

Filesystems, IPCs y Servidores Concurrentes

Lezica, Santiago. (Leg. 49147).
Ballesty, Pablo Andrés. (Leg. 49359).
Pose, Jimena Belén. (Leg. 49015).

Índice

1. Introducción	3
1.1. Objetivo	3
1.2. Enunciado	3
1.3. Actividades	4
2. Desarrollo	6
2.1. Modelado del problema	6
2.2. Implementación de IPCs	6
2.2.1. Fifos	6
2.2.2. Memoria compartida y semáforos	7
2.2.3. Colas de mensajes	7

1. Introducción

1.1. Objetivo

El objetivo de este trabajo es familiarizarse con el uso de sistemas cliente-servidor concurrentes, implementando el servidor mediante la creación de procesos hijos utilizando `fork()` y mediante la creación de threads. Al mismo tiempo, ejercitar el uso de los distintos tipos de primitivas de sincronización y comunicación de procesos (IPC) y manejar con autoridad el filesystem de Linux desde el lado usuario.

1.2. Enunciado

Se desea implementar un simulador de colonia de hormigas simplificado. En el mismo habrá un hormiguero, varias hormigas y comida esparcida a lo largo del mundo. Las hormigas deberán recolectar la comida y traerla al hormiguero. Para ello deberán leer la información del ambiente y comunicarse entre ellas de manera eficiente. El objetivo de la simulación es traer la mayor cantidad de comida al hormiguero. Después de 10000 turnos, o cuando ya no haya más comida en el mundo, la simulación finaliza.

El servidor leerá del archivo de configuración la información acerca del mundo, que tendrá forma de grilla. Particularmente estará la ubicación del hormiguero, cada pieza de comida, su tipo (simple o grande) y las hormigas en su posición inicial. A continuación, empezará la simulación, en donde, por turnos simultáneos, todas las hormigas deberán realizar una acción. Esta acción podría ser o bien:

1. Moverse a un casillero contiguo horizontal o vertical.
2. Oler los casilleros vecinos para detectar rastros, hormigas o comida.
3. Levantar una pieza de comida que esté en un casillero contiguo horizontal o vertical.
4. Moverse a un casillero vecino dejando un rastro.
5. Emitir un grito.

Dos hormigas no podrán ocupar el mismo casillero y 1 hormiga no podrá ocupar el mismo casillero que una pieza de comida sin levantar. En caso de que al finalizar un turno, 2 o más hormigas intenten moverse al mismo casillero, solo una lo logrará y la otra fallará su movimiento. La única excepción a esta regla es el hormiguero. Pueden haber infinitas hormigas en el casillero del hormiguero. Las hormigas pueden dejar y detectar un rastro. Este rastro es un valor decimal entre 0 y 1, en donde 1 es un rastro recién puesto y 0 es "no hay rastro en absoluto". Al avanzar y dejar rastro, el valor de rastro dejado SIEMPRE será de valor 1. y por cada turno el rastro decrementará en 0.01.

Las hormigas SIEMPRE saben la orientación del hormiguero relativa a donde están paradas, no así la distancia. (Es decir, una hormiga puede preguntar, sin invertir turnos en ello, hacia donde está el hormiguero y recibir como respuesta (N,S,E,W,NE,NW,SE,SW)).

Las hormigas tienen una memoria muy limitada y solo pueden recordar 2 posiciones en el mapa, una de ellas siendo siempre el hormiguero. Es decir, una hormiga puede decidir almacenar una posición del tablero para luego preguntarse en que dirección está.

Las hormigas tienen una memoria muy limitada y solo pueden recordar 2 posiciones en el mapa, una de ellas siendo siempre el hormiguero. Es decir, una hormiga puede decidir almacenar una posición del tablero para luego preguntarse en que dirección está. Cuando una hormiga grita, todas las hormigas reciben la posición de la hormiga que grita y la distancia hamiltoniana entre ellas. Al escuchar un grito, una hormiga puede optar por reemplazar su memoria por la posición de la hormiga que gritó.

