Obligatorio Sistemas Operativos

Entrega Final.

Integrantes:

- *Ignacio Villarreal*
- *Erik Hernández*
- *Emiliano Fau*
- *Gabriel Felipe*

Fecha de entrega: 20 de Junio de 2023.

Índice

Abstract

Part 1

The present report outlines the process undertaken to address the challenge of generating a daily report summarizing user activity on a Linux operating system running on a virtual machine. To achieve this, a set of character processing tools were used. A shell and its corresponding scripting language were selected to automate the generation of daily and weekly reports, as well as the weekly backup of the reports in a compressed format.

The report thoroughly documents the decision-making process that informed the selection of the underlying platform, operating system, and toolset, as well as the justification for each choice and the specific technology stack employed. The report also provides a detailed account of the coding and testing methodology employed to validate the solution.

Lastly, the report offers a critical evaluation of the tools employed in the solution, drawing relevant conclusions that may be useful in the presentation and defense of the report.

Part 2.1

To tackle this second stage, we are faced with the task of selecting a key component of the Operating System and recreating it in a simulated manner. In collaboration with the professors, we will determine the approach that this section will take, the technologies we will employ, and the way in which we will represent it, while also considering the level of complexity we wish to address.

Part 2.2

In the third part of this project, we are given the opportunity to choose a synchronization problem that we have studied in the course and develop a variant based on it. Subsequently, we are required to implement the solution to this variant using a programming language of our choice.
Once we have defined the problem variant, we can proceed to implement its solution using a programming language of our choice. Depending on our preferences and knowledge, we could opt for languages like Python, Java, or C#, in which we have experience to address the synchronization problem at hand.
By completing this third part of the project, we will have demonstrated our understanding of synchronization concepts and developed practical skills by implementing a functional solution using a specific programming language.

Resumen ejecutivo

Parte 1

El presente informe describe el proceso llevado a cabo para abordar el desafío de generar un informe diario que resuma la actividad de los usuarios en un sistema operativo Linux que se ejecuta en una máquina virtual. Para lograr esto, se utilizaron herramientas de procesamiento de caracteres. Se seleccionó un shell y su correspondiente lenguaje de scripting para

automatizar la generación de informes diarios y semanales, así como la copia de seguridad semanal de dichos informes en formato comprimido.

El informe documenta detalladamente el proceso de toma de decisiones que orientó la selección de la plataforma subyacente, el sistema operativo y el conjunto de herramientas, así como la justificación de cada elección y la pila de tecnologías específica empleada. El informe también ofrece una descripción detallada de la metodología de codificación y pruebas utilizada para validar la solución.

Por último, el informe ofrece una evaluación crítica de las herramientas empleadas en la solución, extrayendo conclusiones relevantes que pueden ser útiles en la presentación y defensa del informe.

Parte 2.1

Para abordar esta segunda etapa, nos enfrentamos a la tarea de seleccionar un componente clave del Sistema Operativo y recrearlo de manera simulada. En colaboración con los profesores, determinaremos el enfoque que tomará esta sección, las tecnologías que emplearemos y la forma en que lo representaremos, considerando también el nivel de complejidad que deseamos abordar.

Parte 2.2

En la tercera parte de este proyecto, se nos brinda la oportunidad de elegir un problema de sincronización que hayamos estudiado en el curso y desarrollar una variante basada en él. Posteriormente, se nos solicita implementar la solución a esta variante utilizando un lenguaje de programación de nuestra elección.

Una vez que hayamos definido la variante del problema, podemos proceder a implementar su solución utilizando un lenguaje de programación de nuestra elección. Dependiendo de nuestras preferencias y conocimientos, podríamos optar por lenguajes como Python, Java, C# en los cuales tenemos experiencia para abordar el problema de sincronización planteado.

Al completar esta tercera parte del proyecto, habremos demostrado nuestra comprensión de los conceptos de sincronización y habremos desarrollado habilidades prácticas al implementar una solución funcional utilizando un lenguaje de programación específico.

Objetivos

Parte 1

Con el grupo de trabajo nos planteamos los siguientes objetivos para esta primera parte del proyecto:

- Describir las diferentes opciones de plataformas para instalar un sistema operativo de máquina virtual, justificando la elección realizada.
- Presentar los diferentes shells disponibles en Linux, y explicar por qué se eligió el shell utilizado para el desarrollo del script.
- Analizar el problema de generar un reporte diario y semanal de actividad de usuarios en el sistema, y justificar el uso de herramientas de procesamiento de flujos de caracteres para la generación del reporte.

- Detallar el proceso de creación del script para generar el reporte diario y semanal, incluyendo la elección de las herramientas de procesamiento de flujos de caracteres utilizadas, el diseño de la estructura del reporte y la generación de los archivos de texto.
- Explicar la elección del lugar donde se guardan los archivos generados y respaldados semanalmente, y detallar el proceso de respaldo automatizado y compresión de los archivos.
- Documentar todas las decisiones tomadas durante el proceso de desarrollo del script, incluyendo el stack de tecnologías utilizado, el código y el testing, y justificar cada una de ellas.
- Realizar una evaluación crítica de las herramientas y tecnologías utilizadas en el desarrollo del script, extrayendo conclusiones que puedan ser compartidas durante la defensa del informe.

Parte 2.1

- Desarrollar un planificador (scheduler) que simule la gestión de recursos y procesos en un Sistema Operativo moderno.
- Diseñar un scheduler capaz de manejar la multitarea y permitir la ejecución concurrente de múltiples procesos.
- Implementar algoritmos de planificación, como Round-Robin, Prioridad y Batch, para ofrecer opciones y flexibilidad en la simulación.
- Establecer un equilibrio entre eficiencia y justicia en la asignación de recursos, evitando situaciones de inanición y bloqueo.
- Optimizar el rendimiento del sistema en general, minimizando los tiempos de espera y maximizando la utilización de la CPU.
- Realizar pruebas exhaustivas para asegurar la correcta funcionalidad del scheduler y la simulación de gestión de recursos y procesos.
- Evaluar el rendimiento y la eficacia del scheduler en diferentes escenarios de carga de trabajo y configuraciones de sistema.
- Adquirir un conocimiento profundo sobre los mecanismos fundamentales de asignación y gestión de recursos en un Sistema Operativo, ampliando nuestra comprensión de los sistemas operativos modernos.

Parte 2.2

- Desarrollar una simulación realista de una pista de aterrizaje que refleje los desafíos y decisiones que enfrentan los aviones al aterrizar y despegar en un entorno compartido.
- Implementar el uso de semáforos como mecanismo de sincronización para controlar el acceso de los aviones a las pistas y garantizar un flujo ordenado de tráfico aéreo.
- Coordinar las acciones de los aviones en función de la disponibilidad de las pistas y la presencia de otros aviones en el espacio aéreo, simulando así un entorno realista de gestión del tráfico aéreo.
- Establecer una cola de espera para los aviones cuando ambas pistas estén ocupadas, asegurando que los aviones puedan aterrizar o despegar de manera secuencial una vez que una pista esté disponible.

- Demostrar la comprensión de los principios fundamentales de la sincronización y la coordinación en sistemas concurrentes, utilizando semáforos como una herramienta efectiva para resolver problemas de acceso compartido a recursos críticos.
- Garantizar un flujo seguro y eficiente de aviones en el espacio aéreo y en la pista de aterrizaje, minimizando los posibles conflictos y maximizando la capacidad de la pista.

Descripción del problema

Parte 1

Como objetivo para la primera parte de este proyecto lo que debemos hacer es generar un reporte diario de los registros de inicio de sesión del sistema. Para esto generaremos un archivo diario de log resumen y un archivo semanal de log resumen.

Parte 2.1

Para esta parte, hemos tomado la decisión conjunta de desarrollar el planificador (scheduler) y todo lo que implica su implementación, con el objetivo de simular la gestión de recursos y procesos que subyace en las computadoras modernas. El scheduler desempeña un papel crucial en el Sistema Operativo al asignar y administrar eficientemente los recursos del sistema a los diferentes procesos en ejecución.

Nuestro enfoque principal será diseñar un scheduler capaz de manejar la multitarea, permitiendo la ejecución concurrente de múltiples procesos. Implementaremos algoritmos de planificación, como el Round-Robin, el cual asigna a cada proceso un intervalo de tiempo de ejecución, y luego pasa al siguiente proceso en la cola de espera. También consideraremos otros algoritmos, como el Prioridad y el por Lotes (Batch), para brindar opciones y flexibilidad en nuestra simulación.

La complejidad de este proyecto reside en lograr un equilibrio entre la eficiencia y la justicia en la asignación de recursos, para evitar situaciones de inanición o bloqueo (deadlock) y garantizar un rendimiento óptimo del sistema en general.

Parte 2.2

En esta propuesta, decidimos tomar un enfoque diferente al problema discutido en clase, pero con la misma lógica subyacente. Creamos una simulación de una pista de aterrizaje con varios aviones que necesitan aterrizar, pero no todos pueden hacerlo al mismo tiempo. Hay dos pistas disponibles y los aviones pueden aterrizar o despegar según su trayecto. Para sincronizar las acciones de los aviones, utilizamos semáforos.

Imaginamos una situación en la que hay dos pistas de aterrizaje y los aviones deben tomar decisiones basadas en la disponibilidad de las pistas. Para controlar el acceso a las pistas, usamos semáforos.

Los semáforos actúan como señales de tráfico para que los aviones soliciten acceso a una pista. Si ambas pistas están ocupadas, los aviones esperan en una cola hasta que haya una pista libre y el semáforo correspondiente lo permita.

Este enfoque basado en semáforos nos permitió simular la sincronización y coordinación de los aviones durante el aterrizaje y el despegue. De esta manera, replicamos los principios de gestión del tráfico aéreo, asegurando un flujo seguro y ordenado de los aviones.

Marco teórico

Sistema Operativo

Los sistemas operativos llevan a cabo dos funciones.

Como controlador de recursos: El sistema operativo actúa como intermediario entre el hardware y las aplicaciones de software que se ejecutan en el sistema. Se encarga de gestionar los recursos de hardware, como la memoria, el procesador, los dispositivos de entrada y salida, etc. para garantizar que se utilicen de manera eficiente y equitativa entre las aplicaciones que los necesitan. De esta manera, el sistema operativo proporciona una capa de abstracción entre las aplicaciones y el hardware, lo que permite a los desarrolladores escribir programas independientes del hardware subyacente. Tanenbaum sostiene que su principal tarea es la de llevar un registro de la utilización de los recursos, dar paso a las solicitudes de recursos, llevar la cuenta de su uso y mediar entre las solicitudes de conflictos de los distintos programas y usuarios.

Como máquina extendida: El sistema operativo proporciona una capa de software adicional que permite que las aplicaciones accedan a recursos que no están disponibles directamente en el hardware subyacente, como el sistema de archivos, la red, el control de acceso, la autenticación, entre otros. En este sentido, el sistema operativo actúa como una "máquina extendida" que proporciona un conjunto de servicios adicionales que permiten a las aplicaciones interactuar con el entorno del sistema operativo de manera segura y eficiente.

Distribución Linux

Linux es un sistema operativo de código abierto que se encarga de administrar los recursos del hardware de un sistema, como la CPU, la memoria y el almacenamiento, y establece la conexión entre los componentes de software y los recursos físicos para realizar las tareas.

Las distribuciones de Linux son sistemas operativos construidos en base al kernel de Linux que pueden incluir programas de usuarios, bibliotecas y repositorios. Cada distribución es creada por una comunidad o proveedor específico.

Gracias a que Linux está bajo la Licencia Pública General de GNU (GPL), cualquiera puede ejecutar, estudiar, modificar y redistribuir su código fuente, incluso vender copias modificadas. Esta característica lo diferencia de los sistemas operativos propietarios como Unix, Microsoft Windows y MacOS, los cuales tienen limitaciones en cuanto a su modificación.

Shell

Se denomina shell al intérprete de comandos de UNIX. Es la interfaz principal entre un usuario y el sistema operativo a nivel de la terminal. Existen diversos shells, tales como sh, bash, ksh, entre otros.

Al iniciar sesión, se inicia un shell que contiene la terminal como entrada y salida estándar. Comenzando por escribir el indicador de comandos, también conocido como prompt (un carácter, por ejemplo un signo de dólar) el cual indica al usuario que el shell está en espera de aceptar un comando.

Máquina virtual de sistema

Una máquina virtual es un software que permite emular un sistema operativo sobre el sistema operativo base. Este tiene los mismos componentes de hardware que una computadora normal, pero todos ellos son virtuales y se seleccionan en el software de la máquina virtual. El sistema operativo emulado utiliza los componentes de la computadora principal, o también llamado hipervisor.

Scripting

Se trata de escribir una secuencia de comandos en un archivo para luego ejecutarlos con el fin de automatizar procesos. Estos se pueden designar tanto para ser ejecutados al inicializar el sistema operativo, como también manualmente, entre otras opciones.

Bash shell

La linea de comandos de linux, mejor reconocida como la terminal, contiene un programa llamado shell. Si bien esta puede ser configurada y modificada a gusto del consumidor, la shell por defecto en CentOS, que es la que utilizaremos en nuestro caso, es GNU Bourne-Again Shell (bash).

Al interactuar con la shell, esta nos devuelve un signo de pesos “$” si se trata de un usuario no root o una almohadilla “#” si se trata del usuario root, en ambos casos significa que está en espera de un comando enviado por el usuario enviado por el usuario. Esto es conocido como la shell prompt.

Bash script

Un script de bash es considerado como una serie de comandos escritos en un archivo de texto, que luego serán leídos y ejecutados línea por línea por la bash. Un script de bash puede ser ejecutado diversas veces y por convención terminan en “.sh”. A su vez, los scripts se identifican con un shebang (el nombre que reciben el par de caracteres “#!”) seguidos de la ruta de la shell de bash. Esto siempre se encuentra en la primera línea del script.

Scheduler

Según el libro "Sistemas Operativos Modernos" de Andrew S. Tanenbaum, el scheduler, también conocido como planificador, es un componente crucial en un sistema operativo encargado de tomar decisiones sobre la asignación del procesador a los procesos en ejecución. Su función principal es determinar cuál será el próximo proceso que obtendrá el control del procesador.

El scheduler tiene como objetivo optimizar el rendimiento del sistema al maximizar la utilización del procesador y garantizar una respuesta rápida a las solicitudes de los usuarios. Para lograr esto, se basa en diferentes políticas y algoritmos de planificación.

Existen varias políticas de planificación utilizadas en los sistemas operativos, entre las cuales se encuentran:

1. Planificación por prioridad: En este enfoque, cada proceso se asigna una prioridad y se les da acceso al procesador de acuerdo con su prioridad. Los procesos con mayor prioridad obtienen el control del procesador antes que los de menor prioridad. Esta política asegura que los procesos más importantes o urgentes sean atendidos primero.
2. Planificación Round-Robin: En este esquema, se asigna a cada proceso un intervalo de tiempo llamado "quantum". Los procesos se ejecutan secuencialmente durante un quantum y luego se produce un cambio de contexto para permitir que otro proceso tome el control del procesador. Esta política garantiza una distribución justa del tiempo de CPU entre los procesos y evita que un proceso acapare el procesador durante mucho tiempo.
3. Planificación basada en la realización de lotes (Batch): En esta política, los procesos se agrupan en lotes y se ejecutan en secuencia. Cada lote se ejecuta por completo antes de pasar al siguiente. Es común en sistemas que procesan grandes cantidades de datos por lotes, como procesamiento de transacciones o procesamiento de archivos por lotes.

Procesos

Un proceso se refiere a un programa en ejecución en un sistema operativo. Los procesos pueden ser de diferentes tipos, tales como procesos de usuario, procesos de sistema, entre otros. Cada proceso tiene un identificador único, un conjunto de recursos asignados, y puede interactuar con otros procesos a través de la sincronización y la comunicación. Los procesos compiten por el acceso a los recursos de la computadora, lo que puede dar lugar a conflictos y bloqueos. Por lo tanto, el SO debe administrar y coordinar el acceso a los recursos de manera eficiente, a través de políticas de planificación de procesos y de gestión de recursos.

Recurso

En un sistema operativo, los recursos desempeñan un papel fundamental al ser utilizados por los procesos para llevar a cabo sus tareas. Estos recursos abarcan tanto componentes físicos como lógicos de una computadora, y su correcta administración es crucial para garantizar el funcionamiento eficiente y confiable del sistema.

El procesador, es uno de los recursos más importantes. Es responsable de ejecutar las instrucciones de los procesos y realizar cálculos. El sistema operativo debe gestionar el acceso

al procesador y asignar su tiempo de manera justa entre los procesos para optimizar el rendimiento general del sistema.

La memoria es otro recurso esencial en un sistema operativo. Se utiliza para almacenar programas en ejecución, datos y variables temporales. El sistema operativo se encarga de asignar y liberar la memoria de manera eficiente para satisfacer las necesidades de los procesos, evitando conflictos de acceso y maximizando su utilización.

El almacenamiento, tanto en forma de discos duros como de unidades de estado sólido (SSD), proporciona un espacio duradero para almacenar programas, datos y archivos. El sistema operativo administra el acceso a estos dispositivos de almacenamiento, permitiendo la lectura y escritura de datos de manera segura y eficiente.

Los dispositivos de entrada y salida (E/S) son recursos utilizados por los procesos para interactuar con el mundo exterior. Esto incluye dispositivos como teclados, ratones, monitores, impresoras, escáneres y otros periféricos. El sistema operativo facilita la comunicación entre los procesos y estos dispositivos, gestionando su acceso y garantizando una correcta sincronización.

La red, tanto local como global, es otro recurso crucial en los sistemas operativos modernos. Permite la comunicación y el intercambio de información entre computadoras y dispositivos. El sistema operativo gestiona el acceso a la red, asegurando una comunicación confiable y segura entre procesos en diferentes máquinas.

Multitarea

La multitarea es una ilusión creada por el CPU en la que se permite la ejecución concurrente de múltiples tareas aparentemente al mismo tiempo. Aunque el CPU no puede realizar múltiples tareas simultáneamente, su velocidad de procesamiento y su capacidad para realizar rápidos cambios entre tareas generan la sensación de que las tareas se están ejecutando en paralelo.

