using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace SyntaxisOS\_Simulacion

{

// Enumeraciones para el sistema

public enum TipoInvestigacion

{

Fisica,

Biologia,

Quimica,

CienciasComputacion,

Multidisciplinario

}

public enum RolUsuario

{

InvestigadorPrincipal,

Colaborador,

Becario,

Visitante

}

public enum EstadoProceso

{

Configurando,

Listo,

Ejecutando,

Bloqueado,

PausadoCheckpoint,

Finalizado

}

// Clase para representar un proceso científico

public class ProcesoCientifico

{

public int Id { get; set; }

public string Nombre { get; set; }

public EstadoProceso Estado { get; set; }

public TipoInvestigacion Tipo { get; set; }

public int Prioridad { get; set; }

public DateTime FechaCreacion { get; set; }

public double ConsumoMemoria { get; set; }

public ProcesoCientifico(int id, string nombre, TipoInvestigacion tipo)

{

Id = id;

Nombre = nombre;

Tipo = tipo;

Estado = EstadoProceso.Configurando;

Prioridad = 1;

FechaCreacion = DateTime.Now;

ConsumoMemoria = 0;

}

public void Ejecutar()

{

Estado = EstadoProceso.Ejecutando;

Console.WriteLine($"🔬 Proceso '{Nombre}' en ejecución...");

}

public void PausarCheckpoint()

{

Estado = EstadoProceso.PausadoCheckpoint;

Console.WriteLine($"⏸️ Proceso '{Nombre}' en checkpoint - Listo para reanudar");

}

}

// Clase para el planificador de procesos

public class PlanificadorProcesos

{

private List<ProcesoCientifico>[] colas;

private int totalProcesos;

public PlanificadorProcesos()

{

// 3 colas: Tiempo Real, Interactiva, Por Lotes

colas = new List<ProcesoCientifico>[3];

for (int i = 0; i < 3; i++)

{

colas[i] = new List<ProcesoCientifico>();

}

totalProcesos = 0;

}

public void AgregarProceso(ProcesoCientifico proceso, int cola)

{

if (cola >= 0 && cola < 3)

{

colas[cola].Add(proceso);

totalProcesos++;

Console.WriteLine($"📥 Proceso '{proceso.Nombre}' agregado a cola {cola + 1}");

}

}

public ProcesoCientifico SiguienteProceso()

{

// Algoritmo multicolas: prioridad a tiempo real, luego interactiva, luego por lotes

for (int i = 0; i < 3; i++)

{

if (colas[i].Count > 0)

{

var proceso = colas[i][0];

colas[i].RemoveAt(0);

totalProcesos--;

return proceso;

}

}

return null;

}

public void MostrarColas()

{

string[] nombresColas = { "TIEMPO REAL", "INTERACTIVA", "POR LOTES" };

for (int i = 0; i < 3; i++)

{

Console.WriteLine($"\n🎯 Cola {i + 1} - {nombresColas[i]}:");

if (colas[i].Count == 0)

{

Console.WriteLine(" Vacía");

}

else

{

foreach (var proceso in colas[i])

{

Console.WriteLine($" • {proceso.Nombre} [{proceso.Tipo}] - Prioridad: {proceso.Prioridad}");

}

}

}

}

}

// Clase para el sistema de archivos científico

public class SciFileSystem

{

private Dictionary<string, List<string>> estructura;

public SciFileSystem()

{

estructura = new Dictionary<string, List<string>>();

InicializarEstructura();

}

private void InicializarEstructura()

{

estructura["/users/"] = new List<string>();

estructura["/projects/"] = new List<string>();

estructura["/shared/collaborative/"] = new List<string>();

estructura["/repositories/datasets/"] = new List<string>();

estructura["/experiments/"] = new List<string>();

}

public void CrearProyecto(string idProyecto, string investigador)

{

string rutaProyecto = $"/projects/{idProyecto}/";

estructura[rutaProyecto] = new List<string>

{

"raw/",

"processed/",

"results/"

};

Console.WriteLine($"📁 Proyecto '{idProyecto}' creado para investigador '{investigador}'");

Console.WriteLine($" Estructura: {rutaProyecto}");

Console.WriteLine(" ├── raw/");

Console.WriteLine(" ├── processed/");

Console.WriteLine(" └── results/");

}

public void MostrarEstructura()

{

Console.WriteLine("\n🌳 Estructura del Sistema de Archivos SciFS:");

foreach (var directorio in estructura)

{

Console.WriteLine($"\n{directorio.Key}");

foreach (var subdir in directorio.Value)

{

Console.WriteLine($" ├── {subdir}");

}

}

}

}

// Clase para gestión de seguridad

public class SistemaSeguridad

{

private Dictionary<string, Tuple<string, RolUsuario>> usuarios;

public SistemaSeguridad()

{

usuarios = new Dictionary<string, Tuple<string, RolUsuario>>();

// Usuarios de ejemplo

RegistrarUsuario("dr\_rodriguez", "fisica123", RolUsuario.InvestigadorPrincipal);

RegistrarUsuario("lic\_gonzalez", "bio456", RolUsuario.Colaborador);