Existen 2 tipos de comida: chica y grande. La comida chica puede ser transportada por una hormiga sin dificultad y tiene valor 1. La hormiga simplemente tiene que posicionarse en un casillero contiguo y utilizar un turno para levantar la comida. La comida grande vale 5 puntos, puede ser transportada por una hormiga, pero necesita de 2 hormigas para ser levantada, es decir: debe haber 2 hormigas posicionadas contigua a la comida y ambas deben utilizar un turno para levantar o asistir en levantar la comida.

1.3. Actividades

Implemente la simulación utilizando procesos y threads y haga cuatro versiones del sistema, usando las siguientes primitivas de IPC:

1.
 - Pipes o fifos.
 - Colas de mensajes - System V o POSIX.
 - Memoria compartida o mmap(), Semáforos System V o POSIX.
 - Sockets - TCP o de dominio Unix.
2. El archivo de configuración tendrá el siguiente formato:
 - Una línea con la longitud y alto del tablero separados por coma. Ej: 6,8 significa un tablero de 6 columnas y 8 filas.
 - Una línea con la posición del hormiguero separada por coma, teniendo en cuenta que la posición superior izquierda es 0,0. Ej: 3,4 significa que el hormiguero está en la cuarta columna, quinta fila.
 - Una línea con la cantidad de hormigas N. Todas las hormigas empiezan en el hormiguero.
 - Una línea con la cantidad de comida chica M, seguida de M líneas con la posición de cada comida.
 - Una línea con la cantidad de comida grande K, seguida de K líneas con la posición de cada comida.
3. El programa debe permitir la visualización de la simulación en un tablero en la pantalla, iterando turno a turno.

4. Todas las hormigas deben tener la misma programación, es decir, si bien lo que hacen dependerá de su posición y pueden elegir moverse a un lugar al azar, debe haber una única función con la programación para todas las hormigas individuales.
5. La nota del trabajo práctico no dependerá de la eficiencia de la simulación, sin embargo, la cátedra proveerá 3 escenarios de prueba y se otorgará un punto extra al equipo que maximice puntajes en estos 3 escenarios.

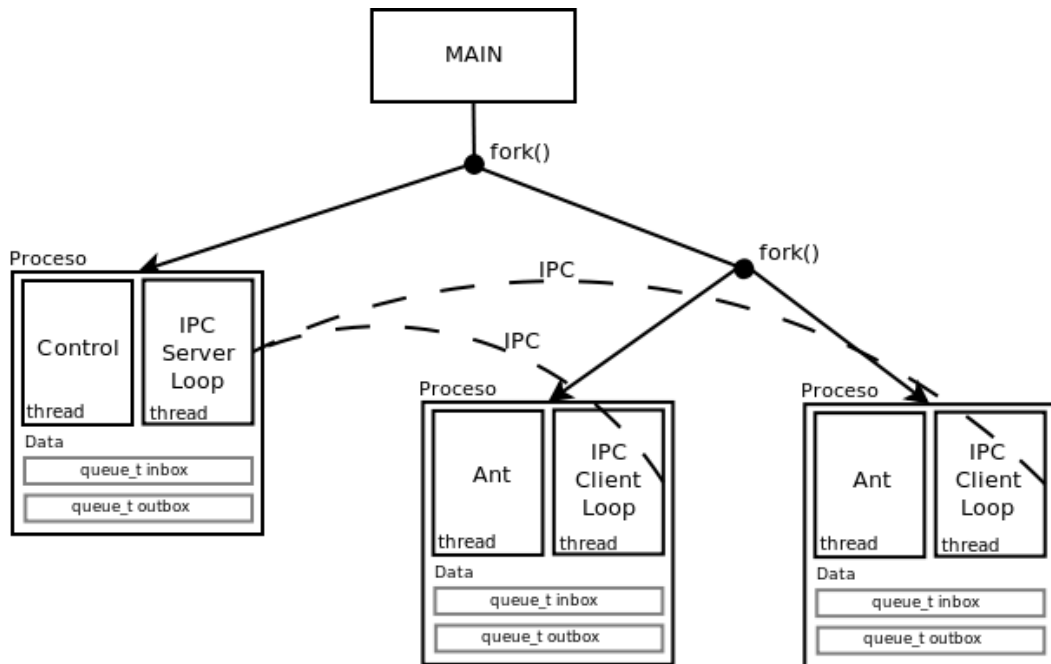
2. Desarrollo

2.1. Modelado del problema

Para el desarrollo de la simulación de hormigas decidimos modelar a cada hormiga como un proceso, el cual dispara un *thread* que se encarga de la comunicación de esta hormiga con el *server*. De la misma manera, el *server* es un proceso que dispara un *thread* que se encarga de la comunicación de éste con las hormigas de este mini-universo.