En un sistema operativo que admite multitarea, se asigna un intervalo de tiempo de CPU a cada tarea, conocido como "quantum", y el CPU alterna rápidamente entre las tareas, asignando un fragmento de tiempo a cada una. Aunque cada tarea puede no completarse en un solo intervalo de tiempo, el cambio rápido entre tareas da la impresión de que todas están ejecutándose simultáneamente.

La multitarea permite maximizar la utilización del CPU y mejorar la eficiencia del sistema, ya que mientras una tarea está esperando un recurso o realizando una operación de entrada/salida, otras tareas pueden avanzar y utilizar el CPU. Además, brinda una experiencia de usuario más fluida y la capacidad de ejecutar múltiples aplicaciones o procesos al mismo tiempo, aumentando la productividad y la capacidad de respuesta del sistema.

Semáforos

En el libro de Andrew S. Tanenbaum, se abordan los semáforos como una herramienta fundamental en los sistemas operativos para la sincronización de procesos y la gestión de recursos compartidos. Los semáforos son variables especiales utilizadas para controlar el acceso a recursos compartidos y coordinar la sincronización entre procesos en un sistema operativo.

Los semáforos son ampliamente utilizados en sistemas operativos para resolver problemas de concurrencia y sincronización, como evitar condiciones de carrera, garantizar la exclusión mutua y coordinar la comunicación entre procesos.

Tanenbaum también discute problemas potenciales asociados con el uso de semáforos, como el bloqueo mutuo y la inanición, y propone soluciones para mitigar estos problemas, como la utilización de semáforos de tipo mutex y semáforos con prioridad.

Problemas de inanición y bloqueo

Inanición: La inanición ocurre cuando un proceso o recurso no puede obtener acceso a los recursos necesarios para completar su ejecución debido a la competencia o priorización de otros procesos. Como resultado, el proceso no puede avanzar, incluso si está en espera activa.

La inanición puede ocurrir en situaciones donde los recursos están asignados de manera injusta o donde hay procesos de mayor prioridad que dominan el acceso a los recursos, dejando a los procesos de menor prioridad en un estado de espera indefinido.

Para mitigar la inanición, es necesario implementar políticas de planificación y asignación de recursos que garanticen una distribución equitativa y justa de los recursos entre los procesos.

Bloqueo (Deadlock): El bloqueo se produce cuando dos o más procesos se quedan atascados esperando recursos que están siendo retenidos por otros procesos en el sistema. En otras palabras, cada proceso está esperando a que se libere un recurso que otro proceso tiene en su posesión, creando un estancamiento en el sistema.

El bloqueo puede ser causado por la falta de sincronización adecuada entre los procesos o por la asignación inapropiada de recursos. Si no se toman medidas para resolver el bloqueo, los procesos involucrados permanecerán en un estado de espera indefinido, sin poder avanzar ni liberar los recursos que retienen.

Para evitar el bloqueo, se utilizan técnicas como la implementación de protocolos de exclusión mutua para garantizar que los recursos sean utilizados adecuadamente por los procesos.

Gestión de procesos

En el libro de Tanenbaum, se aborda la gestión de procesos en los sistemas operativos describiendo los siguientes estados de los procesos:

1. Listo: Cuando un proceso se encuentra en estado listo, significa que está preparado para ejecutarse y está esperando a que el planificador de procesos lo seleccione para su

ejecución. El proceso en estado listo se encuentra en la cola de procesos listos y tiene todos los recursos necesarios disponibles para su ejecución, excepto el procesador.

2. Bloqueado: Un proceso se coloca en estado bloqueado cuando necesita esperar a que se cumpla una determinada condición antes de poder continuar su ejecución. Esto puede ocurrir, por ejemplo, cuando el proceso espera la finalización de una operación de entrada/salida (E/S) o la liberación de un recurso compartido. Cuando un proceso se bloquea, se mueve a la cola de procesos bloqueados y se le retira temporalmente el acceso al procesador. Permanecerá en estado bloqueado hasta que la condición necesaria se cumpla y luego será trasladado nuevamente al estado listo.
3. En ejecución: Cuando un proceso es seleccionado por el planificador de procesos para su ejecución, pasa al estado de ejecución. En este estado, el proceso utiliza activamente el procesador para realizar sus tareas y llevar a cabo sus instrucciones.

Un sistema operativo puede tener varios procesadores o núcleos de procesador, lo que permite la ejecución simultánea de varios procesos en paralelo. Cada procesador se asigna a un proceso en ejecución y puede alternar rápidamente entre diferentes procesos utilizando técnicas de planificación de procesos.

4. Suspendido: En algunas situaciones, un proceso puede ser suspendido, lo que implica que se detiene temporalmente su ejecución y se libera de los recursos del sistema. Esto puede ocurrir por diversas razones, como la necesidad de liberar memoria, un recurso que no está funcionando bien o la priorización de otros procesos.

Cuando un proceso está suspendido, se retira de la memoria principal y se guarda en el almacenamiento secundario, como el disco duro. Esto libera recursos para otros procesos en ejecución. El proceso suspendido puede ser reanudado y devuelto al estado listo en algún momento posterior, donde volverá a estar disponible para su ejecución.

Stack tecnológico

CentOS

Es una distribución estable, manejable, predecible y reproducible derivada de las fuentes de Red Hat Enterprise Linux (RHEL), con la cual busca ser funcionalmente compatible.

Se trata de un software libre impulsado por un equipo de desarrolladores principales apoyados por una comunidad de usuarios activa, cuyo objetivo es aportar una plataforma base con el fin de lograr que las comunidades de código abierto se desarrollen.

Ventajas y Desventajas de CentOS Linux

CentOS es una distribución Linux ideal para servidores debido a su alta estabilidad. Se basa en RHEL, una de las mejores distribuciones para servidores, y se caracteriza por tener una selección cuidadosa de paquetes que garantiza la estabilidad del sistema. En los repositorios por defecto solo se encuentran paquetes con versiones estables de los mismos.

Otra ventaja de CentOS es su velocidad, ya que al eliminar paquetes innecesarios e instalar solo lo esencial, se obtiene un sistema más liviano. Además, el kernel, los módulos y los

servicios más utilizados están configurados para maximizar el rendimiento del hardware del servidor.

CentOS es un sistema muy confiable debido a su estructura robusta. Al utilizar menos paquetes, las actualizaciones son menos frecuentes que en otros sistemas, lo que simplifica la administración del sistema.

En cuanto a respaldo y soporte, aunque CentOS no cuenta con un soporte oficial como RHEL o Ubuntu, cuenta con una amplia comunidad y una extensa documentación disponible. Además, la mayoría de la documentación de RHEL es compatible con CentOS, ya que se basa en ella.

VirtualBox

Es un virtualizador de propósito general para hardware x86, con énfasis en servidores, máquinas de escritorios y embebidos. Al usar un virtualizador, se permite instalar sistemas operativos “invitados” sobre el sistema operativo “host”.

Virtualización

La virtualización es una tecnología que simula un entorno el cual imita el hardware de la computadora principal, con el fin de ejecutar en una misma maquina varios sistemas operativos y aplicaciones independientes de la maquina principal.

La virtualización se utiliza principalmente para la ejecución múltiple de software en una misma máquina virtual, facilitando el uso de varios sistemas en un mismo pc y por ende, ahorrando el costo de tener una computadora diferente para cada sistema. Además, al ser independientes, permite realizar pruebas sin correr el riesgo de dañar el sistema principal.

Bash

Bash es un intérprete de comandos y lenguaje de programación integrado que corre bajo el macroprocesador Shell de Unix.

Este programa ejecuta una a una las órdenes que el usuario pone en una ventana de texto o las que se encuentran contenidas en un script o bash script (archivo con todas las instrucciones), para luego devolver los resultados.

Bash fue diseñado por Stephen Bourne en 1977 y tuvo su primera aparición en Unix v7. En la actualidad, Bash es el intérprete predeterminado en gran parte de los sistemas GNU/Linux y funciona en la mayoría de los sistemas de Unix.

Nano

Nano es un editor de texto diseñado para ser utilizado en la terminal. Este programa forma parte del proyecto GNU y ofrece una interfaz de usuario intuitiva y sencilla, que permite editar archivos de texto de manera eficiente. Nano es compatible con varios idiomas y sistemas operativos, lo que lo hace una herramienta útil para tareas de edición de texto generales y para programación en diferentes lenguajes.

Cron

Es un demonio que lee un archivo de texto que contiene otros comandos o scripts a ejecutar en determinada hora y con determinados permisos. Estos comandos o scripts son ejecutados en segundo plano. verifica minuto a minuto este archivo. El crontab es usado para automatización de procesos como respaldo de datos y actualizaciones del sistema operativo.

Java

Java es un lenguaje de programación multiplataforma orientado a objetos que se ejecuta. Java es actualmente el lenguaje de programación más popular para los desarrolladores de aplicaciones. Java se encontraba entre los primeros lenguajes de programación orientados a objetos. Un lenguaje de programación orientado a objetos organiza su código en torno a clases y objetos, en lugar de funciones y comandos.

IntelliJ IDEA

En términos generales, podríamos describir IntelliJ IDEA como un entorno de desarrollo integrado (IDE, por sus siglas en inglés) ampliamente reconocido.
Ahora, ¿qué significa IDE? Para entender el concepto de IntelliJ IDEA, es importante comprender que un IDE es una herramienta de software diseñada para proporcionar diversos servicios que ayudan y mejoran el proceso de trabajo de los desarrolladores.
Podemos destacar que este IDE ha estado presente en el mercado del desarrollo de aplicaciones durante muchos años, desde su lanzamiento público en 2001. A lo largo del tiempo, ha servido como base para otros entornos de desarrollo.

Swing GUI

Swing GUI (Graphical User Interface) es una biblioteca de componentes gráficos y herramientas de desarrollo de interfaces de usuario en Java. Forma parte del conjunto de bibliotecas y herramientas proporcionadas por Java Foundation Classes (JFC) para construir aplicaciones de escritorio.

Swing GUI ofrece una amplia gama de componentes gráficos, como botones, cajas de texto, etiquetas, paneles, menús desplegables y muchos otros elementos, que permiten crear interfaces visuales interactivas y atractivas. Estos componentes se pueden organizar y combinar en ventanas, marcos y contenedores para construir la estructura visual de una aplicación.

Junit

JUnit es un framework de pruebas unitarias para el lenguaje de programación Java. Es una herramienta ampliamente utilizada en el desarrollo de software para probar el funcionamiento de pequeñas unidades de código, como métodos y clases, de manera aislada y automatizada. Esto implica dividir el código en pequeñas unidades independientes, como métodos o clases, y probar cada una de ellas individualmente.

Shunit2

Shunit2 es un framework desarrollado para realizar pruebas unitarias en scripts y programas escritos en Bash. Permite escribir pruebas de manera estructurada y automatizada. Está inspirado en otros marcos de pruebas populares, como JUnit para Java.

AWS S3

AWS S3 (Amazon Simple Storage Service) es un servicio de almacenamiento en la nube altamente escalable que proporciona un almacenamiento seguro y basado en objetos que permite a las empresas almacenar y recuperar grandes cantidades de datos de manera eficiente. AWS S3 está diseñado para ofrecer durabilidad de objetos del 99.999999999% y disponibilidad del 99.99%. Esto significa que los archivos están altamente protegidos contra pérdidas, y ofrece una variedad de mecanismos de seguridad para proteger los archivos, como el cifrado de datos en reposo y en tránsito, controles de acceso y autenticación de usuarios. Desde un punto de vista económico, solo se paga por el almacenamiento que es consumido y los datos que son transferidos. No hay tarifas mínimas ni contratos a largo plazo.

Modelos, alternativas a considerar, decisiones tomadas

Sistema Operativo

Decidimos utilizar CentOS 7 como sistema operativo debido a que tiene un enfoque hacia servidores. Además, tiene un enfoque hacia la seguridad debido a que contiene SEL (Security Enhanced Linux) que tiene un control de acceso obligatorio y un firewall robusto. Existen otras alternativas como Ubuntu server. Además, tiene soporte hasta el 2024.

Virtualización

Para virtualizar existen dos aplicaciones, VirtualBox y VMWare. La ventaja que tiene VirtualBox es que virtualiza el hardware y el software, y que es open source. VMWare es un poco más rápido, pero es un software privativo. Por lo tanto, decidimos utilizar VirtualBox.

Shell a utilizar

Decidimos utilizar Bash ya que es el intérprete que viene por defecto en Linux, y si se decide migrar el script a otra distribución, no se van a tener que hacer cambios sobre el mismo.

Automatización

Decidimos utilizar cron ya que es la única herramienta para automatizar tareas que viene por defecto en distribuciones Linux. Un punto para considerar es que cron ejecuta los comandos solo si la maquina esta encendida. Es decir, si un script se debe ejecutar todos los días a las 8:00 de la mañana y la maquina se prende a las 9:00, cron no va a ejecutar el script ese día. Al considerar que el sistema operativo va a estar instalado sobre un servidor, y este va a estar corriendo 24 horas por día, esto no es un problema.

Conjunto de datos a guardar

Consideramos que el conjunto de datos que tenemos que guardar es el inicio de sesión de cada usuario e intentos fallidos de cada usuario, que están en el log auth.log, y los archivos creados y eliminados por cada usuario.

Respaldos

Existen varias formas de respaldar datos, cada una con sus ventajas y desventajas, cada una explicadas a continuación. Una opción es respaldar sobre un pendrive o un disco duro externo. La ventaja es que, al ser muy pequeño, se puede llevar a todos lados y además es muy barato. La desventaja es que este debe estar junto a el servidor, por lo menos al momento de hacer el respaldo, y si le llega a pasar algo físicamente al lugar donde esta, se puede perder el pendrive y el servidor. Se podría usar un NAS que este en otro edificio, pero este tiene un costo muy alto. Otra alternativa es almacenarlo en la nube, pero es necesario una conexión estable a

internet y confiar que el proveedor mantenga estándares para proteger esos datos, y la forma en la cual se nos cobra por su uso. En base a lo anterior, consideramos que utilizar una nube, como AWS y su servicio S3 es la mejor opción para esta aplicación donde estamos respaldando archivos de texto de pocos kb cada uno, y su política de precios se basa en el almacenamiento usado, es decir, permite tener escalabilidad con poca inversión.

IntelliJ IDEA

La elección de utilizar IntelliJ IDEA se fundamenta en varios factores. En primer lugar, es un software gratuito, lo cual lo hace accesible para todos los miembros del equipo. Además, todos los integrantes cuentan con experiencia previa en su uso. Por último, se destaca por ofrecer una excelente implementación para el lenguaje de programación Java.

Java

La decisión de utilizar Java se basa en el hecho de que todos los miembros del equipo poseemos experiencia en este lenguaje. Además, debido a su naturaleza orientada a objetos, Java ofrece ciertas ventajas en comparación con otros lenguajes a la hora de implementar el problema.

Swing GUI

Decidimos usar Swing debido a que una de las características clave de Swing es su naturaleza liviana y altamente personalizable. Los componentes Swing están escritos completamente en Java y no dependen de las bibliotecas nativas del sistema operativo, lo que garantiza la portabilidad y consistencia visual en diferentes plataformas.

Con Swing, se pueden crear interfaces de usuario interactivas mediante la manipulación de eventos, como clics de botón o entrada de teclado. También se pueden utilizar administradores de diseño para controlar la disposición y la apariencia de los componentes en la pantalla.

Swing es de las más utilizadas en Java, ofreciendo flexibilidad y facilidad de uso para crear interfaces gráficas intuitivas y funcionales.

Implementación

Parte 1

Para realizar esta primera parte, lo que hicimos fue separar la funcionalidad que queríamos implementar, generar logs diarios y semanales con ciertos datos, en tres scripts diferentes.

El primer script es usuariosValidos.sh (Anexo 1), este lo que hace el script busca usuarios en el archivo /etc/passwd que tengan un ID de usuario mayor o igual a 1000 (ya que en centos el id es mayor a 1000 para todos los usuarios) y que no tengan la palabra "nologin" en el campo de inicio de sesión, ya que indica que el usuario es un demonio del sistema. Luego, muestra el nombre de usuario, el ID de usuario y la ruta de inicio de sesión de aquellos usuarios que cumplan con estas condiciones.

El segundo script es inicioEnFecha.sh (Anexo 3), el script busca ciertos patrones en un archivo especificado por la variable $3, que refiere a la fecha pasada por parámetro al script. Dependiendo del valor de las variables $2 y $3 (que es la fecha en día y mes), busca líneas que contengan una combinación específica de texto. Luego, filtra las líneas que contienen algunas cadenas de texto específicas ('session opened', 'password check failed' o 'incorrect password'). El resultado final se mostrará en la salida.

El ultimo script es el main.sh (Anexo 4), el script funciona de la siguiente manera, primero se define variables para la ruta del directorio principal, la fecha de ayer y el número de semana correspondiente a esa fecha. Luego crea directorios basados en la semana actual y la fecha actual en la ruta especificada. Ejecuta el script inicioEnFecha.sh (Anexo 3) pasando como argumentos el mes y el día obtenidos de la fecha de ayer. Itera sobre una lista de usuarios válidos obtenida del script usuariosValidos.sh (Anexo 1). Por cada usuario, crea un archivo de texto vacío con el nombre del usuario en la ruta correspondiente a la semana y la fecha actual. Obtiene el directorio de inicio del usuario y busca todos los archivos en ese directorio. Si existe un archivo del usuario en la semana y fecha anteriores, se comparan los archivos para determinar las diferencias. Si hay diferencias, se guarda el resultado en el archivo correspondiente a la semana y fecha actual, agregando un + si se agregaron nuevas cosas o un – si se eliminaron. Si no hay diferencias, se guarda el archivo completo. Si no existe un archivo del usuario en la semana y fecha anteriores, se guarda el archivo completo con un signo "+" al inicio de cada línea. El proceso se repite para todos los usuarios válidos. Una vez finalizado, comprimimos los archivos con el comando tar -czvf archivo.tar.gz /ruta/archivo.

