Tomás Raiti, 62.887
Tobías Noceda, 63.422
Lucas Lonegro Gurfinkel, 63.738

Trabajo práctico especial – Arquitectura de Computadoras

Organización de kernel:

En cuanto a la organización de la sección de kernel del trabajo, se intentó seguir lo más posible el estándar de Linux y de las librerías en C. Algunas decisiones tomadas por los integrantes que nos gustaría comentar respecto de esta sección de la entrega son las siguientes.

- Manejo de los buffers de stdin, stdkey y stderr
- Implementación de memoria dinámica.
- Blue Screen (BSOD) en el manejo de excepciones.
- Comportamiento de la letra ‘ñ’

Manejo de buffers:

El manejo de buffers de stdin y de stderr es prácticamente idéntico, el mayor motivo de su separación es evitar que se mezcle la información que el usuario recibirá del teclado y la que recibirá por errores o alguna otra razón (por ejemplo el registers dump) directamente de kernel.

El manejo de estos dos es mediante un buffer circular como se propuso, pero en donde si se tomó una decisión, es en el caso límite en el que se llena el buffer, ante esta situación nos encontramos con dos posibilidades. Una era que al dar la vuelta entera, el llenado del buffer siguiera como si nada hubiera pasado y dejara al índice del “lector” del buffer donde este se encontraba, pero esto nos pareció un comportamiento indeseado ya que como el lector puede como máximo leer hasta el índice del escritor, si este lo excede por ejemplo solo en una letra, el lector ahora solo podría leer el último carácter ingresado perdiéndose todo el resto de información ingresada, en cambio, lo que se hizo es que ante esta situación, el “escritor” siguiera avanzando en el buffer pero que hiciera que el índice del lector también avance junto con él, generando así que la pérdida de información sea menor y que si bien haya información perdida que nos pareció inevitable, es mucho menor y la información que si lea será incompleta pero al menos estará en el orden correcto.

En cuanto a stdkey, es una implementación que actualmente no es usada por userspace, pero nos pareció un comportamiento que valía la pena agregar al kernel ya que éste permite manejar el apretado de varias teclas en simultáneo y que estas tengan efecto en simultáneo, este comportamiento nos pareció útil por ejemplo para la introducción que requieran que mediante el uso de teclas, más de un jugador deba moverse activamente en la pantalla (esto no es necesario en nuestro caso ya que en el juego implementado, el snake, uno no debe indicarle constantemente a la serpiente en qué dirección ir sino que esta se mueve constantemente en línea recta y uno solo debe indicar los cambios de dirección), en la implementación se optó por brindar un máximo de 17 teclas en simultáneo porque nos pareció que esta cantidad sería más que suficiente. En la implementación lo que se tiene es un arreglo de caracteres que se encuentra vacío si no hay letras presionadas pero que ante un cambio en el teclado se actualiza dejando en él siempre un arreglo con todas las teclas apretadas en ese instante.

Implementación de la memoria dinámica:

Si bien sabemos que esta implementación no era estrictamente necesaria, nos pareció útil ya que al tener espacio limitado en memoria para todo lo que es el trabajo (código, data, etc) esta implementación nos permitía usar más eficientemente la memoria permitiendo al usuario

actuar como si tuviera más. Ya que si bien hay información constante y necesaria a lo largo de todo el uso del SO, hay información que puede ser descartada inmediatamente después de ser usada y esto nos permite reutilizarla con facilidad dando la sensación de tener más memoria al usuario.

La implementación se llevó adelante dividiendo en 2 partes iguales la zona de memoria y utilizando una de las mitades como mapeo de la otra, en un inicio, se inicializa toda la zona de mapeo de memoria en cero ya que no hay memoria alocada. Lo que se hace a partir de ahí es indicar en el inicio de la zona que se quiere alocar, el número de espacios reservados en la zona de memoria mapeada, con eso, uno se asegura que al querer alocar un nuevo bloque de memoria, este no se aloque donde ya había algo. Para el caso del free, simplemente se reemplaza el número que indicaba la cantidad de memoria alocada por delante con un cero para que ahora sí, al querer reservar más memoria en otro malloc, ese bloque se encuentre disponible.

