“AÑO DE LA RECUPERACIÓN Y CONSOLIDACIÓN DE LA ECONOMÍA PERUANA”

ASIGNATURA

ESTRUCTURA DE DATOS

Proyecto Grupal

SISTEMA DE GESTIÓN DE PROCESOS

DOCENTE:

Medina Flores, Jimmy Roberto

INTEGRANTES:

- Alarcon Chiquillan Mileydi Diana
- Ruiz Cardenas Angeli Mayli
- Chahuayla Castro Kenny Smith

2025

Índice

I. ANÁLISIS DEL PROGRAMA:

1. 1 Descripción

En la actualidad, los sistemas operativos requieren una administración eficiente de los procesos para asegurar un buen desempeño y una distribución adecuada de los recursos disponibles. Este proyecto busca replicar ese proceso mediante el desarrollo de un sistema capaz de registrar procesos, organizarlos según su nivel de prioridad y gestionar la memoria que se les asigna. El propósito principal es reforzar el uso práctico de estructuras dinámicas lineales y aplicar conocimientos de programación en la solución de situaciones similares a las del mundo real.

1. **2 Requerimientos del sistema:**

1.2.1 Funcionales:

- Permitir el registro de procesos con ID, nombre y prioridad.
- Planificar la ejecución de procesos en base a su prioridad.
- Gestionar el uso de la memoria de forma dinámica por proceso.
- Mostrar visualmente el estado de cada estructura (lista, cola y pila).
- Brindar opciones para modificar, eliminar o buscar procesos.

1.2.2 No funcionales:

- El sistema debe ser rápido y eficiente en la ejecución de operaciones.
- Debe tener una interfaz amigable por consola.
- Debe permitir la persistencia de datos (guardar y recuperar estados).
- El código debe estar bien estructurado, modular y comentado.

1. 3 Estructura de datos propuesta:

- **Lista enlazada**: Para mantener un registro de todos los procesos activos.
- **Cola de prioridad**: Para organizar los procesos que deben ejecutarse según la prioridad.
- **Pila**: Para administrar la memoria asignada y liberada de manera eficiente.

1. 4 Justificación de la elección:

Para el desarrollo del Sistema de Gestión de Procesos, se eligieron tres estructuras dinámicas lineales: lista enlazada, pila y cola de prioridad, debido a que cada una responde eficazmente a los requerimientos funcionales del sistema y simula comportamientos reales observados en los sistemas operativos. A continuación, se detalla la justificación de cada una:

Lista Enlazada – Registro de Procesos

Esta estructura fue seleccionada por su capacidad de almacenar y manipular datos de forma dinámica, permitiendo insertar y eliminar procesos sin necesidad de reorganizar toda la estructura, como ocurriría en un arreglo. Además, facilita la búsqueda y modificación de datos, lo cual es fundamental para la administración de procesos activos.

Cola de Prioridad – Planificador de CPU

La cola fue adaptada para manejar prioridades, simulando así una **cola de prioridad**, donde los procesos con mayor prioridad se ejecutan antes que los de menor prioridad. Esta estructura reproduce fielmente el funcionamiento de los **planificadores de procesos**

de un sistema operativo, donde el orden de ejecución no siempre depende del momento de llegada, sino de la urgencia o importancia del proceso.

Pila – Gestión de Memoria
La pila se utiliza para administrar la memoria asignada a los procesos, ya que sigue el principio **LIFO (Last In, First Out)**. Esto refleja cómo en muchos casos los sistemas operativos manejan bloques de memoria temporal (como el stack de llamadas de función). Su implementación permite **asignar y liberar memoria** de manera eficiente y controlada.

2. DISEÑO DE LA SOLUCIÓN:

2. 1. Descripción de estructuras de datos y operaciones

2.1.1. Lista Enlazada – Registro de Procesos

La lista enlazada permite mantener un registro dinámico de todos los procesos activos en el sistema. Cada nodo de la lista contiene la información de un proceso (ID, nombre, prioridad) y un puntero al siguiente nodo.

