Reporte del proyecto MatComInvasion.

Rodrigo Mederos and Josue R. Naranjo

Contents

1. Introduction

En el marco del proyecto de la asignatura de Sistemas Operativos, se ha desarrollado el juego "MatComInvasion", un arcade estilo "Alien Invaders" programado en C. La humanidad enfrenta una amenaza inminente: una invasión alienígena ha comenzado, y la Facultad de Matemática y Computación de la Universidad de La Habana se encuentra bajo ataque. El jugador toma el control de una nave espacial con la misión de repeler a los invasores y asegurar la supervivencia de la humanidad.

Este proyecto pone en práctica conceptos fundamentales de los sistemas operativos, tales como la programación orientada a eventos, la concurrencia mediante hilos, la gestión de memoria y el diseño de estrategias de planificación. Además, se ha implementado un algoritmo de reemplazo de páginas para determinar las trayectorias de las naves enemigas. El desarrollo y la defensa del proyecto se realizarán en un entorno de Arch Linux, proporcionando un entorno de trabajo realista y coherente para el aprendizaje de los conceptos abordados.

2. Funcionamiento del Juego

El juego *MatComInvasion* es un arcade de estilo clásico en el que el jugador controla una nave espacial que debe defender la Tierra de una invasión alienígena. El objetivo es eliminar las naves enemigas antes de que estas lleguen al suelo, todo mientras se evita ser alcanzado por sus disparos.

El juego comienza con la inicialización de la nave del jugador y la configuración del entorno de juego, que incluye la gestión de los enemigos y las balas. Durante el transcurso del juego, las naves enemigas descienden gradualmente desde la parte superior de la pantalla hacia la Tierra. El jugador puede moverse lateralmente y disparar hacia arriba para destruir estas naves enemigas.

El sistema de juego se basa en una serie de bucles donde se realizan las siguientes acciones clave:

- **Actualización de la Nave del Jugador**: La nave del jugador puede moverse a la izquierda o derecha, y disparar balas hacia los enemigos. Cada acción del jugador es capturada y procesada para actualizar la posición de la nave y la dirección de los disparos.
- **Gestión de los Enemigos**: Las naves enemigas son generadas de manera aleatoria y descienden hacia la Tierra. Cada nave tiene una trayectoria definida y puede disparar balas hacia la nave del jugador. El comportamiento y movimiento de estas naves están determinados por un proceso dedicado a cada una.
- **Detección de Colisiones**: El juego cuenta con un sistema de detección de colisiones que verifica si alguna bala (ya sea del jugador o del enemigo) ha impactado en una nave enemiga o en la nave del jugador. Si una nave enemiga es alcanzada por una bala, se desactiva y se elimina del campo de juego. Si el jugador es alcanzado por un disparo enemigo, pierde una vida. Si se pierden todas las vidas, el juego termina.
- **Rendimiento Gráfico**: El campo de juego es representado gráficamente en la consola, mostrando la posición de las naves, las balas, y otros elementos del juego. Cada actualización de la pantalla incluye la posición actual de la nave del jugador, los enemigos activos, y el estado de las balas en el juego.
- **Planificación y Gestión de Procesos**: El juego utiliza un sistema de planificación de procesos para gestionar de manera eficiente las naves enemigas y la nave del jugador. Cada enemigo y la nave del jugador son tratados como procesos independientes que se planifican y ejecutan en función de su prioridad y estado actual. Esto asegura que el juego maneje múltiples naves y disparos de manera concurrente sin afectar el rendimiento.
- **Condiciones de Fin del Juego**: El juego termina si las naves enemigas logran llegar al suelo o si el jugador pierde todas sus vidas. En este punto, se muestra un mensaje de "Game Over" y se detiene el bucle principal del juego.

En resumen, *MatComInvasion* es un juego que combina acción rápida con conceptos avanzados de sistemas operativos, como la gestión de procesos y la concurrencia, lo que ofrece una experiencia de juego dinámica y desafiante.

3. Ejecución del Juego en un Entorno Arch Linux

Para ejecutar el juego, simplemente inicie sesión en la máquina virtual que utiliza Arch Linux como sistema operativo. Una vez en el entorno, abra una terminal y escriba el comando MatComInvasion. Este comando ejecutará automáticamente el juego sin necesidad de navegar por directorios o especificar rutas largas.

Este comportamiento se logra mediante un script de Bash que se ha configurado previamente en el sistema. El script define un alias que asocia el comando MatComInvasion con la ejecución del archivo binario del juego. La configuración del alias se realiza de la siguiente manera:

```
alias MatComInvasion='cd /ruta/Project && ./game'
```

En este script, el alias MatComInvasion se vincula a una serie de comandos que cambian el directorio actual (cd) a la ubicación del ejecutable del juego y, posteriormente, ejecutan dicho archivo (./game). Es crucial que la ruta especificada en el alias (/ruta/Project) sea la correcta, es decir, que apunte al directorio donde se encuentra el archivo ejecutable del juego. De este modo, al escribir MatComInvasion en la consola, el sistema ejecuta automáticamente el juego desde la ubicación especificada.