Ejemplo no real a modo explicativo:

Mapeo

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Memoria

Paso 1:

aux = malloc(4);

aux = “hola”;

Mapeo

4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	O	l	a	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Memoria

Paso 2:

aux2 = malloc(3);

aux2 = “que”;

Mapeo

4	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h	o	l	a	q	u	e	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?	¿?

Memoria

En este caso, si ahora se hiciera: free(aux) el 4 se reemplazaría por un 0 que “liberaría” el espacio en memoria que contiene “hola”. Malloc lo que hace en cada invocación es buscar una secuencia de N ceros consecutivos y cuando se encuentra un número $n \neq 0$, salta n posiciones para reiniciar su contador. Calloc permite hacer una copia de memoria alocada en una posición y busca pegarlo en una posición donde haya la nueva cantidad deseada de espacios libres y libera el espacio previamente ocupado.

Tomás Raiti, 62.887
Tobías Noceda, 63.422
Lucas Lonegro Gurfinkel, 63.738

Blue Screen (BSOD) en el manejo de excepciones:

Para poder informar que se produjo una excepción, se optó por una pantalla azul que permanece en pantalla durante unos segundos luego de que se produzca la misma, en el caso de tratarse de una de las excepciones que deben manejarse, se indica en ella el número de la excepción producida. En cuanto al contenido de los registros al producirse la excepción, esta información se guarda en stderr y puede accederse al reiniciarse accediendo al módulo que permite ver el contenido de los registros la última vez que se pidió registrarlos.

Comportamiento de la letra ‘ñ’:

Ya que Keyboard Layout es un array de los valores de las teclas del teclado, y para poder manejar caracteres especiales como la Ñ en kernel, se los almacena en formato UTF16 (uinsigned int/uint16_t). Sin embargo, gcc marca con warnings este formato de caracteres ('Ñ'), por lo que se agregó la instrucción pragma para ignorar el mensaje porque es el comportamiento esperado.

Documentación:

Al contar con documentación doxygen en el código, se pudo generar una versión web de la misma el cual también actúa como manual de usuario, a la cual se accede mediante la siguiente URL:

https://retdocs.vercel.app/

Cambio del idioma del teclado y huso horario:

El sistema operativo puede manejar 3 husos horarios y diferentes distribuciones de teclado. Para cambiar entre ES_AR, FR_FR e EN_US se debe presionar la tecla esc. Debido al tiempo, la distribución del teclado permanece igual en las 3 versiones.

Userspace:

Implementaciones de Userspace:

Non-standard-library:

Entrada y salida estándar:

El kernel define dos formas de acceder a entrada estándar: Una que retorna todas las teclas presionadas, una vez por cada presión, y otra que retorna todas las combinaciones de teclas presionadas en cada instante. Mientras que se comentó en la práctica que un teclado intercalaba la tecla comunicada si se mantiene presionada más de una tecla a la vez, en la práctica notamos que este no era el caso, lo que suscitó esta implementación. Sin embargo, la segunda implementación no fue necesaria en Userspace. La primera implementación es manejada por las funciones getChar y readChar, que llaman a la system call read_sys con file descriptor 0 (file descriptor 3 accede al buffer de la segunda implementación). GetChar llama a sys_read hasta recibir un carácter, es decir, hasta que haya un input del usuario. ReadChar retorna un carácter o ‘\0’ de no haber un carácter por leer. Se utiliza la función wait, definida en ‘nstdlib’ como una llamada a la system call ‘halt’ para prevenir el ‘busy-waiting’ en la función getChar.

Al producirse un ‘dump’ de registros, este es leído mediante la función getDumpString que utiliza la system call get_dump. De recibir 0 como la longitud del dump, getDumpString retorna un string vacío.

Gran porción de las funciones de la librería estándar de c fueron implementadas en nstdlib. Mientras que printf y scanf fueron programadas, estas no son utilizadas en userSpace,

pues nstdlib no fue creado con la capacidad de interactuar con el modo video de manera directa en mente. Toda salida estándar directa a modo video es manejada mediante window y painter. En lugar de estas funciones, nstdlib implementa sPrintf, que se comporta como printf y retorna un string formateado en lugar de imprimirlo de manera directa.

sPrintf y sus variantes utilizan la system call malloc para ser independientes de la memoria alocada a los strings que recibe como parámetros. Por esta razón, nstdlib mantiene un registro de los espacios de memoria que reservó mediante malloc. Al llamar a la función freePrints, estos espacios de memoria, junto con el registro que los ordena, son liberados mediante la system call free.