Operaciones principales:

- **Insertar proceso:** Agrega un nuevo nodo al final de la lista con los datos de un nuevo proceso.
- **Mostrar procesos:** Recorre la lista e imprime los procesos almacenados.
- **Buscar proceso:** Permite encontrar un proceso según su ID, devolviendo una referencia a este si existe.
- **Eliminar proceso:** Elimina el nodo correspondiente a un ID específico, ajustando los enlaces de la lista.

2.1.2. Cola de Prioridad – Planificación de Procesos

Descripción:
Esta estructura organiza los procesos pendientes de ejecución según su prioridad. Los procesos con mayor prioridad (valor numérico menor) se atienden primero, simulando el comportamiento de un planificador de CPU real.

Operaciones principales:

- **Insertar proceso:** Añade un nuevo proceso manteniendo el orden según la prioridad.
- **Atender proceso:** Extrae y devuelve el proceso con mayor prioridad para su ejecución.

- **Mostrar cola:** Visualiza el estado actual de los procesos en espera, en orden de prioridad.

2.1.3. Pila – Gestión de Memoria

Descripción:
La pila se utiliza para simular la asignación y liberación de bloques de memoria por proceso. Funciona bajo el principio LIFO (Last In, First Out), apropiado para representar estructuras como la pila de llamadas de funciones en un sistema operativo.

Operaciones principales:

- **Asignar bloque de memoria:** Agrega un nuevo bloque de memoria a la cima de la pila.
- **Liberar bloque de memoria:** Elimina el bloque más recientemente asignado.
- **Mostrar pila:** Muestra el estado actual de la memoria utilizada.
  -

2. 2. Algoritmos principales:

a) Registro de proceso

```
crearNodo(ID, nombre, prioridad);
insertarEnLista(nuevoNodo);
insertarEnColaPrioridad(nuevoNodo);
```

b) Ejecución de proceso

```
proceso = desencolarPrioridad();
if (proceso != NULL) {
   asignarMemoria(proceso);
   ejecutar(proceso);
   liberarMemoria(proceso);
}
```

c) Gestión de memoria

```
void asignarMemoria(Proceso p) {
   apilar(p.ID, bloqueMemoria);
}
void liberarMemoria(Proceso p) {
   desapilar();
}
```

d) Buscar proceso

```
recorrerLista(ID o nombre);
mostrarProceso();
```

e) Modificar proceso

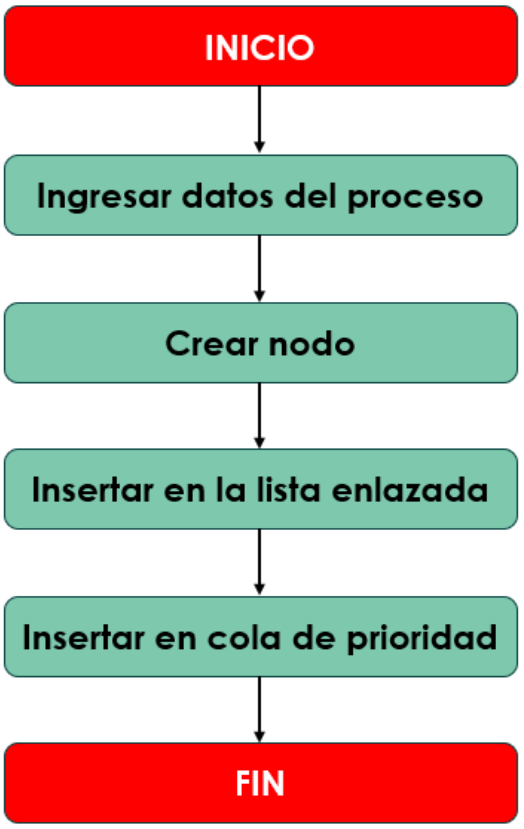
```
buscarEnLista(ID);
si existe -> modificar nombre o prioridad;
```

