Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-322 — Sistemas Operativos I

Trabajo Práctico

Sistema P2P Simple para Compartir Archivos en LAN

Duración Estimada: 15-20 horas.

§1. Introducción

Los sistemas Peer-to-Peer (P2P), o de igual a igual, representan una arquitectura de red descentralizada donde cada participante, conocido como nodo o par, actúa simultáneamente como cliente y como servidor. A diferencia de los modelos cliente-servidor tradicionales, en los que la comunicación fluye a través de un servidor central, los sistemas P2P permiten la comunicación directa entre los nodos, facilitando el intercambio de recursos y servicios sin la necesidad de una entidad intermedia. Cada nodo puede ofrecer archivos para compartir y, al mismo tiempo, descargar archivos de otros nodos. La comunicación entre ellos implica el registro y descubrimiento de otros pares, la consulta de disponibilidad de archivos, y la transferencia directa de los mismos. Los sistemas P2P logran la descarga simultánea de múltiples partes desde diferentes fuentes dividiendo el archivo en chunks, gestionando el descubrimiento de los nodos que poseen esos chunks y estableciendo conexiones paralelas para la transferencia.

§2. Objetivo General

El objetivo de este trabajo práctico es desarrollar un sistema P2P básico diseñado para el intercambio de archivos en una red de área local (LAN). El enfoque principal reside en la implementación de la concurrencia dentro de cada nodo y en la comunicación distribuida entre ellos. Para ello, el proyecto se estructura en etapas progresivas, comenzando con la configuración de un nodo base concurrente que incluye un servidor de conexiones entrantes y un cliente para conexiones salientes. Posteriormente, se avanza hacia el descubrimiento de nodos en la red mediante mensajes UDP , la implementación de un protocolo de búsqueda de archivos distribuida , y finalmente, la funcionalidad de transferencia de archivos y la robustez del sistema, incluyendo consideraciones para el manejo de archivos grandes.

La elección del uso de C o Erlang para la implementación queda a criterio del estudiante.

§2.1. Descripción del Sistema

Cada "Nodo P2P"será una instancia de un programa que actúa como un par en la red. Los nodos pueden ofrecer archivos para compartir y descargar archivos de otros nodos.

Archivos Compartidos Cada nodo tendrá una carpeta local designada como "compartida". Los archivos ubicados en esta carpeta serán los que el nodo ofrece a la red.

Archivos Descargados Habrá una carpeta local designada como "descargas"donde se guardarán los archivos obtenidos de otros nodos.

Identificación Cada Nodo P2P tendrá un ID único (ej., nodo_A, nodo_B).

Archivo de Nodos descubiertos (por ejemplo: nodes_registry.json): Un archivo que mantiene un registro de los IDs de los nodos y sus direcciones IP/puertos.

Comunicación P2P : Los nodos deben poder:

- Requerir un identificador único en la red (una vez elegido no debería cambiar).
- Anunciarse en la red a través de mecanismos de HELLO.
- Consultar la disponibilidad de un archivo específico en la red.
- Ofrecer sus propios archivos cuando se les consulta.
- Transferir archivos entre pares (descarga).

Interfaz de Usuario (CLI) : Una interfaz de línea de comandos básica para interactuar con el nodo local.

§2.2. Partes del Trabajo Práctico

Parte 1 (primer semana): Nodo Base Concurrente (5-7 horas)

Parte 2: Descubrimiento de Nodos y Búsqueda de Archivos (5-7 horas)

Parte 3: Descarga de Archivos y Robustez (3-6 horas)

Parte 4: Presentación final (3-5 horas)

§3. Parte 1

En esta parte, el foco está en la estructura interna de un solo nodo P2P, manejando múltiples tareas concurrentemente. El uso de C o Erlang queda a criterio de los estudiantes.

§3.1. Requerimientos

§3.1.1. Cada nodo debe:

- Iniciar con un ID único en la red. La garantía de unicidad debe ser obtenida de forma consensuada entre los pares de la red a través de algún mecanismo adecuado[1].
- Escanear la carpeta compartida al inicio para construir una lista de archivos compartidos (ej., solo nombres de archivo).
- Mantener una lista de nodos conocidos en la red (inicialmente vacía).
  - Al iniciar, el nodo debe intentar obtener su identificador único.
  - Un archivo podría mantener un mapeo de `{ node\_id}` a `{"ip":"..", "port":..}`. Como es local el formato y nombre del archivo es responsabilidad de cada nodo.

