Universidad
Continental

Presentación
DEL ABR

ESTRUCTURA DE DATOS

Universidad
Continental

Miembros:

1. Cutipa Jara Juan Alex
2. Espinoza Mora Johanna Alexandra
3. Guzmán Condori Flavio Cesar
4. Pérez Olivera Jaasiel Joana
5. Pontecil Alvarez Bruce Anderson

01

Descripción del Problema

Como parte del reto de desarrollar un gestor de procesos, aplicaremos los conocimientos adquiridos durante este periodo para construir un programa que permita administrar procesos de manera eficiente.

Para lograrlo, es fundamental elegir y utilizar correctamente las estructuras de datos adecuadas, como:

Listas enlazadas, para almacenar y recorrer todos los procesos registrados.
Colas, para organizar los procesos que serán ejecutados por la CPU según su prioridad.
Pilas, para simular la asignación y liberación de memoria, siguiendo un orden de último en entrar, primero en salir.

El objetivo es que el sistema funcione de forma ordenada, flexible y rápida, permitiendo al usuario agregar, modificar, buscar y eliminar procesos, además de simular cómo se comportan estos dentro de la CPU y la memoria.

02

Requerimientos del sistema

FUNCIONALES:

1. RF01: El sistema debe permitir crear nuevos procesos con atributos configurables.
2. RF02: El sistema debe asignar automáticamente un ID único a cada proceso.
3. RF03: El sistema debe permitir seleccionar el algoritmo de planificación de procesos (ej. FCFS, Round Robin, Prioridad).
4. RF04: El sistema debe simular el cambio de estados de los procesos (nuevo, listo, ejecutando, bloqueado, finalizado).
5. RF05: El sistema debe permitir pausar, reanudar y terminar procesos manualmente.
6. RF06: El sistema debe mostrar en tiempo real una tabla o lista con el estado actual de todos los procesos.
7. RF07: El sistema debe simular el paso del tiempo para representar la ejecución de procesos.

NO FUNCIONALES:

1. RNF01: El sistema debe estar desarrollado completamente en lenguaje C++.
2. RNF02: El sistema debe contar con una interfaz de usuario en consola clara e intuitiva.
3. RNF03: El sistema debe poder ejecutarse en sistemas operativos Windows, Linux o MacOS sin necesidad de librerías externas.
4. RNF04: El código debe estar modularizado y documentado adecuadamente para facilitar su comprensión y mantenimiento.
5. RNF05: El sistema debe ser capaz de gestionar al menos 20 procesos simultáneamente sin errores ni pérdidas de información.

03

Estructura de datos

Propuestos

Listas enlazadas, para almacenar y recorrer todos los procesos registrados.
Colas, para organizar los procesos que serán ejecutados por la CPU según su prioridad.
Pilas, para simular la asignación y liberación de memoria, siguiendo un orden de último en entrar, primero en salir.

Justificación de la elección

LISTAS ENLAZADAS

Las listas enlazadas se usan para los procesos principalmente porque permiten una gestión dinámica y flexible de la memoria, así como una gestión eficiente de la lista de procesos en ejecución.

Beneficios clave

GESTIÓN DINÁMICA DE LA MEMORIA:

Las listas enlazadas pueden crecer o reducir su tamaño durante la ejecución del programa, lo que es ideal cuando no se conoce de antemano el número de procesos que se ejecutarán.

INSERCIÓN Y ELIMINACIÓN EFICIENTES:

Las listas enlazadas permiten añadir o quitar procesos de la lista sin necesidad de mover grandes cantidades de datos en la memoria, lo que es muy útil en sistemas operativos donde los procesos cambian constantemente de estado.

LISTA DE PROCESOS EN EJECUCIÓN:

En sistemas operativos, los programadores de procesos utilizan listas enlazadas para mantener la lista de todos los procesos que están en ejecución, permitiendo una gestión eficiente de la planificación y la ejecución de los procesos.

GESTIÓN DE LA MEMORIA EN SISTEMAS OPERATIVOS:

Las listas enlazadas ayudan a gestionar la asignación de memoria a los procesos, lo que es crucial para la eficiencia del sistema operativo y la prevención de problemas como fragmentación de memoria.

