Memoria del proyecto: Simulador de Carreras

Rafael Sánchez Sánchez - Sergio Galán Martín
Sistemas Operativos
Universidad Autónoma de Madrid

1 Diagrama de diseño.

El diagrama que muestra la comunicación y uso de todos los ipcs en los procesos se muestra al final de la sección.

El proceso principal crea los distintos ipcs que aparecen en el digrama, y tras ellos genera los procesos hijos en orden de izquierda a derecha.

Encima de cada proceso se indica de que tipo de IPC hace uso.

1.1 Proceso monitor.

El proceso monitor hace uso de todos los semáforos y de la memoria compartida. El semaforo general sirve para sincronizarse con el inicio del resto de hijos, el de caballos permite el acceso de forma correcta a la memoria compartida y los nombrados como "nom" y "tur" sirven para establecer un rendez-vous con el proceso principal para imprimir las tiradas de cada turno de los caballos. Finalmente lee del fichero de apuestas para imprimirlo en pantalla y mostrar el historial de las mismas.

1.2 Proceso gestor.

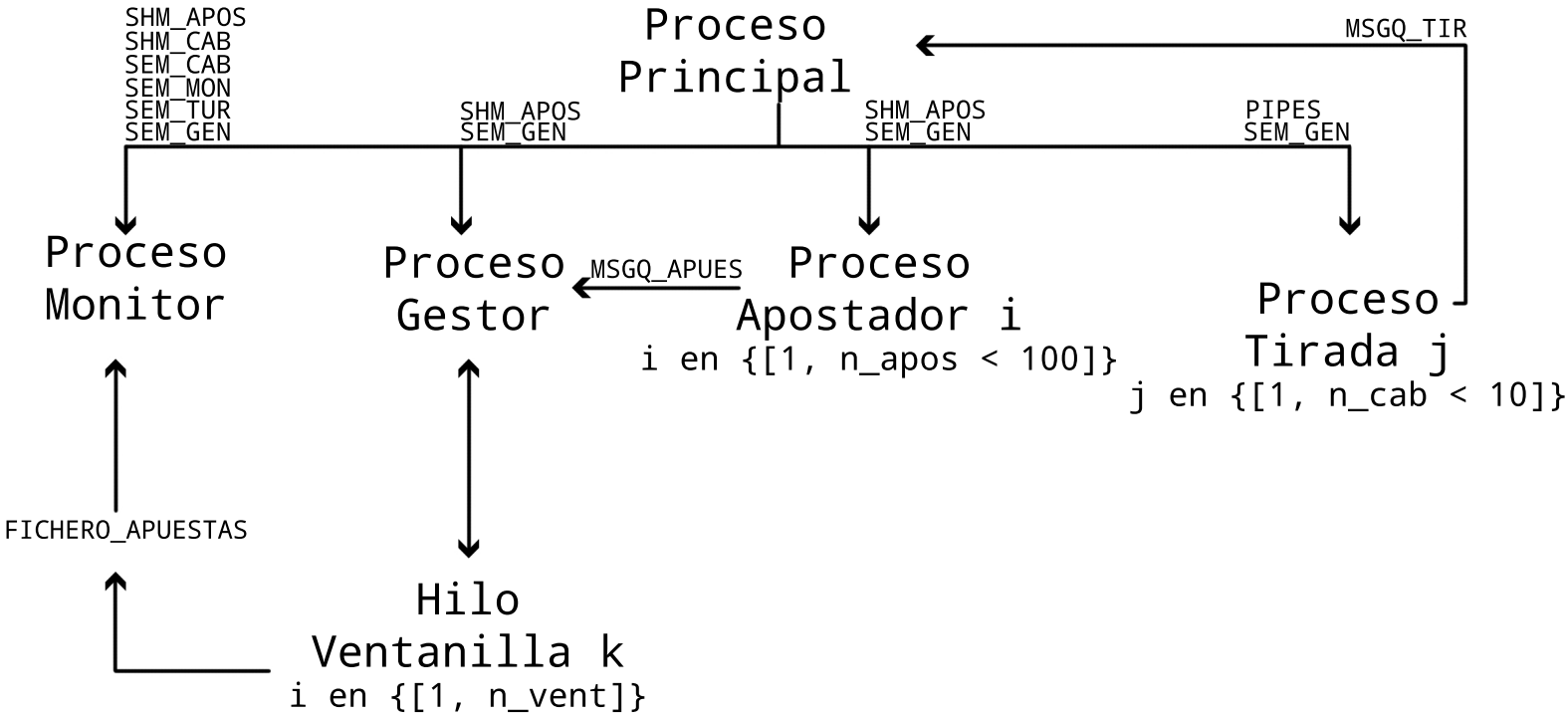
El proceso gestor hace uso del semaforo general y de la memoria compartida de los apostadores. Además, se comunica por medio de una cola de mensajes con el proceso apostador, que envia las apuestas. El proceso gestor crea unos hilos que corresponden a una ventanilla cada uno. La ventanilla es la encargada de recibir los mensajes y de procesar las apuestas que imprimirá en el fichero de apuestas.