§3.1.2. Servidor de Conexiones Entrantes:

Implementar un servidor TCP que escuche en el puerto de red 12345[2]:

Cada vez que una conexión entrante (de otro nodo P2P) es aceptada, debe ser manejada concurrentemente.

- En C: Usar de epoll para manejar múltiples conexiones.

[1] Los estudiantes decidirán/diseñarán en conjunto el protocolo a usar y quedará formalizado en el repositorio `https://github.com/Sistemas-Operativos-1-DCC/Apunte-colaborativo-2025`

[2] en realidad puede usarse cualquier puerto por encima de 1024. El puerto podría ser un parámetro de entrada al iniciar el nodo.

- En Erlang: El proceso principal que maneja el socket de escucha (gen_tcp:listen) sigue siendo el encargado de aceptar nuevas conexiones (gen_tcp:accept). Cada conexión aceptada (Socket) se pasa a un nuevo proceso ligero (spawn) dedicado a manejar la comunicación con ese cliente específico. Erlang maneja intrínsecamente la eficiencia de esta concurrencia sin necesidad de epoll explícito.

El servidor debe ser capaz de manejar múltiples conexiones simultáneas.

§3.1.3. Cliente para Conexiones Salientes

El nodo debe poder iniciar conexiones TCP a otros nodos (como cliente) para enviar solicitudes (ej. buscar archivos, iniciar una descarga). Estas conexiones pueden ser efímeras o persistentes según el diseño.

§3.1.4. Gestión de Datos y Sincronización Interna

Proteger el acceso a la lista de archivos compartidos y la lista de nodos conocidos para garantizar la exclusión mutua.

§3.1.5. Interfaz de Línea de Comandos (CLI)

Un hilo (C) / proceso (Erlang) dedicado para la CLI, debe permitir comandos básicos como: `id_nodo`, `listar_mis_archivos`, `salir`.

Esta CLI debe interactuar de forma segura con los componentes internos del nodo.

§4. Parte 2

Se implementa la comunicación entre nodos para el descubrimiento de la red y la consulta de archivos.

§4.1. Mecanismo de Descubrimiento de Nodos (UDP Broadcast/Multicast)

Cada Nodo P2P debe poder descubrir otros nodos activas en la LAN. Para ello se puede utilizar UDP Broadcast (o Multicast, si se prefiere) para que los nodos envíen periódicamente un mensaje de "presencia"con el siguiente formato:

```
HELLO <nodo_ID_X> <puerto_tcp>\n
```

y escuchen las respuestas de otros. Tener en cuenta que `<nodo_ID_X>` debe ser el identificador único que identifica al nodo en la red y `<puerto_tcp>` corresponde con el puerto en que dicho está esperando conexiones TCP.

Los nodos descubiertos se añaden a la lista de nodos conocidos, gestionada de forma concurrente.

Implementar la concurrencia en la recepción y procesamiento de estos mensajes UDP.

§4.2. Protocolo de Búsqueda de Archivos

Las solicitudes de búsqueda por red se realizaran usando el siguiente mensaje

```
SEARCH_REQUEST  <nodo_ID_X> <nombre_archivo_con_wildcards>\n
```

Las respuestas de búsqueda tendrán el formato:

```
SEARCH_RESPONSE <nodo_ID_X> <nombre_archivo> <tamaño en bytes>\n
```

§4.3. Funcionalidad de Búsqueda Distribuida (buscar <patron_nombre_archivo>)

Implementar un comando buscar en la CLI. Al ejecutar la búsqueda, el nodo local debe:

- Consultar su propia lista de archivos compartidos.
- Enviar la solicitud de búsqueda a todos los nodos conocidos en paralelo (consultar a cada uno por TCP).
- Recopilar las respuestas de todos los nodos y presentar los resultados consolidados al usuario.
  - En C: Utilizar hilos separados para cada conexión de búsqueda saliente. Necesitará un mecanismo de sincronización (ej., variables de condición y un contador de respuestas esperadas) para que el hilo de la CLI espere los resultados o un timeout.
  - En Erlang: Lanzar procesos para cada consulta a un nodo remoto. Recopilar las respuestas mediante receive con after timeout en el proceso de la CLI o en un proceso coordinador, aprovechando la naturaleza asíncrona de los mensajes.

