Manual de Funcionalidades

Simulador de gestor de procesos

Universidad Nacional Autónoma de Honduras

Sistemas Operativos I (IS-412)

Manual de funcionalidades diseñado por María Ramírez, revisado por Kelly Aguilar

Desarrolladores:

Lito Soler

Kelly Aguilar

María Ramírez

Orlando Durán

Mario Gómez

Tegucigalpa M.D.C, 19 de Agosto de 2018

Introducción

El presente reporte es un manual de funcionalidades de un simulador de procesos que imita la funcionalidad de un sistema operativo en cuanto al manejo de los procesos.

Podemos agregar que la información planteada es la estructura interna del programa, un desglose de toda la funcionalidad que el programa posee para el usuario si desea entender de forma coloquial lo que se programó.

Asimismo, cuenta con un análisis del programa utilizado, del entorno, de las variables y de las bases técnicas que existen en el mundo de programación que es necesario documentar.

Contenido

Índice de Tablas

1. Objetivos

1.1 Aplicar los conocimientos teóricos de los procesos que ejecuta un Sistema operativo a nivel de programación para diseñar un simulador de estos.

1.2 Aprender a diseñar un proyecto con todos los requerimientos y que sea funcional.

2. Guía de Instalación

El fichero de instalación y todos los materiales necesarios para la correcta aplicación del programa se encuentran en la carpeta:

Ejecutable > proyectoSO.jar

Es importante tener instalado el JDK de Java, para el correcto funcionamiento del programa. Una vez completado el proceso de instalación, se puede acceder al programa dando doble clic en el archivo.jar.

3. Confección

Simulador de Gestor de Procesos
Versión del Sistema: V.02
Tipo de Manual: Manual de Funcionalidades del Gestor de Procesos
Fecha de elaboración: sábado 4 de agosto de 2018
Área de elaboración: Tegucigalpa, M.D.C., Honduras, C.A

3.1. Fundamentos de las tecnologías Utilizadas

Se eligió el lenguaje de programación: Java, y como entorno para desarrollar el proyecto escogimos NetBeans. Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems.

Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son compiladas a bytecode (clase Java), que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

Se escogió Java por ser un lenguaje fácil de aprender, usar y para la mayor comodidad de todos los integrantes del grupo.

3.2. Actores del Sistema

Nombre del Estudiante	Descripción
Lito Soler	-Creación de las clases -Declaración de variables -Creación de funciones
Kelly Aguilar	-Diseño estructura de iteración -Revisión de Manuales -Personalización de la interfaz
Orlando Duran	-Validación de los procesos -Creación de la interfaz gráfica
Mario Gómez	-Elaboración de Manual de Usuario
María Fernanda Ramírez	-Elaboración de Manual de Funcionalidades

Tabla 1: Actores del sistema

4. Especificaciones funcionales

4.1. Descripción de Requisitos funcionales

- La información de los procesos deberá ser almacenada en un BCP
- Los procesos estarán almacenados en un archivo de texto plano
- El usuario indicara cuantos ciclos del procesador necesita utilizar
- El procesador manejara un temporizador de 5 ciclos
- Se deben representar por lo menos 5 TDA's que contengan los procesos en los estados de Nuevo (0), Listos (1), ejecutando (2), bloqueado (3) y saliente (4)
- Cada proceso manejara una prioridad, siendo la prioridad 1 la más alta y la prioridad 3 la más baja
- Si el mismo proceso está siendo ejecutado durante 3 segmentos seguidos, el gestor deberá bajar la prioridad y continuar con otros procesos de la lista

- La ejecución dentro del procesador se simulará por medio de una estructura de repetición
- El gestor podrá manejar desde 1 hasta 10 procesos

4.2. Requisitos No Funcionales

- Interfaz gráfica más agradable para el usuario
- Los procesos son seguros, ya que se pueden almacenar en una hoja de texto
- Es portable, ya que como se usó Java, se puede ejecutar en diferentes plataformas que posean previamente instalado JAVA.