1.3 Procsesos apostador.

Los procesos apostador hacen uso del semaforo general, de la memoria de apostadores para inicializarla y de la cola de mensajes de apuestas para enviarlas al proceso gestor. Genera un mensaje por segundo durante el tiempo de apuestas.

1.4 Procesos de tirada.

Los procesos de tirada simulan cada uno el funcionamiento de los caballos. Leen cada uno de su pipe correspondiente el tipo de tirada que tienen que ejecutar. Tras ello envian su posicion al proceso principal por la cola de mensajes de tiradas. El proceso principal determina las tiradas y las envia de vuelta a los procesos de tirada por medio de las pipes.


SHM_APOS
SHM_CAB
SEM_CAB
SEM_MON
SEM_TUR
SEM_GEN
Proceso Principal
MSGQ_TIR
SHM_APOS
SEM_GEN
SHM_APOS
SEM_GEN
PIPES
SEM_GEN
Proceso Monitor
Proceso Gestor
MSGQ_APUES
Proceso Apostador i
i en {[1, n_apos < 100]}
Proceso Tirada j
j en {[1, n_cab < 10]}
FICHERO_APUESTAS
Hilo Ventanilla k
i en {[1, n_vent]}

2 Decisiones de diseño y estructura del proyecto.

Hemos orientado el proyecto con la idea de modularizar el código lo máximo posible. De esta forma hemos dividido el proyecto en distintos archivos, cada uno correspondiente a los distintos procesos que se llevan a cabo y cuya función se invoca desde el proceso principal.

Por motivos de legibilidad hemos decidido además separar la memoria en distintos tipos abstractos de datos (caballo, apostador y apuesta), de forma que nos permite encapsular el código de una forma más sencilla. En los TADS aunque la estructura es pública, decidimos acceder a los campos por medio de funciones ya que esto nos permite hacer privada la estructura sin necesitar muchos cambios.

En cuanto a los semaforos, usamos 4, uno para proteger la memoria compartida del caballo, uno general para sincronizar el inicio de los procesos hijo, y 2 para hacer un rendez-vous entre el proceso principal y los procesos hijo.

Además, los parametros de entrada los guardamos en variables externas, declaradas en el fichero sim_carr_lib.h y definidas en sim_carr_lib.c aunque las inicialicemos en el simular_carreras.c. Esto nos permite escribir el código de forma mucho más fácil. Además, hay declaras otras variables globales internas para poder usarlas en los manejadores de las distintas señales. Se establecen mascaras de señales en cada archivo para intentar hacer el código más robusto.

Por último, debido a que el pdf de la práctica no dejaba claro si había que crear un proceso que generara todas las apuestas o un proceso por apostador, hemos decidido tomar la segunda opción. Esto nos permite diferenciar mejor cada apostador aunque resulte algo más dificil su sincronización. Además, no incluimos el uso de `syslog` ya que el debugging lo hemos llevado a cabo con gdb y sus distintas opciones.

3 Comentarios generales.

Hemos hecho uso de todos los tipos de recursos aprendidos durante este curso, por lo que no ha sido muy difícil la codificación y testeo de los mismos.

Sin embargo, hemos tenido ciertos problemas con la modularización y el comportamiento de variables externas, que era nuevo para nosotros.

Finalmente, hemos conseguido que valgrind reporte 0 errores de memoria, para lo que hemos tenido que revisar a fondo el funcionamiento de los hilos.