RegistrarUsuario("est\_perez", "quim789", RolUsuario.Becario);

}

public void RegistrarUsuario(string usuario, string credencial, RolUsuario rol)

{

usuarios[usuario] = new Tuple<string, RolUsuario>(credencial, rol);

}

public bool AutenticarUsuario(string usuario, string credencial, string token = "token\_ciencia")

{

if (usuarios.ContainsKey(usuario) && usuarios[usuario].Item1 == credencial)

{

Console.WriteLine($"🔐 Autenticación exitosa: {usuario} - Rol: {usuarios[usuario].Item2}");

Console.WriteLine($" Token de investigación: {token}");

Console.WriteLine(" ✅ Autenticación multifactor completada");

return true;

}

Console.WriteLine("❌ Error de autenticación");

return false;

}

public bool VerificarPermiso(string usuario, string recurso, string operacion)

{

if (usuarios.ContainsKey(usuario))

{

var rol = usuarios[usuario].Item2;

// Lógica simplificada de control de acceso

if (rol == RolUsuario.InvestigadorPrincipal)

return true;

if (rol == RolUsuario.Colaborador && operacion != "eliminar")

return true;

if (rol == RolUsuario.Becario && operacion == "lectura")

return true;

}

return false;

}

}

// Clase principal del sistema

public class SyntaxisOS

{

private PlanificadorProcesos planificador;

private SciFileSystem sistemaArchivos;

private SistemaSeguridad seguridad;

private List<ProcesoCientifico> procesosActivos;

public SyntaxisOS()

{

planificador = new PlanificadorProcesos();

sistemaArchivos = new SciFileSystem();

seguridad = new SistemaSeguridad();

procesosActivos = new List<ProcesoCientifico>();

}

public void DemostrarFuncionalidades()

{

Console.WriteLine("🚀 INICIANDO SYNTaxisOS - SISTEMA OPERATIVO CIENTÍFICO");

Console.WriteLine("====================================================\n");

// Demostración de seguridad

Console.WriteLine("1. 🔐 SISTEMA DE SEGURIDAD");

Console.WriteLine("----------------------------");

seguridad.AutenticarUsuario("dr\_rodriguez", "fisica123");

Console.WriteLine();

// Demostración de sistema de archivos

Console.WriteLine("2. 📁 SISTEMA DE ARCHIVOS SCIFS");

Console.WriteLine("--------------------------------");

sistemaArchivos.CrearProyecto("FISICA\_001", "dr\_rodriguez");

sistemaArchivos.CrearProyecto("BIO\_045", "lic\_gonzalez");

sistemaArchivos.MostrarEstructura();

Console.WriteLine();

// Demostración de planificación de procesos

Console.WriteLine("3. ⚙️ PLANIFICACIÓN DE PROCESOS CIENTÍFICOS");

Console.WriteLine("-------------------------------------------");

// Crear procesos de ejemplo

var proceso1 = new ProcesoCientifico(1, "Simulación Partículas", TipoInvestigacion.Fisica);

var proceso2 = new ProcesoCientifico(2, "Análisis Genoma", TipoInvestigacion.Biologia);

var proceso3 = new ProcesoCientifico(3, "Procesamiento Datos Climáticos", TipoInvestigacion.Multidisciplinario);

// Agregar a diferentes colas

planificador.AgregarProceso(proceso1, 0); // Tiempo real

planificador.AgregarProceso(proceso2, 1); // Interactiva

planificador.AgregarProceso(proceso3, 2); // Por lotes

Console.WriteLine("\nEstado de las colas:");

planificador.MostrarColas();

// Ejecutar algunos procesos

Console.WriteLine("\n4. 🎯 EJECUCIÓN DE PROCESOS");

Console.WriteLine("----------------------------");

var proxProceso = planificador.SiguienteProceso();

if (proxProceso != null)

{

proxProceso.Ejecutar();

procesosActivos.Add(proxProceso);

}

// Demostrar checkpoint

Console.WriteLine("\n5. 💾 SISTEMA DE CHECKPOINT");

Console.WriteLine("---------------------------");

foreach (var proceso in procesosActivos)

{

proceso.PausarCheckpoint();

}

Console.WriteLine("\n✨ Demostración de SyntaxisOS completada");

}

}

// Programa principal

class Program

{

static async Task Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.UTF8;

// Crear instancia del SO

var syntaxisOS = new SyntaxisOS();

// Ejecutar demostración

syntaxisOS.DemostrarFuncionalidades();

// Esperar antes de cerrar

Console.WriteLine("\n\nPresiona cualquier tecla para salir...");

Console.ReadKey();

}

}

}

Preguntas

El entregable debe incluir:

1. Nombre del sistema operativo y su propósito.

Nombre: SyntaxisOS

Propósito: Sistema operativo especializado para investigación científica multidisciplinaria, diseñado para manejar grandes volúmenes de datos, simulaciones complejas y colaboración en tiempo real entre investigadores de diferentes disciplinas.

