Universidad Nacional Experimental de Guayana

Vice-Rectorado Académico

Coordinación de pregrado

Proyecto de carrera de Ingeniería Informática

Asignatura: Sistemas de Operación

Sección: 01

Proyecto II Bingo

Profesor:

Caniumilla Andrés

Elaborado Por: C.I.:

Carrasco Tomás
23.506.608

Sánchez Stalin
24.183.684

Ciudad Guayana, marzo de 2015.

Marco Teórico

Sistema Operativo

Un Sistema Operativo (SO) es el software básico de una computadora que provee una interfaz entre el resto de programas del ordenador, los dispositivos hardware y el usuario.

Las funciones básicas del Sistema Operativo son administrar los recursos de la máquina, coordinar el hardware y organizar archivos y directorios en dispositivos de almacenamiento.

Los Sistemas Operativos más utilizados son Dos, Windows, Linux y Mac. Algunos SO ya vienen con un navegador integrado, como Windows que trae el navegador Internet Explorer.

El sistema operativo es el programa (o software) más importante de un ordenador. Para que funcionen los otros programas, cada ordenador de uso general debe tener un sistema operativo. Los sistemas operativos realizan tareas básicas, tales como reconocimiento de la conexión del teclado, enviar la información a la pantalla, no perder de vista archivos y directorios en el disco, y controlar los dispositivos periféricos tales como impresoras, escáner, etc.

En sistemas grandes, el sistema operativo tiene incluso mayor responsabilidad y poder, es como un policía de tráfico, se asegura de que los programas y usuarios que están funcionando al mismo tiempo no interfieran entre ellos. El sistema operativo también es responsable de la seguridad, asegurándose de que los usuarios no autorizados no tengan acceso al sistema.

Procesos

Un proceso de unix es cualquier programa en ejecución y es totalmente independiente de otros procesos. El comando de unixps nos lista los procesos en ejecución en nuestra máquina. Un proceso tiene su propia zona de memoria y se ejecuta "simultáneamente" a otros procesos. Es totalmente imposible en unix que un proceso se meta, a posta o por equivocación, en la zona de memoria de otro proceso. Esta es una de las características que hace de unix un sistema fiable. Un programa chapucero o malintencionado no puede fastidiar otros programas en ejecución ni mucho menos a los del sistema operativo. Si el programa chapucero se cae, se cae sólo él.

Dentro de un proceso puede haber varios hilos de ejecución (varios threads). Eso quiere decir que un proceso podría estar haciendo varias cosas "a la vez". Los hilos dentro de un proceso comparten toda la misma memoria. Eso quiere decir que si un hilo toca una variable, todos los demás hilos del mismo proceso verán el nuevo valor de la variable.

Un proceso es, por tanto, más costoso de lanzar, ya que se necesita crear una copia de toda la memoria de nuestro programa. Los hilos son más ligeros.

Cola de Mensajes

Las colas de mensajes, junto con los semáforos y la memoria compartida son los recursos compartidos que pone unix a disposición de los programas para que puedan intercambiarse información.

En C para unix es posible hacer que dos procesos (dos programas) distintos sean capaces de enviarse mensajes (estructuras de datos) y de esta forma pueden intercambiar información. El mecanismo para conseguirlo es el de una cola de mensajes. Los procesos introducen mensajes en la cola y se van almacenando en ella. Cuando un proceso extrae un mensaje de la cola, extrae el primer mensaje que se introdujo y dicho mensaje se borra de la cola.

También es posible hacer "tipos" de mensajes distintos, de forma que cada tipo de mensaje contiene una información distinta y va identificado por un entero. Por ejemplo, los mensajes de tipo 1 pueden contener el saldo de una cuenta de banco y el número de dicha cuenta, los de tipo 2 pueden contener el nombre de una sucursal bancaria y su calle, etc. Los procesos luego pueden retirar mensajes de la cola selectivamente por su tipo. Si un proceso sólo está interesado en saldos de cuentas, extraería únicamente mensajes de tipo 1, etc.