Notablemente, realloc no es implementado en kernel-space, sino que nstdlib crea un pseudo-realloc utilizando las system calls free y malloc programadas en kernel-space. Esta decisión es menos eficiente que la convencional, pero brinda al usuario transparencia acerca del comportamiento de realloc.

Shell se encuentra en un loop infinito (escapado únicamente por el comando “exit\n”), ralentizado por llamadas a la función wait. Mantiene dos buffers, uno estándar y uno de comandos con longitud limitada. Al recibir un caracter, si es ‘\n’, se pasa el buffer de comandos a CommandHandler mediante la función passCommand, se vacía el buffer de comandos, y se agrega el retorno de passCommand al buffer estándar. De ser ‘\b’, si no se encuentra sobre el prompt de comandos (“:~ “), se reduce en uno el índice de ambos buffers y se indica a Painter que borre un caracter. De ser otro caracter, se agrega a ambos buffers. Tras agregar un caracter al buffer estándar, Shell siempre intenta que Painter dibuje el caracter agregado desde la ubicación del cursor actual. De no poder dibujarse, Shell acorta el buffer una línea, llama a la función blank de painter, y redibuja el buffer completo desde la pantalla vaciada. Tras agregar un string al buffer estándar, la shell pregunta a Painter si este string puede ser dibujado desde la ubicación actual del cursor. De ser posible, lo hace dibujar. De no ser posible se dibuja la cantidad posible y se pide input al usuario para llamar a la función blank, acortar el buffer, y redibujar el buffer acortado; este es un comportamiento adicional descrito en mayor profundidad más adelante.

CommandHandler tiene una función de uso externo, handleCommand, que recibe y retorna un string, junto con el booleano mustRedraw, utilizado para indicar a la shell si el comando ejecutado requiere que la shell redibuje el buffer tras ejecutarlo. Tiene un arreglo de Command, estructura con un string ‘código’, un string ‘help’, y un puntero a función que recibe y retorna los mismo parámetros que handleCommand. Al recibir un string, handleCommand lo compara contra los códigos de todos sus comandos almacenados, de encontrar un código que coincide retorna el retorno de ejecutar el puntero a función del comando, pasándole mustRedraw y el string que recibió desplazado hasta la próxima palabra. El string ‘help’ es utilizado por la estructura Command help para producir su string de retorno.

Manejo del tiempo:

La system call get_tick, que retorna un número que es incrementado por kernel cada tick (18 veces por segundo, en la documentación de pure64), es utilizada para crear las funciones timeHasPassed, que sirve para el manejo de velocidad en el juego snake y la función sleep, que se utiliza en las canciones de ‘sing’ y piano. Notablemente, sleep no es una system call, y, a diferencia de la implementación regular, no espera únicamente en kernel-space, sino que utiliza la función wait para prevenir ‘busy-waiting’. La función wait utiliza la system call halt, por lo que gran parte de la espera sí ocurre en kernel-space.

Sonido:

En conjunto con la función sleep, la system call beep es utilizada para crear las funciones play_for y shut, utilizada para las canciones del comando ‘sing’ y el piano. La

desventaja de la función play_for es que no es asincrónica: hace un sleep de la duración de nota indicada. Por lo tanto, el snake usa la system call beep y la función shut de manera directa, actualizando el sonido (o silencio) a reproducir en cada ciclo de juego.

Window y Painter:

La única manera de dibujar a pantalla creada para userSpace es mediante la system call draw, que recibe un arreglo de colores de 32 bytes cada uno en el formato OORRGGBB, las dimensiones de ese arreglo, y la posición en pantalla donde ubicar el vértice superior izquierdo del arreglo. Cada color representa un pixel: OO su opacidad, RRGGBB su color en formato HEX.