COLAS

Las colas se usan para la gestión de procesos en la CPU en sistemas operativos y para la ejecución de tareas en orden en diversas aplicaciones. En sistemas operativos, las colas, como la cola de listos, organizan y priorizan los procesos que deben ejecutarse. En aplicaciones como chatbots o servidores de mensajería, las colas permiten manejar mensajes entrantes de manera ordenada.

GESTIÓN DE PROCESOS EN LA CPU:

En un sistema operativo, los procesos no se ejecutan de inmediato cuando están listos. Se colocan en una cola (cola de listos) para que la CPU pueda elegir qué proceso ejecutar a continuación.

ORDEN DE EJECUCIÓN:

Las colas siguen el principio FIFO (First In, First Out), es decir, el primer elemento en entrar es el primero en salir. Esto garantiza que los procesos se ejecuten en el orden en que fueron colocados en la cola.

PRIORIZACIÓN:

La cola de listas puede utilizar algoritmos de programación (como el programador de CPU) para priorizar ciertos procesos, dando más tiempo de CPU a aquellos que lo necesitan más o que están más cerca de terminar.

PILAS

Las pilas en la memoria, también conocidas como estructuras de datos de tipo "último en entrar, primero en salir" (LIFO), se utilizan principalmente para gestionar la memoria durante la ejecución de programas, especialmente al realizar llamadas a funciones y al manejar la recursividad. Permiten almacenar información temporalmente, como variables locales, parámetros de funciones y direcciones de retorno, para luego restaurar el estado del programa cuando la función termina.

04

Descripción de estructuras de datos y operaciones

Listas enlazadas:

- Estructura dinámica formada por nodos, donde cada nodo contiene un dato (proceso) y un puntero al siguiente nodo.
- Permite almacenar procesos en memoria no contigua manteniendo un orden.
- Operaciones básicas: creación, inserción (inicio, medio, final), búsqueda, eliminación y recorrido.
- Ventajas: gestión dinámica de memoria y procesos sin tamaño fijo, facilitando inserciones y eliminaciones eficientes según cambian los estados o prioridades de los procesos.

Colas:

- Estructura lineal que sigue el principio FIFO (primero en entrar, primero en salir).
- Operaciones básicas: encolar (agregar al final), desencolar (extraer del frente) y consulta del frente.
- Uso: organiza procesos para ejecución en orden de llegada o prioridad, fundamental en planificación de procesos como FCFS o Round Robin.

Pilas:

- Estructura que sigue el principio LIFO (último en entrar, primero en salir).
- Operaciones básicas: apilar (insertar en la cima), desapilar (remover de la cima) y consulta de cima.
- Uso: simula asignación y liberación de memoria en ejecución de procesos, especialmente para llamadas a funciones, variables locales y recursividad, gestionando memoria de forma ordenada y eficiente.

05

Algoritmos

principales

PSEUDOCÓDIGO PARA AGREGAR PROCESO:

INICIO
Crear función insertarProceso con parámetros:
int identificador
string nombre
int prioridad
Crear un nuevo objeto nuevoProceso de tipo Proceso (puntero).
Asignar atributos a nuevoProceso:
nuevoProceso->identificador = identificador
nuevoProceso->nombre = nombre
nuevoProceso->prioridad = prioridad
nuevoProceso->siguiente = NULL // indica que será el último nodo

SI cabeza es NULL (lista vacía) ENTONCES:
 cabeza = nuevoProceso // nuevoProceso es el primer nodo

SINO:
 Crear puntero temporal y asignarlo a cabeza
 MIENTRAS temporal->siguiente sea distinto de NULL HACER:
 temporal = temporal->siguiente // avanzar al siguiente nodo
 FIN MIENTRAS
 Asignar temporal->siguiente = nuevoProceso // agregar al final de la lista
Imprimir: "Su proceso fue añadido correctamente (^^)"
FIN

PSEUDOCÓDIGO PARA CAMBIAR EL ESTADO DEL PROCESO:

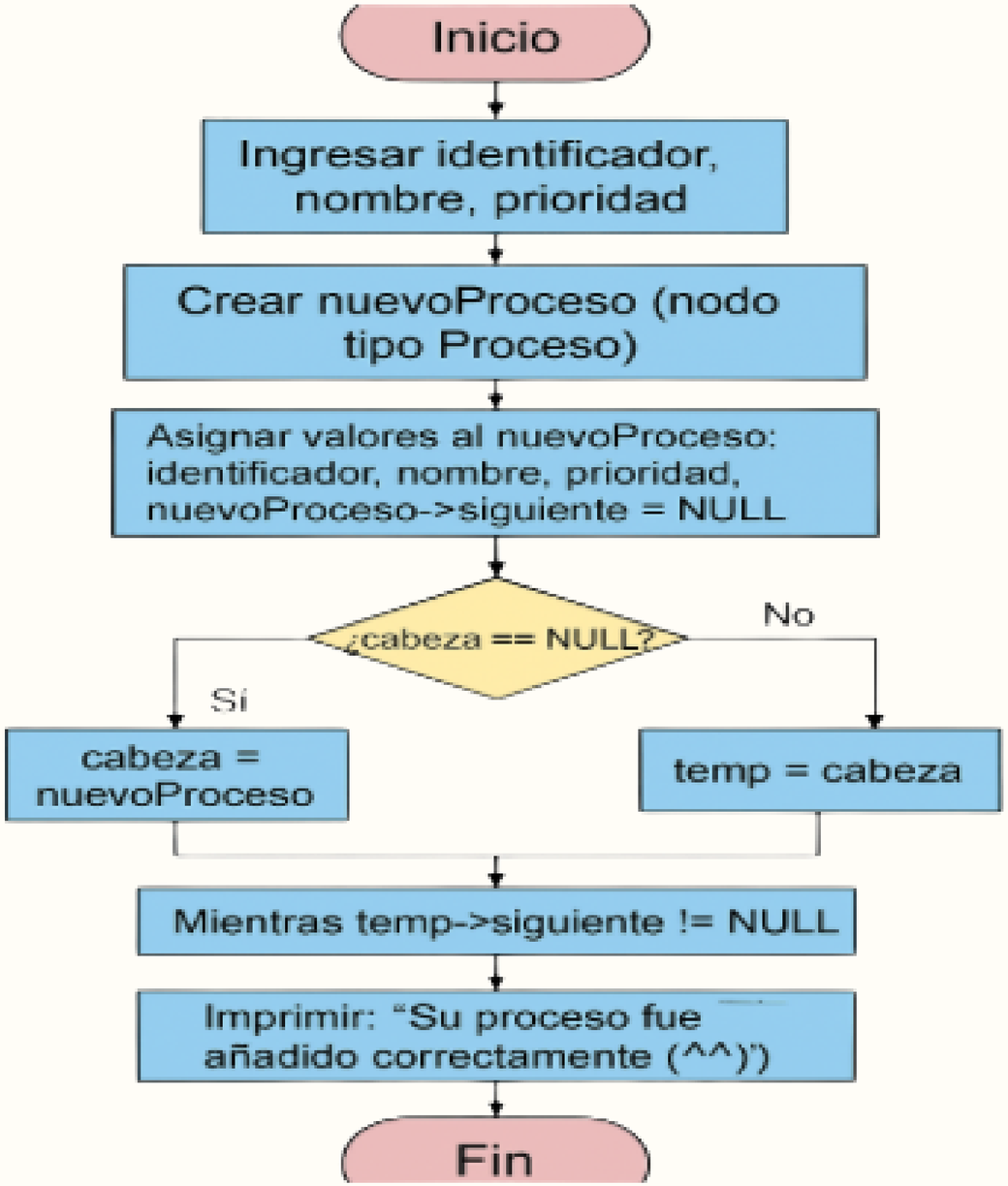
```
INICIO
Función modificarPrioridad(identificador, nuevaPrioridad)
    temporal ← cabeza //temporal apunta al nodo cabeza
    MIENTRAS temporal ≠ NULO

SI temporal.identificador = identificador ENTONCES
            temporal.prioridad ← nuevaPrioridad // nuevaPrioridad reemplaza al atributo prioridad
            IMPRIMIR "Su prioridad fue modificada correctamente (^^)"
  RETORNAR
        FIN SI
        temporal ← temporal.siguiente //temporal avanza al siguiente nodo de la lsita
    FIN MIENTRAS
    IMPRIMIR "Su proceso no fue encontrado (T-T)"
FIN
```