f) Eliminar proceso

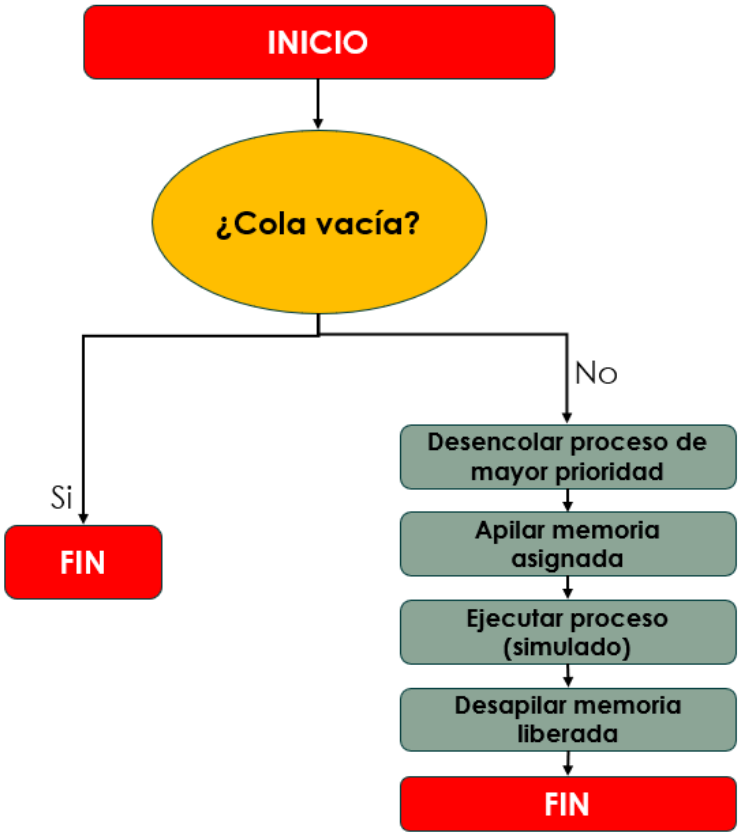
```
buscarYEliminarDeLista(ID);
eliminarDeCola(ID);
si en ejecución -> liberarMemoria();
```

2. 3. Diagramas de flujo:

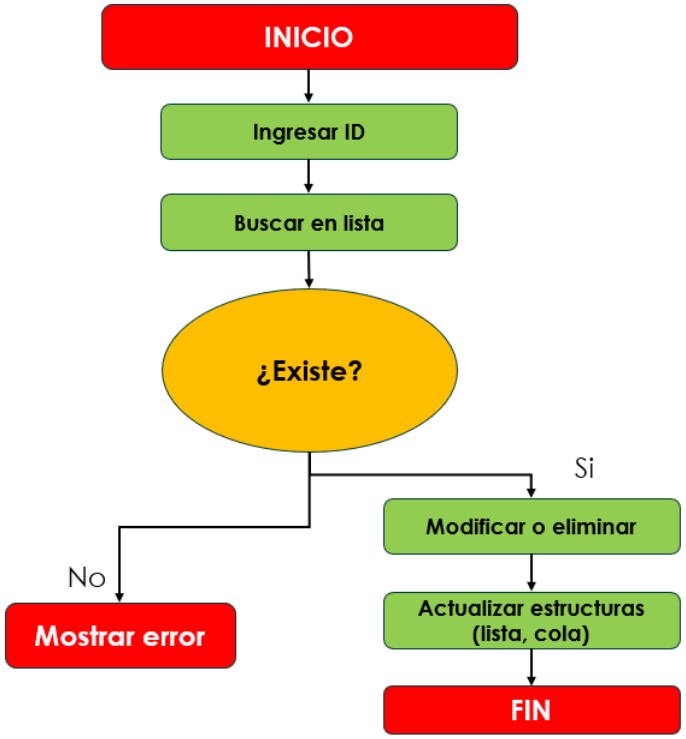
Registro de proceso:



Planificación de proceso:



Modificación y/o eliminación de proceso:



2. 4. Justificación del diseño:

Este diseño propone una solución modular, clara y alineada con los principios de la programación orientada a objetos y estructuras dinámicas. Cada estructura cumple una función específica:

- **Lista Enlazada** permite la gestión dinámica y flexible de procesos activos.
- **Cola de Prioridad** simula fielmente el planificador de CPU, optimizando la ejecución según urgencia.
- **Pila se adapta** perfectamente a la simulación del uso de memoria en sistemas reales, permitiendo un control efectivo de asignaciones temporales.