5. Vista Funcional

5.1. Parte Proceso

Nombre de la función	**Resumen**	**Precondiciones**	**Poscondiciones**	**Requisitos Especiales**
crearBCP	Se declaran las variables pertinentes para inicializar el proceso, y luego un switch que capta los procesos y los bloquea según los parámetros asignados	La mayoría de las variables están inicializadas con valores establecidos, y otra condición que es random	Depende del número que se haya escogido con el random, si pasa a bloqueado y las diferentes categorías del programa	No hay requerimientos especiales
validarProceso	Validamos las distintas etapas de los procesos, para tener uno final, sin repeticiones de los mismos y la validez de los procesos	Como precondiciones contamos con los números que nos puede arrojar el programa, y	Retornamos la validez del proceso, así como la asignación de su respectivo id, y un mensaje para	No hay requerimientos especiales

		validar según sea la cantidad	dar a conocer que era válido	
crearTipoEvento	Tenemos una variable evento con la cual en un while se decide sobre ese valor lo que acontecerá después y el tipo de bloqueo	Se tiene la variable evento, con un valor inicial de 4, para luego tomar decisiones	Se retorna la variable evento según la decisión del random 3 y 5	No hay requerimientos especiales
crearAleatorioBCP	Se les asigna ciertos valores a las variables y luego con un switch se determina qué hacer en cada caso con los valores y generar un proceso válido	Como precondiciones contamos con los valores random y un tipo de evento como objeto	Tenemos dos randoms y otros valores que le asignamos para terminar el proceso	No hay requerimientos especiales
get y set	Obtiene el valor de dicha petición o lo devuelve	Según sea en tipo de variable, obtendremos su valor	Se devuelve el valor de la solicitud	No hay requerimientos especiales
toString	Con esta función tratamos de el formato de la impresión del proceso	Ocupamos el id, estado y la prioridad del proceso	Se convierten estos valores a los números que el usuario verá	No hay requerimientos especiales

Tabla 2: Vista funcional de la clase Proceso

5.2. Parte Formulario

Nombre de la función	**Resumen**	**Precondiciones**	**Poscondiciones**	**Requisitos Especiales**
Formulario	Esta función sólo sirve para tener un fondo de color blanco	Inicializar los componentes	Colocar el fondo blanco, con Background	No hay requerimientos especiales
btnSimulacionActionPerformed	Primero, se limpia la cola, luego se establecen nuevos procesos y los valida, luego decide si los añade a nuevos procesos si es menor que 10, sino a los demás procesos, por último, imprime las colas	Como precondiciones contamos con la creación de dicho proceso y la validación del mismo	Se le añade un id al último proceso (0), y se imprime las colas en la pantalla	No hay requerimientos especiales
btnGuardarActionPerformed	Sólo manda a llamar la función explorar	Sólo manda a llamar la función explorar	Sólo manda a llamar la función explorar	No hay requerimientos especiales
btnnuevaActionPerformed	Se comienza limpiando la cola y leer el archivo txt, entra a un ciclo while que verifica si están vacíos los procesos y luego un if que si la suma de todos los procesos es mayor que 9, comienza a a	Como precondiciones contamos con limpiar colas y leer el archivo while y luego la validación de los procesos	Tenemos imprimir la cola de los procesos y poder ejecutar el botón	No hay requerimientos especiales

	validar todos los procesos, entran en un case que decide en qué parte los añadirá, Luego imprime colas. En caso de error, entra en un catch que pide el nombre de la clase.			
btnEjecutarActionPerformed	Verifica si el usuario ingresó una cantidad de ciclos y luego comienza a verificar si están vacíos, y se ejecuta	La cantidad de ciclos ingresados por el usuario	La función de ejecución	No hay requerimientos especiales
exportar	Crea un nuevo archivo con la ruta procesos.txt y luego comienza a guardar cada proceso en ese archivo	El archivo con la ruta procesos.txt	Un catch con un mensaje de error	No hay requerimientos especiales
ordenarListas	Ordena las listas utilizadas para el manejo de los procesos	Creación previa de las listas para los procesos	Devuelve las listas ordenadas	No hay requerimientos especiales
obtenerProcesos	Va a prepararse para leer el archivo y cuando lo comienza a leer lo va a almacenar en un acumulador y va a crear	Una cadena de datos, un acumulador y el archivo que va a leer	Cerrar el archivo y crear el proceso	No hay requerimientos especiales

