SOPER:
Práctica 3

Adrián Lorenzo Mateo
Ari Handler Gamboa

Grupo 221, Equipo 07

1. ¿Qué diferencias observas al crear la cola de mensajes usando IPC_PRIVATE, en lugar de una clave constante?. Prueba a ejecutarlo varias veces usando IPC_PRIVATE.

Si se usa el flag **IPC_PRIVATE** se creará una cola de mensajes distinta por cada llamada a la función **msgget**. En cambio, si se usa una clave constante se creará una sola vez.

2. ¿Qué hace este código? Ejecútalo y compruébalo usando el comando ipcs.

Este código genera una cola de mensajes con la key 1555 y envia el mensaje “Hola Mundo! “ con tipo=1 mediante **msgsnd**. Cada vez que se ejecute el programa se añadirá a la misma cola de mensajes (key 1555) el mensaje “Hola Mundo! “.

3. Modifica el código anterior para que al finalizar elimine la cola de mensajes.

Para eliminar una cola de mensajes creada mediante msgget se utiliza la función **msgctl** junto con el flag **IPC_RMID** que borra del sistema la cola cuyo identificador es el id pasado como parámetro.

4. Ejecuta este código. Ahora cambia el valor del define LOOP por un número mayor, hasta que el programa se bloquee. Desde otra consola ejecuta ipcrm para eliminar la cola creada, y explica porqué se ha bloqueado.

El programa se ha bloqueado debido a que no hay espacio en la cola para enviar un mensaje.

5 Compílalo y ejecútalo en una consola. Ahora sin parar este proceso, abre una nueva consola y ejecuta el programa ej4. ¿Qué sucede? ¿Es posible ahora bloquear el programa ej4 cambiando el valor de LOOP?

El programa 4 ejecuta todo el bucle debido a que el programa 5 está recibiendo los mensajes y eliminándolos de la cola, con lo cual nunca se llena.

6. Sin parar el ej5 desde otra consola elimina las colas de mensajes. ¿Qué sucede?

Al eliminar la cola de mensajes en el ejecicio 5 ocurre un error en la función msgrcv devolviendo en errno el código EIDRM que significa que la cola fue borrada.

7. Como afecta a los programas ej2 y ej5 añadir el flag IPC_NOWAIT a la función msgsnd y/o msgrcv.

En **msgsnd** si la cola está llena y el flag **IPC_NOWAIT** está activo la función devuelve el control retornando un -1 (el mensaje no se envía).

En **msgrcv** si no hay ningún mensaje del tipo especificado la función devuelve el control retornando un -1 (el mensaje no se recibe).

8. Ejecuta en una consola el programa ej8b y en otra el ej8. ¿Qué ocurre? Páralos y elimina las colas de mensajes, ejecuta ej8b con argumento 2 y ej8 con argumento 1 en consolas separadas. Explica por qué el comportamiento es diferente.

El comportamiento es diferente ya que si se le pasa de parámetro un 2 a ej8b espera un mensaje de tipo 2 y ej8 envia mensajes de tipo 1. Con lo que sin el flag **IPC_NOWAIT** el programa ej8b se queda esperando la llegada de un mensaje tipo 2.

9. Elimina las colas de mensajes y ejecuta en tres consolas diferentes el programa ej8, a cada una dale argumentos diferentes (primera consola -> ej8 1, segunda consola -> ej8 3 y tercera consola -> ej8 2). Ahora en una cuarta consola ejecuta el programa ej8b con argumento –2. Explica qué sucede.

El programa ej8b lee todos los mensajes con tipo menor o igual que el valor absoluto de -2 ya el -2 se utilizará en la función **msgrcv** como msgtyp (el tipo de mensaje esperado). El funcionamiento de **msgtyp** es el siguiente:

Msgtyp=0 lee el primer mensaje que haya en la cola.

Msgtyp>0 lee el primer mensaje que haya en la cola del tipo msgtyp.

Msgtyp<0 lee el primer mensaje cuyo tipo sea menor o igual al valor absoluto de msgtyp y sea el de menor valor de los que cumplan esta condición.

10. Modifica el ej8b original para que el define SIZE sea 15, elimina las colas y vuelve a ejecutar los programas ej8 y ej8b en diferentes consolas. ¿Qué pasa ahora? ¿En qué estado ha quedado la cola de mensajes?

El ej8b imprime por pantalla el error “Argument list too long”, con errno igual a **E2BIG** que significa que la longitud del texto es mayor que msgsz de la función **msgrcv**. La cola de mensajes queda ocupada con los mensajes enviados por el ej8.

11. Con el ej8b modificado en el ejercicio anterior añade el flag MSG_NOERROR a la llamada a la función msgrcv y vuelve a repetir el comportamiento del ejercicio 10. Comenta la diferencia con el anterior.

MSG_NOERROR trunca el mensaje de texto si es más largo que msgsz, permitiendo la recepción del mensaje (parcialmente) por parte del ej8.