Window administra la estructura window, que guarda un arreglo de colores y sus dimensiones. Mediante la función drawWindow, se llama a la system call draw para dibujar el arreglo de una window en una posición dada. Window utiliza las implementaciones de fast_clear y quick_fill_screen de kernel mediante las funciones quickDraw y clear: si kernel recibe dimensiones de una window que exceden ambos el ancho y la altura de la pantalla, copia el arreglo completo a pantalla sin verificar la opacidad de cada color, si recibe como arreglo un puntero null, rápidamente pone todos los pixeles de la pantalla en negro.

Window posee una gran variedad de funciones que permiten el mapeo de ventanas a otras ventanas, de arreglos de char a ventanas mediante mapas de colores, y mapeos parciales de arreglos de char a ventanas. También permite transformaciones, sean factores de escala o rotaciones. La implementación del manejo de dibujo casi únicamente en userSpace permite esta gran variedad de manipulaciones de pixeles. Asimismo, mientras que el kernel maneja un arreglo de pixeles sub-dividido en tres elementos de 1 bytes cada uno, Window manejo un arreglo de pixeles sin sub-dividir, de forma que manipulaciones de pantalla son más rápidas en la implementación Window, con la única desventaja de tener que comunicar estas manipulaciones a kernel mediante system call tras haberlos consolidado.

FontInterface es la interfaz mediante la cual, en conjunto con Window, Painter es capaz de crear dibujos de cada letra cuando recibe instrucciones de la shell/terminal. FontInterface accede a los dibujos en arreglo de char en el compilado fonts para mapear, mediante Window, estos arreglos de char al window de Painter.

Painter es la interfaz mediante la que la shell/terminal escribe en pantalla. Maneja una única estructura window cuyo tamaño es incrementado o reducido mediante malloc si se cambia el tamaño de la letra. Sobre esta única window se pinta cada caracter para luego llamar a la system call draw mediante la función drawWindow. Painter mantiene mediante variables estáticas un cursor que maneja dónde será dibujada la próxima letra. De alcanzar el cursor el vértice inferior derecha de la pantalla, el pintor retornará falso al pedírsele dibujar una letra o un string. Esto indica a la shell que debe llamar a blank, función de Painter que vacía la pantalla y resetea el cursor, y acortar su buffer de forma que entre en pantalla nuevamente.

Snake:

Se modeló un tablero mediante una matriz de casillas. Cada casilla es una estructura que guarda el número del jugador asociado, las direcciones necesarias para su dibujo, su identidad de dibujo, y su vida restante. El ciclo de actualización es manejado por un ciclo while en frontSnake que se rompe cuando backSnake indica que han muerto la misma cantidad de serpientes con las que el juego comenzó. Llamadas a la función wait previenen ‘busy-waiting’. La función timeHasPassed, de nstdlib, permite administrar la velocidad de actualización de backSnake mediante la función update, un llamado a update corresponde con un movimiento de cada serpiente. La función update recibe parámetros de comunicación con front madeApple y deathCount, punteros a int utilizados para comunicar la cantidad de serpientes muertas y de manzanas creadas en cada actualización, también recibe el parámetro snakeCount, que indica la

cantidad de serpientes en el juego. SnakeBack es capaz de administrar n serpientes, pero snakeFront únicamente maneja hasta tres serpientes. De llamar a playSnake con más de tres serpientes, únicamente será posible manejar tres serpientes, el resto morirán estrelladas rápidamente. Este comportamiento no se considera incorrecto, pues no genera excepciones ni rompe el juego, por lo que no es limitado por playSnake, sino por commandHandler.

