UNIVERSIDAD AUTÓNOMA DE MADRID

SISTEMAS OPERATIVOS

(2018-2019)

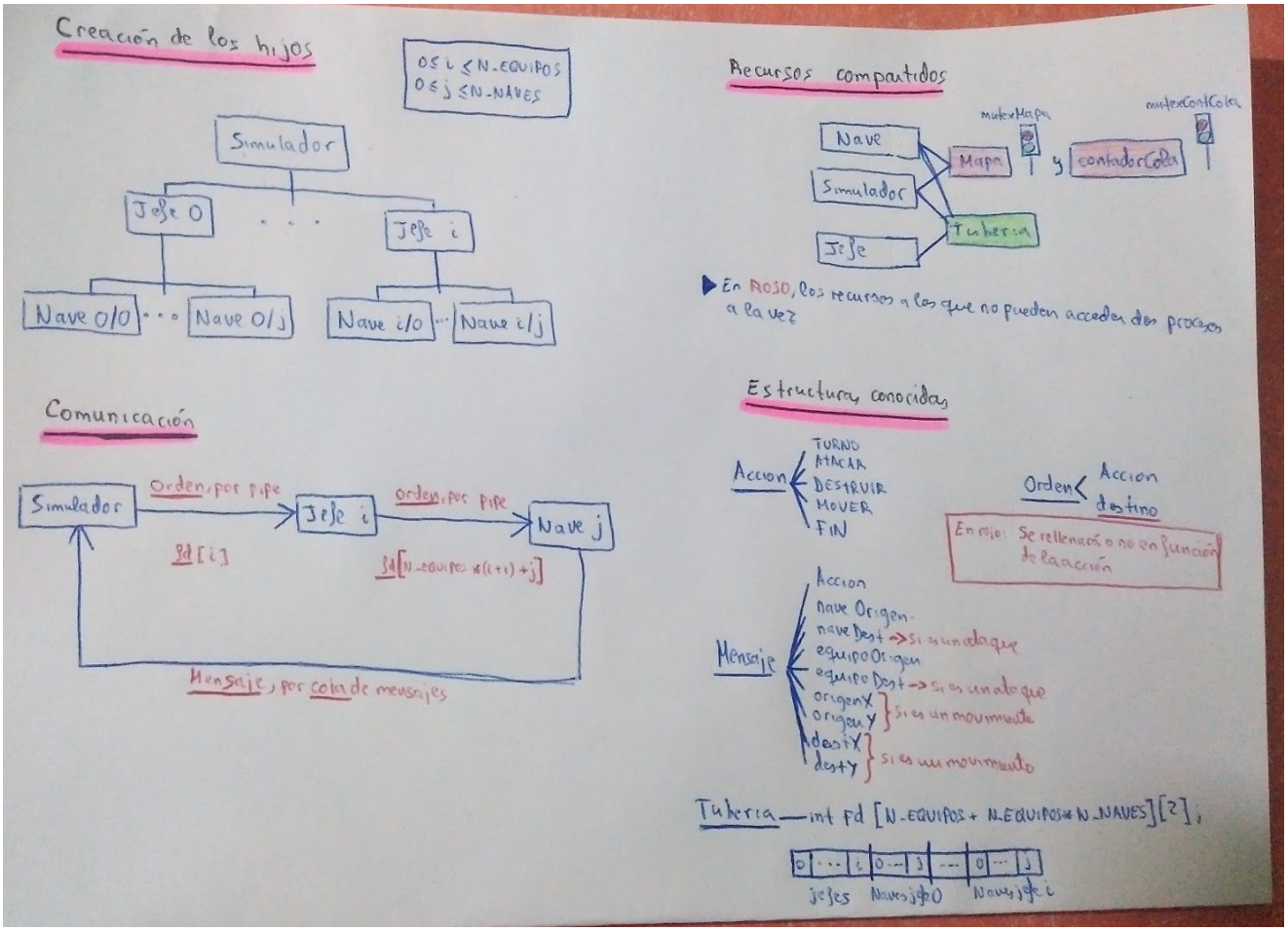
PROYECTO FINAL

Alba Ramos Pedroviejo
Andrea Salcedo López

Grupo 2213

Madrid, 9 de Mayo de 2019

Diagrama



Explicación

En la documentación generada con Doxygen (carpeta /doc/), hemos incluido exclusivamente los módulos creados por nosotras y lo que hayamos modificado (librería simulador.h). No se incluyen mapa.c, mapa.h, gamescreen.c ni gamescreen.h.

Se ha decidido que los jefes, naves y simulador serán procesos separados en ficheros diferentes, para mayor legibilidad del código. En simulador.h, se han modificado algunas macros para facilitar nuestras pruebas: las naves tienen 5 de vida, el tablero es de 10x10 y la distancia máxima de ataque es 8 posiciones.

Para liberar recursos en el control de errores, se ha creado una función liberar() en naves, jefes y simulador, que facilita la legibilidad del código.

A continuación se explican el resto de decisiones de diseño tomadas.

Monitor

En el monitor tenemos un semáforo para que no empiece a imprimir cosas por pantalla hasta que el simulador no tenga todos los recursos, incluido el mapa, inicializados. Este semáforo se crea tanto en el monitor como en el simulador (realmente lo creará uno de los dos, pero al no poner el flag O_EXCL, no dará error si lo crea uno primero y luego solicita crearlo el otro), inicializado a 0, por lo que, lo inicialice quien lo inicialice, el monitor esperará.
Después, llama a screen_init y entra en un bucle infinito a imprimir el mapa, esperando un tiempo determinado por SCREEN_REFRESH entre cada impresión.

