INFORME TÉCNICO: SISTEMA DE GESTIÓN DE PROCESOS

Asignatura: Estructura de Datos

Ciclo: 2025-10

Grupo: GRUPO 3

Integrantes:

- Huaraca Huaraca Jhafeth Frank

CAPÍTULO 1: ANÁLISIS DEL PROBLEMA

1.1 Descripción del problema

El presente proyecto desarrolla un Sistema de Gestión de Procesos que simula un entorno de sistema operativo mediante estructuras de datos dinámicas lineales. El objetivo es gestionar tareas (procesos) considerando tres componentes fundamentales:

- Gestor de Procesos: almacenamiento y administración de todos los procesos activos.
- Planificador de CPU: ejecución ordenada de procesos según su prioridad.
- Gestor de Memoria: asignación y liberación temporal de bloques de memoria.

Este sistema permite aplicar conceptos teóricos de estructuras de datos a un problema real de ingeniería, cumpliendo con los principios de eficiencia, modularidad y persistencia.

1.2 Requerimientos del sistema

Funcionales

- Registrar procesos con: ID único, nombre, prioridad (1–10), estado y memoria asignada.
- Eliminar procesos por ID.
- Buscar procesos por ID o nombre.
- Modificar la prioridad de un proceso existente.
- Enviar procesos a una cola de prioridad (1 = más alta).
- "Ejecutar" el proceso de mayor prioridad (desencolar).
- Asignar memoria a procesos mediante push en una pila.
- Liberar memoria mediante pop (LIFO).
- Mostrar el estado actual de procesos, cola y pila.
- Guardar y cargar el estado del sistema en un archivo binario (`estado.dat`).

No funcionales

- Implementación en DEV C++.
- Código modularizado, comentado y sin librerías predefinidas para estructuras (`<stack>`, `<queue>`, `<list>` prohibidas).
- Interfaz de consola con validación de entradas.
- IDs consecutivos (1, 2, 3, …).
- Persistencia funcional entre sesiones.

1.3 Estructuras de datos propuestas

Gestor de Procesos	Lista enlazada	Permite inserción/eliminación eficiente sin reasignar memoria. Ideal para número dinámico de procesos.
Planificador de CPU	Cola de prioridad(lista ordenada)	Garantiza que el proceso de mayor prioridad esté siempre al frente para ejecución inmediata.
Gestor de Memoria	Pila	Modelo LIFO (último en entrar, primero en salir) refleja cómo los SO gestionan memoria temporal.

1.4 Justificación de la elección

Estas estructuras son dinámicas, lo que significa que no tienen límite fijo de tamaño (a diferencia de los arreglos estáticos). Esto es esencial porque el número de procesos en un sistema operativo real no es conocido a priori.

Además:

- La lista enlazada permite recorrer, buscar y modificar procesos en tiempo lineal.

- La cola de prioridad ordenada al insertar evita reordenar constantemente, optimizando la ejecución.
- La pila es la estructura natural para la gestión de memoria LIFO (ej: pilas de llamadas, memoria temporal).

CAPÍTULO 2: DISEÑO DE LA SOLUCIÓN

2.1 Descripción de estructuras y operaciones

Lista enlazada (ListaEnlazada)

- Nodos con Proceso y puntero al siguiente.
- Operaciones: insertar(), eliminar(), buscar(ID), buscar(nombre), mostrar(), guardarEnArchivo(), cargarDeArchivo().

Cola de prioridad (ColaPrioridad)

- Lista ordenada por prioridad (1 = más alta).
- Operaciones: encolar() (inserta ordenado), desencolar(), mostrar().

Pila (PilaMemoria)

- Nodos con BloqueMemoria (idProceso, tamaño).
- Operaciones: push(), pop(), mostrar().