06

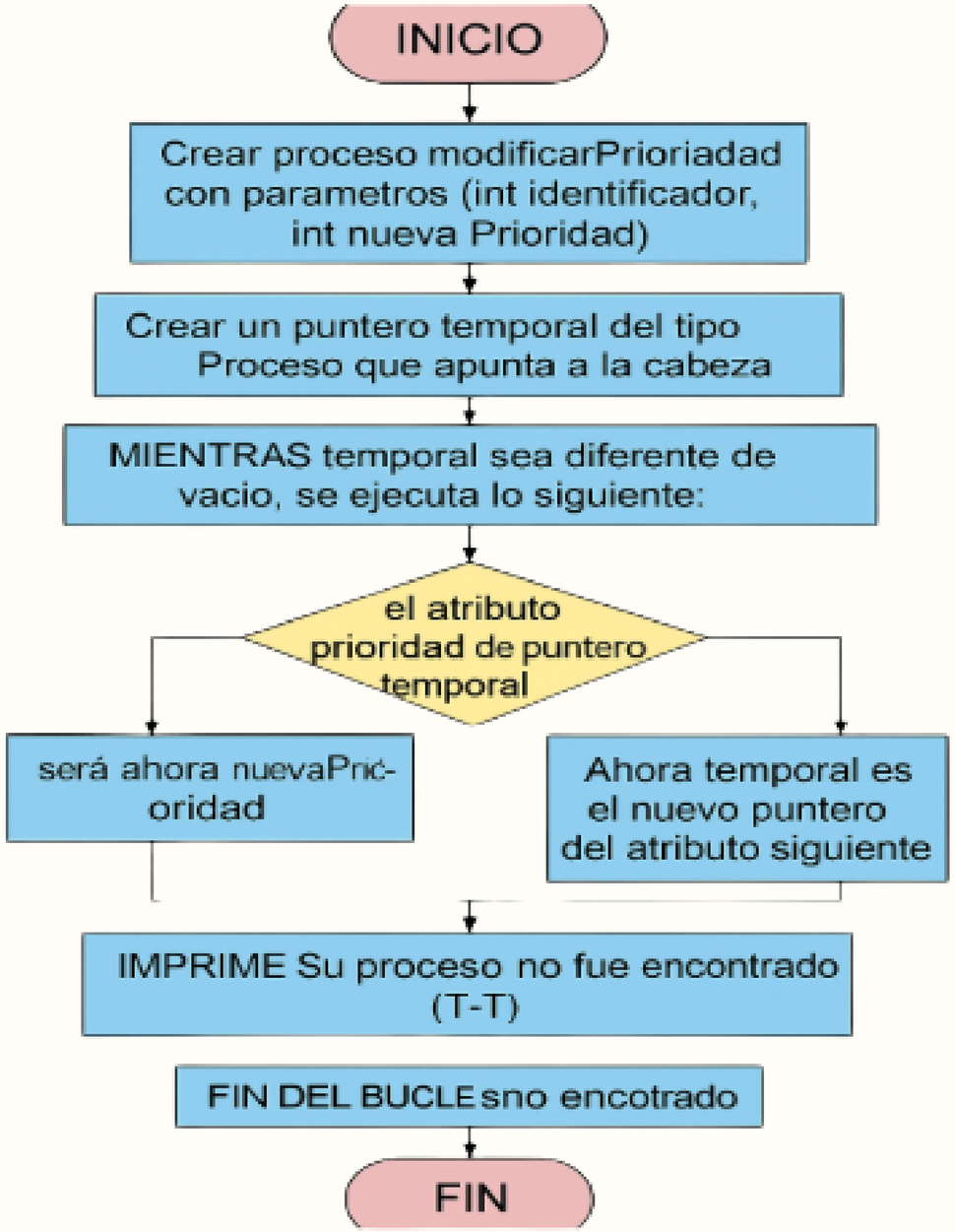
Diagramas

de flujo

- **PRIMER PSEUDOCÓDIGO**



• SEGUNDO PSEUDOCÓDIGO



07

Justificación del diseño

VENTAJAS DEL DISEÑO:

- Inserciones y eliminaciones eficientes:

Las listas enlazadas permiten insertar y eliminar procesos (nodos) de manera eficiente, especialmente cuando la posición es conocida. Modificar un puntero es mucho más rápido que reestructurar toda una estructura de datos como ocurre en los arrays. Esto es especialmente útil en sistemas donde los procesos pueden entrar y salir dinámicamente.

- Tamaño dinámico:

No es necesario definir el tamaño máximo de la lista de procesos al inicio. La lista puede crecer o decrecer según sea necesario, adaptándose al número real de procesos activos en cada momento. Esto optimiza el uso de memoria y evita la sobreasignación o la necesidad de redimensionar estructuras estáticas.

- Flexibilidad en la gestión:

Puedes añadir procesos en cualquier posición de la lista (por ejemplo, según prioridad), y modificar atributos como la prioridad de manera sencilla recorriendo la lista hasta encontrar el proceso deseado.

EFICIENCIA EN OPERACIONES ESPECÍFICAS:

- Inserción: Si el punto de inserción es conocido, se realiza en tiempo constante.
- Eliminación o modificación: Recorrer la lista para encontrar el nodo deseado, pero una vez localizado, la operación es inmediata.
- Bajo acoplamiento de memoria: Los nodos no necesitan estar almacenados de forma contigua, lo que reduce problemas de fragmentación de memoria y permite un uso más eficiente de la misma.

EFICIENCIA EN EL CONTEXTO DE PLANIFICACIÓN DE PROCESOS

- Gestión de prioridades: El uso de listas enlazadas permite implementar fácilmente algoritmos de planificación por prioridad, donde cada proceso tiene un atributo de prioridad y la lista puede recorrer para encontrar o modificar procesos según este criterio.

- Facilidad de actualización: El segundo diagrama muestra cómo modificar la prioridad de un proceso recorriendo la lista hasta encontrar el identificador correspondiente. Esta operación es directa y no requiere reestructuración masiva de la estructura de datos, solo cambiar el valor de un campo.

CONSIDERACIONES ADICIONALES

- Consumo de memoria: Cada nodo requiere espacio adicional para almacenar el puntero al siguiente nodo, lo que puede aumentar el uso de memoria en comparación con arrays, pero este costo es compensado por la flexibilidad y eficiencia en inserciones y eliminaciones.

- Acceso secuencial: El acceso a elementos específicos es secuencial, lo que puede ser menos eficiente que el acceso aleatorio de un array. Sin embargo, en sistemas donde las operaciones más frecuentes son inserciones, eliminaciones y modificaciones, esta desventaja es mínima frente a las ventajas obtenidas.

08

Captura de pantalla

de ejecución

VALIDACIÓN DE DATOS:

```
===== Sistema de gestion de procesos =====
1. Insertar proceso (^^)
2. Eliminar proceso (T-T)
3. Buscar proceso (O.O)
4. Modificar prioridad (^^)
5. Mostrar todos los procesos (O.O)
6. Encolar proceso en CPU (^^)
7. Desencolar y ejecutar proceso (O.O)
8. Mostrar cola de CPU (O.O)
9. Asignar memoria (^^)
10. Liberar memoria (T-T)
11. Verificar estado de memoria (O.O)
0. Salir (T-T)
=========================================
Ingrese una opción:
```

INSERTAR PROCESO:

```
Ingrese una opcion: 1
Ingrese ID del proceso: 123456
Ingrese nombre del proceso: Proceso 1
Ingrese prioridad del proceso: 1
Su proceso fue agregado correctamente (^^)
```