Esta configuración no solo simplifica la ejecución del juego, sino que también demuestra una comprensión de la automatización en entornos de desarrollo basados en Unix, utilizando herramientas nativas del sistema operativo para mejorar la experiencia del usuario.

4. Mejoras en la Gestión de Procesos y Memoria

A continuación se enumeran las principales mejoras realizadas en la gestión de procesos y memoria para el proyecto *MatComInvasion*:

1. **Creación Dinámica de Procesos:** Implementación de un sistema donde cada vez que aparece una nueva nave enemiga, se crea un proceso correspondiente. Esto permite una mayor modularidad y un mejor control de cada entidad en el juego.
2. **Liberación de Procesos y Memoria:** Cada proceso que representa a una nave enemiga es eliminado una vez que la nave es destruida o llega al final de su trayectoria. Esto libera recursos de memoria, evitando sobrecargas y permitiendo la creación de nuevas naves sin problemas de memoria.
3. **Identificación de Procesos mediante Punteros:** Utilización de punteros como identificadores de procesos para una gestión más eficiente de las entidades en el juego, permitiendo una rápida búsqueda y manipulación de los procesos activos.
4. **Algoritmo de Reemplazo de Páginas:** Implementación de un algoritmo de reemplazo de páginas que optimiza el uso de memoria durante la ejecución del juego, asegurando que las páginas de memoria más relevantes estén siempre disponibles para el sistema.
5. **Gestión de Señales y Comunicación entre Procesos:** Mejora en la comunicación entre procesos mediante señales, asegurando una coordinación efectiva entre las naves enemigas y el jugador, y respondiendo de manera eficiente a los eventos en el juego.

5. Creación Dinámica de Procesos

En el proyecto "MatComInvasion", la creación y gestión de procesos es un componente crucial que permite la concurrencia y el control eficiente de las entidades del juego, como la nave del jugador y las naves enemigas. Esta sección detalla el manejo de procesos en el juego, que se realiza mediante un administrador de procesos (process-manager.c), diseñado para operar en un entorno de Arch Linux.

5.1. Estructura del Administrador de Procesos

El administrador de procesos utiliza colas de prioridad (ProcessQueue) para organizar los procesos en diferentes niveles, lo que facilita la planificación y ejecución de los mismos. Se definen múltiples colas, cada una correspondiente a un nivel de prioridad, permitiendo una gestión ordenada y eficiente de los procesos. Los procesos son instanciados dinámicamente a lo largo del juego, y su ejecución es gestionada mediante señales del sistema, como SIGSTOP y SIGCONT, que permiten pausar y continuar los procesos según sea necesario.

5.2. Creación de Procesos

Los procesos en "MatComInvasion" se crean utilizando la llamada al sistema fork(), que genera un nuevo proceso hijo. Este proceso hijo es responsable de ejecutar las tareas específicas asociadas con la entidad correspondiente en el juego. Por ejemplo, el proceso del jugador es iniciado con el siguiente fragmento de código:

```
pid_t pid = fork();
if (pid == 0) {
    execlp("./game", "game", "player", NULL);
    perror("Error ejecutando el proceso del jugador");
    exit(1);
} else if (pid > 0) {
    addProcess(pid, 0, 0); // Añadir proceso a la cola
} else {
    perror("Error al crear el proceso del jugador");
}
```

Aquí, execlp se utiliza para ejecutar el binario del juego con un argumento específico que indica el tipo de proceso (en este caso, "player"). Similarmente, se crean procesos para las naves enemigas, donde se asocia cada proceso con una instancia de la estructura NaveEnemiga, que almacena los atributos específicos de cada enemigo.

5.3. Gestión y Eliminación de Proceso

Cada proceso se añade a la cola correspondiente mediante la función addProcess, que lo organiza en el nivel de prioridad adecuado. Los procesos pueden ser terminados dinámicamente cuando una nave enemiga es destruida o alcanza su destino. Esto se logra mediante la función removeProcess, que envía una señal SIGKILL al proceso correspondiente y reorganiza la cola para mantener un manejo eficiente de los recursos del sistema:

```
void removeProcess(int enemy_id) {
    for (int i = 0; i < enemy_process_count; i++) {
        if (enemy_processes[i].enemy_id == enemy_id) {
            kill(enemy_processes[i].pid, SIGKILL);
            waitpid(enemy_processes[i].pid, NULL, 0);
            // Reorganizar la lista de procesos
            for (int j = i; j < enemy_process_count - 1; j++)
                enemy_processes[j] = enemy_processes[j + 1];
            }
            enemy_process_count--;
            break;
        }
    }
}
```

