Práctica de Sistemas Operativos 2019 – Proyecto Final

FECHA DE ENTREGA: JUEVES 9 DE MAYO (HORA LÍMITE DE ENTREGA,23:00 HORAS).

Introducción

En este ejercicio se debe construir un simulador estelar. Para reducir la complejidad de la práctica, se van a entregar ficheros donde ya están definidos diferentes aspectos de la práctica, tales como: número de equipos, número de naves, la estructura del mapa del simulador. Además, se entregan fuentes que implementan algunos aspectos del simulador. Es muy importante conocer todo el material contenido en estos ficheros antes de abordar cualquier implementación.

El proceso inicial (ejecutado por el usuario) es el encargado de gestionar los recursos compartidos, coordinar a los diferentes componentes involucrados en la simulación y marcar la llegada de un nuevo turno. Este proceso principal ("simulador") generará por cada equipo, un proceso hijo "jefe" que será encargado de coordinar y dar ordenes a sus naves. Cada nave será un proceso hijo de un proceso jefe y tendrá que ejecutar las ordenes enviadas por éste dentro de cada turno.

Se deberá entregar en un fichero zip o tar.gz, además del código, y un documento que incluya un diagrama de la solución junto con un texto explicativo que describa los elementos del diagrama y aspectos relevantes de la solución implementada. Este diseño hará hincapié en los componentes y elementos vistos en la asignatura (procesos o threads y jerarquías, los mecanismos de comunicación y sincronización, etc). El documento deberá incluir los nombres de los miembros del grupo.

Estructura

La práctica se estructura en 1 proceso "simulador", n procesos "jefe" (tantos como equipos), m procesos "nave". Además, existirá un proceso monitor, que mostrará el desarrollo de la simulación y que tendrá que ser ejecutado desde un terminal aparte.

A continuación, se detalla los principales mecanismos de comunicación entre los diferentes procesos. Pueden ser necesarios mecanismos adicionales de comunicación y sincronización tales como señales, semáforos, etc. **El alumno es**

libre de añadir los mecanismos que considere necesarios además de los aquí especificados.

- La comunicación comandos entre el proceso "simulador" y los procesos "jefe" se realizará mediante tuberías.
- La comunicación de ordenes entre cada proceso "jefe" y los procesos "nave" correspondientes se realizará mediante tuberías.
- La comunicación de acciones entre los procesos nave y el proceso simulador se realizará mediante cola de mensajes.
- Todo el estado y mapa de la simulación se mantendrá en memoria compartida para que otros procesos que lo necesiten puedan hacer uso de esta información.
- El turno se implementará mediante una señal de alarma en el proceso "simulador", de tal forma que éste pueda estar procesando acciones hasta la llegada de un nuevo turno.

Consideraciones sobre la implementación

1. En el terminal donde se ejecuta "simulador" y, por consiguiente, todos sus procesos hijo, sebe mostrarse información (printf) de todo lo que va ocurriendo con el siguiente formato de salida (en función del proceso):
    - "Simulador: Mensaje"
    - "Jefe E: Mensaje"
        - E es el número de equipo
    - "Nave E/N: Mensaje"
        - E es el número de equipo
        - N el número de nave
    - "ACCION ATAQUE [Nave] Oy,Ox -> Ty,Tx: …"
        - 'Nave' es el símbolo del equipo + el número de nave de la nave (A1, B0,..)
        - Oy y Ox son las coordenadas origen de la acción
        - Ty y Tx son las coordenadas objetivo de la acción

    Se incluye un fichero ejemplo_salida.txt a modo orientativo de lo que se pide.
2. No pueden quedar procesos huérfanos al finalizar la simulación
3. Incluir un delay (puede ser aproximadamente un usleep(100000)) entre el procesamiento de cada acción recibida por el proceso simulador, mejora mucho la visualización de la simulación.
4. Leed bien los ficheros fuente aportados (sobre todo los .h). Hay mucha funcionalidad ya implementada. Si queréis podéis añadir nuevas funciones a los ficheros existentes.
5. La visualización está hecha con ncurses. Toda la funcionalidad necesaria ya está incluida en los ficheros fuente que os damos. Aún así, si alguien quiere

añadir elementos a la visualización puede consultar alguno de estos pequeños tutoriales:
    a. http://www.cs.ukzn.ac.za/~hughm/os/notes/ncurses.html
    b. https://github.com/fundamelon/terminal-game-tutorial
    c. http://www6.uniovi.es/cscene/CS3/CS3-08.html
6. En el enunciado sólo se describe la simulación requerida. Por ejemplo sólo hay 2 acciones por turno, sólo hay 2 tipos de acción (mover o atacar) y 2 tipos de órdenes (MOVER_ALEATORIO y ATACAR), pero por supuesto hay más que se podrían añadir (RETIRADA, ATACAR_OBJETIVO tipo todos a por uno, etc) o acciones como paralizar (siguiente turno sin hacer nada), cegar (dejar sin visión de enemigo), etc, o aumentar el número de acciones por turno.
7. Localizar naves enemigas es una función que es necesaria para poder hacer un ataque decente.

Fuentes aportados

Como parte del material se entregan una serie de fuentes que implementan una importante cantidad de funciones necesarias en la práctica y que es **MUY IMPORTANTE** conocer.