§5. Parte 3

Se implementará la funcionalidad central de transferencia de archivos y se considera la robustez del sistema.

§5.1. Descarga de Archivos (descargar < nombre_archivo> <nodo_origen_id>)

- Implementar un comando descargar en la CLI.
- El nodo que inicia la descarga debe establecer una nueva conexión TCP con el nodo origen (cuya IP y puerto debe haber obtenido previamente de la búsqueda o de su lista de nodos conocidos) para solicitar el archivo. El mensaje para solicitar la descarga será:

  ```
  DOWNLOAD_REQUEST  <nombre_archivo>\n
  ```

- El nodo origen si tiene el archivo (en su hilo/proceso manejador de cliente) debe leer el archivo del disco y enviarlo a través del socket. Para hacerlo primero indicará que ha encontrado el archivo con un código que significa OK. Luego de lo cúal indicará el tamaño del archivo en binario en un entero de 32 bits en formato *big-endian*.

  Por ejemplo, para enviar un archivo de tamaño 3145728 , el mensaje tiene la forma:

101	0	48	0	0	*(bytes del archivo)*

  Donde 101 es el código de `OK`; los 4 bytes siguientes representan la longitud de la clave ($0 \times 256^3 + 48 \times 256^2 + 0 \times 256^1 + 0 \times 256^0 = 3145728$); y luego viene el archivo en sí.

  Nota: Pueden usar `ntohl` y `htonl` para convertir desde/hacia big-endian.

- Si el nodo origen no tiene el archivo disponible responderá NOTFOUND a través de su código de error 112.
- Si el archivo está disponible el nodo que descarga debe recibir el archivo y guardarlo en su downloads_folder.
- La descarga debe ser concurrente: el nodo que descarga debe poder seguir interactuando con la CLI mientras el archivo se transfiere.
- Consideraciones: Para manejar archivos grandes se transferiran en chunks.

Si el archivo es mayor a 4MB. El envío hará transfiendo chunks(también en binario).

Primero se envía una sola vez al inicio de la transferencia, si el archivo existe el siguiente mensaje:

`101 <tamaño_total_archivo_bytes> <tamaño_chunk_estandar_bytes>`

Donde:

`<tamaño_total_archivo_bytes>`: Tamaño del archivo completo en bytes (entero de 32 bits en big-endian, como se indicó más arriba.

`<tamaño_chunk_estandar_bytes>`El tamaño fijo que el servidor está usando para sus chunks (por ejemplo, 1048576 para 1MB).

Luego se envían repetidamente:

`111 <indice_chunk> <tamaño_chunk_actual_bytes><datos_binarios_del_chunk>`

Donde:

- 111 (CHUNK) indicar que es un chunk de datos.
- `<indice_chunk>`: El número secuencial del chunk (0 para el primer chunk, 1 para el segundo, etc.). Este valor sería un entero de 2 bytes (16 bits) ya que 4096 chunks caben en 16 bits.
- `<tamaño_chunk_actual_bytes>`: El tamaño real de este chunk específico en bytes. El último chunk del archivo puede ser menor que el `tamaño_chunk_estandar_bytes` de los metadatos. Este valor sería un entero de 2 bytes.

§5.2. Gestión de Errores y Robustez Básica

Tener en cuenta:

- Manejo de desconexiones de nodos (ej. si el nodo origen se desconecta durante una descarga).
- Manejo de archivos no encontrados o no disponibles para descarga en el nodo origen.
- Implementar timeouts para las operaciones de red para evitar bloqueos indefinidos.
  - En C: Manejo explícito de errores de socket y lectura/escritura de archivos.
  - En Erlang: El modelo de "let it crashz los supervisores pueden usarse para reiniciar procesos de manejo de conexiones que fallen. Los links y monitors entre procesos pueden detectar desconexiones.