Semáforos

A veces es necesario que dos o más procesos o hilos (threads) accedan a un recurso común (escribir en un mismo fichero, leer la misma zona de memoria, escribir en la misma pantalla, etc). El problema es que si lo hacen simultáneamente y de forma incontrolada, pueden "machacar" el uno la operación del otro (y dejar el fichero o la memoria con un contenido inservible o la pantalla ilegible).

Para evitar este problema, están los semáforos. Un semáforo da acceso al recurso a uno de los procesos y se lo niega a los demás mientras el primero no termine. Los semáforos, junto con la memoria compartida y las colas de mensajes, son los recursos compartidos que suministra UNIX para comunicación entre procesos.

El funcionamiento del semáforo es como el de una variable contador. Imaginemos que el semáforo controla un fichero y que inicialmente tiene el valor 1 (está "verde"). Cuando un proceso quiere acceder al fichero, primero debe decrementar el semáforo. El contador queda a 0 y como no es negativo, deja que el proceso siga su ejecución y, por tanto, acceda al fichero.

Ahora un segundo proceso lo intenta y para ello también decrementa el contador. Esta vez el contador se pone a -1 y como es negativo, el semáforo se encarga de que el proceso quede "bloqueado" y "dormido" en una cola de espera. Este segundo proceso no continuará por tanto su ejecución y no accederá al fichero.

Supongamos ahora que el primer proceso termina de escribir el fichero. Al acabar con el fichero debe incrementar el contador del semáforo. Al hacerlo, este contador se pone a 0. Como no es negativo, el semáforo se encarga de mirar el la cola de procesos pendientes y "desbloquear" al primer proceso de dicha cola. Con ello, el segundo proceso que quería acceder al fichero continua su ejecución y accede al fichero.

Cuando este proceso también termine con el fichero, incrementa el contador y el semáforo vuelve a ponerse a 1, a estar "verde".

Es posible hacer que el valor inicial del semáforo sea, por ejemplo, 3, con lo que pasarán los tres primeros procesos que lo intenten. Pueden a su vez quedar muchos procesos encolados simultáneamente, con lo que el contador quedará con un valor negativo grande. Cada vez que un proceso incremente el contador (libere el recurso común), el primer proceso encolado despertará. Los demás seguirán dormidos.

Como vemos, el proceso de los semáforos requiere colaboración de los procesos. Un proceso debe decrementar el contador antes de acceder al fichero e incrementarlo cuando termine. Si los procesos no siguen este "protocolo" (y pueden no hacerlo), el semáforo no sirve de nada.

Algoritmos Utilizados. Descomposición Modular

El juego consta de 3 procesos bien diferenciados, cantador, juez y postor.

El cantador se encargara de sacar y cantar los números que el mismo saca del bombo (los números que se encuentran dentro del bombo son escogidos al azar), también se encargara de avisarle a uno de los jueces que verifique un bingo en caso de que alguno de los jugadores cante.

El juez se encargara de verificar si un jugador que ha cantado bingo en verdad lo tiene, cada juez tiene un número limitado de jugadores que le pueden ser asignados, de pasarse de este números el juez creara jueces hijos que verificaran a los demás jugadores. Cabe destacar que solo el juez padre se puede comunicar con el cantor, por lo que si uno de los jueces hijo verifica un bingo, este tiene que comunicárselo al juez padre y este a su vez al cantor.

El postor se encargara de tachar los números en sus tarjetas, el tendrá 2 niveles de dificultad: manual en donde el mismo deberá elegir que números de sus tarjetas quiere tachar, y automático en donde la computadora de forma automática tachara los números que sean cantados y se encuentren en alguna de sus tarjetas y cantara bingo.