5.4. Planificación de Procesos

La planificación de los procesos se realiza mediante la función scheduleProcesses, que asigna tiempo de CPU a cada proceso en función de su nivel de prioridad. La función pausa el proceso en ejecución y selecciona el siguiente en la cola para ser ejecutado, asegurando que todos los procesos reciban tiempo de CPU de manera justa y eficiente:

```
void scheduleProcesses() {
    static int current_process = 0;
    for (int i = 0; i < QUEUE_LEVELS; i++) {
        if (queues[i].count > 0) {
            ProcessInfo pInfo = queues[i].processes[current_pr
            kill(pInfo.pid, SIGCONT);
            sleep(1);
            current_process++;
            if (current_process >= queues[i].count) {
                current_process = 0;
            }
            break;
        }
    }
}
```

Este enfoque de gestión de procesos permite un control robusto y dinámico de las diferentes entidades dentro del juego, optimizando el uso de recursos y manteniendo la fluidez de la experiencia de juego.

6. Gestión de Hilos en MatComInvasion

En el proyecto "MatComInvasion", los hilos (threads) juegan un papel fundamental en la gestión de la concurrencia, especialmente en el manejo de la entrada del usuario y la actualización de la nave del jugador en tiempo real. Esta sección describe de manera detallada el uso de hilos en el proyecto y cómo estos se coordinan para asegurar una respuesta eficiente a las acciones del jugador.

6.1. Estructura y Coordinación de Hilos

El código implementa un patrón de productor-consumidor utilizando hilos en C, donde un hilo es responsable de capturar la entrada del usuario, mientras que otro se encarga de procesarla y actualizar el estado del juego. La sincronización entre estos hilos se gestiona mediante el uso de mutexes (pthread-mutex-t) y variables de condición (pthread-cond-t), asegurando que los recursos compartidos se accedan de manera segura y ordenada.

6.2. Captura de Entrada del Usuario

El hilo productor se encarga de capturar la entrada del usuario utilizando la función -getch(), que espera una tecla del teclado sin necesidad de presionar "Enter". Este hilo, definido en la función get-input, opera de la siguiente manera:

1. Bloquea el acceso al buffer compartido utilizando un mutex (pthread-mutex-lock(mutex-buffer)).
2. Si el buffer está lleno, el hilo espera hasta que esté disponible para escritura mediante pthread-cond-wait(can-read, mutex-buffer).
3. Almacena la entrada del usuario en una variable compartida (user-input-fr) y señala al hilo consumidor que hay nueva información disponible mediante pthread-cond-signal(can-consume).
4. Finalmente, desbloquea el mutex para permitir el acceso a otros hilos.

6.3. Procesamiento de la Entrada

El hilo consumidor, implementado en la función consume-input, se encarga de procesar la entrada del usuario:

1. Bloquea el acceso tanto al buffer de entrada como a la nave del jugador mediante mutexes (pthread-mutex-lock(mutex-buffer) y pthread-mutex-lock(user-ship)).
2. Si no hay entradas disponibles en el buffer, el hilo espera mediante pthread-cond-wait(can-consume, mutex-buffer)
3. Una vez que recibe la señal del productor, verifica si la tecla presionada es válida (corresponde a un movimiento o acción del juego).
4. Si la entrada es válida, el hilo señala que la nave del jugador puede ser actualizada (pthread-cond-signal(can-change-ship)).
5. Finalmente, el buffer se limpia (buffer-index = 0) y los mutexes se desbloquean para que otros hilos puedan acceder.

6.4. Implementación del Manejador de Entrada

La función user-input-handler es responsable de la creación y coordinación de los hilos productor y consumidor. Se utilizan pthread-create para iniciar los hilos, y pthread-join para asegurar que el programa principal espere a que ambos hilos terminen su ejecución antes de continuar. Este manejo asegura que la entrada del usuario se procese de manera concurrente sin interferir con la lógica del juego:

```
pthread_t get_input_tid, consume_input_tid;

if (pthread_create(&get_input_tid, ...) != 0) {
    perror("Error al crear el hilo de get_input");
    exit(EXIT_FAILURE);
}

if (pthread_create(&consume_input_tid, ...) != 0) {
    perror("Error al crear el hilo consume_input");
    exit(EXIT_FAILURE);
}

if (pthread_join(get_input_tid, NULL) != 0) {
    perror("Error al unir el get_input");
    exit(EXIT_FAILURE);
}

if (pthread_join(consume_input_tid, NULL) != 0) {
    perror("Error al unir el hilo consume_input");
    exit(EXIT_FAILURE);
}
```

6.5. Conclusion de la implementacion de hilos

El manejo de hilos en "MatComInvasion" permite una respuesta ágil y eficiente a las interacciones del usuario, asegurando que las acciones en el juego se procesen sin demoras perceptibles. La sincronización mediante mutexes y variables de condición garantiza que los recursos compartidos se gestionen de manera segura, evitando condiciones de carrera y otros problemas de concurrencia, lo cual es esencial para el correcto funcionamiento del juego en un entorno multitarea.