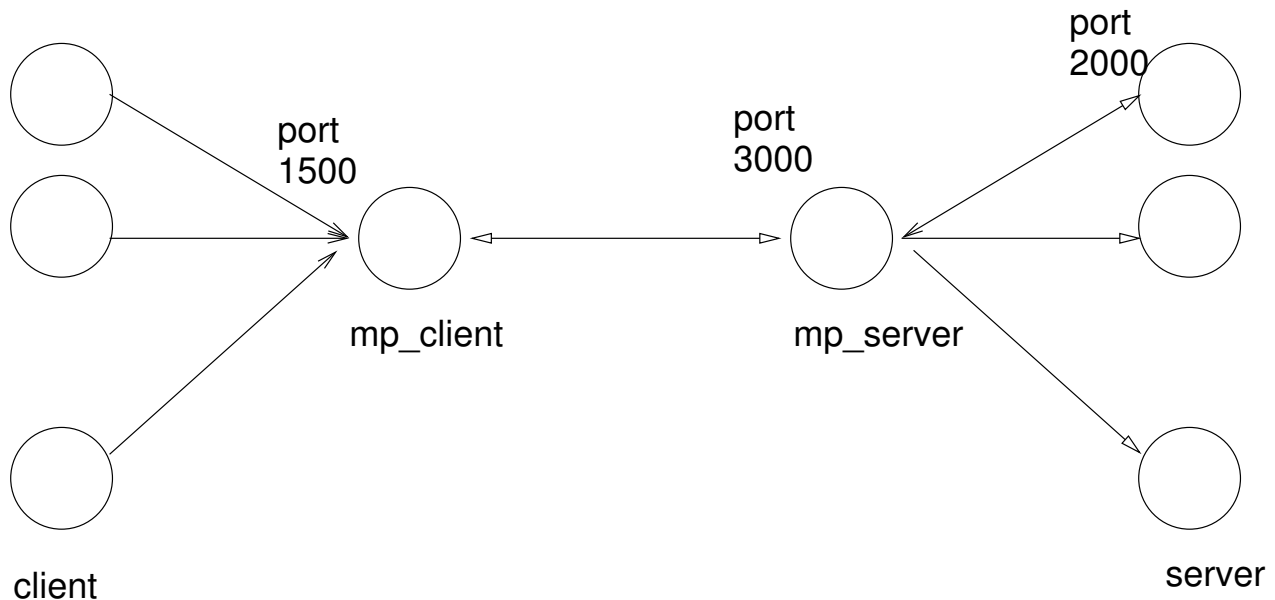


1. Objetivos

Esta tarea persigue construir un Multiplexor, parecido a un servidor proxy, que se intercale entre el cliente y el servidor de la tarea1 y permita acceder a servidores en Intranets, por ejemplo. El medio de comunicación es sin errores, pero la API para el Framing layer será diferente para permitir manejar paquetes en el multiplexor. El cliente y el servidor anteriores no deberían ser modificados, salvo para pasarle el port de conexión al cliente.

El multiplexor también se divide en un cliente y un servidor, el servidor está provisto y está funcionando en dichato:



Ustedes deben programar y entregar el `mp_client` completo. Entre los clientes y el `mp_client` así como entre el `mp_server` y todos los servidores no hay framing y nunca habrá errores, ahí solo van los datos en sockets directamente.

Entre ambos multiplexores utilizaremos un nivel de encapsulamiento o framing en paquetes para identificar a qué conexión corresponde cada uno, pero también para recuperarnos de errores en tareas posteriores.

2. Framing

Esta capa deberá ser provista por el alumno. La misión principal es encapsular los datos: comienzo de paquete (SOH: 'a'), fin de paquete (EOP: 'e') y “escapar” los caracteres de control dentro de los datos (ESC: '\'). Los paquetes llevan un campo largo de un byte que cuenta los bytes de datos que vienen.

```
SOH size data..... EOP
```

El multiplexor lee una secuencia de bytes desde un cliente o servidor y debe enviarlos en un paquete a través del socket compartido agregándole un header inicial que indique a qué conexión pertenece. Al revés, cuando recibe un paquete desde el socket compartido debe sacarle el header y enviar los datos recibidos hacia el cliente o servidor respectivo. Para esto, se requiere una API que será la siguiente:

- `Fput_packet(int fd, char *buf, int size)`
Envía los `size` bytes de `buf`, encapsulándolos en un solo paquete. El tamaño máximo de `size` es 255. El `fd` es el socket entre los multiplexores.
- `size = Fget_packet(int fd, char *buf)`
Lee el próximo paquete y retorna su tamaño. El `fd` es el socket entre los multiplexores.

Cuando aparece un SOH, EOP o ESC en los datos o en el byte de tamaño, debemos escapearlo. El largo no cuenta los caracteres de escape, solo los bytes reales de datos.

3. Multiplexión (`mp_client`)

El `mp_client` se conecta con el `mp_server` y ese socket será el de multiplexión. Luego espera conexiones de los clientes. El `mp_client` actúa como un servidor hacia el lado de los clientes, y todo lo que recibe debe encapsularlo en el socket multiplexor, en un frame (comenzando con SOH, escapando los caracteres y terminando con EOP) de la siguiente forma:

- Conexión de cliente: Al recibir una nueva conexión, el multiplexor debe informar en el socket para que se cree la conexión y el servidor respectivo al otro lado. Para eso envía un paquete de control con 2 bytes:

```
C id
```

Donde `id` es un byte (entre 0 y 255) que identifica en forma única esta conexión (`mp_client` debe asignarlo).

- Término de conexión: Al detectar el término de una conexión (EOF) debemos informar al otro extremo que esa conexión se cerró:

```
S id
```

- Paquete de Datos: Cada paquete de datos entre cliente y servidor para la conexión id, va en el multiplexor como:

```
D id data ... ..
```

El multiplexor puede implementarse con threads o con select, pero no con fork y procesos pesados, puesto que el socket multiplexor debe compartirse.

4. Ejecución y Debugging

En `dichato.dcc.uchile.cl` hay un servidor en el puerto 3000 ejecutando este protocolo (por ahora, supongan que no hay errores de ningún tipo) y que se conecta con un servidor de la tarea1 que escucha en el puerto 2000.

Pueden usar los clientes de la tarea1 y deben construir el `mp_client` en esta tarea. Les incluyo un `mp_client` de prueba.

Si van a probar todo el sistema en un solo PC, les dejo los ejecutables en la página del curso.