3. SOLUCIÓN FINAL:

3. 1. Código:

Código principal:main.cpp

```cpp
#include <iostream>
using namespace std;
#include "proceso.h"
#include "lista_enlazada.h"
#include "cola_prioridad.h"
#include "pila_memoria.h"

int main() {
    setlocale(LC_CTYPE,"Spanish");
    ListaEnlazada lista;
    ColaPrioridad cola;
    PilaMemoria pila;
    int opcion;
    do {
        cout << "\n--- MENÚ PRINCIPAL ---\n";
        cout << "1. Registrar proceso\n";
        cout << "2. Ejecutar proceso\n";
        cout << "3. Liberar memoria\n";
        cout << "4. Mostrar estructuras\n";
        cout << "5. Salir\n";
        cout << "Ingrese opción: ";
        cin >> opcion;
        switch(opcion) {
            case 1: {
                int id, prioridad;
                string nombre;
                cout << "ID: "; std::cin >> id;
                cout << "Nombre: "; std::cin >> nombre;
                cout << "Prioridad (1=Alta, 2=Media, 3=Baja): "; std::cin >> prioridad;
                Proceso p(id, nombre, prioridad);
                lista.insertar(p);
                cola.encolar(p);
                pila.asignarMemoria(id);
                break;
            }
            case 2:
                cola.ejecutar();
                break;
            case 3:
                pila.liberarMemoria();
                break;
            case 4:
                lista.mostrar();
                cola.mostrar();
                pila.mostrar();
                break;
        }
    } while (opcion != 5);
    return 0;
}
```

LIBRERÍAS:

cola_prioridad:

```cpp
#ifndef COLA_PRIORIDAD_H
#define COLA_PRIORIDAD_H

#include <queue>
#include <vector>
#include <iostream>
#include "proceso.h"

```

```
class ColaPrioridad {
private:
    struct Comparador {
        bool operator()(Proceso a, Proceso b) {
            return a.prioridad > b.prioridad; // menor valor = mayor prioridad
        }
    };
    std::priority_queue<Proceso, std::vector<Proceso>, Comparador> cola;
public:
    void encolar(Proceso p) {
        cola.push(p);
    }
    void ejecutar() {
        if (!cola.empty()) {
            Proceso p = cola.top();
            cola.pop();
            std::cout << "Ejecutando proceso ID " << p.id << ": " << p.nombre << "\n";
        } else {
            std::cout << "No hay procesos por ejecutar\n";
        }
    }
    void mostrar() {
        std::cout << "Procesos listos para ejecuciÃ³n: " << cola.size() << "\n";
    }
};

#endif
```

lista_enlazada:

```
#ifndef LISTA_ENLAZADA_H
#define LISTA_ENLAZADA_H

#include <iostream>
#include "proceso.h"

using namespace std;

struct Nodo {
    Proceso proceso;
    Nodo* siguiente;

    Nodo(Proceso p) : proceso(p), siguiente(NULL) {}  //
};

class ListaEnlazada {
private:
    Nodo* cabeza;

public:
    ListaEnlazada() : cabeza(NULL) {}  //

    void insertar(Proceso p) {
        Nodo* nuevo = new Nodo(p);
        if (!cabeza) {
            cabeza = nuevo;
        } else {
            Nodo* temp = cabeza;
```

```
            Nodo* temp = cabeza;
            while (temp->siguiente)
                temp = temp->siguiente;
            temp->siguiente = nuevo;
        }
    }

    void mostrar() {
        Nodo* temp = cabeza;
        cout << "\n--- Lista de procesos ---\n";
        while (temp) {
            cout << "ID: " << temp->proceso.id
                 << ", Nombre: " << temp->proceso.nombre
                 << ", Prioridad: " << temp->proceso.prioridad << endl;
            temp = temp->siguiente;
        }
    }

    Proceso* buscar(int id) {
        Nodo* temp = cabeza;
        while (temp) {
            if (temp->proceso.id == id)
                return &temp->proceso;
            temp = temp->siguiente;
        }
        return NULL;
    }
```

```
    bool eliminar(int id) {
        Nodo* temp = cabeza;
        Nodo* anterior = NULL;
        while (temp) {
            if (temp->proceso.id == id) {
                if (anterior)
                    anterior->siguiente = temp->siguiente;
                else
                    cabeza = temp->siguiente;
                delete temp;
                return true;
            }
            anterior = temp;
            temp = temp->siguiente;
        }
        return false;
    }

    ~ListaEnlazada() {
        while (cabeza) {
            Nodo* temp = cabeza;
            cabeza = cabeza->siguiente;
            delete temp;
        }
    }
};

#endif
```