- simulador.h: Constantes y tipos básicos para la simulación
- gamescreen.h: Funciones básicas para gestionar la pantalla (basado en ncurses)
- mapa.h: Funciones para obtener o fijar información del mapa y sus casillas

Proceso “simulador”

Es el proceso principal de la práctica, que será ejecutado por el usuario y tendrá la responsabilidad de:

- Inicializar recursos
- Inicializar el mapa con una configuración de naves. Eres libre de elegir cuál. Por ejemplo, cada equipo en una esquina del mapa, o cada equipo en un lateral del mapa, etc.
- Crear tantos procesos “jefe” como equipos haya definidos. Y esperar su muerte cuando finalice la simulación.
- La comunicación con los procesos “jefe” será a través de tuberías y se notificará:
    - La llegada de un nuevo turno: “TURNO”
    - DESTRUIR <número nave>: Manda “DESTRUIR” la nave <número nave> al jefe que corresponda.
    - La finalización de la simulación: “FIN”
- La detección de que la simulación debe finalizar. Puede deberse a:
    - Hay un equipo campeón (el resto de las naves han sido destruidas).

    - Ha recibido un Ctrl-C desde el teclado.
- El procesamiento (realización) de las acciones que son enviadas por las naves en cada turno. La recepción de las acciones se realizará en una cola de mensajes.
    1. Recibir la acción
    2. Comprobar de qué tipo es
    3. Comprobar si se cumplen todos los requisitos para poder realizar la acción
    4. Si cumple los requisitos, realizar la acción. Si no, no realizarla.
    5. Si se ha realizado la acción, actualizar el mapa y mandar los comandos que correspondan.
- Si llega un nuevo turno, el proceso simulador deberá:
    1. Restaurar el mapa: Eliminar símbolos que se hayan puesto para mostrar acciones del turno anterior (por ejemplo símbolos de ataque) y dejar el mapa listo para un nuevo turno. Es decir que sólo contenga símbolos de las naves vivas e información que el usuario considere. Podéis utilizar **mapa_restore()** .
    2. Comprobar si hay una condición de ganador
        - Si hay un ganador, deberá mostrar por pantalla quién es el ganador y ordenar la finalización de la simulación
    3. Comunicar un nuevo turno a todos los jefes.

Procesamiento de acciones

Una de las funciones principales del proceso "simulador" consiste en procesar las acciones que las naves le mandan. Una de las actividades a realizar en una acción consiste en generar una animación en el mapa, así como modificar el mapa acorde a los resultados de la acción. A continuación, se detalla las posibles acciones a recibir:

- MOVER: Consiste mover la nave que se indica a la posición que se indica
    1. Requisitos:
        - La casilla objetivo debe estar dentro del mapa
        - La casilla objetivo debe estar vacía
    2. Acción: Mover la nave a la posición objetivo
- ATACAR: Consiste en generar un ataque a la nave situada en la casilla objetivo.
    1. Requisitos:
        - La casilla objetivo debe estar en el rango del mapa
        - La casilla objetivo debe estar dentro de la distancia de ataque
    2. Acción: Atacar la casilla objetivo. Usar (mapa_send_misil)
    3. Consecuencias:
        - Si la casilla objetivo está vacía (la nave objetivo puede haberse movido desde que se mandó la acción de ataque). Es un disparo "agua"

        - Se marca la casilla objetivo con SYMB_AGUA. Podéis utilizar **mapa_set_symbol()** .
    - Si en la casilla objetivo hay una nave.
        - Se descuenta el daño de ataque de la vida de la nave
        - Si la vida <=0
            - Se marca la casilla objetivo con SYMB_DESTRUIDO
            - Se ordena DESTRUIR la nave
            - Se actualiza la información del mapa relativo al número de naves vivas en el equipo (usar mapa_set_num_naves) y al estado de la nave (nivel de vida, atributo 'viva', etc)
        - Si la vida > 0
            - Se marca la casilla objetivo con SYMB_TOCADO

Proceso jefe de equipo

Hay un proceso jefe por cada equipo. Este proceso es creado por el proceso simulador y su función es crear, coordinar y dar órdenes a sus naves.

Tendrá la responsabilidad de:

- Crear tantos procesos "nave" como naves tenga el equipo. En este ejercicio todas las naves son iguales.
- Recibir las instrucciones del proceso simulador ("TURNO", "DESTRUIR" o "FIN") y generar las ordenes que sean necesarias a sus naves. La comunicación con las naves se realizará mediante tuberías.
    - TURNO: Puede mandar 2 acciones (las que considere) a cada nave por turno.
    - DESTRUIR <número nave>: Manda "DESTRUIR" a la nave <número nave> para que se destruya.
    - FIN: Manda finalizar a los procesos nave mediante señal SIGTERM
- Mandar orden a las naves:
    - "ATACAR": Ataca a una nave enemiga
    - "MOVER_ALEATORIO": Mueve 1 posición en una dirección

Proceso nave

Es un proceso que es hijo del proceso "jefe" de su equipo y simula el comportamiento de una de sus naves. En este punto, la combinación de órdenes y acciones es enorme, pero lo hemos simplificado en:

- "ATACAR": Busca una nave enemiga en el mapa que esté dentro del rango de ataque y manda un mensaje con la acción ATACAR al proceso simulador.

- “MOVER_ALEAORIO”: Elige una dirección de movimiento de forma aleatoria (usando un rand) a una casilla válida y libre y manda un mensaje con la acción MOVER al proceso simulador.

- “DESTRUIR”: Finaliza.

Proceso monitor

Muestra el estado de la simulación en el terminal.

Se entrega una serie de funciones que simplifican la implementación. Su descripción puede verse en “gamescreen.h” y “mapa.h”.

El proceso monitor debe ser ejecutado en un terminal diferente.

El orden de inicio del proceso simulador y el proceso monitor debe ser indiferente. Si se ejecuta antes el monitor, debe de esperar por el simulador. Para ello se recomienda utilizar un semáforo.

El proceso monitor debe refrescar el estado del mapa cada SCREEN_REFRESH microsegundos (usleep)

En cada refresco, el proceso monitor debe mostrar el símbolo de cada casilla en pantalla.