ELIMINAR PROCESO:

```
Ingrese una opcion: 2
Ingrese ID del proceso a eliminar: 123456
Su proceso fue eliminado correctamente (^^)
```

```
Ingrese una opci├│n: 2
Ingrese ID del proceso a eliminar: 123321
Su proceso no fue encontrado (T-T)
```

BUSCAR PROCESO:

```
Ingrese una opci├|n: 3
Ingrese ID del proceso a buscar: 12345
ID: 12345 | Nombre: Proceso 2 | Prioridad: 1 (0.0)
```

```
Ingrese una opcion: 3
Ingrese ID del proceso a buscar: 12345
Su proceso no fue encontrado (T-T)
```

MODIFICAR PRIORIDAD:

```
Ingrese una opcion: 4
Ingrese ID del proceso: 12345
Ingrese nueva prioridad: 2
Su prioridad fue modificada correctamente (^^)
```

MOSTRAR PROCESO:

```
Ingrese una opcion: 5
ID: 12345, Nombre: Proceso 2, Prioridad: 2 (0.0)
```

ENCOLAR PROCESO EN CPU:

```
Ingrese una opcion: 6
Ingrese ID del proceso a encolar: 12345
Su proceso fue encontrado correctamente (^^)
```

DESENCOLAR Y EJECUTAR PROCESO:

```
Ingrese una opcion: 7
Ejecutando proceso: ID 12345, Nombre Proceso 2, Prioridad 2 (0.0)
```

MOSTRAR COLA DE CPU:

```
Ingrese una opcion: 8
ID: 54321, Nombre: Proceso 3, Prioridad: 4 (O.O)
ID: 13425, Nombre: Proceso 5, Prioridad: 3 (O.O)
```

ASIGNAR MEMORIA:

```
Ingrese una opcion: 9
Ingrese ID del proceso a asignar memoria: 12345
La memoria fue asignada correctamente (^^)
```

LIBERAR MEMORIA:

```
Ingrese una opcion: 10
La memoria fue liberada correctamente (^^)
```

VERIFICAR ESTADO DE MEMORIA:

```
Ingrese una opcion: 11
La memoria esta vacia (T-T)
```

```
Ingrese una opcion: 11
El estado actual de la memoria es: (O.O)
ID: 12345, Nombre: Proceso 2, Prioridad: 1
```

SALIR DEL MENÚ:

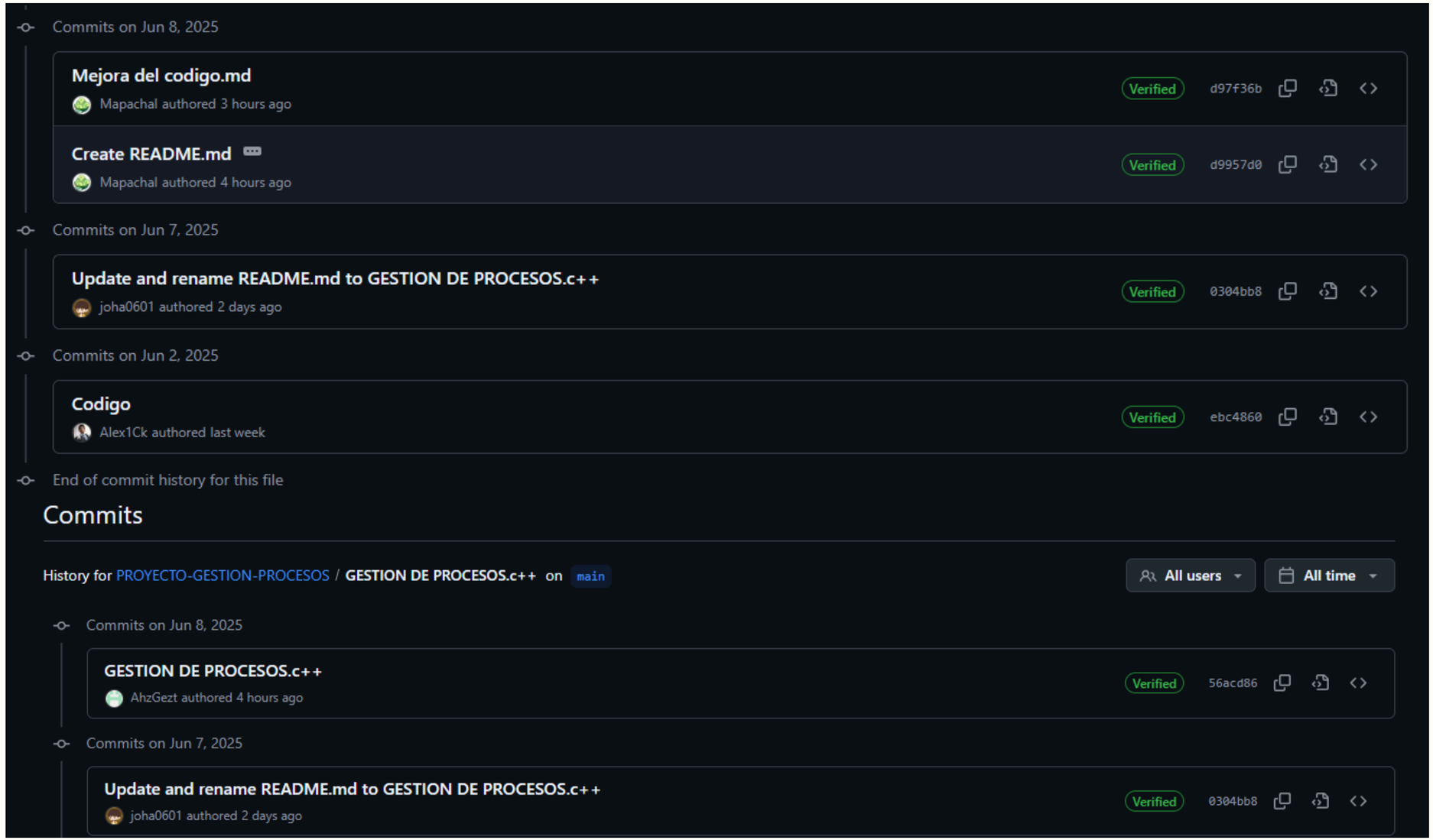
```
Ingrese una opcion: 0
Hasta luegoo... ^^
```

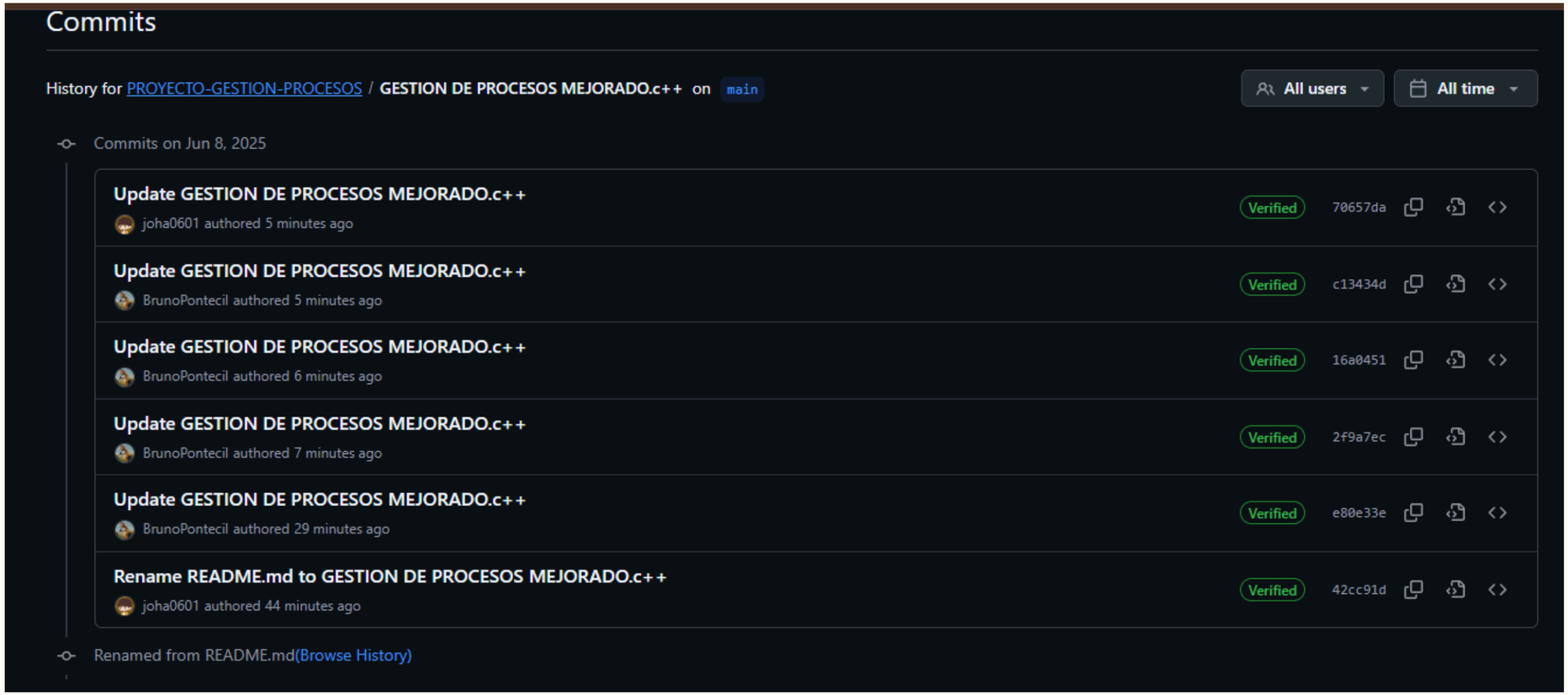
MANUAL DE USUARIOS:
HTTPS://DOCS.GOOGLE.COM/DOCUMENT/D/1SNTS-R6PWOOZPDFE-KA7QZJIDXFDKGKLWS7WNQLLWYE/EDIT?USP=SHARING