pila_memoria:

```cpp
#ifndef PILA_MEMORIA_H
#define PILA_MEMORIA_H

#include <stack>
#include <iostream>
using namespace std;
class PilaMemoria {
private:
    stack<int> memoria;
public:
    void asignarMemoria(int procesoID) {
        memoria.push(procesoID);
        cout << "Memoria asignada al proceso ID " << procesoID << "\n";
    }
    void liberarMemoria() {
        if (!memoria.empty()) {
            int id = memoria.top();
            memoria.pop();
            cout << "Memoria liberada del proceso ID " << id << "\n";
        } else {
            cout << "No hay memoria para liberar\n";
        }
    }
    void mostrar() {
        cout << "Procesos con memoria asignada: " << memoria.size() << "\n";
    }
};

#endif
```

proceso:

```cpp
#ifndef PROCESO_H
#define PROCESO_H

#include <string>

struct Proceso {
    int id;
    std::string nombre;
    int prioridad;
    Proceso(int i, std::string n, int p) : id(i), nombre(n), prioridad(p) {}
};

#endif
```

3. 2. Capturas:

```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 1
ID: 487
Nombre: panda
Prioridad (1=Alta, 2=Media, 3=Baja): 1
Memoria asignada al proceso ID 487
```

```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 1
ID: 744
Nombre: youtube
Prioridad (1=Alta, 2=Media, 3=Baja): 2
Memoria asignada al proceso ID 744
```

```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 1
ID: 784
Nombre: facebook
Prioridad (1=Alta, 2=Media, 3=Baja): 1
Memoria asignada al proceso ID 784
```

```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 2
Ejecutando proceso ID 487: panda

--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 4

--- Lista de procesos ---
ID: 487, Nombre: panda, Prioridad: 1
ID: 744, Nombre: youtube, Prioridad: 2
ID: 784, Nombre: facebook, Prioridad: 1
Procesos listos para ejecución: 2
Procesos con memoria asignada: 3

--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 2
Ejecutando proceso ID 784: facebook
```

```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 2
Ejecutando proceso ID 744: youtube

--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 4

--- Lista de procesos ---
ID: 487, Nombre: panda, Prioridad: 1
ID: 744, Nombre: youtube, Prioridad: 2
ID: 784, Nombre: facebook, Prioridad: 1
Procesos listos para ejecución: 0
Procesos con memoria asignada: 3
```

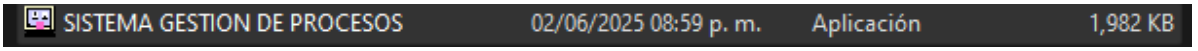
```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 3
Memoria liberada del proceso ID 784

--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 4

--- Lista de procesos ---
ID: 487, Nombre: panda, Prioridad: 1
ID: 744, Nombre: youtube, Prioridad: 2
ID: 784, Nombre: facebook, Prioridad: 1
Procesos listos para ejecución: 0
Procesos con memoria asignada: 2
```

```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 4

--- Lista de procesos ---
ID: 487, Nombre: panda, Prioridad: 1
ID: 744, Nombre: youtube, Prioridad: 2
ID: 784, Nombre: facebook, Prioridad: 1
Procesos listos para ejecución: 0
Procesos con memoria asignada: 2

--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 3
Memoria liberada del proceso ID 744
```