Automatización

Para que se generen los archivos todos los días a las 4:00 de la mañana, pusimos en el crontab lo siguiente:

0 4 * * * main.sh.

Parte 2.1

Scheduler

Un scheduler es una parte del sistema operativo encargado de la ejecución de tareas y procesos. Su objetivo principal es determinar el orden y el momento de ejecución de las tareas, asignar los recursos del sistema y coordinar la ejecución de tareas concurrentes.

El scheduler es responsable de administrar la asignación de recursos de manera eficiente y garantizar que las tareas se ejecuten de acuerdo con las prioridades establecidas. Puede manejar diferentes tipos de tareas, como procesos en un sistema operativo, hilos de ejecución en un programa o eventos en una aplicación.

Un scheduler puede utilizar diferentes algoritmos y políticas para tomar decisiones sobre qué tarea ejecutar y cuándo hacerlo. Algunos de los algoritmos comunes utilizados en los schedulers son:

Round Robin: Las tareas se ejecutan en un orden circular, asignando un tiempo de ejecución limitado a cada tarea antes de pasar a la siguiente.

Prioridad: Las tareas se ejecutan según su prioridad asignada. Las tareas con mayor prioridad se ejecutan antes que las de menor prioridad.

FIFO (First-In, First-Out): Las tareas se ejecutan en el orden en que llegan, sin considerar la prioridad.

SJF (Shortest Job First): Las tareas más cortas se ejecutan primero, lo que minimiza el tiempo de espera promedio.

Además, los schedulers pueden implementar políticas de planificación más avanzadas, como la planificación en tiempo real, la planificación basada en eventos o la planificación de tareas dependiendo de la disponibilidad de recursos específicos.

Solución

Para llevar a cabo esta tarea de modelar un scheduler, se realizaron diversas acciones basadas en las lecturas proporcionadas por los profesores. Estas lecturas fueron una fuente fundamental para comprender en profundidad y con precisión las características y capacidades involucradas en el sistema de scheduling. Además, se llevó a cabo una exhaustiva investigación sobre las responsabilidades asociadas al scheduler, lo cual permitió obtener un conocimiento más amplio y detallado del tema.

Contamos con diversas clases para modelar el scheduller, la primera y principal es el propio scheduller la cual para su implementacion decidimos usar diferentes estados para los procesos: "listo", "en ejecución", "bloqueado" y "suspendido" (Anexo 9).

El estado "listo" se utiliza para colocar aquellos procesos que están preparados para ser ejecutados, es decir, aquellos que cumplen con todos los requisitos necesarios para su inicio. Esto implica que se hayan completado todas las tareas previas requeridas y que se disponga de los recursos necesarios.

En cuanto a la ejecución de los procesos, se tiene en cuenta tanto la prioridad correspondiente de cada proceso como su tiempo de ejecución estimado. Para lograr un comportamiento

intercalado y equitativo entre los procesos, se implementó un mecanismo de alternancia basado en dichos criterios. De esta manera, se busca maximizar la eficiencia del sistema y evitar situaciones de bloqueo o inanición de procesos.

Cuando un proceso llega al estado "bloqueado", se aplica una funcionalidad que permite agregar o quitar recursos asociados a dicho proceso. Si un proceso carece de los recursos necesarios en un momento dado, se bloquea hasta que dichos recursos estén disponibles. Esto garantiza un uso adecuado y eficiente de los recursos del sistema.

En cuanto al estado "suspendido", se asigna un valor booleano que indica si el recurso asociado a un proceso está en funcionamiento o no. Si el recurso está averiado o no está disponible por algún motivo, el proceso se traslada al estado "suspendido". En este estado, el proceso queda en espera hasta que el recurso se repare o esté nuevamente disponible. Esto evita la ejecución de procesos en condiciones desfavorables y garantiza un uso óptimo de los recursos disponibles.

En el desarrollo del proyecto, se incluyó una clase adicional llamada "Proceso". Esta clase tiene un parámetro que almacena el estado actual del proceso, el cual puede ser uno de los estados mencionados previamente: "listo", "en ejecución", "bloqueado" y "suspendido". Además, la clase cuenta con dos métodos que permiten la ejecución del proceso.

El primer método se utiliza cuando el proceso no excede el tiempo establecido como "timeout" y se ejecuta de manera completa en una sola pasada. En cambio, el segundo método va decrementando el tiempo de duración del proceso utilizando el tiempo del "timeout". Esto hace que el proceso no se finalice por completo, sino que regrese al final de la cola para su ejecución posterior, hasta que se haya completado su tiempo total de ejecución.

Además, la clase Proceso incluye un atributo que almacena todos los recursos necesarios para su ejecución. Basándose en esta información, el scheduler tomará decisiones sobre si el proceso puede ser ejecutado o si debe ser colocado en otro estado, como "bloqueado" o "suspendido".

Por último, se implementó una interfaz llamada "IRecurso" que define métodos comunes para todos los recursos que la implementen. Esta interfaz permite que los recursos, como impresoras, teclados, monitores, y otros, sean tratados de manera uniforme por el scheduler.

La interfaz "IRecurso" incluye cuatro métodos: dos getters y dos setters. Los getters permiten obtener información sobre el estado del recurso, como si está roto o no, y si está disponible para ser utilizado por otro recurso. Los setters permiten cambiar estos parámetros, lo que facilita la administración y gestión de los recursos por parte del scheduler.

Parte 2.2

Sincronización de pistas de aeropuertos

Este problema implica una o varias pistas que deben utilizarse de manera exclusiva por cada avión, ya que cualquier intento de uso simultáneo podría resultar en una colisión. Además, debemos considerar la posibilidad de aviones que requieren aterrizar de emergencia, y en tales casos, es necesario otorgarles prioridad de acceso a la pista sobre otros aviones. Decidimos

modelar este problema en base al aeropuerto de carrasco. Este aeropuerto cuenta con dos pistas, la pista 01-19 y pista 06-24, y con sus respectivas taxiways (pistas de taxeo).

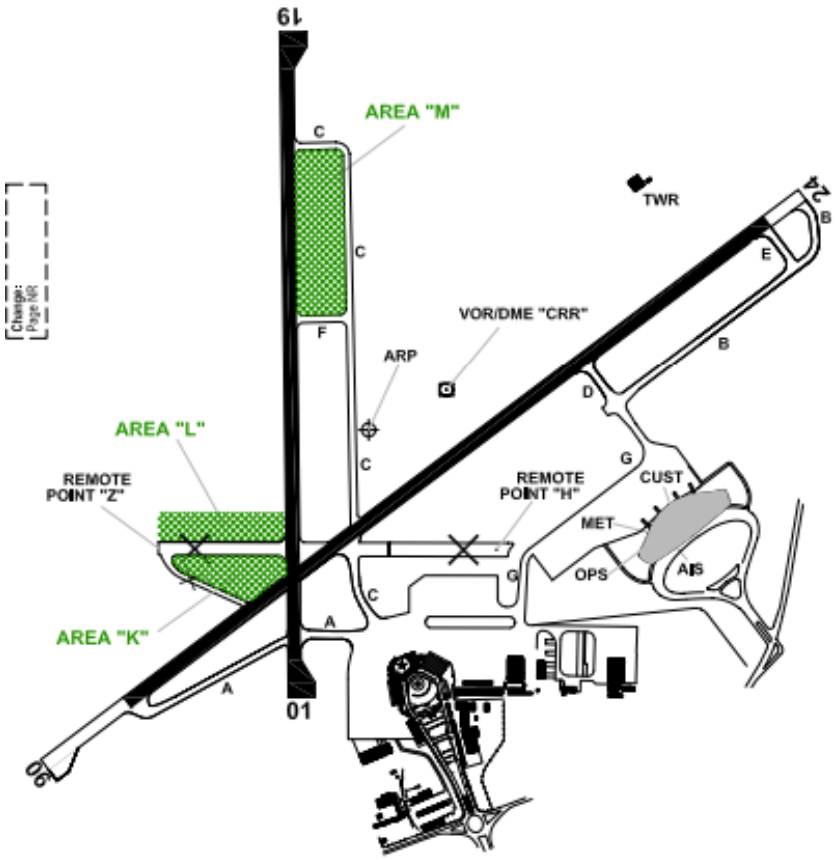

Ilustración 1- Pistas de aterrizaje y taxeo del aeropuerto de carrasco

Con el propósito de resolver este desafío en particular, hemos tomado la decisión de emplear un mecanismo de sincronización conocido como semáforo. Este enfoque nos permite asegurar que la pista de aterrizaje sea utilizada exclusivamente por un avión a la vez, evitando así cualquier posibilidad de colisión. Además, hemos adoptado una estructura de datos conocida como cola de prioridad, la cual nos permite gestionar los aviones que desean utilizar la pista, organizándolos en función de su orden de llegada y su nivel de prioridad. De esta forma, aquellos aviones con mayor prioridad serán atendidos antes que aquellos con menor prioridad.

En cuanto a la implementación práctica de esta solución, hemos elegido utilizar el lenguaje de programación Java. La razón detrás de esta elección radica en que Java ya proporciona una implementación de semáforos, lo cual simplifica considerablemente el desarrollo de esta funcionalidad. Además, contamos con experiencia previa en la programación en Java, lo cual nos brinda un marco sólido para abordar eficientemente este proyecto.

Con el propósito de lograr una manipulación eficiente de cada avión como un proceso individual mediante el semáforo, hemos optado por emplear la clase Thread. Dicha clase nos permite representar tanto procesos como hilos dentro del programa de manera efectiva.

Cada avión, a su vez, cuenta con diferentes estados, los cuales son los siguientes:

- Despegando01, Despegando06, Despegando19, Despegando24: Estos estados representan el avión en proceso de despegue, siendo los números asociados las direcciones y las diferentes pistas utilizadas.
- Esperando: Este estado implica que el avión se encuentra a la espera de que la pista esté disponible para aterrizar.
- Aterrizando01, Aterrizando06, Aterrizando19, Aterrizando24: Estos estados indican que el avión se encuentra en proceso de aterrizaje en una pista específica, siendo los números asociados las direcciones y las distintas pistas utilizadas.
- Taxeando01Porton, Taxeando06Porton, Taxeando19Porton, Taxeando24Porton: Estos estados representan el avión luego de aterrizar, taxeando de la pista especifica por la cual aterrizó, yendo al portón.
- TaxeandoPorton01, TaxeandoPorton06, TaxeandoPorton19, TaxeandoPorton24: Estos estados representan al avión taxeando desde el portón a la pista por la cual va a despegar.
- EnPorton: Este estado representa al avión en el portón.

Solución

Nuestra solución implementa una máquina de estados para los aviones, que se encuentra en la clase AvionAnimator (Anexo 14), donde están recorriendo determinadas posiciones en pantalla hasta que reciban permiso para usar la pista, o lleguen a determinada posición donde se actualiza su estado. Este permiso es dado por un semáforo de capacidad 1 (Anexo 21) que sincroniza el uso de las pistas para que los aviones despeguen, aterricen o esperen antes de cruzar una pista. La decisión de utilizar un único semáforo parte de que no puede haber aviones que despeguen y aterricen al mismo tiempo, y tampoco aviones que crucen la pista cuando otro está aterrizando o despegando. Es decir, no pueden usar el mismo recurso al mismo tiempo. Asimismo, tenemos otro semáforo con capacidad 1 en una clase Recorrer (Anexo 22) que tiene como responsabilidad recorrer la cola de prioridad y cambiar el estado del avión a aterrizando. Utilizamos un semáforo en esta clase para prevenir que vacié toda la cola de prioridad de una sola vez, ya que si lo hace, el usuario no puede pedir para cambiar de prioridad sobre los aviones, porque no hay más aviones en la cola. Al usar un semáforo en Recorrer, este selecciona el avión con mayor prioridad en el momento, le da permiso para aterrizar, y luego de haber aterrizado libera el semáforo para que siga con el próximo. Antes de aterrizar, hacemos un P sobre el semáforo que sincroniza las pistas y al momento que salimos de la pista, hacemos un V para que otro lo pueda usar. Antes de despegar, se hace un P sobre el semáforo que sincroniza las pistas, y al momento que el avión llegó a una posición determinada, se hace un V. Antes de cruzar una pista, cuando se está taxeando desde el portón a la pista 19 por ejemplo, hacemos un P y luego de cruzar un V.

Esta solución usa un patrón de diseño llamado Model-View-Controller. Este patrón divide a una aplicación en 3 partes, donde cada parte tiene una responsabilidad. El Model (Anexo 16) se encarga de gestionar los datos de los aviones (Anexo 13) y define como se pueden accesar y manipular esos datos. El View (Anexo 24) se encarga de presentarle los datos al usuario e

interactúa con él, es la interfaz. El Controller (Anexo 14) hace de intermediario entre el Model y el View ya que recibe datos de entrada del View, lo procesa, y actualiza el modelo acorde. El flujo generalmente se da de la siguiente manera:

1. El usuario interactúa con la View, como hacer clic en un botón.
2. La View notifica al Controller sobre las acciones del usuario.
3. El Controller recibe la entrada, la procesa y actualiza el Model.
4. El Model notifica a la View sobre los cambios en los datos.
5. La View actualiza la interfaz en función a los datos del Model actualizado.

La interfaz de usuario cuenta con una tabla con los aviones en la esquina inferior derecha, con sus respectivos estados y prioridades. Para pedir que un avión aterrice con prioridad, basta hacer click sobre la prioridad del avión en la tabla. Este se va a cambiar a 0, indicando que tiene prioridad. La prioridad 1, es la prioridad por defecto. Luego de aterrizar, su prioridad cambia automáticamente a 1. Además, la interfaz cuenta con un botón sobre la tabla indicando la pista activa. Al clickear sobre el botón, se elige una pista de forma aleatoria y se la pone como activa.

Testing

Parte 1

Durante nuestras pruebas (Anexo 5) para asegurarnos de que el script principal funcione correctamente, hemos llevado a cabo diferentes verificaciones. En primer lugar, nos hemos asegurado de que los archivos correspondientes a varios días en los que se ejecuta el script existan.

Además, hemos realizado comprobaciones de los contenidos de los archivos generados durante la ejecución del script. Esto implica leer el contenido de los archivos y verificar si los datos son los esperados o cumplen ciertas condiciones requeridas por el script.

Para llevar a cabo las verificaciones y pruebas mencionadas anteriormente, hemos decidido utilizar el framework de pruebas unitarias shunit2. Este framework nos proporciona una estructura organizada y herramientas específicas para realizar pruebas en scripts de shell, lo cual resulta beneficioso por varias razones como la facilidad de uso y la facilidad que tiene a la hora de configurarlo.

Parte 2.1

En este caso decidimos realizar pruebas (Anexo 11) sobre procesos cuya prioridad es la misma y sobre procesos cuya prioridad es diferente. Guardamos los valores dados de ejecutar ciertos parámetros en sus respectivos logs y, con el fin de comprobar la veracidad de estos logs, elaboramos logs de ejemplo con los resultados esperados de ejecutar dichos valores.

Parte 2.2

En relación a las pruebas (Anexo 26) realizadas en esta parte, hemos tomado la decisión de implementar dos pruebas específicas. La primera prueba consiste en aterrizar con todos los aviones teniendo la misma prioridad, mientras que la segunda prueba otorga prioridad a un avión en particular para verificar si se ejecuta primero utilizando la cola de prioridad. Es importante destacar que cada una de estas pruebas debe ejecutarse de forma independiente debido a que los no pueden ser reestablecidos, lo que podría afectar la ejecución adecuada de cada prueba.

No hemos llevado a cabo pruebas relacionadas con los aviones en proceso de despegue. Esto se debe a que el orden de los aviones puede variar, lo que dificulta la verificación y predicción del avión que efectivamente realizará el despegue. No obstante, hemos realizado pruebas exhaustivas en diversas situaciones críticas y el programa ha demostrado una adecuada administración de los diferentes procesos.

Conclusiones

Este proyecto nos permitió profundizar nuestros conocimientos sobre CentOS. Aprendimos a configurar el entorno y ejecutar scripts, así como a utilizar las herramientas y comandos necesarios para la administración del sistema. Esta experiencia nos proporcionó una base sólida en el manejo de sistemas operativos basados en Linux.

La simulación del scheduler fue una oportunidad única para comprender en profundidad cómo funcionan los algoritmos de planificación y cómo el sistema operativo asigna los recursos del sistema a los procesos. A través de esta simulación, pudimos experimentar con diferentes políticas de planificación y evaluar su impacto en el rendimiento y la eficiencia del sistema.

Los problemas de sincronización sobre el aeropuerto fue un desafío emocionante. Los semáforos proporcionaron una forma eficaz de coordinar y sincronizar los aviones que interactúan con las pistas en el aeropuerto, evitando conflictos y asegurando un acceso ordenado y seguro. La sincronización mediante semáforos también nos brindó la oportunidad de explorar el concepto de concurrencia y paralelismo. Pudimos comprender cómo los procesos pueden ejecutarse de forma simultánea y cómo los semáforos desempeñan un papel esencial en la coordinación de las actividades de estos procesos.

Gracias a la utilización de semáforos, pudimos resolver de manera efectiva diversos problemas de sincronización de procesos que requieren el acceso a un recurso compartido. Esta solución ha demostrado su eficacia en la resolución del problema planteado, donde se requería una correcta administración de los aviones en un aeropuerto para evitar colisiones. Además, luego de resolver dicho problema, pudimos ver que los semáforos no son útiles solo para resolver problemas de procesos en un sistema operativo, sino que también hemos reconocido su versatilidad y aplicabilidad en una amplia gama de situaciones. Estas herramientas se han revelado eficaces en la administración de procesos e hilos, así como en la resolución de problemas más complejos, como la gestión de un aeropuerto. En este último caso, la correcta implementación de automatizaciones se vuelve fundamental para garantizar la seguridad y evitar posibles colisiones entre aviones.