2.2 Algoritmos principales (Pseudocódigo)

Agregar proceso

INICIO

id ← últimoId + 1

LEER nombre, prioridad

VALIDAR prioridad entre 1 y 10

ESTADO ← "listo"

MEMORIA ← 0

INSERTAR en listaEnlazada

GUARDAR últimoId

FIN

Encolar por prioridad

INICIO encolar(Proceso p)

CREAR nuevoNodo

SI cola vacía O p.prioridad < frente.prioridad

nuevoNodo.siguiente = frente

frente = nuevoNodo

SINO

actual = frente

MIENTRAS actual.siguiente ≠ NULL Y actual.siguiente.prioridad ≤ p.prioridad

actual = actual.siguiente

FIN MIENTRAS

nuevoNodo.siguiente = actual.siguiente

actual.siguiente = nuevoNodo

FIN SI

FIN

Asignar memoria (push)

INICIO push(Bloque b)

CREAR nuevoNodo

nuevoNodo.dato = b

nuevoNodo.siguiente = tope

tope = nuevoNodo

FIN

2.3 Diagramas de flujo (descripción textual)

- Menú principal:
  Inicio → Mostrar opciones → Leer opción → Ejecutar función → Volver al menú → Salir y guardar.
- Encolar:
  Insertar → ¿Es el más prioritario? → Sí → Inserción al inicio → No → Buscar posición → Insertar.

2.4 Justificación del diseño

- Ventajas:
  - Modularidad: cada estructura es independiente.
  - Eficiencia: operaciones críticas (pop, desencolar) son O(1).
  - Escalabilidad: se pueden agregar más atributos sin romper la lógica.
- Persistencia: se guarda el ultimo Id y los procesos, evitando duplicados.

CAPÍTULO 3: SOLUCIÓN FINAL

3.1 Código fuente

El código está implementado en 5 archivos modulares:

- main.cpp: menú e interacción
- proceso.h: definición de struct Proceso
- lista_enlazada.h: gestor de procesos

- cola_prioridad.h: planificador de CPU
- pila_memoria.h: gestor de memoria

Características:

- Sin STL.
- Compatibilidad con DEV C++ (usa 0 en lugar de nullptr).
- IDs consecutivos (1, 2, 3…).
- Persistencia en estado.dat.

3.2 Manual de usuario

1. Ejecute el programa en DEV C++.
2. Use el menú:
    - Opción 1: Agregar proceso.
    - Opción 6: Enviar a cola de prioridad.
    - Opción 7: Ejecutar siguiente proceso.
    - Opción 8/9: Asignar/liberar memoria.
    - Opción 0: Guardar y salir.

3. Al reiniciar, se carga el estado previo.

CAPÍTULO 4: ANÁLISIS DE COMPLEJIDAD

⚠ Nota: No se mide en tiempo real. Se explica qué hace cada estructura de control.

Buscar proceso por ID	`while (actual)`	Recorre la lista nodo por nodo hasta encontrar el ID → O(n)
Encolar por prioridad	`while (actual->siguiente && ...)`	Busca la posición correcta comparando prioridades → O(n)
Desencolar	Sin bucle	Elimina el primer nodo directamente → O(1)
Push / Pop	Sin bucle	Inserta o elimina del tope → O(1)
Guardar estado	`while (actual)`	Recorre toda la lista para escribir en archivo → O(n)

Este análisis demuestra que el diseño equilibra eficiencia y funcionalidad, eligiendo el método adecuado para cada operación.

CAPÍTULO 5: EVIDENCIAS DE TRABAJO EN EQUIPO

5.1 Repositorio GitHub

- Enlace público: https://github.com/frank7774/SistemaProcesos-Dev
- Commits de todos los integrantes.
- Estructura organizada: codigo/, presentacion/, GRUPO3_Evaluacion01_Con2.pdf

5.2 Plan de trabajo

- Semana 1: Diseño + roles.
- Semana 2: Implementación + retroalimentación.
- Semana 3: Pruebas + documentación.

5.3 Roles y actas

- Roles definidos y rotativos.
- Actas de reunión semanales registradas.
- Uso de GitHub Projects para seguimiento.