UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA DE SISTEMAS

INFORME DE PROYECTO
“PROYECTO DE UNIDAD I”

Que se presenta para el curso:
“SISTEMAS OPERATIVOS I”

Estudiante:
Denisthon Leonel Mamani Mamani

Docente:

Msc. Ing. Hugo Manuel Barraza Vizcarra

TACNA – PERÚ
2025

1. INTRODUCCION

La planificación del CPU y la gestión de memoria son componentes esenciales de cualquier sistema operativo, ya que determinan la eficiencia con la que los procesos utilizan los recursos de hardware.

El presente proyecto consiste en el desarrollo de un simulador en C++17 que permite comparar el comportamiento de tres algoritmos clásicos de planificación de CPU: First-Come, First-Served (FCFS), Shortest Process Next (SPN) y Round Robin (RR) con quantum configurable. El simulador lee un conjunto de procesos de un archivo de configuración, ejecuta el algoritmo seleccionado y genera métricas clave como tiempo de respuesta, tiempo de espera, tiempo de retorno y throughput.

Este trabajo busca no solo demostrar el funcionamiento práctico de cada estrategia, sino también ofrecer una herramienta flexible para observar sus diferencias de rendimiento en diversos escenarios de carga.

2. OBJETIVOS

- **Objetivo General**
  - Desarrollar un simulador de planificación de CPU y gestión de procesos en C++ que implemente los algoritmos FCFS, SPN y Round Robin con quantum configurable, permitiendo analizar y comparar su desempeño mediante métricas de tiempo y rendimiento.

- **Objetivos Específicos**
  - Implementar los algoritmos de planificación FCFS, SPN y Round Robin, asegurando su correcto comportamiento según la teoría.
  - Diseñar una estructura de datos eficiente (struct Proceso, colas, vectores) para almacenar y procesar la información de cada tarea.
  - Calcular y presentar métricas de evaluación: tiempo de respuesta, tiempo de espera, tiempo de retorno y throughput.
  - Permitir la selección del algoritmo y parámetros (como quantum) a través de un archivo de configuración o entrada del usuario.
  - Validar el funcionamiento del simulador mediante casos de prueba que evidencien las diferencias de rendimiento entre los algoritmos.
  - Documentar el proceso de desarrollo, justificación de decisiones y resultados obtenidos en un informe técnico.

3. MARCO TEÓRICO

3.1. Procesos en un Sistema Operativo

Un proceso es la unidad fundamental de trabajo en un sistema operativo: un programa en ejecución junto con su contexto de ejecución (registros, memoria, variables y recursos). Cada proceso es representado internamente mediante un Bloque de Control de Proceso (PCB), que contiene:

- PID (Process Identifier): número único para distinguirlo.
- Estado del proceso: nuevo, listo, en ejecución, en espera o terminado.
- Contador de programa y registros de CPU: indican la instrucción a ejecutar.
- Información de planificación: prioridad, tiempos de llegada y servicio.
- Información de memoria: direcciones de espacio asignado.

La correcta administración de procesos permite al sistema operativo multiprogramar y mantener el máximo rendimiento del procesador.

3.2. Planificación de la CPU

Cuando múltiples procesos están listos para ejecutarse, el planificador de CPU decide cuál obtiene el procesador.

Los objetivos principales son:

- Eficiencia: maximizar el uso de CPU y throughput (procesos completados/tiempo).
- Equidad: asegurar que todos los procesos obtengan tiempo de CPU.
- Tiempo de respuesta: especialmente importante en sistemas interactivos.
- Tiempo de espera y tiempo de retorno (turnaround): métricas clave para el rendimiento percibido.

Para evaluar un algoritmo se miden:

- Tiempo de respuesta: inicio − llegada.
- Tiempo de espera: fin − llegada − servicio.
- Tiempo de retorno: fin − llegada.
- Throughput: número de procesos completados / tiempo total del sistema.

3.3. Algoritmos de Planificación

3.3.1. First-Come, First-Served (FCFS)

- Tipo: no expropiativo (el proceso no se interrumpe hasta terminar).
- Funcionamiento: los procesos se atienden en orden de llegada a la cola de listos.

- Ventajas:
  - Implementación simple con una cola FIFO.
  - Adecuado cuando las cargas de trabajo son homogéneas.
- Desventajas:
  - Efecto convoy: un proceso largo puede retrasar a todos los demás.
  - No prioriza procesos cortos o interactivos.