Podemos concluir que, a lo largo del proyecto, nos enfrentamos a diversos desafíos y obstáculos, pero cada uno de ellos representó una oportunidad de aprendizaje. Nos encontramos con situaciones en las que los procesos se bloqueaban o se producían condiciones de espera innecesarias. Sin embargo, gracias a la utilización adecuada de semáforos, logramos superar estos problemas.

Bibliografía

- Atareao. (s.f.). Nano, un editor de texto para la terminal. Recuperado el 11 de abril de 2023, de https://atareao.es/software/programacion/nano-un-editor-de-texto-para-la-terminal/
- AWS. (2023). Amazon S3. Recuperado el 17 de junio de 2023, de https://aws.amazon.com/es/s3/
- Azure (2023, 17 de mayo). ¿Que es java?. https://azure.microsoft.com/es-mx/resources/cloud-computing-dictionary/what-is-java-programming-language/
- Camik, R. (2009). Rotated Icon. Recuperado el 10 de junio de 2023 de https://tips4java.wordpress.com/2009/04/06/rotated-icon/
- CentOS Project. (n.d.). About CentOS. Recuperado el 11 de abril, 2023, de https://www.centos.org/about/
- cOder. (2019). Use multiple threads to update GUI. Recuperado el 17 de junio de 2023 de https://stackoverflow.com/questions/56714951/use-multiple-threads-to-update-gui
- Crehana. (2022, 29 de enero). VMware vs VirtualBox: ¿Cuál es la mejor opción para la virtualización? [Entrada de blog]. Recuperado el 11 de abril de 2023, de https://www.crehana.com/blog/transformacion-digital/vmware-vs-virtualbox/
- Dinacia. (2021). SUMU – MONTEVIDEO/Intl of Carrasco “Gral. Cesáreo L. Berisso”. Recuperado el 17 de junio de 2023, de https://dinacia.gub.uy/sites/default/files/aip/2022-03/Ad2-9_0.pdf
- Ericksen, J. (2012). Java: calculating the angle between two points in degrees. Recuperado el 13 de junio de 2023 de https://stackoverflow.com/questions/9970281/java-calculating-the-angle-between-two-points-in-degrees
- Hosting Diario. (2021, April 23). CentOS Linux: características, ventajas y desventajas. Recuperado el 11 de abril, 2023, de https://hostingdiario.com/centos-linux/
- IBM. (s. f.). Operating system shells. En IBM AIX 7.2 documentation. Recuperado el 11 de abril de 2023, de https://www.ibm.com/docs/es/aix/7.2?topic=administration-operating-system-shells
- InterviewBit. (2021, 28 de octubre). WMware vs VirtualBox: what´s the difference? Recuperado el 11 de abril de 2023 de https://www.interviewbit.com/blog/vmware-vs-virtualbox/
- KeepCoding Team (2022, 28 de febrero). ¿Qué es Bash (Shell) y cómo funciona?
- KeepCoding. https://keepcoding.io/devops/que-es-bash-shell-y-como-funciona/
- KeepCoding.io. (2022). Qué es IntelliJ IDEA [Blog]. https://keepcoding.io/blog/que-es-intellij-idea/
- Kumar, S. (2023). MVC Design Pattern. Recuperado el 17 de junio de 2023 de https://www.geeksforgeeks.org/mvc-design-pattern/
- Noviantika, G. (s.f). What is CentOS – A beginner´s Guide. Recuperado el 14 de abril de 2023 de https://www.hostinger.com/tutorials/what-is-centos#CentOS_Linux_Distribution_Key_Features
- Oracle. (s.f.). About the Tutorial: Java Swing. Recuperado el 14 de junio de 2023, de https://docs.oracle.com/javase/tutorial/uiswing/start/about.html

- Red Hat. (s. f.). What's the best Linux distro for you? Recuperado el 11 de abril de 2023, de https://www.redhat.com/es/topics/linux/whats-the-best-linux-distro-for-you
- Tanenbaum, A. S. (2016). Sistemas operativos modernos (4ta ed.). Pearson.
- Xataka. (2020, 31 de enero). Máquinas virtuales: qué son, cómo funcionan y cómo utilizarlas. Xataka. https://www.xataka.com/especiales/maquinas-virtuales-que-son-como-funcionan-y-como-utilizarlas
- Zaira Hira (2022, Marzo 31). Shell scripting crash course: How to write Bash scripts in Linux. freeCodeCamp. https://www.freecodecamp.org/news/shell-scripting-crash-course-how-to-write-bash-scripts-in-linux/

Anexos

Anexo 1 – usuariosValidos.sh

```
#!/bin/bash

awk -F ':' '{ if ($7 !~ /nologin/ && $3>=1000) print $1":"$3":"$6 }' /etc/passwd
```

Anexo 2 – diferenciaArchivos.sh

```
#!/bin/bash

diff -u -b --ignore-all-space --ignore-space-change $1 $2|grep -v "+++\|---\|@@\|No newline at end of file"
```

Anexo 3 – inicioEnFecha.sh

```
#!/bin/bash


if [ `echo $2 | cut -c1` -eq "0" ]; then

typeset -i dia=`echo $2 | cut -c2`

grep -i "$1  $dia" $3 | egrep 'session opened|password check failed|incorrect password'

else

grep -i "$1 $2" $3 | egrep 'session opened|password check failed|incorrect password'

fi
```

Anexo 4 – main.sh

```
#!/bin/bash

ruta="/home/ivillarreal/Escritorio/ObligatorioSO-main"

ayer=`date +%Y-%m-%d -d "yesterday"`

semana=`date -d "$ayer" +%W`

mkdir $ruta/"`date +%W`" 2>/dev/null

mkdir $ruta/"`date +%W`/`date +%Y-%m-%d`" 2>/dev/null

touch "$ruta/`date +%W`/`date +%Y-%m-%d`/inicioSesion.txt"

bash $ruta/inicioEnFecha.sh `date -d "$ayer" +%b` `date -d "$ayer" +%d` "/var/log/secure" > "$ruta/`date +%W`/`date +%Y-%m-%d`/inicioSesion.txt"


for user in `bash $ruta/usuariosValidos.sh`
```

```
do
archivoHoy=`mktemp`
archivoAyer=`mktemp`
archivoDiferencia=`mktemp`
archivoCut=`mktemp`
usuario=`echo $user | cut -d: -f1`
touch "$ruta/`date +%W`/`date +%Y-%m-%d`/$usuario.txt"
directorio=`echo $user | cut -d: -f3`
find $directorio > $archivoHoy

if test -f "$ruta/$semana/$ayer/$usuario.txt"; then
        grep -v '^-' "$ruta/$semana/$ayer/$usuario.txt" > $archivoAyer
        cut -c2- $archivoAyer > $archivoCut
        bash "$ruta/diferenciaArchivos.sh" $archivoCut $archivoHoy > $archivoDiferencia

        if [ -s $archivoDiferencia ]; then
                cat $archivoDiferencia > "$ruta/`date +%W`/`date +%Y-%m-%d`/$usuario.txt"
        else
                sed -e 's/^/ /' $archivoHoy > "$ruta/`date +%W`/`date +%Y-%m-
%d`/$usuario.txt"
        fi
else
        sed -e 's/^/+/' $archivoHoy > "$ruta/`date +%W`/`date +%Y-%m-%d`/$usuario.txt"
fi
done
```

Anexo 5 – Tests scripts

```
#!/bin/bash
test_archivo_semana21(){
   ruta="/home/ivillarreal/Escritorio/ObligatorioSO-main"
```

```
    date --set "2023-05-26"
    bash "$ruta/main.sh"

    assertTrue "Existe la carpeta: " " [ -d $ruta/21/2023-05-26 ] "
}

test_archivo_semana22(){
    ruta="/home/ivillarreal/Escritorio/ObligatorioSO-main"
    date --set "2023-05-29"
    bash "$ruta/main.sh"

    assertTrue "Existe la carpeta: " " [ -d $ruta/22/2023-05-29 ] "
}

test_archivo_semana_inexistente(){
    ruta="/home/ivillarreal/Escritorio/ObligatorioSO-main"

    assertFalse "Existe la carpeta: " " [ -d $ruta/21/2023-05-27 ] "
}

test_archivo_(){
    ruta="/home/ivillarreal/Escritorio/ObligatorioSO-main"
    archivo=`mktemp`
    diff -u -b --ignore-all-space --ignore-space-change "$ruta/archivoPruebaTest2" "$ruta/21/2023-05-26/ivillarreal.txt" > $archivo

    tamano=$(wc -c < "$archivo")
    assertTrue "El archivo esta vacio" "[[ $tamano -eq 0 ]]"

}
```

```
. /home/ivillarreal/Escritorio/ObligatorioSO-main/shunit2/shunit2
```

Anexo 6 – IRecurso

```
public interface IRecurso {
    public Boolean siendoUsado();
    public Boolean getEstaRoto();
    public void setEstaRoto(boolean bool);

    public void cambiarEstadoUsando();
    public void cambiarEstadoDisponible();
}
```

Anexo 7 – Impresora

```
public class Impresora implements IRecurso {
    //Creación de variables
    public Boolean siendoUtilizado;
    public Boolean estaRoto;

    //Inicialización de variables
    public Impresora() {
        this.siendoUtilizado = false;
        this.estaRoto = false;
    }

    //Métodos para ver el estado de la impresora (getters)
    @Override
    public Boolean siendoUsado(){
        return siendoUtilizado;
```

```
    }

    @Override
    public Boolean getEstaRoto() {
        return estaRoto;
    }

    //Métodos para establecer el estado de la impresora (setters)
    @Override
    public void setEstaRoto(boolean bool) {
        this.estaRoto = bool;
    }

    @Override
    public void cambiarEstadoUsando() {
        siendoUtilizado = true;
    }
    @Override
    public void cambiarEstadoDisponible() {
        siendoUtilizado = false;
    }
}
```

Anexo 8 – Proceso

```
import java.util.List;
import java.util.PriorityQueue;
public class Proceso implements Runnable, Comparable<Proceso>{
    //Creación de variables
    public long tiempoEjecucion;
    public String id;
```

```
    public int prioridad;
    public List<IRecurso> recursosUsados;
    public Estados estado;
    public String ruta;

    //Inicialización de variables
    public Proceso(String id, long tiempoEjecucion, int prioridad, List<IRecurso> recursosUsados, Estados estado, String ruta) {
        this.id = id;
        this.tiempoEjecucion = tiempoEjecucion;
        this.prioridad = prioridad;
        this.recursosUsados = recursosUsados;
        this.estado = estado;
        this.ruta = ruta;
    }

    public Estados getEstado(){
        return this.estado;
    }

    public void setEstado(Estados estado){
        this.estado = estado;
    }

    //Ejecuta el proceso si su tiempo es menor o igual al del TimeOut
    public void run() {
        ManejadorDeArchivos.escribirArchivo(ruta, "Ejecutando " + id + "...");
        System.out.println("Ejecutando " + id + "...");
        try {
            Thread.sleep(tiempoEjecucion);
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + id + " se termino...");
        System.out.println(id + " terminado.");
    }

    //Ejecuta un proceso si su tiempo es mayor al del TimeOut y tiene que ejecutarse dos o más veces
    public void runMax() {
        ManejadorDeArchivos.escribirArchivo(ruta, "Ejecutando " + id + "...");
        System.out.println("Ejecutando " + id + "...");
        try {
            long currentTime = System.currentTimeMillis();
            Thread.sleep(currentTime);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + id + " se termino...");
        System.out.println(id + " terminado.");
    }

    //M+etodo diseñado para generar una comparación entre procesos teniendo en cuenta su prioridad
    public int compareTo(Proceso otroProceso){
        return Integer.compare(this.prioridad, otroProceso.prioridad);
    }

    //Posibles estados de un proceso
    public enum Estados{
        Listo,
        Ejecucion,
```

```
        Bloqueado,
        Terminado,
        SuspendidoListo,
        SuspendidoBloqueado
    }
}
```

Anexo 9 – Scheduller

```
import java.util.*;
import java.util.concurrent.Semaphore;

public class Scheduller {
    private PriorityQueue<Proceso> colaLista;
    private LinkedList<Proceso> listaBloqueado;
    private LinkedList<Proceso> listaSuspendidoListo;
    private LinkedList<Proceso> listaSuspendidoBloqueado;

    private LinkedList<Proceso> listaProcesosTerminados;
    private Semaphore semaforo;
    private List<IRecurso> recursosDisponibles;
    public String ruta;


    //Inicializamos variables y el semáforo en 1 para que sólo un proceso
    //acceda a la vez.
    public Scheduller(List<IRecurso> recursos, String ruta) {
        this.colaLista = new PriorityQueue<>();
        this.listaBloqueado = new LinkedList<>();
        this.listaSuspendidoListo = new LinkedList<>();
        this.listaSuspendidoBloqueado = new LinkedList<>();
```

```
        this.listaProcesosTerminados = new LinkedList<>();
        this.semaforo = new Semaphore(1);
        this.recursosDisponibles = recursos;
        this.ruta = ruta;
    }


    //Agregamos todos los procesos a las listas y colas correspondientes
    //según su estado.
    public void agregarProceso(Proceso proceso) throws InterruptedException {
        if (proceso.estado == Proceso.Estados.Listo) {
            colaLista.offer(proceso);
        } else if (proceso.estado == Proceso.Estados.Bloqueado) {
            ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + proceso.id + " fue bloqueado...");
            listaBloqueado.offer(proceso);
        }
    }


    //Método principal en el que se ejecutan los procesos, si no se
    //pueden ejecutar pasan a las otras listas, bloqueándolos o
    //suspendiéndolos, o ambas.
    public void ejecutarProcesos() throws InterruptedException {
        while (!colaLista.isEmpty()) {
            long currentTime = System.currentTimeMillis();
            Proceso proceso = colaLista.peek();
            long tiempoRestante = proceso.tiempoEjecucion - currentTime;
            if (tiempoRestante <= 0 && solicitarRecurso(proceso) == true) { //Si el tiempo restante es negativo el proceso debe ejecutarse de inmediato
                semaforo.acquire(); //Adquirimos el semáforo antes de ejecutar el proceso
                proceso.run();
```

```
            liberarRecurso(proceso);
            semaforo.release(); //Liberamos el semáforo para permitir que otro proceso lo adquiera
        }else if (tiempoRestante > 0 && solicitarRecurso(proceso) == true){
            semaforo.acquire(); //Adquirimos el semáforo antes de ejecutar el proceso
            proceso.runMax();
            for (IRecurso r: proceso.recursosUsados) {
                r.cambiarEstadoDisponible();
            }
            colaLista.poll();
            proceso.tiempoEjecucion -= currentTime;
            colaLista.offer(proceso);
            semaforo.release(); //Liberamos el semáforo para permitir que otro proceso lo adquiera
        }
        //Checkeo los bloqueados, para ver si van denuevo a la cola lista
        List<Proceso> procesosParaEliminar = new ArrayList<>();
        for (Proceso p: listaBloqueado) {
            salidaBolqueado(p,procesosParaEliminar);
        }
        listaBloqueado.removeAll(procesosParaEliminar);

        List<Proceso> procesosParaSuspender = new ArrayList<>();
        for (Proceso p: listaSuspendidoListo) {
            salidaSuspendidoListo(p,procesosParaSuspender);
        }
        listaSuspendidoListo.removeAll(procesosParaSuspender);
    }
}

//Chequeo de bloqueados, para sacarlos de este estado y pasarlos
```

```
    //a listo
    public void salidaBolqueado(Proceso proceso, List<Proceso> procesosParaEliminar){
        boolean disponible = true;
        boolean ruptura = false;
        for (IRecurso recursos : proceso.recursosUsados){
            if (recursos.getEstaRoto()){
                ruptura = true;
            }
            if(recursos.siendoUsado()) {
                disponible = false;
                break;
            }
        }
        if (ruptura && disponible == false){
            ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + proceso.id + " fue suspendido...");
            proceso.estado = Proceso.Estados.SuspendidoBloqueado;
            listaSuspendidoBloqueado.add(proceso);
            procesosParaEliminar.add(proceso);
            return;
        }
        if (disponible && ruptura == false) {
            ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + proceso.id + " pasa de bloqueado a listo...");
            proceso.estado = Proceso.Estados.Listo;
            colaLista.add(proceso);
            procesosParaEliminar.add(proceso);
        }
    }


    //Chequeo de suspendidosListo, para sacarlos de este estado y pasarlos
```

```
    //a listo
    public void salidaSuspendidoListo(Proceso proceso, List<Proceso> procesosParaSuspender){
        Boolean ruptura = true;
        for (IRecurso recursos : proceso.recursosUsados){
            if (recursos.getEstaRoto() == false){
                ruptura = false;
            }
        }
        if (ruptura == false){
            ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + proceso.id + " sale del estado suspendido y vuelve al estado listo...");
            listaSuspendidoListo.remove(proceso);
            proceso.estado = Proceso.Estados.Listo;
            colaLista.offer(proceso);
        }
    }

    //Chequea si los recursos de un proceso están disponibles para ser
    //ejecutado, sino lo pone en otros estados, ya sea bloqueado como,
    //suspendido bloqueado.
    public boolean solicitarRecurso(Proceso proceso) {
        boolean disponible = true;
        boolean ruptura = false;
        for (IRecurso r: proceso.recursosUsados) {
            if (r.siendoUsado()){
                disponible = false;
            }
            if (r.getEstaRoto()){
                ruptura = true;
            }
```

```
        }
        if (ruptura == true) {
            ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + proceso.id + " fue suspendido...");
            colaLista.poll();
            listaSuspendidoListo.add(proceso);
            proceso.estado = Proceso.Estados.SuspendidoListo;
            return !ruptura;
        }else if (disponible != false){
            for (IRecurso r: proceso.recursosUsados) {
                r.cambiarEstadoUsando();
            }
        }else{
            ManejadorDeArchivos.escribirArchivo(ruta, "El proceso " + proceso.id + "fue bloqueado...");
            colaLista.poll();
            listaBloqueado.add(proceso);
            proceso.estado = Proceso.Estados.Bloqueado;
        }
        return disponible;
    }

    //Una vez finalizado un proceso, libera los recursos de este
    public void liberarRecurso(Proceso proceso) {
        for (IRecurso r: proceso.recursosUsados) {
            r.cambiarEstadoDisponible();
        }
        proceso.estado = Proceso.Estados.Terminado;
        colaLista.poll();
        listaProcesosTerminados.add(proceso);
    }
```

```
}
```