Los *threads* aprovechando que comparten los datos con el proceso que los dispara, se encargarán de recibir y enviar constantemente los mensajes entrantes y salientes, a través de los distintos sistemas de **IPC**, estos mensajes entrantes y salientes serán debidamente encolados o desencolados de las colas *inbox* y *outbox* que posee cada proceso en ejecución.

A continuación se coloca una ilustración de un server conectado con dos hormigas:



2.2. Implementación de IPCs

2.2.1. Fifos

Para entablar una comunicación entre procesos mediante el sistema de IPC implementado con **FIFOS**, se tomaron las siguientes decisiones:

- Al iniciar el proceso *server* creará en el directorio “/tmp”, todas las fifos necesarias para entablar comunicación con las n hormigas participantes.
- Los nombres de cada fifo corresponden a la siguiente regla: La hormiga con id lógico n enviará mensajes a través de la fifo “/tmp/fifo_c_w_n” y recibirá mensajes a través de la fifo “/tmp/fifo_c_r_n”. De esta manera entabla una comunicación bidireccional con el *server*.

- Se asume que en el directorio “/tmp” no hay fifos creadas con los mismos nombres, el programa se asegura crear y eliminar todas las fifos utilizadas.

2.2.2. Memoria compartida y semáforos

Para la implementación del IPC con **Memoria compartida** se utilizó la propuesta de **System V** para allocar memoria compartida, y 2 semáforos binarios o locks. Se crearon dos buffers circulares, en uno el proceso *server* escribe los mensajes para los clientes y en el otro lee los mensajes que le envían. En cada acceso a cada buffer, se utiliza un lock para evitar problemas de sincronización. A su vez cada cliente, lee del buffer donde el server escribe, y escribe en el buffer de donde lee el server, siempre lockeando en los accesos. De esta manera se logra una comunicación bidireccional entre un proceso cliente y un proceso server.

2.2.3. Colas de mensajes

Para la versión del sistema que utiliza **Colas de mensajes** se decidió utilizar **System V** para poder tener una única cola y manejar los mensajes con prioridades. De esta manera se mantiene siempre una prioridad para cada proceso, siendo esta su sid (simulation id). En el caso del servidor, el sid siempre es 1 (Debido a que si se intentan levantar mensajes con prioridad 0 de la cola, esta devuelve el de menor prioridad, es decir que se toma como prioridad por defecto), y para las hormigas es $n + 1$.

Cuando se inicia el servidor se dispara un thread que entra en un loop infinito levantando mensajes con la prioridad del servidor e insertándolos en la cola con el número de prioridad a quien está dirigido el mensaje (indicado en la estructura *st_msg_t*), a no ser que esté dirigido a él, en ese caso lo inserta en la *inbox* del servidor.

Por otro lado, al iniciarse cada cliente se dispara un nuevo thread, que también entra en un loop infinito, levanta mensajes de la cola con la prioridad del cliente y los inserta en su *inbox*. Este mismo thread a la vez agarra los mensajes de la *outbox* del cliente y los inserta en la cola de mensajes con la prioridad del servidor.

Por último, si se intenta levantar un mensaje de la cola con una prioridad determinada y la cola no tiene ningún mensaje con esta prioridad se decidió que el proceso no se bloquee hasta encontrar dicho mensaje, si no que siga pidiendo mensajes hasta que haya uno, para que el resto de los procesos puedan seguir enviando y recibiendo mensajes normalmente.

Gracias a la propuesta de **System V** se logró utilizar una única cola de mensajes para comunicar varios procesos.