3.3.2. Shortest Process Next (SPN) / Shortest Job First (SJF no expropiativo)

- Tipo: no expropiativo.
- Funcionamiento: se selecciona el proceso con menor tiempo de servicio estimado; en caso de empate, se elige el de menor tiempo de llegada.
- Ventajas:
  - Minimiza el tiempo de espera promedio (teóricamente óptimo).
  - Beneficia a procesos cortos.
- Desventajas:
  - Requiere conocer o estimar el tiempo de servicio, lo cual puede ser inexacto.
- Riesgo de *starvation* para procesos largos si siguen llegando trabajos cortos.

3.3.3. Round Robin (RR)

- Tipo: expropiativo.
- Funcionamiento: cada proceso recibe un *quantum* (porción de tiempo).
- Si no termina en su quantum, se interrumpe y vuelve al final de la cola.
- Ventajas:
  - Alta equidad; ideal para sistemas interactivos.
  - Buen tiempo de respuesta cuando el quantum es pequeño.
- Desventajas:
  - Quantum muy pequeño ⇒ excesivos cambios de contexto y sobrecarga.
  - Quantum muy grande ⇒ se comporta como FCFS.
- Parámetro crítico: elección adecuada del quantum (típicamente de 10 a 100 ms en sistemas reales).

3.4. Gestión de Memoria

La memoria principal es un recurso limitado que el sistema operativo administra para alojar procesos. Las estrategias de asignación contigua más comunes son:

- First-Fit: se asigna el primer bloque de memoria lo suficientemente grande.
  - Ventaja: rápido, recorrido lineal.
  - Desventaja: puede dejar fragmentación externa.
- Best-Fit: se busca el bloque que deje el menor espacio libre sobrante.
  - Ventaja: reduce el desperdicio de memoria.
  - Desventaja: búsqueda más lenta y puede fragmentar más rápidamente en bloques pequeños.

Estas políticas muestran el equilibrio entre velocidad de asignación y eficiencia en el uso de memoria, conceptos clave en sistemas operativos.

3.5. Simulación en C++

El lenguaje C++ proporciona:

- Estructuras de datos (struct, vector, queue) para manejar procesos y métricas.
- E/S de archivos (fstream) para leer configuraciones JSON.
- Bibliotecas externas como nlohmann/json que facilitan la lectura y escritura de datos estructurados.

La simulación permite emular el comportamiento real del planificador sin requerir hardware especializado, favoreciendo la experimentación con diferentes escenarios y parámetros (por ejemplo, distintos *quantum* en RR).

4. ANALISIS Y DISEÑO DEL SIMULADOR

4.1. Requerimientos del Sistema

Requerimientos funcionales

- Entrada: archivo JSON con:
  - Lista de procesos (PID, llegada, servicio).
  - Algoritmo de planificación (FCFS, SPN o RR) y quantum (cuando aplique).
  - Tamaño de memoria y solicitudes de asignación.
- Procesamiento: simulación del planificador de CPU y cálculo de métricas:
  - Respuesta, espera, retorno y throughput.
- Salida:
  - Tabla detallada de procesos con todas las métricas.

- Resumen de métricas globales.
- Información de asignación de memoria (bloque y tamaño o mensaje de error).

Requerimientos no funcionales

- Lenguaje: C++17.
- Portabilidad: ejecutable en Windows, Linux o macOS.
- Facilidad de extensión: permitir agregar nuevos algoritmos de planificación.

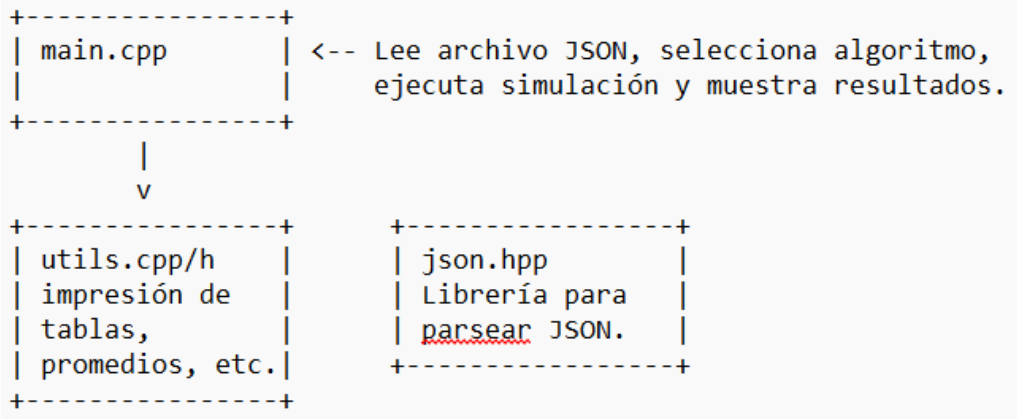
4.2. Analisis de Procesos

El sistema debe manejar una lista de procesos con sus atributos. Cada proceso se representa con un struct Proceso que contiene:

- pid: identificador único.
- llegada: tiempo de arribo.
- servicio: tiempo de CPU requerido.
- inicio, fin, respuesta, espera, retorno: métricas calculadas en tiempo de simulación.