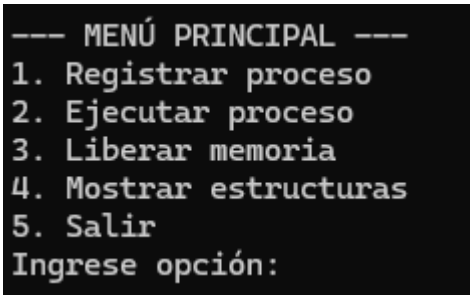
```
--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 3
Memoria liberada del proceso ID 487

--- MENÚ PRINCIPAL ---
1. Registrar proceso
2. Ejecutar proceso
3. Liberar memoria
4. Mostrar estructuras
5. Salir
Ingrese opción: 4

--- Lista de procesos ---
ID: 487, Nombre: panda, Prioridad: 1
ID: 744, Nombre: youtube, Prioridad: 2
ID: 784, Nombre: facebook, Prioridad: 1
Procesos listos para ejecución: 0
Procesos con memoria asignada: 0
```

3. 3. Manual de usuario:

1°. Iniciar el programa**.**



2°. Una vez iniciado el programa aparece el menú de opciones, donde tienes que seleccionar una opción (1, 2, 3, 4, 5)**.**



3°. La primera opción es “Registrar proceso” donde vas a agregar los datos del proceso que son: ID, Nombre y la prioridad del proceso (1=Alta, 2=Media, 3=Baja).
4°. La segunda opción es “Ejecutar proceso” donde los procesos se van a ejecutar según el nivel de prioridad.
5°. La tercera opción “Liberar memoria” donde se va a liberar según el orden que hayas almacenado los procesos.
6°. La cuarta opción “Mostrar estructuras”, esta opción es para que te muestre un informe.
7°. La quinta opción “Salir” es para finalizar la ejecución del programa.

4. EVIDENCIAS DEL TRABAJO EN EQUIPO:

4.1. Repositorio con control de versiones:

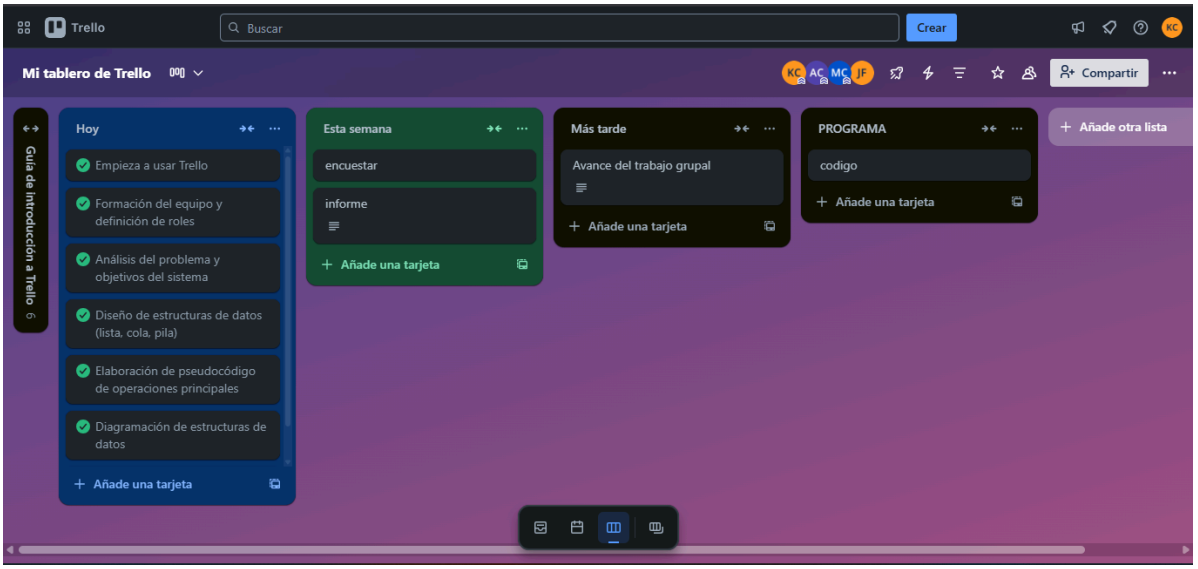
https://github.com/kennychahuayla/Sistema-de-gesti-n-de-proceso.git