1. Tipo de núcleo elegido (monolítico, microkernel, híbrido, etc.) y justificación. Tipo: Microkernel con extensiones híbridas

Justificación:

Microkernel base: Proporciona estabilidad y seguridad al ejecutar servicios esenciales en espacio de usuario

Extensiones híbridas: Permiten alto rendimiento para operaciones científicas intensivas

Beneficios: Mayor modularidad, fácil actualización de componentes, tolerancia a fallos

Específico para ciencia: Aislamiento de experimentos críticos, capacidad de reiniciar servicios sin afectar investigaciones en curso

1. Gestión de procesos: planificación, estados, concurrencia.

Planificación: Algoritmo multicolas adaptativo con prioridades dinámicas

Cola de tiempo real: Simulaciones críticas y adquisición de datos

Cola interactiva: Análisis en tiempo real y visualización

Cola por lotes: Procesamiento de datos masivos

Estados de procesos:

Nuevo estado: "Configurando" - preparación de entorno experimental

Estados tradicionales: Listo, Ejecutando, Bloqueado

Estado especial: "Pausado/Checkpoint" - para experimentos largos

Concurrencia:

Modelo de actores para procesamiento distribuido

Semántica de transacciones para operaciones críticas

Memoria compartida controlada para colaboración

4. Gestión de memoria: segmentación, paginación, asignación.

Esquema: Paginación jerárquica con segmentación lógica

Nivel 1: Paginación tradicional para sistema operativo

Nivel 2: Segmentación por tipo de investigación (física, biología, etc.)

Nivel 3: Paginación específica por proyecto

Asignación:

Asignación dinámica con pre-reserva para experimentos programados

Compartición controlada entre proyectos colaborativos

Swapping inteligente basado en patrones de acceso

Características especiales:

Memoria no volátil integrada para resultados críticos

Caché multinivel para datasets frecuentes

5. Sistema de archivos: estructura, permisos, jerarquía.

/syntaxis/

├── /users/[investigador]/ # Espacio personal

├── /projects/[id\_proyecto]/ / # Datos de proyecto

│ ├── /raw/ , # Datos crudos

│ ├── /processed/ # Datos procesados

│ └── /results/ # Resultados finales

├── /shared/collaborative/ # Espacio colaborativo

├── /repositories/datasets/ # Datasets públicos └── /experiments/[timestamp]/ # Ejecuciones experimentales

Permisos:

Basados en roles: Investigador Principal, Colaborador, Becario

Control de acceso temporal (experimentos con fecha de expiración) Herencia contextual de permisos

CARACTERISTICAS:

Versionado automático de datos científicos

Metadatos extensibles para documentación de experimentos Búsqueda semántica en datasets

6. Mecanismos de seguridad: autenticación, control de acceso, cifrado.

Autenticación:

Multifactor: credenciales + biometría + token de investigación Autenticación federada entre instituciones

Control de Acceso:

Modelo RBAC (Role-Based Access Control) científico

Políticas de acceso basadas en ética de investigación Auditoría completa de acceso a datos sensibles

Cifrado:

Cifrado extremo a extremo para datos en reposo y tránsito

Claves específicas por proyecto de investigación Eliminación segura certificada

1. Interfaz de usuario: CLI, GUI, voz, gestos, etc.

Modo Universal: Adaptable al contexto de investigación

CLI Avanzada: SyntaxisShell con autocompletado contextual y pipelines científicos

GUI Holográfica: Visualización 3D de datos complejos y simulaciones

Interfaz por Voz: Para hands-free durante experimentos de laboratorio

Realidad Aumentada: Superposición de datos en equipos experimentales

Características de UI:

Notebooks ejecutables integrados

Dashboards personalizables por disciplina

Colaboración síncrona con edición concurrente

1. Comparación con un sistema operativo real (Linux, Windows, Android, etc.).

Windows VS Android vs. Linux para Ciencia:

Ventajas: Especialización científica nativa, gestión de datos integrada, colaboración built-in

Desventajas: Menor soporte de hardware general, curva de aprendizaje más pronunciada

Android vs. Linux vs. Windows:

Ventajas: Optimización específica para carga científica, seguridad por diseño, costo-beneficio Desventajas: Menor ecosistema de aplicaciones generales

Puntos innovadores:

Gestión nativa de flujos de trabajo científicos

Integración de ética y cumplimiento normativo Colaboración multidisciplinaria facilitada

1. Reflexión final: ¿Qué aprendiste al diseñar tu propio SO?
2. El diseño de SO va más allá de lo técnico: Debe resolver problemas reales de usuarios específicos
3. Las compensaciones (trade-offs) son inevitables: Cada decisión de diseño implica ganar algo y perder algo else
4. La especialización requiere profundidad: Un SO para ciencia necesita entender profundamente los flujos de trabajo científicos
5. La usabilidad es crucial: incluso en entornos técnicos, la interfaz determina la productividad
6. La seguridad debe ser por diseño: No puede ser una consideración posterior en sistemas que manejan datos sensibles

Aprendizaje más valioso: Entender que un sistema operativo es un facilitador de actividades humanas. No es solo un administrador de recursos, sino un amplificador de capacidades humanas en contextos específicos. Esta perspectiva holística - técnica pero centrada en el usuario - es lo que diferencia un buen diseño de uno excepcional.