4.3. Arquitectura General

El simulador sigue una arquitectura modular:



4.4. Diseño de Algoritmos de Planificación

Se implementaron tres políticas en funciones separadas:

- **FCFS**
  - Cola de procesos ordenada por llegada.
  - Cada proceso se ejecuta en su totalidad antes de pasar al siguiente.
- **SPN**

    - Usa una cola de listos que en cada paso selecciona el proceso con menor servicio.
    - En caso de empate, elige el de menor llegada.
- **Round Robin (RR)**
    - Cola circular.
    - Cada proceso ejecuta min(quantum, servicio_restante) y, si no finaliza, se reencola.
    - Control del tiempo global y cambios de contexto.

4.5. Gestión de Memoria

- Se modela una memoria lineal con tamaño configurable (por ejemplo, 1 MiB).
- Se implementan First-Fit y Best-Fit para asignar bloques solicitados:
    - Se mantiene una lista de bloques libres (posición y tamaño).
    - Al llegar una solicitud, se recorre la lista para encontrar el bloque adecuado.
    - Si no se encuentra espacio, se muestra mensaje de error.

4.6. Flujo de Datos

- Entrada: archivo config/entrada.json.
- Parsing: se carga en un objeto json de la librería nlohmann/json.
- Ejecución:
    - Selección del algoritmo.
    - Simulación paso a paso del planificador.
    - Cálculo de métricas.
- Salida: impresión en consola de tabla de resultados y métricas globales.

4.7. Justificacion del diseño

- C++17: brinda rendimiento y control de memoria, necesario para simular planificación y manejar estructuras complejas.
- nlohmann/json: permite separar la lógica de simulación de la definición de procesos, facilitando pruebas con distintos escenarios.
- Modularidad: utils.* concentra funciones de impresión y cálculo de promedios, permitiendo mantener limpio el main.cpp y agregar algoritmos nuevos sin alterar el resto del código.

5. IMPLEMENTACIÓN

La simulación se desarrolló en C++17 siguiendo una arquitectura modular para facilitar la lectura y el mantenimiento del código.

Se utilizaron tres archivos principales:

- main.cpp: punto de entrada; recibe los datos de procesos, selecciona el algoritmo de planificación y muestra los resultados en consola.
- utils.h / utils.cpp: definen la estructura Proceso, funciones de utilidad y las rutinas de los algoritmos de planificación.

5.1. Estructura de Datos

Cada proceso se representa con la estructura Proceso, que almacena: pid, llegada, servicio, inicio, fin, respuesta, espera, retorno y restante.

Esta información permite calcular métricas como tiempo de respuesta, espera, retorno y throughput.

5.2. Algoritmos Implementados

Se implementaron tres planificadores:

- First-Come, First-Served (FCFS, no expropiativo)
    - Ordena por tiempo de llegada y ejecuta cada proceso hasta terminar.
    - Complejidad: O(n log n) por la ordenación inicial.
- Shortest Process Next (SPN, no expropiativo)
    - Selecciona, entre los procesos llegados, el de menor tiempo de servicio.
    - Resuelve empates por menor llegada y luego por PID.
    - Ventaja: reduce el tiempo de retorno promedio frente a FCFS.
- Round Robin (RR, expropiativo)
    - Utiliza un quantum configurable (mínimo 2).
    - Mantiene una cola FIFO; los procesos que no terminan en su turno se re-encolan.
    - Propósito: garantizar equidad en entornos multiprogramados.

Cada algoritmo calcula al final los promedios de Respuesta, Espera, Retorno y el Throughput.

5.3. Flujo General

- Lectura de procesos (PID, llegada y servicio).
- Selección del algoritmo (FCFS, SPN o RR) y, si aplica, del quantum.
- Ejecución del planificador: actualización de tiempos y métricas.
- Presentación de resultados en tabla y promedios en consola.

5.4. Justificación

- C++17: ofrece portabilidad, uso eficiente de memoria y librerías estándar para manejo de colecciones y ordenamiento.
- Diseño modular: separa lógica de planificación y utilidades, facilitando pruebas y futuras mejoras.

6. RESULTADOS Y PRUEBAS

6.1. Datos de Entrada

Para todas las pruebas se usó el archivo config/entrada.json con la siguiente configuración de procesos:

```
{
  "cpu": { "algoritmo": "", "quantum": 4 },
  "procesos": [
    { "pid": 1, "llegada": 0, "servicio": 12 },
    { "pid": 2, "llegada": 1, "servicio": 5 },
    { "pid": 3, "llegada": 2, "servicio": 8 }
  ]
}
```

- Llegada: tiempo en el que cada proceso entra al sistema.
- Servicio: tiempo total de CPU requerido.
- Para Round Robin se utilizó un quantum de 4 unidades de tiempo.