§5.3. Mejoras (Opcional ahora / Obligatorias para las mesas flotantes o Diciembre)

- Remoción de nodos inactivos: Implementar un mecanismo para detectar y eliminar nodos que ya no responden a los mensajes "HELLO"(ej. tras varios timeouts).
- Verificación de Integridad: Calcular y enviar un checksum/hash del archivo para que el receptor pueda verificar la integridad de la descarga.
- Implementar descargas de múltiples fuentes:
  - Modificar el protocolo de búsqueda: Las respuestas de búsqueda (SEARCH_RESPONSE) deberían incluir no solo si el nodo tiene el archivo, sino también (opcionalmente, en un sistema más avanzado) qué partes o chunks de ese archivo tiene.
  - Gestionar el estado de la descarga: El nodo que descarga necesitaría un mapa de los chunks que ya tiene y los que le faltan.

- Múltiples conexiones salientes: Cuando se inicie una descarga, el nodo cliente establecería múltiples conexiones TCP con diferentes nodos que tengan el archivo (o partes de él).
- Coordinación de chunks: El cliente asignaría diferentes chunks a cada conexión activa, solicitando y recibiendo partes en paralelo.
- Reensamblaje: Una vez recibidas todas las partes, el nodo las concatenaría en el orden correcto.

§6. Parte 4

Redacción de la documentación e informe final.

§7. Entregables

A través de un repositorio git:

- Código Fuente: Todo el código del proyecto, bien organizado y comentado. con su Makefile adecuado. Deberá ser posible compilar y ejecutar el nodo en al menos dos máquinas diferentes dentro de la misma LAN para demostrar la capacidad P2P.
- Documentación (README.md) que incluya:
  - Instrucciones de Uso: Cómo compilar, configurar (IDs, puertos, carpetas) y ejecutar el sistema.
  - Comandos CLI: Lista y descripción de todos los comandos implementados.

Por plataforma:

- Link al repo.
- Informe técnico:
  - Diseño de Concurrencia: Descripción detallada del modelo de concurrencia utilizado (hilos/mutexes/cond.vars en C vs. procesos/paso de mensajes en Erlang) y justificación de las decisiones clave. ¿Cómo se asegura la exclusión mutua y la sincronización interna? Discusión sobre posibles deadlocks (C) o bloqueos (Erlang) y cómo se evitaron.
  - Robustez: Descripción de los errores manejados y las estrategias de recuperación (ej., timeouts, detección de desconexiones).
  - Comparación de Paradigmas (Fundamental): Un análisis crítico y comparativo de cómo el lenguaje elegido (C o Erlang) simplificó o complejizó ciertas partes del diseño de concurrencia y distribución, en contraste con el otro lenguaje. ¿Cuáles son las ventajas y desventajas que encontraron para este tipo de aplicación con cada paradigma?

§8. Aclaraciones de la forma de trabajo y evaluación

Los grupos de trabajo se fusionarán y formarán equipos de hasta cuatro integrantes. Para organizar mejor sus tareas los equipos pueden usar trello[3] La entrega final de hará por plataforma y a través de un repositorio git privado al que cada miembro del equipo tenga acceso y al cúal le darán acceso a los miembros de la cátedra. El repositorio deberá permanecer privado hasta agosto del 2026.

Vamos a definir una fecha de entrega en la que los equipos se reunirán en el laboratorio y pondrán a prueba sus sistemas de manera simultánea.

[3] `https://trello.com/`

Si algún programa presentara fallas se podrá corregir en vivo para hacerlo funcionar. Si el programa presentara errores graves o insalvabless, entonces la entrega de dicho tp quedará para ser presentado en alguna mesa de examen. Si el programa cumple los requerimientos y no contiene errores los miembros del equipo tendrán aprobado el tp final.

Pasada la fecha de entrega inicial, para quienes no lo hayan entregado u aprobado. La entrega del tp será requisito para el examen final pero a condición de ser entregado por no más de 2 personas. La entrega del mismo se considerará parte de la instancia de examen.