Aparte de estos 3 procesos también existe una memoria compartida (La sala). La sala de juego se compone esencialmente de una sala, con aforo limitado a MAX_PARTICIPANTES (30) jugadores, en donde estarán:

- Además de los participantes, el locutor y al menos un juez.
- El bombo, con una cantidad finita (MAX_BOLAS) de 75 bolas, cuyos valores variarán en el intervalo MIN_NUMERO y MAX_NUMERO (1 y 75).
- La pila de tarjetas, con un número fijo de números impresos (NUMEROS) de 25. Cada participante podrá jugar simultáneamente con un número máximo de tarjetas, indicado por

MAX_TARJETAS_JUGADOR (2), siendo por tanto MAX_PARTICIPANTES *

MAX_TARJETAS_JUGADOR el número máximo de tarjetas existentes.

Que la sala tenga un aforo limitado implica que nunca podrá haber más participantes que el máximo número permitido, lo cual no implica que deba llenarse para poder comenzar el juego. Si un participante desea jugar estado el aforo completo, deberá esperar a que finalice el juego actual (alguien cante bingo) y a que haya un hueco para poder entrar a la partida, teniendo siempre prioridad la gente que actualmente ha jugado.

Descripción de la Estructura de Datos Utilizada

Estructuras

TARJETA: representa la tarjeta en la que los jugadores tachan los números que están siendo cantados y está compuesta por los números que se encuentran en ella y el pid del usuario que la posee.

SALA: representa la sala donde se está jugando el bingo, guarda el pid de los jugadores, jueces, el ganador, el número de participantes, las tarjetas y el bombo.

MENSAJE: está compuesto por el id y el número de bola, esto vendría siendo el mensaje que se le pasa a los jugadores para que sepan que numero de bola salió en el bombo.

Especificaciones de Entrada

Botón Tachar: la tarjeta está representada por una matriz de botones, la cual para poder tachar presionamos el botón del número que queremos tachar.

Modo de Juego: hay 2 opciones a elegir Manual en donde el usuario es el que elige que números tachar en su tarjeta, y Automático en donde la maquina se

encargara de tachar los números que sean cantados y se encuentren en su tarjeta.

Botón Cantar: para que el jugador pueda cantar bingo necesita presionar este botón, que después de presionado el juez que le corresponda procederá a verificar si el jugador efectivamente tiene bingo.

Salir: Para poder salir de alguno de los programas instanciados el usuario deberá presionar CTRL+C.

Especificaciones de Salida

Ventana Jugador: en esta ventana se le mostrara al jugador que ha cantado el cantor, en caso de haber elegido el modo manual podrá elegir que parte de sus tarjetas quiere tachar (en caso de querer hacerlo) y se le dará la opción de cantar bingo.

Decisión: en caso de que algún jugador cante bingo el juez que tiene asignado determinara si el jugador tiene o no tiene bingo.

Resultado: al final de cada partida se muestra el resultado que incluye al ganador (en caso de haberlo).

Diccionario de Variables

i->numeroEnColumna: variable de tipo entero que sirve para recorrer una tarjeta en busca de un número.

i, j,k->tarjetaRepetida: variables de tipo entero que se usa para recorrer las tarjetas.

bandera->tarjetaRepetida: variable de tipo entero para marcar donde haya números repetidos.

auxiliar->tarjetaRepetida: variable de tipo Tarjeta utilizada como auxiliar para hacer un intercambio de valores entre 2 variables.

i,j,k->llenarTarjetas: variables de tipo entero que se usa para recorrer las tarjetas.

aleatorio->llenarTarjetas: variable de tipo entero donde se guardan los números generados de forma aleatoria antes de ser guardados en las tarjetas.

i,j,k->inicializarTarjetas: variables de tipo entero que se usa para recorrer las tarjetas.

i,j,k->desmarcarTarjetas: variables de tipo entero que se usa para recorrer las tarjetas.

i,j,k->imprimirTarjeta: variables de tipo entero que se usa para recorrer las tarjetas.

i->bolaEnBombo: variable de tipo entero que se usa para recorrer las bolas que se encuentran en el bombo.

i->imprimirBombo: variable de tipo entero que se usa para recorrer las bolas que se encuentran en el bombo.

i->inicializarBombo:variable de tipo entero que se usa para recorrer las bolas que se encuentran en el bombo.