Simulador

Hemos incluido en simulador.h las estructuras Tuberia, Orden, Mensaje y el enumerado Accion. Su funcionamiento se irá explicando a lo largo de este apartado. También se ha modificado la estructura tipo_mapa proporcionada, incluyendo en ella un contador de perdedores (que contará cuántos equipos se han quedado sin naves) y un contador de mensajes en cola. Hemos decidido incorporarlos a esta estructura para reutilizar la memoria compartida que creamos para compartir la estructura del mapa tanto en el simulador como en las naves (esta decisión se explicará en el apartado de las naves), y así evitar tener que crear más zonas de memoria compartida solo para estas variables.

Cuando el simulador detecta que una nave se ha quedado sin vida, comprobará si su equipo sigue teniendo naves vivas. Si no las tiene, el contador de perdedores se incrementa. Esto nos permitirá que, cuando llega la señal de turno, se consulte esta variable para comprobar si han perdido todos los equipos menos uno, lo que implica que hay un ganador (el cual se busca de entre todos los equipos comprobando el número de naves vivas que le quedan) y que la simulación debe terminar.

El contador de mensajes en la cola se ha utilizado porque la llamada a mq_receive es bloqueante, entonces ocurría un interbloqueo cuando el simulador trataba de leer y no había nada. Para solucionarlo hemos utilizado este contador, y lo incrementamos cuando las naves envían mensajes, y lo decrementamos cuando el simulador los recibe.

Para manejar las tuberías simulador-jefe y jefe-nave, hemos decidido crear una estructura Tuberia, que almacena un array con tantos descriptores de fichero como tuberías queramos tener: N_EQUIPOS + (N_EQUIPOS * N_NAVES), es decir, tuberías para todos los jefes y para las naves de todos los equipos. Para compartir estas tuberías, utilizamos una zona de memoria compartida, a la que todos podrán acceder. Con esta estructura, es sencillo indexar en la tubería correspondiente para leer y escribir: i = (N_EQUIPOS*(equipo + 1)) + numNave. Las tres primeras serían las de los jefes, y las siguientes tres las de las naves del jefe 0, y así sucesivamente (3 por el máximo de equipos, en nuestro caso).

Cuando llega una señal de alarma, las llamadas bloqueantes se desbloquean. Esto podría causarnos problemas a la hora de, por ejemplo, hacer un wait en un semáforo si justo ha llegado una alarma. Para controlar esto, protegemos las llamadas bloqueantes con un while

y if(errno != EINTR), y así no entrará ni se saltará la condición bloqueante cuando errno nos indique que ha llegado una interrupción. También en relación con las alarmas, hemos bloqueado la señal de alarma para que, cuando el simulador entre al manejador de SIGINT, no le interrumpan mientras manda terminar a todos los jefes.

Las naves pueden decidir atacar a una posición ocupada, pero cuando el simulador procese esta acción concreta esa nave ya podría haberse movido. Es por eso que, en cada acción recibida de las naves, volvemos a comprobar que la posición o el objetivo que escogieron estén disponibles. Hemos incluido un usleep antes de hacer mapa_set_nave, porque los cambios en los símbolos de las naves cuando eran tocadas o destruidas se veían demasiado rápido. Ahora también van rápido, pero ya se pueden ver mejor.

Todas las estructuras nuevas (y las que hemos modificado) que hemos comentado se encuentran incluidas en simulador.h, para que tengan acceso a ellas desde las naves y los jefes importando la librería.

Jefe

Para evitar que el proceso acaba cuando llega SIGINT, y que solo la procese el simulador, hemos creado un sigaction que ignore la señal.
Cuando recibe la orden TURNO del simulador, generará un número aleatorio para decidir qué acción de las dos posibles enviar a todas las naves. Simplemente enviará la acción escogida, y serán las naves las que decidan a dónde moverse y a quién atacar, consultando el mapa para ello.
También puede recibir DESTRUIR, en cuyo caso enviará esta misma orden a la nave correspondiente y ésta terminará.
Por último, puede recibir FIN, y ordenará destruirse a todas sus naves, esperando al final para no dejar procesos huérfanos. Aunque alguna ya estuviese muerta, no habría problemas, ya que no espera a que la nave haya leído de la tubería ni nada, sino que al final hace un wait para todas las naves y ahí las recoge a todas. Antes de acabar, libera sus recursos.

Nave

Hemos decidido compartir el mapa con las naves. Esto se debe a que las naves deciden a qué posición moverse, donde controlamos que la casilla a la que se quieren mover esté vacía y que la posición a la que se quiere acceder no se sale fuera del rango del mapa, ya predefinido en las macros del *simulador.h*. Creamos un número aleatorio para decidir si movernos al Norte, Sur, Este u Oeste.

También deciden a qué equipo atacar y a cual de sus naves quitar vida todo ello mediante números aleatorios, donde se comprueba que el número aleatorio sea distinto del propio equipo y que además el equipo elegido tenga naves vivas. si este equipo posee naves vivas se crea el número aleatoria para decidir a qué nave atacar y por último se le envía toda la información al simulador.

Cuando la nave reciba la instrucción destruir por la tubería, liberará todos los recursos utilizados llamando a la función y finalizará su programa.

Para evitar que el proceso acaba cuando llega SIGINT, y que solo la procese el simulador, hemos creado un sigaction que ignore la señal. En cambio, cuando al proceso le llegue la señal SIGTERM, que implica que ha habido un equipo ganador, el manejador que controla la misma señal cerrará los recursos abiertos y terminará.