Anexo 10 – Main Scheduller

```
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        //Se crean los recursos, en este caso, impresoras
        IRecurso r1 = new Impresora();
        IRecurso r2 = new Impresora();
        //El segundo recurso está roto
        r2.setEstaRoto(true);
        //Se agregan los recursos disponibles a una lista
        List<IRecurso> recursosDisponibles = new ArrayList<>();
        recursosDisponibles.add(r1);
        //Se agregan los recursos suspendidos a una lista
        List<IRecurso> recursosSuspendido = new ArrayList<>();
        recursosSuspendido.add(r2);

        //Se crean los procesos
        Proceso p1 = new Proceso("1111",5000,2,recursosDisponibles,
Proceso.Estados.Listo,"src/Logs.txt");
        Proceso p2 = new Proceso("1112",3000,3,recursosDisponibles,
Proceso.Estados.Listo,"src/Logs.txt");
        Proceso p3 = new Proceso("1113",3320,1, recursosSuspendido,
Proceso.Estados.Listo,"src/Logs.txt");
        Proceso p4 = new Proceso("1114",3320,1, recursosSuspendido,
Proceso.Estados.Listo,"src/Logs.txt");
        Proceso p5 = new Proceso("1115",3320,1, recursosSuspendido,
Proceso.Estados.Bloqueado,"src/Logs.txt");
```

```
        //Se inicializan los procesos, el p3 no se debería correr debido a que el recurso que necesita está roto
        Scheduller scheduller = new Scheduller(recursosDisponibles,"src/Logs.txt");
        scheduller.agregarProceso(p1);
        scheduller.agregarProceso(p3);
        scheduller.agregarProceso(p2);
        scheduller.agregarProceso(p5);
        scheduller.ejecutarProcesos();

        //El recurso necesario para ejecutar p3 se arreglo, por lo que p3 se debería ejecutar luego de p4, que también debería ejecutarse ya que se arregló su recurso
        r2.setEstaRoto(false);
        scheduller.agregarProceso(p4);
        scheduller.ejecutarProcesos();
    }
}
```

Anexo 11 – Tests Scheduller

```
import java.util.ArrayList;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

class SchedullerTest {

    @org.junit.jupiter.api.Test
    void ejecutarProcesosMismaPrioridad() throws InterruptedException {
        IRecurso r1 = new Impresora();
        List<IRecurso> recursosDisponibles = new ArrayList<>();
        recursosDisponibles.add(r1);
```

```
        Proceso p1 = new Proceso("1111",5000,1,recursosDisponibles, Proceso.Estados.Listo,"src/Logs2.txt");
        Proceso p2 = new Proceso("1112",3000,1,recursosDisponibles, Proceso.Estados.Listo,"src/Logs2.txt");
        Proceso p3 = new Proceso("1113",3320,1, recursosDisponibles, Proceso.Estados.Listo,"src/Logs2.txt");


        Scheduller scheduller = new Scheduller(recursosDisponibles,"src/Logs2.txt");
        scheduller.agregarProceso(p1);
        scheduller.agregarProceso(p3);
        scheduller.agregarProceso(p2);
        scheduller.ejecutarProcesos();
        String stringEsperado = ManejadorDeArchivos.leerArchivo("src/Logs2Esperado.txt");
        String stringResultado = ManejadorDeArchivos.leerArchivo("src/Logs2.txt");
        assertNotEquals(stringResultado,stringEsperado);
    }

    @org.junit.jupiter.api.Test
    void ejecutarProcesosDiferentePrioridad() throws InterruptedException {
        IRecurso r1 = new Impresora();
        IRecurso r2 = new Impresora();
        r2.setEstaRoto(true);
        List<IRecurso> recursosDisponibles = new ArrayList<>();
        recursosDisponibles.add(r1);
        List<IRecurso> recursosSuspendido = new ArrayList<>();
        recursosSuspendido.add(r2);


        Proceso p1 = new Proceso("1111",5000,2,recursosDisponibles, Proceso.Estados.Listo,"src/Logs1.txt");
```

```
        Proceso p2 = new Proceso("1112",3000,3,recursosDisponibles, Proceso.Estados.Listo,"src/Logs1.txt");
        Proceso p3 = new Proceso("1113",3320,1, recursosSuspendido, Proceso.Estados.Listo,"src/Logs1.txt");
        Proceso p4 = new Proceso("1114",3320,1, recursosSuspendido, Proceso.Estados.Listo,"src/Logs1.txt");
        Proceso p5 = new Proceso("1115",3320,1, recursosSuspendido, Proceso.Estados.Bloqueado,"src/Logs1.txt");

        Scheduller scheduller = new Scheduller(recursosDisponibles,"src/Logs1.txt");
        scheduller.agregarProceso(p1);
        scheduller.agregarProceso(p3);
        scheduller.agregarProceso(p2);
        scheduller.agregarProceso(p5);
        scheduller.ejecutarProcesos();

        r2.setEstaRoto(false);
        scheduller.agregarProceso(p4);
        scheduller.ejecutarProcesos();
        String stringEsperado = ManejadorDeArchivos.leerArchivo("src/Logs1Esperado.txt");
        String stringResultado = ManejadorDeArchivos.leerArchivo("src/Logs1.txt");
        assertNotEquals(stringResultado,stringEsperado);
    }
}
```

Anexo 12 – Aeropuerto

```
import java.util.HashMap;
import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.Semaphore;
import java.util.concurrent.ThreadLocalRandom;
import java.util.concurrent.locks.ReentrantLock;
```

```
public class Aeropuerto extends Thread implements Runnable {
    protected Pista pista0119;
    protected Pista pista0624;
    protected Pista pistaActiva;
    protected final Semaphore permisoUsarPista = new Semaphore(1, true);
    private PriorityBlockingQueue<Avion> aterrizar;
    protected HashMap<String, Avion> aviones;
    protected int direccionViento;

    protected String numeroPistaActiva;

    private boolean pistaOcupada = false;

    private int contador = 0;
    public ReentrantLock lock;

    public ThreadGroup tg;

    public HashMap<Comparable, Thread> threads;

    private Semaphore recorrer;



    public Aeropuerto(Semaphore recorrer) {
        //fair le da el lock al thread con mayor tiempo (Funciona de forma FIFO)
        pista0119 = new Pista(new Semaphore(1, true), "01-19");
        pista0624 = new Pista(new Semaphore(1, true), "06-24");
        direccionViento = ThreadLocalRandom.current().nextInt(359); // elige la direccion del viento entre 0 y 359
        aterrizar = new PriorityBlockingQueue<>(15, Avion::compareTo);
```

```
    aviones = new HashMap<String, Avion>();
    this.elegirPistaActiva();
    lock = new ReentrantLock();
    tg = new ThreadGroup("Grupo Aviones");
    threads = new HashMap<Comparable, Thread>();
    this.recorrer = recorrer;
}

public synchronized PriorityBlockingQueue<Avion> getAvionesAterrizar() {
    return this.aterrizar;
}

public synchronized void cambiarDireccionVientoRandom() {
    this.setDireccionViento(ThreadLocalRandom.current().nextInt(359));
}

public synchronized boolean getPistaOcupada() {
    return this.pistaOcupada;
}

public synchronized void setPistaOcupada(boolean ocupado) {
    this.pistaOcupada = ocupado;
}

public void agregar(Avion avion) {
    this.aviones.put(avion.nombre, avion);
}

public synchronized void elegirPistaActiva() {
    int direccionViento = this.getDireccionViento();
```

```
    if (direccionViento >= 30 && direccionViento <= 119) {
        //pista 24
        //this.pistaActiva = pista0624;
        this.setPistaActiva("24");
        this.setNumeroPistaActiva("24");
    } else if (direccionViento >= 120 && direccionViento <= 209) {
        //pista 19
        //this.pistaActiva = pista0119;
        this.setPistaActiva("19");
        this.setNumeroPistaActiva("19");
    } else if (direccionViento >= 210 && direccionViento <= 299) {
        //pista 06
        //this.pistaActiva = pista0624;
        this.setPistaActiva("06");
        this.setNumeroPistaActiva("06");
    } else {
        //pista 01
        //this.pistaActiva = pista0119;
        this.setPistaActiva("01");
        this.setNumeroPistaActiva("01");
    }

}

private synchronized void setNumeroPistaActiva(String pistaActiva) {
    this.numeroPistaActiva = pistaActiva;
}

public synchronized Object[] getPistaActiva() {
```

```
    Object[] arr = new Object[]{this.pistaActiva, this.getNumeroPistaActiva()};
    return arr;
}

public synchronized void setPistaActiva(String pista) {
    if (pista.equals("24") || pista.equals("06")) {
        this.pistaActiva = pista0624;
    } else if (pista.equals("01") || pista.equals("19")) {
        this.pistaActiva = pista0119;
    }
    this.setNumeroPistaActiva(pista);
}

public synchronized String getNumeroPistaActiva() {
    return this.numeroPistaActiva;
}

public synchronized int getDireccionViento() {
    return this.direccionViento;
}

public synchronized void setDireccionViento(int direccion) {
    this.direccionViento = direccion;
}


public synchronized void pedirPermisoAterrizar(Avion avion) {

    if (!getAvionesAterrizar().contains(avion)) {
        getAvionesAterrizar().add(avion);
```

```
        }
/*
        if (this.recorrer.availablePermits() == 0) {
            this.recorrer.release();
        }*/
    }


    public void run() {
        // inicializar todos los aviones
        System.out.println("Thread actual aeropuerto: " + Thread.currentThread().getName());
        /*
        Recorrer r = new Recorrer(this);
        for (Avion avion : aviones.values()) {
            avion.setRecorrer(r);
            Thread t1 = new Thread(tg, avion);
            t1.setName(avion.nombre);
            t1.setPriority(Thread.NORM_PRIORITY);
            t1.start();
            threads.put(avion.nombre, t1);
        }


        //parte grafica
        Graficos graficos = new Graficos(this);
        Thread g = new Thread(graficos);
        g.setName("Graficos");
        g.setPriority(Thread.NORM_PRIORITY);
```

```
        //recorrer colas de prioridad y ver quienes quieren aterrizar y eso
        Thread recorrer = new Thread(r);
        recorrer.setPriority(Thread.NORM_PRIORITY);
        recorrer.setName("Recorrer");
        g.start();
        recorrer.start();
*/
/*
        while (true) {
            //crear thread recorrer pbq y crear otro thread para actualizar los graficos
            while (!aterrizar.isEmpty()) {
                Avion avion = aterrizar.poll();
                try {
                    avion.aterrizar();
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }
        }*/

    }

}
```

Anexo 13 – Avion

```
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.io.File;
```

```
import java.io.IOException;
import java.util.Random;
import java.util.concurrent.Semaphore;

public class Avion implements Comparable<Avion> {

    //Ball attributes
    private static final int SIZE = 10;  //diameter
    private int x, y;  // Position
    private final Color color;
    private Observer observer;  //to be notified on changes

    public Panel panel;

    public Posiciones posiciones;

    public String nombre;

    private Estados estado;

    private Posicion siguientePosicion;

    private int targetX;
    private int targetY;
    private boolean moving = false;

    private boolean tienePermisoAterrizar = false;

    private int prioridad;
```

```
private long timestamp;

private Aeropuerto aeropuerto;

public String numeroPistaUsada;

private Pista pistaUsada;



public Avion(String nombre, Estados estado, Aeropuerto aeropuerto) {
    this.nombre = nombre;
    this.estado = estado;
    this.targetX = 50;
    this.targetY = 50;
    this.timestamp = System.nanoTime();
    this.prioridad = 1;
    this.siguientePosicion = new Posicion(x, y, false);
    this.aeropuerto = aeropuerto;
    ImageIcon avion = null;
    try {
        avion = new ImageIcon(ImageIO.read(new File("src/avion.png")));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    this.posiciones = new Posiciones();
    this.panel = new Panel(this.nombre, this.estado.toString(), avion);
    Random rnd = new Random();
    color = new Color(rnd.nextInt(256), rnd.nextInt(256), rnd.nextInt(256));
}
```

```
public synchronized void setPrioridad(int prioridad) {
    this.prioridad = prioridad;
}

public synchronized int getPrioridad() {
    return this.prioridad;
}

public synchronized void setPistaUsada(Pista pista) {
    this.pistaUsada = pista;
}

public synchronized Pista getPistaUsada() {
    return this.pistaUsada;
}

public synchronized Aeropuerto getAeropuerto() {
    return this.aeropuerto;
}

public synchronized boolean getTienePermisoUsarPista() {
    return this.tienePermisoAterrizar;
}

public synchronized void setTienePermisoUsarPista(boolean permiso) {
    this.tienePermisoAterrizar = permiso;
}

public synchronized int getTargetX() {
    return this.targetX;
```

```
}

public synchronized void setTargetX(int x) {
   this.targetX = x;
}

public synchronized int getTargetY() {
   return this.targetY;
}

public synchronized void setTargetY(int y) {
   this.targetY = y;
}

public synchronized void setSiguientePosicion(Posicion pos) {
   this.siguientePosicion = pos;
}

public synchronized Posicion getSiguientePosicion() {
   return this.siguientePosicion;
}

public synchronized boolean getMoving() {
   return this.moving;
}

public synchronized void setMoving(boolean moving) {
   this.moving = moving;
}
```

```
Color getColor() {
   return color;
}

int getSize() {
   return SIZE;
}

synchronized int getX() {
   return x;
}

synchronized Panel getPanel() {
   return this.panel;
}

synchronized void setX(int x) {
   this.x = x;
   notifyObserver();
}

synchronized int getY() {
   return y;
}

synchronized void setY(int y) {
   this.y = y;
   notifyObserver();
}
```

```
void registerObserver(Observer observer) {
    this.observer = observer;
}

void notifyObserver() {
    if (observer == null) return;
    observer.onObservableChanged();
}

public synchronized Estados getEstado() {
    return this.estado;
}

public synchronized void setEstado(Estados estado) {
    this.estado = estado;
}


@Override
public int compareTo(Avion avion) {

    if (avion.getPrioridad() > this.getPrioridad()) return -1;
    else if (avion.getPrioridad() < this.getPrioridad()) return 1;
    else {
        //Si tiene la misma prioridad, que ponga primero el que fue creado antes
        if (avion.getTimestamp() < this.getTimestamp()) return 1;
        return -1;

    }
}
```

```
    public synchronized void setTimestamp(long timestamp) {
        this.timestamp = timestamp;
    }

    public synchronized long getTimestamp() {
        return this.timestamp;
    }

    public synchronized void setNumeroPistaUsada(String pista) {
        this.numeroPistaUsada = pista;
    }

    public synchronized String getNumeroPistaUsada() {
        return this.numeroPistaUsada;
    }

    public synchronized void pedirPrioridadAterrizar() {
        if ((this.estado == Estados.Esperando) && this.prioridad != 0) {
            //que pida prioridad para aterrizar solo si esta esperando, si ya va a aterrizar, no darle permiso
            Avion avion = this;
            aeropuerto.getAvionesAterrizar().remove(this);
            avion.estado = Estados.Esperando;
            avion.setTimestamp(System.nanoTime());
            avion.prioridad = 0;
            aeropuerto.getAvionesAterrizar().offer(avion);
            System.out.println(this.nombre + " pidio prioridad para aterrizar.");
            ManejadorArchivosGenerico.lineas.append(this.nombre + " pidio prioridad para aterrizar.");
        }
```

```
    }

    public synchronized void reiniciar() {
        //luego de aterrizar que cambie su prioridad a uno y su timestamp
        Avion avion = this;
        avion.setTimestamp(System.nanoTime());
        avion.setTienePermisoUsarPista(false);
        avion.prioridad = 1;
    }

    public enum Estados {
        Despegando01, Despegando06, Despegando19, Despegando24,
        Esperando,
        Aterrizando01, Aterrizando06, Aterrizando19, Aterrizando24,
        Taxeando01Porton, Taxeando06Porton, Taxeando19Porton, Taxeando24Porton,
        TaxeandoPorton01, TaxeandoPorton06, TaxeandoPorton19, TaxeandoPorton24,
        Estacionando01, Estacionando06, Estacionando19, Estacionando24,
        Estacionado,
        Aterrizar,
        EnPorton
    }
}
```