09

Repositorio con

Control de versiones


Commits on Jun 8, 2025
Mejora del codigo.md
Mapachal authored 3 hours ago
Verified
d97f36b
Create README.md
Mapachal authored 4 hours ago
Verified
d9957d0
Commits on Jun 7, 2025
Update and rename README.md to GESTION DE PROCESOS.c++
joha0601 authored 2 days ago
Verified
0304bb8
Commits on Jun 2, 2025
Codigo
Alex1Ck authored last week
Verified
ebc4860
End of commit history for this file
Commits
History for PROYECTO-GESTION-PROCESOS / GESTION DE PROCESOS.c++ on main
All users
All time
Commits on Jun 8, 2025
GESTION DE PROCESOS.c++
AhzGezt authored 4 hours ago
Verified
56acd86
Commits on Jun 7, 2025
Update and rename README.md to GESTION DE PROCESOS.c++
joha0601 authored 2 days ago
Verified
0304bb8


Commits
History for PROYECTO-GESTION-PROCESOS / GESTION DE PROCESOS MEJORADO.c++ on main
All users
All time
Commits on Jun 8, 2025
Update GESTION DE PROCESOS MEJORADO.c++
joha0601 authored 5 minutes ago
Verified
70657da
Update GESTION DE PROCESOS MEJORADO.c++
BrunoPontecil authored 5 minutes ago
Verified
c13434d
Update GESTION DE PROCESOS MEJORADO.c++
BrunoPontecil authored 6 minutes ago
Verified
16a0451
Update GESTION DE PROCESOS MEJORADO.c++
BrunoPontecil authored 7 minutes ago
Verified
2f9a7ec
Update GESTION DE PROCESOS MEJORADO.c++
BrunoPontecil authored 29 minutes ago
Verified
e80e33e
Rename README.md to GESTION DE PROCESOS MEJORADO.c++
joha0601 authored 44 minutes ago
Verified
42cc91d
Renamed from README.md(Browse History)

Plan de Trabajo y Roles Asignados

Todos: código

Alex y Flavio: Informe

Johanna y Bruce : Manual de usuario

Jaasiel: Presentación

Link del GitHub

Todos: código

Alex y Flavio: Informe

Johanna y Bruce : Manual de usuario

Jaasiel: Presentación

Muchas

GRACIAS

www.unsitiogenial.es