aleatorio ->inicializarBombo:variable de tipo entero donde se guardan los números generados de forma aleatoria antes de ser guardados en el bombo.

i->inicializarJueces:variable de tipo entero que se usa para recorrer a los jueces.

i->inicializarIdentificadores:variable de tipo entero que se usa para recorrer a los participantes.

i->bomboVacio: variable de tipo entero que se usa para recorrer las bolas que se encuentran en el bombo.

i->cantar: variable de tipo entero que se usa para recorrer a los participantes.

aleatorio->cantar:variable de tipo entero donde se guardan los números generados de forma aleatoria.

mensaje->cantar:variable de tipo Mensaje que se usa para guardar el mensaje que se va a enviar.

i->posicionDeTarjeta1:variable de tipo entero que se usa para recorrer las tarjetas.

i,j,k->comprobarBingo:variables de tipo entero que se usa para recorrer las tarjetas.

bandera->comprobarBingo: variable de tipo entero que se usa para saber si hay o no bingo.

posTarjeta->comprobarBingo: variable de tipo entero que guarda la posición de una tarjeta.

i,j->tacharNumero,tacharNumero2: variables de tipo entero que se usa para recorrer las tarjetas de un jugador.

columna->tacharNumero,tacharNumero2: variable de tipo entero que se guarda el valor de una columna a la que pertenece un determinado número.

i->obtenerPosicionDeTarjetas: variable de tipo entero que se usa para recorrer las tarjetas de un jugador.

contador->obtenerPosicionDeTarjetas: variable de tipo entero se usa para saber cuántas tarjetas tiene un jugador .

semaforoTarjetas->G_tacharNumero: variable de tipo entero que representa un semáforo.

semaforoGanado->G_cantarBingo: variable de tipo entero que representa un semáforo.

i->meterJugadorASala: variable de tipo entero que se usa para recorrer a los jugadores.

contador->meterJugadorASala: variable de tipo entero que se usa para contar cuantas tarjetas tiene un jugador.

i->matarJuecesHijos: variable de tipo entero que se usa para recorrer a los jueces.

i->meterJuezASala: variable de tipo entero que se usa controlar el número de jueces que se van a crear.

hijos->meterJuezASala: variable de tipo entero que se usa para saber cuántos hijos se van a crear.

pid->meterJuezASala: variable de tipo entero que se usa para saber el pid de los jueces hijos.

i->designarJuez: variable de tipo entero que se usa para recorrer a los jugadores.

juezDesignado->designarJuez: variable de tipo entero que representa el juez asignado para determinado jugador.

i->quitarTarjetasAJugador: variable de tipo entero que se usa para recorrer las tarjetas de un jugador.

contador->quitarTarjetasAJugador: variable de tipo entero que se usa para contar las tarjetas que tiene un jugador.

i->finalizarPostor: variable de tipo entero que se usa para recorrer las tarjetas de un jugador..

bandera->finalizarPostor: variable de tipo entero que se usa para controlar si se encontró al jugador que quiere salir.

i,j->tarjetaLlena: variable de tipo entero que se usa para recorrer una tarjeta.

Tabla de Funciones Definidas

Nombre	Tipo	Parámetros	¿Qué hace?
numeroEnColumna	Entera	intnumero, int columna, Tarjeta tarjeta	Sirve para saber si un determinado número se encuentra en una determinada columna.
tarjetaRepetida	Entera	Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR], intposTarjeta	Verifica si una tarjeta se encuentra repetida.
llenarTarjetas	Vacía	Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR]	Se encarga de llenar las tarjetas con números aleatorios.
inicializarTarjetas	Vacía	Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR]	Se encarga de inicializar con valores por defecto las tarjetas.
desmarcarTarjetas	Vacía	Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR]	Se encarga de limpiar todas las tarjetas.
imprimirTarjetas	Vacía	Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR]	Imprime la información de las tarjetas.
bolaEnBombo	Entera	int bola, int bombo[MAX_BOLAS]	Averigua si una bola se encuentra en el bombo.
imprimirBombo	Vacía	int bombo[MAX_BOLAS]	Imprime todas las bolas que se encuentran en el bombo.
inicializarBombo	Vacía	intbombo[MAX_BOLAS]	Inicializa el bombo.
inicializarJueces	Vacia	pid_t	Inicializa a los