En cada actualización, snakeBack recorre toda la matriz de casillas, reduce la vida de las casillas con vida distinta de cero, administra las identidades de dibujo de cada casilla dependiendo de su actual identidad de dibujo y su vida. Cuando encuentra una casilla en la que se encuentra la cabeza de una serpiente, calcula la siguiente posición de la cabeza de la serpiente, si es una posición válida para la cabeza de una serpiente, la guarda en un arreglo para convertir esa casilla en cabeza tras haber recorrido todo el arreglo, si en esa posición se encuentra una manzana, se incrementa la vida de la serpiente. Si la posición es inválida, no se guarda su cabeza, se indica la serpiente como muerta para no ser actualizada en el futuro, se reduce la vida de todos los segmentos de la serpiente correspondiente y se cambia su identificador de dibujo a fondo, de manera que frontSnake dibuja esas casillas como fondo y en el próximo ciclo de actualización backSnake identifica todos esos segmentos como casillas sin serpiente. Tras haber recorrido todo el arreglo, se crea una manzana nueva por cada manzana consumida en la actualización. Al ubicar la cabeza de una serpiente en una casilla, se modifica la vida de esa casilla a la vida de la serpiente, asegurando que esa casilla tendrá vida distinta de 0 por tantas actualizaciones como tarde su serpiente en morir o como vida tenga la serpiente.

En cada actualización, snakeFront pasa a snakeBack el último input de cada jugador, para ser registrado como el movimiento de cada serpiente. snakeBack rechaza (y debe rechazar) cambios de dirección de 180°, de manera que las direcciones deben ser actualizadas una única vez por ciclo de actualización para prevenir que dos cambios de dirección de 90° sean registrados sin un movimiento de la serpiente entremedio. SnakeFront luego llama a update, función de backSnake y actualiza la cantidad de serpientes muertas y la puntuación total (no se especifica que se debe mantener una puntuación separada por serpiente). Luego, llama a getBoard de backSnake, que retorna el tablero de juego, y usa la identidad de dibujo, junto con sus direcciones para dibujar fondo, cabeza de serpiente, cuerpo de serpiente, cola de serpiente, giro de serpiente, manzana, o nada. Finalmente, redibuja la puntuación.

Comportamiento adicional de Userspace:

En Userland o Userspace, se decidió agregar comportamiento extra al pedido. El comportamiento agregado es el siguiente:

- Permitir cambiar el fondo y la forma de las serpientes previo al inicio del juego
- Incluir un piano que permite el uso de una octava de piano y un número de canciones que al oprimir un botón dentro del mismo se reproducen por completo.
- Decisiones de comportamiento de la terminal.

Este comportamiento agregado se encuentra debidamente explicado en el comando help de la terminal.

Cambio de fondo y formas de las serpientes en snake:

Ya que qué el fondo y el cuerpo de las serpientes eran a fin de cuentas arreglos o matrices de píxeles se decidió brindarle al jugador de snake la posibilidad de cambiar el fondo y las serpientes con distintas temáticas ya que se decidió por ejemplo agregar un tema relacionado con Mario que cambia el fondo y la forma de la serpiente a algo relacionado con la famosa saga de videojuegos. Para facilitar este proceso se programó un parser de imágenes que a partir de

Tomás Raiti, 62.887
Tobías Noceda, 63.422
Lucas Lonegro Gurfinkel, 63.738

imágenes jpg produce una matriz de char y un mapa de colores en formato c que juntos codifican una versión comprimida de la imagen que Window puede dibujar mediante las funciones fromCharArray y overlayFromCharArray. Este nos pareció un agregado que hace la experiencia de juego del snake mucho más amena y divertida para el jugador.

Inclusión de un piano entre las opciones:

La inclusión de un piano nos pareció una forma útil de demostrar que se comprendió la forma de uso del parlante ya que nos pareció una forma de demostrar que se entendió que se puede afectar la frecuencia con la que se emite el sonido más allá de la capacidad de acceder al sonido del parlante en sí mismo, ya que en los sonidos durante el snake o al inicio del sistema, al ser simplemente un solo sonido podría parecer que se puso un número aleatorio como parámetro de la función.

Decisiones de comportamiento de la terminal:

Ya que se debía permitir cambiar el tamaño de la letra, esto generaba que con ciertos tamaños de letras, el output de ciertos comandos no entrara no solo en el espacio restante de la terminal sino, que no entraban ni aunque se usara toda la pantalla, y ya que se decidió no entrar en el manejo de un historial por complejidad y espacio, se estandarizo que si el output de un comando no entra en el restante de terminal, el “shifteo” del texto de la terminal se da un 10 renglones a la vez y a partir de un input del usuario, es decir se le informa al usuario que se espera un input para ver el resto de los renglones del output del comando y ante cada input se muestra una línea nueva.