos de alto de la distribución en el espacio
        float * values; // Total de valores de la distribución ( x * y * z)
};
```

Apellido y Nombres	Legajo	# de Hojas

NOTA: Los cuatro primeros miembros de la estructura funcionan como una suerte de encabezamiento del archivo. En el archivo el campo unidades es de ancho fijo: 32 bytes. Su codificación es ASCII. Los bytes del final si no se emplean para gruargar información deben estar en 00h. Luego los tres enteros *x*, *y*, y *z* están en formato de entero (4 bytes c/u little endian). Seguidamente el vector de números de punto flotante cuya cantidad es *x * y * z.*

server:
Escucha pedidos por el port TCP 2345. Por cada pedido crea un hijo, y vuelve a escuchar por el socket. El proceso hijo lee por el socket duplicado el nombre del archivo a procesar. Lo envía por el *Named FIFO* ubicado en **~/pipes/readerd_IN_FIFO**, y espera por el otro *Named FIFO* (**~/pipes/readerd_OUT_FIFO**). A medida que le llegan los datos por el pipe, los retransmite por el socket.
No deben quedar procesos zombies.
client:
Inicia una conexión con el server. Envía el nombre del archivo a requerir cuando el server conecta. Recibe los datos y los presenta en pantalla a razón de 8 números por línea separados por tabuladores. Los datos se presentan con 5 dígitos decimales y 2 enteros.

Se pide:
1. Escribir los archivos fuente de cada uno de los tres módulos acorde con las condiciones especificadas mas abajo.
2. Completar el archivo ***Makefile***, provisto con el examen
3. Utilizar el archivo de cabecera ***2doParcial.h*** provisto por la cátedra **sin modificaciones** a lo existente. Si necesita agregar alguna otra definición Usar los campos delimitados por los comentarios

Condiciones:
Cada uno de los tres procesos tiene dos archivos fuentes como mínimo. Uno de ellos contiene solamente la función main (). En el server, el código del proceso hijo lleva un archivo por separado.
Usar la biblioteca de sockets provista por la cátedra.

Apellido y Nombres	Legajo	# de Hojas

Teoría

1. La estructura del problema teórico ***en una máquina de 64 bits*** ocupa en memoria:

- [ ] a. 16 bytes
- [ ] b. 24 bytes
- [ ] c. 28 bytes
- [ ] d. 8 bytes
- [ ] e. Ninguna de las anteriores

2. Utilizando la misma estructura del punto práctico la recibimos en una función como argumento por referencia en la variable **r**. Responder cuales de las siguientes afirmaciones son correctas (puede haber mas de una)

- [ ] a. La cantidad de floats apuntados por values se calcula con **r.x * r.y * r.z**
- [ ] b. El tercer float del arreglo se accede mediante **r.values[3]**
- [ ] c. Para apuntar al siguiente float del arreglo basta con: **(r->values) ++**
- [ ] d. Los paréntesis son redundantes en el punto c.
- [ ] e. Para mostrar las unidades: **printf ("%s\n", *r->unidades);**
- [ ] f. **printf ("%f",*r->values + *(*r->values + 1));** dá la suma del valor actual de float mas el siguiente.

3. Cual será el resutado de la función open () , sobre el Named FIFO **~/pipes/reader_IN_FIFO** del problema práctico cuando la ejecute *server* y cuando la ejecute *reader*. Justificar.

4. ¿Como se resuelve la espera de varios procesos bloqueantes (Ej: leer un socket y un FIFO), sin perder información de ninguno de los dos?

5. Señalar cuales afirmaciones son correctas (puede haber mas de una)

- [ ] a. kill -9 termina un proceso zombie.
- [ ] b. Cuando un proceso padre termina, sus hijos tienen como padre a **init**.
- [ ] c. Un proceso recibe varias señales SIGCHLD, pero solo ejecuta una vez el handler
- [ ] d. Escribir **while (1);** , equivale a **while (1) sleep(1);**
- [ ] e. Si un proceso termina sus hijos zombie también lo hacen.

Apellido y Nombres	Legajo	# de Hojas