		jueces[MAX_PARTICIPANTES/MAX_JUGADORES_JUEZ +1]	jueces.
inicializarIdentificadores	Vacía	pid_t identificadores[MAX_PARTICIPANTES]	Inicializa los identificadores de los jugadores.
inicializarSala	Vacía	Sala* sala	Inicializa la Sala.
bomboVacio	Entera	int bombo[MAX_BOLAS]	Dice si el bombo se encuentra vacío (ya no le quedan bolas).
inicializarPuntos	Vacía	Nodo **puntos	Inicializa los puntos de ambos jugadores en 0
cantar	Vacía	intidCola, int participantes, pid_t identificadores[MAX_PARTICIPANTES], int bombo[MAX_BOLAS]	Se encarga de realizar el canto.
posicionDeTarjeta1	Entera	intpidPropietario, Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR]	Devuelve la posición de la primera tarjeta de un determinado jugador.
comprobarBingo	Entera	intpidGanador, Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR], int bombo[MAX_BOLAS]	Se encarga de comprobar si el jugador que canto el bingo, en verdad tiene bingo.
columnaPerteneciente	Entera	intnumero	Dice a qué columna pertenece un determinado número.
tacharNumero	Entera	intnumero, intpidPropietario, Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR], int *fi,	Se encarga de tachar un número de la tarjeta (multiplicándolo por -1).

		int *co	
cantarBingo	Vacía	int *ganador, pid_tpidJuez	Avisa al juez y cantor que un jugador canto bingo y coloca su pidcomo posible ganador.
obtenerPosicionDeTarjetas	Vacía	pid_tpidPropietario, intposTarjetas[MAX_TARJETAS_JUGADOR], Tarjeta tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR]	Se encarga de obtener las posiciones de cada una de las tarjetas de un jugador.
G_tacharNumero	Vacía	GtkWidget* btnNumero, Sala* sala	Se encarga de tachar el número en modo gráfico.
G_cantarBingo	Vacía	GtkWidget* btnBingo, Sala* sala	Se encarga de cantar bingo en modo gráfico.
G_inicialiarTarjeta	Vacía	G_Tarjeta *G_tarjeta, Tarjeta tarjeta1, Tarjeta tarjeta2, Sala *sala	Inicializa la tarjeta en modo gráfico.
meterJugadorASala	Vacía	pid_tpid, Sala *sala, intposTarjetas[MAX_TARJETAS_JUGADOR], G_Tarjeta* G_tarjeta	Se encarga de meter un jugador en la sala.
matarJuecesHijos	Vacía	pid_t jueces[MAX_PARTICIPANTES/MAX_JUGADORES_JUEZ +1]	Se encarga de matar a los jueces hijos.
meterJuezASala	Vacía	Sala *sala	Se encarga de meter un juez en la sala.
designarJuez	Entera	pid_tpidGanador, pid_t identificadores[MAX_PARTICIPANTES]	Se encarga de designar juez a un jugador.
quitarTarjetasAJugador	Vacía	pid_tpidPropietario, Tarjeta	Se encarga de quitarle las

		tarjetas[MAX_PARTICIPANTES*MAX_TARJETAS_JUGADOR]	tarjetas a un jugador.
finalizarPostor	Vacía	pid_tpidSaliente, Sala *sala	Se encarga de sacar a un jugador de la partida y de liberar los recursos que este haya reservado.
destruirTodosSemaforos	Vacía	intsemaforoGanador, intsemaforoTarjetas, intsemaforoCantor, intsemaforoJueces, intsemaforoIdentificadores, intsemaforoParticipantes, intsemaforoBombo, intsemaforoAforo	Se encarga de destruir todos los semáforos.
imprimirEstado	Vacía	intsemaforoGanador, intsemaforoTarjetas, intsemaforoCantor, intsemaforoJueces, intsemaforoIdentificadores, intsemaforoParticipantes, intsemaforoBombo, intsemaforoAforo, Sala *sala	Imprime el estado de los semáforos.
bajarSemaforoDeSala	Vacía	intsemaforoGanador, intsemaforoTarjetas, intsemaforoCantor, intsemaforoJueces,	Se encarga de bajar todos los semáforos.