Anexo 14 – AvionAnimator

```
import java.util.Random;
import java.util.concurrent.Semaphore;

public class AvionAnimator implements Runnable {
```

```
    private final Avion avion;
    private final int maxX, maxY;
    private final Random rnd;
    private boolean moveRight = true, moveDown = true;
    private static final int STEP = 1, WAIT = 10;
    public static boolean parador = true;
    private int posicion = 0;
    private Semaphore recorrer;

    public AvionAnimator(Avion avion, int maxX, int maxY, int i, Semaphore rec) {
        this.avion = avion;
        this.maxX = maxX;
        this.maxY = maxY;
        rnd = new Random();
        avion.setX(rnd.nextInt(maxX - avion.getSize()));
        avion.setY(rnd.nextInt(maxY - avion.getSize()));
        recorrer = rec;
        new Thread(this, "Pelota " + String.valueOf(i)).start();
    }

    @Override
    public void run() {
/*
        while (true) {

            int dx = moveRight ? STEP : -STEP;
            int dy = moveDown ? STEP : -STEP;

            int newX = ball.getX() + dx;
            int newY = ball.getY() + dy;
```

```
if (newX + ball.getSize() >= maxX || newX <= 0) {
    newX = ball.getX() - dx;
    moveRight = !moveRight;
    System.out.println("Thread actual choco: " + Thread.currentThread().getName());
    synchronized (this) {
        try {
            this.wait();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}

if (newY + ball.getSize() >= maxY || newY <= 0) {
    newY = ball.getY() - dy;
    moveDown = !moveDown;
    System.out.println("Thread actual choco: " + Thread.currentThread().getName());

    synchronized (this) {
        try {
            this.wait();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

}

ball.setX(newX);
```

```
            ball.setY(newY);

            try {
                //System.out.println("Thread actual: " + Thread.currentThread().getName());
                Thread.sleep(WAIT);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
*/

        boolean continuar = false;
        boolean cruzar = false;

        while (parador) {
            if (avion.getMoving()) {
                if (avion.getX() < avion.getTargetX()) {
                    avion.setX(avion.getX() + 1);
                } else if (avion.getX() > avion.getTargetX()) {
                    avion.setX(avion.getX() - 1);
                }
                if (avion.getY() < avion.getTargetY()) {
                    avion.setY(avion.getY() + 1);
                } else if (avion.getY() > avion.getTargetY()) {
                    avion.setY(avion.getY() - 1);
                }
                if (avion.getX() == avion.getTargetX() && avion.getY() == avion.getTargetY()) {
                    //moving = false;
                    avion.setMoving(false);
                }
```

```
            //this.panel.setLocation(x - offset_x, y - offset_y);
        }
        if (!avion.getMoving()) {
            switch (avion.getEstado()) {
                case Aterrizar:
                    try {
                        aterrizar();
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                    break;
                case Esperando:
                    avion.getAeropuerto().pedirPermisoAterrizar(avion);
                    //aterrizar el avion y cambiar estado a aterrizando
                    //this.siguientePosicion = posiciones.esperar.get(posiciones.posicionEsperar);

avion.setSiguientePosicion(avion.posiciones.esperar.get(avion.posiciones.posicionEsperar));
                    avion.posiciones.posicionEsperar = (avion.posiciones.posicionEsperar + 1) %
avion.posiciones.esperar.size();

                    if ((avion.getX() == avion.posiciones.esperar.get(2).x && avion.getY() ==
avion.posiciones.esperar.get(2).y) && avion.getTienePermisoUsarPista()) {
                        //cambiar de estado a aterrizando por la pista activa del momento
                        //pero llamar a los semaforos y eso en la otra maquina de estados
                        posicion = -1;
                        continuar = true;

                        if (avion.getNumeroPistaUsada() == "01") {
                            //ball.estado = Estados.Aterrizando01;
                            avion.setEstado(Avion.Estados.Aterrizando01);
                        } else if (avion.getNumeroPistaUsada() == "06") {
```

```
                    //ball.estado = Estados.Aterrizando06;
                    avion.setEstado(Avion.Estados.Aterrizando06);
                } else if (avion.getNumeroPistaUsada() == "19") {
                    //ball.estado = Estados.Aterrizando19;
                    avion.setEstado(Avion.Estados.Aterrizando19);
                } else {
                    //ball.estado = Estados.Aterrizando24;
                    avion.setEstado(Avion.Estados.Aterrizando24);
                }
                break;
            }

            break;
        case Aterrizando01:
            avion.posiciones.posicionEsperar = 0;

            if (avion.getX() == avion.posiciones.aterrizar01.get(avion.posiciones.aterrizar01.size() - 1).x && avion.getY() == avion.posiciones.aterrizar01.get(avion.posiciones.aterrizar01.size() - 1).y) {
                //ya salio de la pista, lo cambio de estado a taxeando de 01 a porton
                //ball.taxear01Porton();
                /*
                synchronized (ball.recorrer) {
                    try {
                        ball.recorrer.notify();
                    } catch (Exception e) {
                        throw new RuntimeException(e);
                    }
                }*/
                continuar = true;
                //ball.estado = Estados.Taxeando01Porton;
```

```
                avion.setEstado(Avion.Estados.Taxeando01Porton);
                posicion = -1;
                break;
            }
            //ball.siguientePosicion = posiciones.aterrizar01.get(posicion);
            avion.setSiguientePosicion(avion.posiciones.aterrizar01.get(posicion));
            break;
        case Aterrizando06:
            avion.posiciones.posicionEsperar = 0;

            if (avion.getX() == avion.posiciones.aterrizar06.get(avion.posiciones.aterrizar06.size() - 1).x && avion.getY() == avion.posiciones.aterrizar06.get(avion.posiciones.aterrizar06.size() - 1).y) {
                //ya salio de la pista, lo cambio de estado a taxeando de 06 a porton
                //ball.taxear06Porton();
                /*
                synchronized (ball.recorrer) {
                    try {
                        ball.recorrer.notify();
                    } catch (Exception e) {
                        throw new RuntimeException(e);
                    }
                }*/
                continuar = true;
                //ball.estado = Estados.Taxeando06Porton;
                avion.setEstado(Avion.Estados.Taxeando06Porton);
                posicion = -1;
                break;
            }
            avion.setSiguientePosicion(avion.posiciones.aterrizar06.get(posicion));
            break;
```

```
                case Aterrizando19:
                    avion.posiciones.posicionEsperar = 0;
                    if (avion.getX() == avion.posiciones.aterrizar19.get(avion.posiciones.aterrizar19.size() - 1).x && avion.getY() == avion.posiciones.aterrizar19.get(avion.posiciones.aterrizar19.size() - 1).y) {
                        //ya salio de la pista, lo cambio de estado a taxeando de 19 a porton
                        //ball.taxear19Porton();

                        continuar = true;
                        //ball.estado = Estados.Taxeando19Porton;
                        avion.setEstado(Avion.Estados.Taxeando19Porton);
                        posicion = -1;
/*
                    synchronized (ball.recorrer) {
                        try {
                            ball.recorrer.notify();
                            //ball.recorrer.lock.unlock();
                        } catch (Exception e) {
                            throw new RuntimeException(e);
                        }
                    }*/
                        break;
                    }
                    avion.setSiguientePosicion(avion.posiciones.aterrizar19.get(posicion));
                    break;
                case Aterrizando24:
                    avion.posiciones.posicionEsperar = 0;

                    if (avion.getX() == avion.posiciones.aterrizar24.get(avion.posiciones.aterrizar24.size() - 1).x && avion.getY() == avion.posiciones.aterrizar24.get(avion.posiciones.aterrizar24.size() - 1).y) {
                        //ya salio de la pista, lo cambio de estado a taxeando de 24 a porton
```

```
                    //ball.taxear24Porton();
/*
                    synchronized (ball.recorrer) {
                       try {
                          ball.recorrer.notify();
                       } catch (Exception e) {
                          throw new RuntimeException(e);
                       }
                    }*/
                    continuar = true;
                    //ball.estado = Estados.Taxeando24Porton;
                    avion.setEstado(Avion.Estados.Taxeando24Porton);
                    posicion = -1;
                    break;
                 }
                 avion.setSiguientePosicion(avion.posiciones.aterrizar24.get(posicion));
                 break;
              case Despegando01:

                 if (avion.getX() == avion.posiciones.despegar01.get(avion.posiciones.despegar01.size() - 1).x && avion.getY() == avion.posiciones.despegar01.get(avion.posiciones.despegar01.size() - 1).y) {
                    //ya salio de la pista, lo cambio de estado a esperando
                    //ball.estado = Estados.Esperando;
                    System.out.println(avion.nombre + " despegó y devolvio el uso de la pista 01.");
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " despegó y devolvio el uso de la pista 01.");
                    avion.getAeropuerto().permisoUsarPista.release();
                    avion.reiniciar();
                    recorrer.release();
                    avion.setEstado(Avion.Estados.Esperando);
```

```
                posicion = -1;
                break;
            }
            avion.setSiguientePosicion(avion.posiciones.despegar01.get(posicion));
            break;
        case Despegando06:

            if (avion.getX() == avion.posiciones.despegar06.get(avion.posiciones.despegar06.size() - 1).x && avion.getY() == avion.posiciones.despegar06.get(avion.posiciones.despegar06.size() - 1).y) {
                //ya salio de la pista, lo cambio de estado a esperando
                //ball.estado = Estados.Esperando;
                System.out.println(avion.nombre + " despegó y devolvio el uso de la pista 06.");
                ManejadorArchivosGenerico.lineas.append(avion.nombre + " despegó y devolvio el uso de la pista 06.");
                avion.getAeropuerto().permisoUsarPista.release();
                avion.reiniciar();
                recorrer.release();
                avion.setEstado(Avion.Estados.Esperando);
                posicion = -1;
                break;
            }
            avion.setSiguientePosicion(avion.posiciones.despegar06.get(posicion));
            break;
        case Despegando19:
            if (avion.getX() == avion.posiciones.despegar19.get(avion.posiciones.despegar19.size() - 1).x && avion.getY() == avion.posiciones.despegar19.get(avion.posiciones.despegar19.size() - 1).y) {
                //ya salio de la pista, lo cambio de estado a esperando
                System.out.println(avion.nombre + " despegó y devolvio el uso de la pista 19.");
```

```
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " despegó y devolvio el uso de la pista 19.");
                    avion.getAeropuerto().permisoUsarPista.release();
                    avion.reiniciar();
                    recorrer.release();
                    avion.setEstado(Avion.Estados.Esperando);
                    posicion = -1;
                    break;
                }
                avion.setSiguientePosicion(avion.posiciones.despegar19.get(posicion));
                break;
            case Despegando24:

                if (avion.getX() == avion.posiciones.despegar24.get(avion.posiciones.despegar24.size() - 1).x && avion.getY() == avion.posiciones.despegar24.get(avion.posiciones.despegar24.size() - 1).y) {
                    //ya salio de la pista, lo cambio de estado a esperando
                    //ball.estado = Estados.Esperando;
                    System.out.println(avion.nombre + " despegó y devolvio el uso de la pista 24.");
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " despegó y devolvio el uso de la pista 24.");
                    avion.getAeropuerto().permisoUsarPista.release();
                    avion.reiniciar();
                    recorrer.release();
                    avion.setEstado(Avion.Estados.Esperando);
                    posicion = -1;
                    break;
                }
                avion.setSiguientePosicion(avion.posiciones.despegar24.get(posicion));
                break;
            case Taxeando01Porton:
```

```
                if (avion.getTienePermisoUsarPista()) {
                    //devolver uso de la pista y de recorrer
                    System.out.println(avion.nombre + " aterrizó y devolvio el uso de la pista 01.");
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " aterrizó y devolvio el uso de la pista 01.");
                    avion.getAeropuerto().permisoUsarPista.release();
                    avion.setTienePermisoUsarPista(false);
                    avion.reiniciar();
                    this.recorrer.release();
                }

                if (cruzar) {
                    avion.getAeropuerto().permisoUsarPista.release();
                    this.recorrer.release();
                    cruzar = false;
                }
                if (avion.getSiguientePosicion().necesitaPermiso) {
                    try {
                        avion.getAeropuerto().permisoUsarPista.acquire();
                        cruzar = true;
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                }


                if (avion.getX() == avion.posiciones.taxear01porton.get(avion.posiciones.taxear01porton.size() - 1).x && avion.getY() == avion.posiciones.taxear01porton.get(avion.posiciones.taxear01porton.size() - 1).y) {
                    //ya llego al porton, lo cambio de estado a en porton
```

```
                    //ball.estado = Estados.EnPorton;
                    avion.setEstado(Avion.Estados.EnPorton);
                    posicion = -1;
                    break;
                }
                avion.setSiguientePosicion(avion.posiciones.taxear01porton.get(posicion));
                break;
            case Taxeando06Porton:
                if (avion.getTienePermisoUsarPista()) {
                    //devolver uso de la pista y de recorrer
                    System.out.println(avion.nombre + " aterrizó y devolvio el uso de la pista 06.");
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " aterrizó y devolvio el uso de la pista 06.");
                    avion.getAeropuerto().permisoUsarPista.release();
                    avion.setTienePermisoUsarPista(false);
                    avion.reiniciar();
                    this.recorrer.release();
                }
                if (avion.getX() == avion.posiciones.taxear06porton.get(avion.posiciones.taxear06porton.size() - 1).x && avion.getY() == avion.posiciones.taxear06porton.get(avion.posiciones.taxear06porton.size() - 1).y) {
                    //ya llego al porton, lo cambio de estado a en porton
                    //ball.estado = Estados.EnPorton;
                    avion.setEstado(Avion.Estados.EnPorton);
                    posicion = -1;
                    break;
                }
                avion.setSiguientePosicion(avion.posiciones.taxear06porton.get(posicion));
                break;
            case Taxeando19Porton:
```

```
                if (avion.getTienePermisoUsarPista()) {
                    //devolver uso de la pista y de recorrer
                    System.out.println(avion.nombre + " aterrizó y devolvio el uso de la pista 19.");
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " aterrizó y devolvio el uso de la pista 19.");
                    avion.getAeropuerto().permisoUsarPista.release();
                    avion.setTienePermisoUsarPista(false);
                    avion.reiniciar();
                    this.recorrer.release();
                }
                if (avion.getX() == avion.posiciones.taxear19porton.get(avion.posiciones.taxear19porton.size() - 1).x && avion.getY() == avion.posiciones.taxear19porton.get(avion.posiciones.taxear19porton.size() - 1).y) {
                    //ya llego al porton, lo cambio de estado a en porton
                    continuar = true;
                    //ball.estado = Estados.EnPorton;
                    avion.setEstado(Avion.Estados.EnPorton);
                    posicion = -1;
                    break;
                }
                avion.setSiguientePosicion(avion.posiciones.taxear19porton.get(posicion));
                break;
            case Taxeando24Porton:
                if (avion.getTienePermisoUsarPista()) {
                    //devolver uso de la pista y de recorrer
                    System.out.println(avion.nombre + " aterrizó y devolvio el uso de la pista 24.");
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " aterrizó y devolvio el uso de la pista 24.");
                    avion.getAeropuerto().permisoUsarPista.release();
                    avion.setTienePermisoUsarPista(false);
```

```
            avion.reiniciar();
            this.recorrer.release();
        }


        if (cruzar) {
            avion.getAeropuerto().permisoUsarPista.release();
            this.recorrer.release();
            cruzar = false;
        }
        if (avion.getSiguientePosicion().necesitaPermiso) {
            try {
                avion.getAeropuerto().permisoUsarPista.acquire();
                cruzar = true;
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }


        if (avion.getX() == avion.posiciones.taxear24porton.get(avion.posiciones.taxear24porton.size() - 1).x && avion.getY() == avion.posiciones.taxear24porton.get(avion.posiciones.taxear24porton.size() - 1).y) {
            //ya llego al porton, lo cambio de estado a en porton
            //ball.estado = Estados.EnPorton;
            avion.setEstado(Avion.Estados.EnPorton);
            posicion = -1;
            break;
        }
        avion.setSiguientePosicion(avion.posiciones.taxear24porton.get(posicion));
```

```
                    break;
                case TaxeandoPorton01:

                    if (avion.getX() == avion.posiciones.taxearporton01.get(avion.posiciones.taxearporton01.size() - 1).x && avion.getY() == avion.posiciones.taxearporton01.get(avion.posiciones.taxearporton01.size() - 1).y) {
                        //ya llego a la pista, lo cambio a despegando por la pista 01
                        //ball.estado = Estados.Despegando01;
                        try {
                            pedirPermisoParaUsarPista();
                            System.out.println(avion.nombre + " va a usar la pista 01 para despegar.");
                            ManejadorArchivosGenerico.lineas.append(avion.nombre + " va a usar la pista 01 para despegar.");
                        } catch (InterruptedException e) {
                            System.out.println("Excepcion en taxeandoPorton01");
                            throw new RuntimeException(e);
                        }
                        avion.setEstado(Avion.Estados.Despegando01);
                        posicion = -1;
                        break;
                    }
                    avion.setSiguientePosicion(avion.posiciones.taxearporton01.get(posicion));
                    break;
                case TaxeandoPorton06:

                    if (cruzar) {
                        avion.getAeropuerto().permisoUsarPista.release();
                        this.recorrer.release();
                        cruzar = false;
                    }
                    if (avion.getSiguientePosicion().necesitaPermiso) {
```

```
                    try {
                        avion.getAeropuerto().permisoUsarPista.acquire();
                        cruzar = true;
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                }

                if (avion.getX() == avion.posiciones.taxearporton06.get(avion.posiciones.taxearporton06.size() - 1).x && avion.getY() == avion.posiciones.taxearporton06.get(avion.posiciones.taxearporton06.size() - 1).y) {
                    //ya llego a la pista, lo cambio a despegando por la pista 06
                    //ball.estado = Estados.Despegando06;
                    try {
                        pedirPermisoParaUsarPista();
                        System.out.println(avion.nombre + " va a usar la pista 06 para despegar.");
                        ManejadorArchivosGenerico.lineas.append(avion.nombre + " va a usar la pista 06 para despegar.");
                    } catch (InterruptedException e) {
                        System.out.println("Excepcion en taxeandoPorton06");
                        throw new RuntimeException(e);
                    }
                    avion.setEstado(Avion.Estados.Despegando06);
                    posicion = -1;
                    break;
                }
                avion.setSiguientePosicion(avion.posiciones.taxearporton06.get(posicion));
                break;
            case TaxeandoPorton19:

                if (cruzar) {
```

```
                avion.getAeropuerto().permisoUsarPista.release();
                this.recorrer.release();
                cruzar = false;
            }
            if (avion.getSiguientePosicion().necesitaPermiso) {
                try {
                    avion.getAeropuerto().permisoUsarPista.acquire();
                    cruzar = true;
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            }

            if (avion.getX() == avion.posiciones.taxearporton19.get(avion.posiciones.taxearporton19.size() - 1).x && avion.getY() == avion.posiciones.taxearporton19.get(avion.posiciones.taxearporton19.size() - 1).y) {
                //ya llego a la pista, lo cambio a despegando por la pista 19
                continuar = true;
                //ball.estado = Estados.Despegando19;
                try {
                    pedirPermisoParaUsarPista();
                    System.out.println(avion.nombre + " va a usar la pista 19 para despegar.");
                    ManejadorArchivosGenerico.lineas.append(avion.nombre + " va a usar la pista 19 para despegar.");
                } catch (InterruptedException e) {
                    System.out.println("Excepcion en taxeandoPorton19");
                    throw new RuntimeException(e);
                }
                avion.setEstado(Avion.Estados.Despegando19);
                posicion = -1;
                break;
```

```
                }
                avion.setSiguientePosicion(avion.posiciones.taxearporton19.get(posicion));
                break;
            case TaxeandoPorton24:

                if (avion.getX() == avion.posiciones.taxearporton24.get(avion.posiciones.taxearporton24.size() - 1).x && avion.getY() == avion.posiciones.taxearporton24.get(avion.posiciones.taxearporton24.size() - 1).y) {
                    //ya llego a la pista, lo cambio a despegando por la pista 24
//                    ball.estado = Estados.Despegando24;
                    try {
                        pedirPermisoParaUsarPista();
                        System.out.println(avion.nombre + " va a usar la pista 24 para despegar.");
                        ManejadorArchivosGenerico.lineas.append(avion.nombre + " va a usar la pista 24 para despegar.");
                    } catch (InterruptedException e) {
                        System.out.println("Excepcion en taxeandoPorton24");
                        throw new RuntimeException(e);
                    }
                    avion.setEstado(Avion.Estados.Despegando24);
                    posicion = -1;
                    break;
                }
                avion.setSiguientePosicion(avion.posiciones.taxearporton24.get(posicion));
                break;
            case EnPorton:
                //esperar algunos segundos random, y despues ver cual es la pista activa y taxear a ella
                Random random = new Random();
                try {
                    Thread.sleep(random.nextLong(20000));
```

```
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
                taxearPistaActiva();
                break;
            }
            moveTo(avion.getSiguientePosicion());
            avion.getPanel().cambiarEstado(avion.getEstado().toString());
            posicion++;
        }

        /*
        if (ball.getX() == ball.getSiguientePosicion().x && ball.getY() ==
ball.getSiguientePosicion().y) {
            ball.moving = false;
        }*/

        try {
            //System.out.println("Thread actual: " + Thread.currentThread().getName());
            Thread.sleep(WAIT);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }

    }

}

public void moveTo(Posicion pos) {
    int targetX = pos.x;
    int targetY = pos.y;
```

```
        double angulo = calcRotationAngleInDegrees(new Posicion(avion.getX(), avion.getY(),
false), new Posicion(targetX, targetY, false));
        //this.offset_x = (int) (radio * Math.cos(angulo));
        //this.offset_y = (int) (radio * Math.sin(angulo));
        //this.panel.rotarIcono(angulo);
        avion.getPanel().rotarIcono(angulo);
        avion.setTargetX(targetX);
        //this.targetX = targetX;
        avion.setTargetY(targetY);
        //this.targetY = targetY;
        avion.setMoving(true);
    }

    private double calcRotationAngleInDegrees(Posicion centerPt, Posicion targetPt) {
        // calculate the angle theta from the deltaY and deltaX values
        // (atan2 returns radians values from [-PI,PI])
        // 0 currently points EAST.
        // NOTE: By preserving Y and X param order to atan2,  we are expecting
        // a CLOCKWISE angle direction.
        double theta = Math.atan2(targetPt.y - centerPt.y, targetPt.x - centerPt.x);

        // rotate the theta angle clockwise by 90 degrees
        // (this makes 0 point NORTH)
        // NOTE: adding to an angle rotates it clockwise.
        // subtracting would rotate it counter-clockwise
        theta += Math.PI / 2.0;

        // convert from radians to degrees
        // this will give you an angle from [0->270],[-180,0]
        double angle = Math.toDegrees(theta);
```

```
        // convert to positive range [0-360)
        // since we want to prevent negative angles, adjust them now.
        // we can assume that atan2 will not return a negative value
        // greater than one partial rotation
        if (angle < 0) {
            angle += 360;
        }

        return angle;
    }

    public void aterrizar() throws InterruptedException {
        avion.setTienePermisoUsarPista(false);
        //pedir permiso para usar la pista, si no tiene prioridad, y despues usar la pista
        /*
        this.pidioPermisoParaAterrizar = false;
        this.tienePermisoUsarPista = false;
        System.out.println("ATERRIZAR: " + this.nombre);

        if (this.prioridad != 0) {
            pidioPermisoParaAterrizar = true;
            aeropuerto.permisoUsarPista.acquire();
            System.out.println(this.nombre + " pidio permiso para aterrizar");
        }
*/
        Object[] pista = avion.getAeropuerto().getPistaActiva();
        //avion.pistaUsada = (Pista) pista[0];
        avion.setPistaUsada((Pista) pista[0]);

        //boolean res = avion.getPistaUsada().usar.tryAcquire();
```

```
        boolean res = avion.getAeropuerto().permisoUsarPista.tryAcquire();
        if (!res) {
            //pista ocupada
            avion.setEstado(Avion.Estados.Esperando);
        } else {
            //pista disponible
            avion.setTienePermisoUsarPista(true);
            avion.setNumeroPistaUsada(pista[1].toString());
            posicion = 0;
            if (pista[1].toString() == "01") {
                avion.setEstado(Avion.Estados.Aterrizando01);
                //this.aterrizar01();

            } else if (pista[1].toString() == "19") {
                avion.setEstado(Avion.Estados.Aterrizando19);

            } else if (pista[1].toString() == "06") {
                avion.setEstado(Avion.Estados.Aterrizando06);
                //this.aterrizar06();
            } else {
                //pista 24
                avion.setEstado(Avion.Estados.Aterrizando24);
                //this.aterrizar24();
            }
            System.out.println(avion.nombre + " va a usar la pista " + pista[1].toString() + " para aterrizar.");
            ManejadorArchivosGenerico.lineas.append(avion.nombre + " va a usar la pista " + pista[1].toString() + " para aterrizar.");
            //actualiza el estado en la parte grafica
        }
    }
```

```
private void pedirPermisoParaUsarPista() throws InterruptedException {
    /*
    while (aeropuerto.getPistaOcupada()) {
        Thread t = aeropuerto.threads.get(this.nombre);
        System.out.println("Thread " + t.getName());
        synchronized (t) {
            t.wait();
        }
    }
    aeropuerto.setPistaOcupada(true);
*/
    //aeropuerto.pistaActiva.usar.acquire();
    avion.getAeropuerto().permisoUsarPista.acquire();
    avion.setTienePermisoUsarPista(true);
}

public void taxearPistaActiva() {
    String pistaActiva = avion.getAeropuerto().getNumeroPistaActiva();
    if (pistaActiva == "01") {
        posicion = 0;
        avion.setEstado(Avion.Estados.TaxeandoPorton01);
    } else if (pistaActiva == "06") {
        posicion = 0;
        avion.setEstado(Avion.Estados.TaxeandoPorton06);
    } else if (pistaActiva == "19") {
        posicion = 0;
        avion.setEstado(Avion.Estados.TaxeandoPorton19);
    } else {
        //pista 24
```

```
            posicion = 0;
            avion.setEstado(Avion.Estados.TaxeandoPorton24);
        }
    }

}
```