EJERCICIO FINAL SUDOKU

Diseño:

La aplicación no implementa un sudoku propiamente dicho, si no una simulación de éste. Se podrá introducir en las celdas cualquier número con tal de que la casilla esté vacía, contando como jugada correcta.

Bibliotecas implementadas:

Sudoku:

Contiene las estructuras, constantes y funciones necesarias para el funcionamiento del juego.

Mensajería:

Contiene todas las funciones necesarias para el envio y recepción de mensajes del juego por parte de los procesos.

Semáforos:

Contiene todos los recursos para la utilización de semáforos.

Recursos Compartidos:

Memoria compartida:

Zona Sudoku (tablero):
Alamacena el tablero del sudoku.

Zona Jugador:
Alamacena la información de los jugadores.

Semáforos:

SEM_MAX_JUGADORES: Controla el máximo de jugadores.
SEM_SUDOKU_MUTEX: Controla el acceso a la zona Sudoku.
SEM_JUGADOR_MUTES: Controla el acceso a la zona Jugador.

Cola de mensajes:

- Peticiones de Jugador: serán de tipo 1.
- Peticiones de Jugada: serán de tipo 99.
- Peticiones de Ordenador: serán de tipo 4.
- Turno: serán del tipo (pid de cada jugador).
- Jugada Ordenador: serán del tipo 5.
- Bloque completo: serán del tipo 6.

Ordenador:

Sistema que “dará pistas” a los jugadores que se pondrá en funcionamiento al no producirse ninguna jugada de parte de los jugadores.

El proceso realiza las siguientes funciones:

- Se engancha a los recursos compartidos.
- Inicializa la estructura Ordenador con el pid del proceso.
- Envia esa estructura al proceso Juez mediante un mensaje.
- Ejecuta un bucle en el que:
  - Espera recibir un mensaje de comienzo de turno.
  - Obtiene el tablero desde la memoria compartida.
  - Realiza una jugada correcta.
  - Introduce el tablero en memoria compartida.

La finalización del proceso se producirá al enviarle la señal SIGUSR1 o SIGINT, liberando los recursos.

Jugador:

Proceso mediante el cual el jugador/es podrán resolver el sudoku.

El proceso realiza los siguientes pasos:

- Se engancha a los recursos compartidos.
- Inicializa la estructura Jugador con el pid y el nombre del jugador.
- Baja el semáforo que representa el máximo de jugadores.
- Envía la petición de entrar a resolver el sudoku.
- Ejecuta un bucle en el que:
  - Espera recibir un mensaje de comienzo de turno.
  - Obtiene el tablero desde la memoria compartida.

- Imprime el tablero.
- Comienza la alarma de fin de turno.
- Introduce las coordenadas y el valor de la casilla.
- Si ha saltado la alarma envía un mensaje de fin de turno a juez y vuelve al inicio del bucle.
- Inicializa la jugada.
- Envia la jugada.
- Si es correcta se le automáticamente el turno.

La finalización de Jugador supondrá la recepción de la señal SIGINT o SIGUSR1, liberando los recursos (y subiendo el semáforo).

Si la partida ha terminado correctamente el jugador recibirá una señal para imprimir un mensaje según el número de jugadas correctas realizadas. Si es el ganador recibirá la señal SIGUSR2, si no la SIGBUS.

Juez:

Inicializa los recursos, y controla el paso de turnos y las jugadas correctas.

El proceso juez realiza un bucle infinito en el que:

- Inicializa la tabla de jugadores (para mandarles señales).
- Inicializa la tabla de hijos (para mandarles señales).
- Bucle de captación de jugadores:
  - Activa el temporizador, si este se activa comienza el juego.
  - Si recibe una peticion de jugador
    - Lo introduce en la memoria compartida.
    - Vuelve a reiniciar el temporizador
  - Cuando el temporizador llegue a 0 o el número de jugadores introducidos sea el máximo permitido comienza la partida.
- Genera los procesos hijos, que van a realizar un sondeo de cada bloque, viendo si está completo.
- Si se ha completado el bloque el hijo manda una señal al padre.
- El proceso padre coordina la partida mediante un bucle:
  - Saca al jugador de la memoria compartida
  - Le manda el turno
  - Chequea si el jugador ha mandado una jugada a un fin de turno (agotado) (cambiando el turno).
  - Si es una jugada se comprueba si es correcta (si no hay ningún numero anterior en la casilla).
  - Si es correcta le vuelve a mandar el turno.

        - Si es incorrecta cambia el turno (saca al siguiente jugador).
        - Si recibe un mensaje de bloque completado incremente el contador de bloques.
        - Si el contador de bloques llega al máximo de bloques la partida ha finalizado.
            - Por cada jugador, se saca de memoria y según sea ganador o perdedor se le manda una señal.
        - Finaliza el bucle de control de partida.
    - Vuelve al inicio del bucle principal.

El proceso termina cuando recibe una señal SIGINT, mandando a su vez una señal a los proceso hijos, jugadores y ordenador para su finalziación y finalmente libera los recursos compartidos.