		intsemaforoIdentificadores, intsemaforoParticipantes, intsemaforoBombo	
subirSemaforoDeSala	Vacía	intsemaforoGanador, intsemaforoTarjetas, intsemaforoCantor, intsemaforoJueces, intsemaforoIdentificadores, intsemaforoParticipantes, intsemaforoBombo	Se encarga de subir todos los semáforos.
finalizarCantador	Vacía	Sala *sala	Se encarga de finalizar al cantadory de liberar los recursos que este haya reservado..
finalizarJuez	Vacía	pid_t jueces[MAX_PARTICIPANTES/MAX_JUGADORES_JUEZ +1]	Se encarga de matar a todos los juecesy de liberar los recursos que estos hayan reservado.
manejadorCantador	Vacía	intsigNum	Se encarga de manejar las señales que puede recibir el cantor.
manejadorJuez	Vacía	intsigNum	Se encarga de manejar las señales que puede recibir un juez.
manejadorPostor	Vacía	intsigNum	Se encarga de manejar las señales que puede recibir un postor.
escucharNumerosManual	Vacía		Sirve para escuchar los números en modo

			manual.
tarjetaLlena	Entera	Tarjeta tarjeta	Verifica si la tarjeta está llena.
escucharNumerosAutomatico	Vacía	G_Tarjeta *G_tarjeta	Sirve para escuchar los números en modo automático.

Consideraciones.

- El bombo solo tiene 75 bolas en su interior. Y sus valores van desde el 1 al 75.
- Cada tarjeta tiene solo 25 números.
- Existen solo 60 tarjetas.
- Existen solo 2 modos de juego: manual y automático.

Restricciones

- Solo pueden haber como máximo 30 postores jugando al mismo tiempo.
- Cada participante solo puede tener como máximo 2 tarjetas.
- Cada juez solo puede supervisar a un máximo de 5 jugadores.

Conclusiones

La comunicación entre procesos es una función básica de los sistemas operativos. Los procesos pueden comunicarse entre sí a través de compartir espacios de memoria, ya sean variables compartidas o buffers. La comunicación entre procesos provee un mecanismo que permite a los procesos comunicarse y sincronizarse entre sí, normalmente a través de un sistema de bajo nivel de paso de mensajes.

Un semáforo es unatipo abstracto de datos que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento del sistema o variables del código fuente) en un entorno de multiprocesamiento.

Cada proceso puede crear una o más estructuras llamadas colas; Cada estructura puede esperar uno o más mensajes de diferentes tipos, que pueden venir de diferentes fuentes y pueden contener información de toda naturaleza; todos pueden enviar un mensaje a las colas de las cuales conozcan su identificador. Los procesos pueden acceder secuencialmente a la cola, leyendo los mensajes en orden cronológico (desde el más antiguo, el primero, al más reciente, el último en llegar), pero selectivamente, esto es, considerando sólo

los mensajes de un cierto tipo: esta última característica nos da un tipo de control de la prioridad sobre los mensajes que leemos.

El uso de las colas es similar a una implementación simple de un sistema de correo: cada proceso tiene una dirección con la cual puede operar con otros procesos. El proceso puede entonces leer los mensajes mandados a su dirección en un orden preferencial y actuando de acuerdo con lo que le ha sido notificado.

La sincronización entre dos procesos puede ser así llevada a cabo simplemente usando mensajes entre los dos: los recursos seguirán aun teniendo semáforos para permitir a los procesos conocer sus estados, pero la temporización entre procesos será realizada directamente. Entenderemos de forma inmediata que el uso de las colas de mensajes simplifica muchísimo lo que en un principio era un problema extremadamente complejo.