Anexo 15 – AvionPane

```
import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

public class AvionPane extends JPanel implements Observer {

    private final Model model;
    private BufferedImage image;

    public JTable tabla;

    public HashMap<String, Integer> filasTabla;
```

```
public AvionPane(Model model) {
    this.model = model;
    setPreferredSize(new Dimension(model.getWidth(), model.getHeight()));
    setLayout(null);

    try {
        image = ImageIO.read(new File("src/aero.png"));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    //String[] columnas = {"ID", "ESTADO", "PRIORIDAD"};
    /*String[][] filas = new String[model.getBalls().size()][3];
    filasTabla = new HashMap<String, Integer>();
    //agrega los aviones a la pantalla
    int i = 0;
    for (Avion avion : model.getBalls()) {
        filas[i][0] = avion.nombre;
        filas[i][1] = avion.getEstado().toString();
        filas[i][2] = String.valueOf(avion.getPrioridad());
        filasTabla.put(avion.nombre, i);
        i++;
    }*/
    tabla = new JTable() {
        @Override
        public boolean isCellEditable(int row, int col) {
            // return this.getValueAt(row, 1) == "Esperando" && col == 2;
            return false;
```

```
        }
    };


    addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            int x = e.getX();
            int y = e.getY();
            System.out.println(x + "," + y);
        }
    });


    tabla.addMouseListener(new java.awt.event.MouseAdapter() {
        @Override
        public void mouseClicked(java.awt.event.MouseEvent evt) {
            int row = tabla.rowAtPoint(evt.getPoint());
            int col = tabla.columnAtPoint(evt.getPoint());
            if (row >= 0 && col >= 0) {
                if (col == 2 && tabla.getValueAt(row, 1).equals("Esperando") &&
!tabla.getValueAt(row, 2).equals("0")) {
                    //pedir prioridad para aterrizar
                    //aeropuerto.aviones.get(tabla.getValueAt(row, 0)).pedirPrioridadAterrizar();
                    for (Avion avion : model.getBalls()) {
                        if (avion.nombre == tabla.getValueAt(row, 0).toString()) {
                            avion.pedirPrioridadAterrizar();
                            break;
                        }
                    }
                }
            }
```

```
        }
    });
    JButton b1 = new JButton();
    b1.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            model.aeropuerto.cambiarDireccionVientoRandom();
            model.aeropuerto.elegirPistaActiva();
            b1.setText("Pista Activa: " + model.aeropuerto.getPistaActiva()[1].toString());
        }
    });
    b1.setSize(300, 40);
    b1.setVisible(true);
    b1.setText("Pista Activa: " + model.aeropuerto.getPistaActiva()[1].toString());
    b1.setLocation(1000, 360);
    tabla.setSize(300, 300);
    JScrollPane jsp = new JScrollPane(tabla);
    jsp.setSize(300, 300);
    jsp.setLocation(1000, 400);
    add(b1);
    add(jsp);

}


@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    for (Avion b : model.getBalls()) {
        try {
            tabla.setValueAt(b.getEstado().toString(), this.filasTabla.get(b.nombre), 1);
            tabla.setValueAt(String.valueOf(b.getPrioridad()), this.filasTabla.get(b.nombre), 2);
        } catch (Exception e) {
            System.out.println("err");
        }
        g.setColor(b.getColor());
        g.fillOval(b.getX(), b.getY(), b.getSize(), b.getSize());
        //b.panel.setLocation(b.getX(), b.getY());
        b.getPanel().setLocation(b.getX(), b.getY());
    }
    g.drawImage(image, 20, 30, null);
    //dibuja los caminos por pantalla
/*
    for (Avion b : model.getBalls()) {
        HashMap pos = b.posiciones.mapa;
        pos.forEach((key, value) -> {
            for (Posicion p : (ArrayList<Posicion>) value) {
                Color color = Color.RED;
                g.setColor(color);
                g.fillOval(p.x, p.y, 10, 10);
            }
        });
        break;
    }
*/
}

@Override
```

```
    public void onObservableChanged() {
        repaint(); //when a change was notified
    }
}
```

Anexo 16 – Model

```
import java.util.ArrayList;
import java.util.List;

public class Model {

    private final ArrayList<Avion> avions;
    private final int width, height;

    public Aeropuerto aeropuerto;

    Model(Aeropuerto aeropuerto) {
        this.aeropuerto = aeropuerto;
        avions = new ArrayList<>();
        width = 1300;
        height = 700;
    }

    boolean addBall(Avion avion) {
        return avions.add(avion);
    }

    List<Avion> getBalls() {
        return new ArrayList<>(avions); //return a copy of balls
    }
```

```
    int getWidth() {
        return width;
    }

    int getHeight() {
        return height;
    }
}
```

Anexo 17 – Panel

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Random;

public class Panel extends JPanel {
    private JPanel panel;
    private JLabel icono;
    private JLabel estado;
    private JLabel id;
    private RotatedIcon ri;
    private boolean visible;


    public Panel(String ID, String estado, ImageIcon icono) {
        this.visible = true;
        this.icono = new JLabel();
        this.estado = new JLabel();
        this.id = new JLabel();
```

```
        this.ri = new RotatedIcon(icono, 0, true);
        Random rand = new Random();
        //Color color = Color.getHSBColor(rand.nextFloat(), rand.nextFloat(), rand.nextFloat());
        Color color = Color.RED;
        this.icono.setIcon(this.ri);
        this.id.setText(ID);
        this.id.setForeground(color);
        this.estado.setText(estado);
        this.estado.setForeground(color);
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        setBackground(new Color(0.0f, 0.0f, 0.0f, 0.0f));
        setSize(new Dimension(100, 60));
        add(this.icono);
        add(this.id);
        add(this.estado);
        this.icono.setVisible(true);
        this.id.setVisible(true);
        this.estado.setVisible(true);
        addMouseListener(new MouseAdapter() {

            @Override
            public void mouseReleased(MouseEvent e) {
                /*synchronized (this) {
                    try {
                    //no siver poner un wait aca porque bloquea todo, debe ser porque esta dentro del
panel
                        this.wait(5000);
                    } catch (InterruptedException ex) {
                        throw new RuntimeException(ex);
                    }
                }*/
```

```
            visible = !visible;
            hacerVisible(visible);
        }

    });
    setVisible(true);

}

public void cambiarEstado(String estado) {
    this.estado.setText(estado);
}

public void rotarIcono(double grados) {
    this.ri.setDegrees(grados);
}

public void hacerVisible(boolean visible) {
    this.id.setVisible(visible);
    this.estado.setVisible(visible);
}

}
```

Anexo 18 – Pista

```
import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.ReentrantLock;

public class Pista {
    protected Semaphore usar;
    protected String nombre;
```

```
    public Pista(Semaphore semaforo, String nombre) {
        this.usar = semaforo;
        this.nombre = nombre;
    }

}
```

Anexo 19 – Posicion

```
public class Posicion {
    protected int x;
    protected int y;
    //indica si debe pedir permiso para ir a la posicion
    public boolean necesitaPermiso;

    public Posicion(int x, int y, boolean necesitaPermiso) {
        this.x = x;
        this.y = y;
        this.necesitaPermiso = necesitaPermiso;
    }

}
```

Anexo 20 – Posiciones

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class Posiciones {

    public ArrayList<Posicion> aterrizar01;
    public ArrayList<Posicion> aterrizar06;
```

```
public ArrayList<Posicion> aterrizar19;
public ArrayList<Posicion> aterrizar24;

public ArrayList<Posicion> despegar01;
public ArrayList<Posicion> despegar06;
public ArrayList<Posicion> despegar19;
public ArrayList<Posicion> despegar24;
public ArrayList<Posicion> esperar;
public ArrayList<Posicion> taxear01porton;
public ArrayList<Posicion> taxear06porton;
public ArrayList<Posicion> taxear19porton;
public ArrayList<Posicion> taxear24porton;


public ArrayList<Posicion> taxearporton01;
public ArrayList<Posicion> taxearporton06;
public ArrayList<Posicion> taxearporton19;
public ArrayList<Posicion> taxearporton24;

public HashMap<String, ArrayList<Posicion>> mapa;

public int posicionEsperar = 0;


public Posiciones() {

    mapa = new HashMap<String, ArrayList<Posicion>>();
    aterrizar01 = new ArrayList<Posicion>();
    aterrizar06 = new ArrayList<Posicion>();
    aterrizar19 = new ArrayList<Posicion>();
```

```
aterrizar24 = new ArrayList<Posicion>();

despegar01 = new ArrayList<Posicion>();
despegar06 = new ArrayList<Posicion>();
despegar19 = new ArrayList<Posicion>();
despegar24 = new ArrayList<Posicion>();

esperar = new ArrayList<Posicion>();

taxear01porton = new ArrayList<Posicion>();
taxear06porton = new ArrayList<Posicion>();
taxear19porton = new ArrayList<Posicion>();
taxear24porton = new ArrayList<Posicion>();


taxearporton01 = new ArrayList<Posicion>();
taxearporton06 = new ArrayList<Posicion>();
taxearporton19 = new ArrayList<Posicion>();
taxearporton24 = new ArrayList<Posicion>();

mapa.put("src/Posiciones/Aterrizar01.txt", aterrizar01);
mapa.put("src/Posiciones/Aterrizar06.txt", aterrizar06);
mapa.put("src/Posiciones/Aterrizar19.txt", aterrizar19);
mapa.put("src/Posiciones/Aterrizar24.txt", aterrizar24);

mapa.put("src/Posiciones/Despegar01.txt", despegar01);
mapa.put("src/Posiciones/Despegar06.txt", despegar06);
mapa.put("src/Posiciones/Despegar19.txt", despegar19);
mapa.put("src/Posiciones/Despegar24.txt", despegar24);
```

```
    mapa.put("src/Posiciones/Esperar.txt", esperar);

    mapa.put("src/Posiciones/Taxear01Porton.txt", taxear01porton);
    mapa.put("src/Posiciones/Taxear06Porton.txt", taxear06porton);
    mapa.put("src/Posiciones/Taxear19Porton.txt", taxear19porton);
    mapa.put("src/Posiciones/Taxear24Porton.txt", taxear24porton);



    mapa.put("src/Posiciones/TaxearPorton01.txt", taxearporton01);
    mapa.put("src/Posiciones/TaxearPorton06.txt", taxearporton06);
    mapa.put("src/Posiciones/TaxearPorton19.txt", taxearporton19);
    mapa.put("src/Posiciones/TaxearPorton24.txt", taxearporton24);

    cargarPosiciones(this.mapa);

}

public static void cargarPosiciones(HashMap<String, ArrayList<Posicion>> mapa) {
    for (Map.Entry<String, ArrayList<Posicion>> entry : mapa.entrySet()) {

        String[] datos = ManejadorArchivosGenerico.leerArchivo(entry.getKey());
        for (String s : datos) {
            String[] arr = s.split(",");
            int x = Integer.valueOf(arr[0]);
            int y = Integer.valueOf(arr[1]);
            boolean permiso = false;
            if (arr.length == 3) {
                permiso = Boolean.valueOf(arr[2]);
            }
            entry.getValue().add(new Posicion(x, y, permiso));
```

```
        }
    }

}

}
```