	un nuevo proceso con esos datos			
crearProcesos	Lee la cadena y la va a separar cada vez que encuentre ";", cuando la esté separando, cada uno de esos elementos será un dato de ese proceso y luego los añade para que el usuario pueda verlos	El string de proceso	Crear un nuevo proceso con los datos del anterior para que el usuario pueda verlos	No hay requerimientos especiales
guardarOrdenEjecucionProcesos	Va a ingresar en un try y va a crear el archivo ordenEjecucion.txt, y luego entra en un ciclo if para verificar que el archivo existía, si es así, simplemente va a guardar los datos. Cuando entra en el catch, hay un mensaje de error, luego el archivo está listo para el historial de ejecución	String ruta = "./ordenEjecucion.txt"	Almacenar esa información en txtHistorialEjecucion	No hay requerimientos especiales
verificarCantCiclos	Primero va a entrar a un try y va a convertir el dato a un int para leerlo y verificar que sea mayor que 0, y cuando	El txt de la cantidad de ciclos que el usuario ingresó	Ninguna	No hay requerimientos especiales

	entra en el catch es porque no hay cantidad de ciclos			
ejecucion	Primero se mandan a una caja de texto el número que ingresa el usuario para la cantidad de ciclos que quiere para la ejecución, y luego comienza a verificar si no están vacíos, luego obtiene todos los estados y según sea el caso, le elimina ese estado para añadirles la extensión hacia donde corresponden. Luego va verificando si a las colas dónde los mandó están listas y viendo cuáles son los procesos que tiene que mandar a bloqueo y los de prioridad más alta	Obtener y convertir el número que el usuario ingresó para saber cuál es la cantidad de ciclos que el usuario quiere	Imprime las colas de todos los procesos según donde correspondan y guarda esas información en una variable que luego será leída para saber en dónde quedó la ejecución	No hay requerimientos especiales
limpiarListas	Todos los parámetros los limpia, incluyendo las cajas de texto que el usuario mira	Todos los parámetros txt de impresión y las colas	Ninguna	No hay requerimientos especiales

imprimirListas	Inicializa los campos de texto en vacío, luego, en cada ciclo, verifica si la cola de impresión no está vacía, y luego manda a imprimir el proceso a dicho campo de texto asignado y le agrega un salto de línea para verse más ordenado	Inicializa los campos de texto de la impresión de colas en vacío al igual que los procesos simulados	Imprime las colas terminadas y la variable de procesos simulados, se llena con los datos de los procesos terminados	No hay requerimientos especiales

Tabla 3: Vista funcional Formulario

6. Vista Lógica

6.1. Patrón de Creación

Estos patrones crean objetos, evitando dicha función y la instanciación directa por parte de los desarrolladores. Nosotros, al usar Java, contamos con Abstract Factory y Factory, por su funcionalidad en las clases.

El patrón Abstract Factory está aconsejado cuando se prevé la inclusión de nuevas familias de productos, pero puede resultar contraproducente cuando se añaden nuevos productos o cambian los existentes, puesto que afectaría a todas las familias creadas. Factory es un patrón de diseño creacional y que sirve para construir una jerarquía de clases**.**

6.2. Patrón Estructural

Guían el desarrollo de la aplicación bajo una determinada estructura. Nosotros, al usar Java, contamos con el Patrón de datos de clase privada.

El patrón de diseño de datos de clase privado busca reducir la exposición de atributos al limitar su visibilidad. Reduce el número de atributos de clase al encapsularlos en un solo objeto de Datos. Permite que el diseñador de clase elimine el privilegio de escritura

de los atributos que están destinados a establecerse solo durante la construcción, incluso desde los métodos de la clase objetivo.

6.3. Patrón de Comportamiento

Se enfoca principalmente en algoritmos y en la asignación de responsabilidades entre los objetos.

- Nivel de clases: Usan la herencia como vía para distribuir el comportamiento. El Interpreter y el Template Method constituyen ejemplos de este tipo de patrón.

Nuestro sistema tiene que ser capaz de reconocer sentencias de un lenguaje previamente conocido (mediante su gramática), poder evaluar expresiones de este y ser capaz de ejecutar las sentencias recibidas. Se aplica cuando:

- Debemos trabajar con sentencias de un lenguaje que nuestro lenguaje de programación no reconoce automáticamente.
- La gramática del lenguaje con el que debemos trabajar es sencilla.
- No debe utilizarse si ya existe alguna clase nativa que interprete éste lenguaje.
- Nivel de objetos: Usan la composición, más que la herencia para llevar a cabo las tareas.

Por norma general definiremos los métodos que sean accedidos desde otra clase como public y los métodos que sean usados dentro de la propia clase como private.