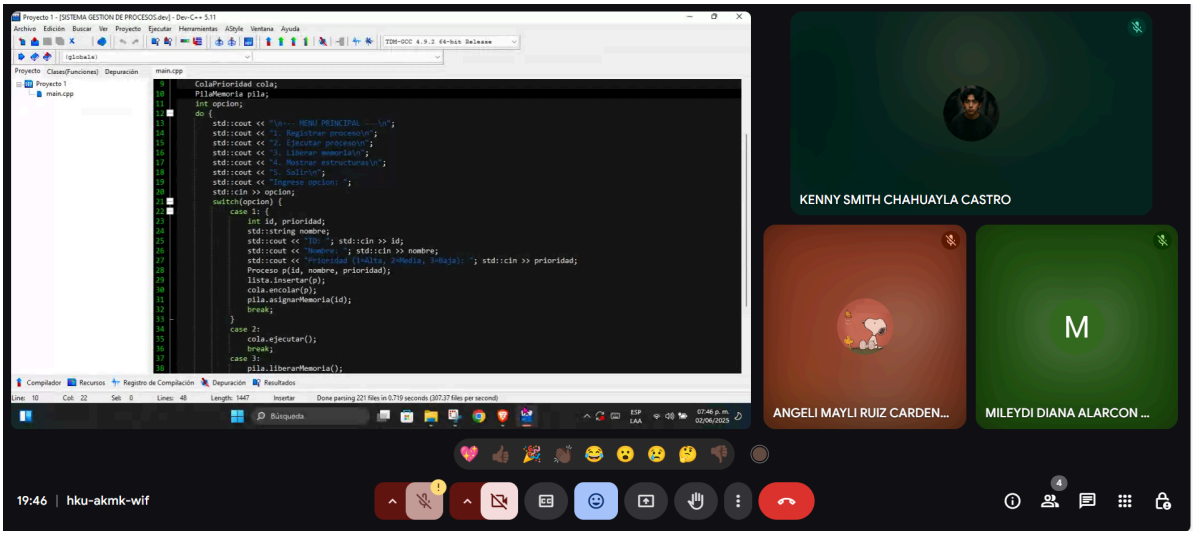
4.2. Plan de trabajo y roles asignados:

Roles:
Alarcon Chiquillan Mileydi Diana (Lider - programador)
Ruiz Cardenas Angeli Mayli (diseñadora)
Chahuayla Castro Kenny Smith (programador)
Herramientas usadas:

- Meet
- Trello
- Google docs
- Dev c++
- ppt: https://www.canva.com/design/DAGpQMGIbKI/S7BDClH7P8z_q8a0PL725g/edit?utm_content=DAGpQMGIbKI&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Actas de reunión:


Trello
Buscar
Crear
Mi tablero de Trello
Compartir
Guía de introducción a Trello
Hoy
Empieza a usar Trello
Formación del equipo y definición de roles
Análisis del problema y objetivos del sistema
Diseño de estructuras de datos (lista, cola, pila)
Elaboración de pseudocódigo de operaciones principales
Diagramación de estructuras de datos
Añade una tarjeta
Esta semana
encuestar
informe
Añade una tarjeta
Más tarde
Avance del trabajo grupal
Añade una tarjeta
PROGRAMA
codigo
Añade una tarjeta
Añade otra lista


KENNY SMITH CHAHUAYLA CASTRO
ANGELI MAYLI RUIZ CARDEN...
MILEYDI DIANA ALARCON ...
M
19:46 | hku-akmk-wif