Anexo 21 – Main Aeropuerto

```
import java.util.concurrent.Semaphore;



public class Proyecto {
    public static int numAviones = 10;


    public static void main(String[] args) {
        ManejadorArchivosGenerico.lineas = new StringBuffer();
        Semaphore rec = new Semaphore(1); //semaforo de recorrer para que solo permita que uno aterrice por vez
        Aeropuerto aeropuerto = new Aeropuerto(rec);
        Model model = new Model(aeropuerto);
        View view = new View(model);
        for (int i = 0; i < numAviones; i++) {
            Avion b = new Avion("Avion " + i, Avion.Estados.Esperando, aeropuerto); //construct a ball
            model.addBall(b);    //add it to the model
            b.registerObserver(view.getObserver());  //register view as an observer to it
            new AvionAnimator(b, model.getWidth(), model.getHeight(), i, rec); //start a thread to update it
        }
        new Recorrer(aeropuerto, rec).start();
        view.createAndShowGui();
    }
```

```
}

interface Observer {
    void onObservableChanged();
}
```

Anexo 22 – Recorrer

```
/*
Recorre constantemente la PriorityBlockingQueue con los aviones que pidieron permiso para aterizar,
y les da permiso.
 */

import java.util.concurrent.Semaphore;
import java.util.concurrent.locks.ReentrantLock;

public class Recorrer extends Thread implements Runnable {
    private Semaphore semaforo;
    public static boolean parador = true;
    private Aeropuerto aeropuerto;
    private final int DELAY = 100;

    public Recorrer(Aeropuerto aeropuerto, Semaphore s) {
        this.aeropuerto = aeropuerto;
        semaforo = s;
        new Thread(this).start();
    }

    @Override
    public void run() {
```

```
        while (parador) {
            //recorrer las tres listas de prioridad, una para aterrizar, otra para usar la pista1 y otra para usar la pista2
            while (!aeropuerto.getAvionesAterrizar().isEmpty()) {
                try {
                    semaforo.acquire();
                    Avion avion = aeropuerto.getAvionesAterrizar().poll();
                    if (avion != null) {
                        avion.setEstado(Avion.Estados.Aterrizar);
                    }
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }

/*
                synchronized (this) {
                    try {
                        this.wait();
                        System.out.println("LEVANTO");
                    } catch (InterruptedException e) {
                        throw new RuntimeException(e);
                    }
                }
                /*
                try {
                    //verificar cual es la pistaActiva del momento
                    //avion.aterrizar();
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }*/
            }
```

```
        }
    }

}
```

Anexo 23 – RotatedIcon

```
import javax.swing.*;
import java.awt.*;

/**
 *  The RotatedIcon allows you to change the orientation of an Icon by
 *  rotating the Icon before it is painted. This class supports the following
 *  orientations:
 *
 * <ul>
 * <li>DOWN - rotated 90 degrees
 * <li>UP (default) - rotated -90 degrees
 * <li>UPSIDE_DOWN - rotated 180 degrees
 * <li>ABOUT_CENTER - the icon is rotated by the specified degrees about its center.
 * </ul>
 */
public class RotatedIcon implements Icon
{
    public enum Rotate
    {
        DOWN,
        UP,
        UPSIDE_DOWN,
        ABOUT_CENTER;
    }
```

```
private Icon icon;

private Rotate rotate;

private double degrees;
private boolean circularIcon;

/**
 *  Convenience constructor to create a RotatedIcon that is rotated DOWN.
 *
 *  @param icon  the Icon to rotate
 */
public RotatedIcon(Icon icon)
{
    this(icon, Rotate.UP);
}

/**
 *  Create a RotatedIcon
 *
 *  @param icon     the Icon to rotate
 *  @param rotate  the direction of rotation
 */
public RotatedIcon(Icon icon, Rotate rotate)
{
    this.icon = icon;
    this.rotate = rotate;
}

/**
```

```
     *  Create a RotatedIcon. The icon will rotate about its center. This
     *  constructor will automatically set the Rotate enum to ABOUT_CENTER.
     *
     *  @param icon     the Icon to rotate
     *  @param degrees   the degrees of rotation
     */
    public RotatedIcon(Icon icon, double degrees)
    {
        this(icon, degrees, false);
    }

    /**
     *  Create a RotatedIcon. The icon will rotate about its center. This
     *  constructor will automatically set the Rotate enum to ABOUT_CENTER.
     *
     *  @param icon     the Icon to rotate
     *  @param degrees   the degrees of rotation
     *  @param circularIcon treat the icon as circular so its size doesn't change
     */
    public RotatedIcon(Icon icon, double degrees, boolean circularIcon)
    {
        this(icon, Rotate.ABOUT_CENTER);
        setDegrees( degrees );
        setCircularIcon( circularIcon );
    }

    /**
     *  Gets the Icon to be rotated
     *
     *  @return the Icon to be rotated
```

```
 */
public Icon getIcon()
{
   return icon;
}

/**
 *  Gets the Rotate enum which indicates the direction of rotation
 *
 *  @return the Rotate enum
 */
public Rotate getRotate()
{
   return rotate;
}

/**
 *  Gets the degrees of rotation. Only used for Rotate.ABOUT_CENTER.
 *
 *  @return the degrees of rotation
 */
public double getDegrees()
{
   return degrees;
}

/**
 *  Set the degrees of rotation. Only used for Rotate.ABOUT_CENTER.
 *  This method only sets the degress of rotation, it will not cause
 *  the Icon to be repainted. You must invoke repaint() on any
```

```
     *  component using this icon for it to be repainted.
     *
     *  @param degrees the degrees of rotation
     */
    public void setDegrees(double degrees)
    {
        this.degrees = degrees;
    }

    /**
     *  Is the image circular or rectangular? Only used for Rotate.ABOUT_CENTER.
     *  When true, the icon width/height will not change as the Icon is rotated.
     *
     *  @return true for a circular Icon, false otherwise
     */
    public boolean isCircularIcon()
    {
        return circularIcon;
    }

    /**
     *  Set the Icon as circular or rectangular. Only used for Rotate.ABOUT_CENTER.
     *  When true, the icon width/height will not change as the Icon is rotated.
     *
     *  @param circularIcon true for a circular Icon, false otherwise
     */
    public void setCircularIcon(boolean circularIcon)
    {
        this.circularIcon = circularIcon;
    }
```

```
//
//  Implement the Icon Interface
//

    /**
     *  Gets the width of this icon.
     *
     *  @return the width of the icon in pixels.
     */
    @Override
    public int getIconWidth()
    {
        if (rotate == Rotate.ABOUT_CENTER)
        {
            if (circularIcon)
                return icon.getIconWidth();
            else
            {
                double radians = Math.toRadians( degrees );
                double sin = Math.abs( Math.sin( radians ) );
                double cos = Math.abs( Math.cos( radians ) );
                int width = (int)Math.floor(icon.getIconWidth() * cos + icon.getIconHeight() * sin);
                return width;
            }
        }
        else if (rotate == Rotate.UPSIDE_DOWN)
            return icon.getIconWidth();
        else
            return icon.getIconHeight();
```

```
}

/**
 *  Gets the height of this icon.
 *
 *  @return the height of the icon in pixels.
 */
@Override
public int getIconHeight()
{
   if (rotate == Rotate.ABOUT_CENTER)
   {
      if (circularIcon)
         return icon.getIconHeight();
      else
      {
         double radians = Math.toRadians( degrees );
         double sin = Math.abs( Math.sin( radians ) );
         double cos = Math.abs( Math.cos( radians ) );
         int height = (int)Math.floor(icon.getIconHeight() * cos + icon.getIconWidth() * sin);
         return height;
      }
   }
   else if (rotate == Rotate.UPSIDE_DOWN)
      return icon.getIconHeight();
   else
      return icon.getIconWidth();
}

/**
```

```
     *  Paint the icons of this compound icon at the specified location
     *
     *  @param c The component on which the icon is painted
     *  @param g the graphics context
     *  @param x the X coordinate of the icon's top-left corner
     *  @param y the Y coordinate of the icon's top-left corner
     */
    @Override
    public void paintIcon(Component c, Graphics g, int x, int y)
    {
        Graphics2D g2 = (Graphics2D)g.create();

        int cWidth = icon.getIconWidth() / 2;
        int cHeight = icon.getIconHeight() / 2;
        int xAdjustment = (icon.getIconWidth() % 2) == 0 ? 0 : -1;
        int yAdjustment = (icon.getIconHeight() % 2) == 0 ? 0 : -1;

        if (rotate == Rotate.DOWN)
        {
            g2.translate(x + cHeight, y + cWidth);
            g2.rotate( Math.toRadians( 90 ) );
            icon.paintIcon(c, g2,  -cWidth, yAdjustment - cHeight);
        }
        else if (rotate == Rotate.UP)
        {
            g2.translate(x + cHeight, y + cWidth);
            g2.rotate( Math.toRadians( -90 ) );
            icon.paintIcon(c, g2,  xAdjustment - cWidth, -cHeight);
        }
        else if (rotate == Rotate.UPSIDE_DOWN)
```

```
        {
            g2.translate(x + cWidth, y + cHeight);
            g2.rotate( Math.toRadians( 180 ) );
            icon.paintIcon(c, g2, xAdjustment - cWidth, yAdjustment - cHeight);
        }
        else if (rotate == Rotate.ABOUT_CENTER)
        {
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
            g2.setClip(x, y, getIconWidth(), getIconHeight());
            g2.translate((getIconWidth() - icon.getIconWidth()) / 2, (getIconHeight() -
icon.getIconHeight()) / 2);
            g2.rotate(Math.toRadians(degrees), x + cWidth, y + cHeight);
            icon.paintIcon(c, g2, x, y);
        }

        g2.dispose();
    }
}
```

Anexo 24 – View

```
import javax.imageio.ImageIO;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.util.HashMap;

public class View {
    private final AvionPane avionPane;
```

```
private Model model;

public View(Model model) {
    ImageIcon avion = null;
    this.model = model;
    try {
        avion = new ImageIcon(ImageIO.read(new File("src/avion.png")));
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
    avionPane = new AvionPane(model);
}

void createAndShowGui() {
    JFrame frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocationRelativeTo(null);

    ImageIcon img = new ImageIcon("src/avion.png");
    frame.setIconImage(img.getImage());

    frame.add(avionPane);
    frame.setTitle("Sincronización Aeropuerto");

    String[] columnas = {"ID", "ESTADO", "PRIORIDAD"};
    String[][] filas = new String[model.getBalls().size()][3];
    int i = 0;
    DefaultTableModel m = (DefaultTableModel) avionPane.tabla.getModel();
    m.addColumn(columnas[0].toString());
    m.addColumn(columnas[1].toString());
```

```
        m.addColumn(columnas[2].toString());
        avionPane.filasTabla = new HashMap<String, Integer>();
        for (Avion b : model.getBalls()) {
            //agrega los aviones a la pantalla
            avionPane.filasTabla.put(b.nombre, i);
            i++;
            avionPane.add(b.getPanel());
            m.addRow(new Object[]{b.nombre, b.getEstado().toString(),
String.valueOf(b.getPrioridad())});
        }

        //frame.pack();

        frame.setSize(model.getWidth() + 15, model.getHeight() + 15);
        frame.setResizable(false);
        frame.setVisible(true);
    }

    Observer getObserver() {
        return avionPane;
    }

}
```

Anexo 25 – ManejadorDeArchivosGenericos

```
import java.io.*;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.StringTokenizer;
```

```
public class ManejadorArchivosGenerico {
    public static StringBuffer lineas;
    /**
     * @param nombreCompletoArchivo
     * @param listaLineasArchivo    lista con las lineas del archivo
     * @throws IOException
     */
    public static void escribirArchivo(String nombreCompletoArchivo, String[] listaLineasArchivo) {
        FileWriter fw;
        try {
            fw = new FileWriter(nombreCompletoArchivo, true);
            BufferedWriter bw = new BufferedWriter(fw);
            for (int i = 0; i < listaLineasArchivo.length; i++) {
                String lineaActual = listaLineasArchivo[i];
                bw.write(lineaActual);
                bw.newLine();
            }
            bw.close();
            fw.close();
        } catch (IOException e) {
            System.out.println("Error al escribir el archivo " + nombreCompletoArchivo);
            e.printStackTrace();
        }
    }

    public static String[] leerArchivo(String nombreCompletoArchivo) {
        FileReader fr;
        ArrayList<String> listaLineasArchivo = new ArrayList<String>();
        try {
            fr = new FileReader(nombreCompletoArchivo);
```

```
        BufferedReader br = new BufferedReader(fr);
        String lineaActual = br.readLine();
        while (lineaActual != null) {
            listaLineasArchivo.add(lineaActual);
            lineaActual = br.readLine();
        }
    } catch (FileNotFoundException e) {
        System.out.println("Error al leer el archivo " + nombreCompletoArchivo);
        e.printStackTrace();
    } catch (IOException e) {
        System.out.println("Error al leer el archivo " + nombreCompletoArchivo);
        e.printStackTrace();
    }
    //System.out.println("Archivo leido satisfactoriamente");

    return listaLineasArchivo.toArray(new String[0]);
}

/**
 * Metodo encargado de filtrar un texto, dejando solamente letras validas.-
 *
 * @param unaPalabra Palabra a evaluar
 * @return Cadena de caracteres limpia
 */
public static String filtrarPalabra(String unaPalabra) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < unaPalabra.length(); i++) {
        char caracter = unaPalabra.charAt(i);
        if ((caracter >= 'A' && caracter <= 'Z') ||
                (caracter >= 'a' && caracter <= 'z'))
```

```
            sb.append(caracter);
        }

        return sb.toString();
    }

    /**
     * Lee una string y devuelve las palabras que se encuentren dentro.
     *
     * @param strLine L�nea a procesar
     * @return Listado de palabras procesadas
     */
    public static String[] palabrasPorLinea(String strLine) {
        StringTokenizer st = new StringTokenizer(strLine);
        ArrayList<String> words = new ArrayList<String>();
        while (st.hasMoreTokens()) {
            String temp = st.nextToken();
            String word = temp.replaceAll("�", "a");
            word = word.replaceAll("�", "e");
            word = word.replaceAll("�", "i");
            word = word.replaceAll("�", "o");
            word = word.replaceAll("�", "u");
            word = word.replaceAll("�", "n");
            word = word.replaceAll("\\W+", "");
            if (word.compareTo("") != 0) {
                words.add(word);
                //System.out.println(word);
            }
        }
        return words.toArray(new String[0]);
```

```
    }
}
```

Anexo 26 – Test Aeropuerto

```
import org.junit.Assert;
import org.junit.Test;

import java.util.concurrent.Semaphore;

import static org.junit.Assert.*;

public class AvionTest {

    @Test
    public void TestAterrizar() {
        ManejadorArchivosGenerico.lineas = new StringBuffer();
        int numAviones = 5;
        Semaphore rec = new Semaphore(1); //semaforo de recorrer para que solo permita que uno aterrice por vez
        Aeropuerto aeropuerto = new Aeropuerto(rec);
        aeropuerto.setPistaActiva("19");
        Model model = new Model(aeropuerto);
        View view = new View(model);

        for (int i = 0; i < numAviones; i++) {
            Avion b = new Avion("Avion " + i, Avion.Estados.Esperando, aeropuerto); //construct a ball
            model.addBall(b);    //add it to the model
            b.registerObserver(view.getObserver());  //register view as an observer to it
            new AvionAnimator(b, model.getWidth(), model.getHeight(), i, rec); //start a thread to update it
        }
```

```
        Recorrer r = new Recorrer(aeropuerto, rec);
        r.parador = true;
        AvionAnimator.parador = true;
        r.start();
        try {
            Thread.sleep(50000);
            r.parador = false;
            AvionAnimator.parador = false;
            String linea[] = ManejadorArchivosGenerico.lineas.toString().split("\\.");
            assertEquals(linea[0], "Avion 0 va a usar la pista 19 para aterrizar");
            assertEquals(linea[1], "Avion 0 aterrizó y devolvio el uso de la pista 19");
            assertEquals(linea[2], "Avion 1 va a usar la pista 19 para aterrizar");
            assertEquals(linea[3], "Avion 1 aterrizó y devolvio el uso de la pista 19");
            assertEquals(linea[4], "Avion 2 va a usar la pista 19 para aterrizar");
            assertEquals(linea[5], "Avion 2 aterrizó y devolvio el uso de la pista 19");
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    @Test
    public void TestAterrizarConPrioridad() {
        ManejadorArchivosGenerico.lineas = new StringBuffer();
        int numAviones = 5;
        Semaphore rec = new Semaphore(1); //semaforo de recorrer para que solo permita que uno aterrice por vez
        Aeropuerto aeropuerto = new Aeropuerto(rec);
        aeropuerto.setPistaActiva("19");
        Model model = new Model(aeropuerto);
        View view = new View(model);
```

```
        for (int i = 0; i < numAviones; i++) {
            Avion b = new Avion("Avion " + i, Avion.Estados.Esperando, aeropuerto); //construct a ball
            model.addBall(b);    //add it to the model
            b.registerObserver(view.getObserver());  //register view as an observer to it
            new AvionAnimator(b, model.getWidth(), model.getHeight(), i, rec); //start a thread to update it
        }
        Avion b = new Avion("Avion 5", Avion.Estados.Esperando, aeropuerto); //construct  a ball
        b.setPrioridad(0);
        model.addBall(b);    //add it to the model
        b.registerObserver(view.getObserver());  //register view as an observer to it
        new AvionAnimator(b, model.getWidth(), model.getHeight(), 5, rec); //start a thread to update it
        Recorrer r = new Recorrer(aeropuerto, rec);
        r.parador = true;
        AvionAnimator.parador = true;
        r.start();
        try {
            Thread.sleep(40000);
            r.parador = false;
            AvionAnimator.parador = false;
            String linea[] = ManejadorArchivosGenerico.lineas.toString().split("\\.");
            assertEquals(linea[0], "Avion 5 va a usar la pista 19 para aterrizar");
            assertEquals(linea[1], "Avion 5 aterrizó y devolvio el uso de la pista 19");
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```