6.2. Prueba 1: FCFS (First-Come, First-Served)

- Comando:

  g++ -std=c++17 src/main.cpp src/utils.cpp -Iinclude -o simulador

  ./simulador # cpu.algoritmo = "FCFS"

- Salida:

PID	Llegada	Servicio	Inicio	Fin	Respuesta	Espera	Retorno
1	0	12	0	12	0	0	12
2	1	5	12	17	11	11	16
3	2	8	17	25	15	15	23

- Promedio de Respuesta: 8.67
- Promedio de Espera: 8.67

- Promedio de Retorno: 17.0
- Throughput: 3/25 ≈ 0.12 procesos/unidad de tiempo.

Análisis: Se observa el *efecto convoy*; el primer proceso largo (P1) retrasa a los demás.

6.3. Prueba 2: SPN (Shortest Process Next)

- Comando:

  ./simulador # cpu.algoritmo = "SPN"

- Salida:

PID	Llegada	Servicio	Inicio	Fin	Respuesta	Espera	Retorno
1	0	12	13	25	13	13	25
2	1	5	1	6	0	0	5
3	2	8	6	14	4	4	12

- Promedio de Respuesta: 5.67
- Promedio de Espera: 5.67
- Promedio de Retorno: 14.0
- Throughput: 3/25 ≈ 0.12

Análisis: Al ejecutar primero los procesos de menor servicio, el tiempo de espera promedio mejora respecto a FCFS, aunque P1 termina más tarde.

6.4. Prueba 3: Round Robin (Quantum 4)

- Comando:

  ./simulador # cpu.algoritmo = "RR", quantum = 4

- Salida:

PID	Llegada	Servicio	Inicio	Fin	Respuesta	Espera	Retorno
1	0	12	0	25	0	13	25
2	1	5	4	17	3	11	16
3	2	8	8	21	6	11	19

- Promedio de Respuesta: 3.0
- Promedio de Espera: 11.67
- Promedio de Retorno: 20.0
- Throughput: 3/25 ≈ 0.12

Análisis: La respuesta promedio es la mejor, ya que todos los procesos comienzan pronto, pero la espera total es mayor debido a las múltiples rotaciones.

6.5. Comparación Global

Algoritmo	Resp. Prom.	Espera Prom.	Retorno Prom.	Observación principal
FCFS	8.67	8.67	17	Sencillo pero sensible a procesos largos.
SPN	5.67	5.67	14	Minimiza la espera, favorece trabajos cortos.
RR	3	11.67	20	Respuesta inicial rápida, mayor tiempo total.

6.6. Conclusiones de Pruebas

- FCFS es simple pero poco eficiente con procesos heterogéneos.
- SPN ofrece menor espera promedio, ideal cuando se conoce el tiempo de servicio.
- Round Robin reparte el CPU equitativamente y mejora la interactividad, a costa de un mayor tiempo de retorno y más cambios de contexto.

7. CONCLUSIONES

- **Comprensión de algoritmos de planificación**

  La implementación práctica de FCFS, SPN y Round Robin permitió verificar de forma experimental las características teóricas de cada política. Se evidenció cómo la elección del algoritmo impacta directamente en métricas clave como tiempo de espera, tiempo de respuesta y retorno.

- **FCFS (First-Come, First-Served)**
  - Ventaja: simplicidad de implementación y predictibilidad.
  - Desventaja: sufre del "efecto convoy", donde un proceso largo retrasa a todos los posteriores.
  - Adecuado para entornos por lotes con procesos de duración similar.

- **SPN (Shortest Process Next)**
  - Minimiza el tiempo de espera promedio al priorizar trabajos cortos.
  - Puede provocar inanición (starvation) de procesos largos si continuamente llegan procesos cortos.
  - Requiere conocer de antemano el tiempo de servicio, lo que limita su uso en sistemas reales.

- **Round Robin (RR)**
  - Ofrece la mejor respuesta inicial, favoreciendo la interactividad.
  - El quantum es crítico: valores bajos aumentan cambios de contexto y espera total; valores altos lo acercan a FCFS.
  - Es el algoritmo más equilibrado para sistemas multiprogramados y de tiempo compartido.

- **Aprendizaje obtenido**
  - Se fortaleció el dominio de estructuras de datos, manejo de colas, y control de tiempo en simulaciones.
  - La separación en módulos (main, utils) facilitó pruebas, reutilización y escalabilidad del proyecto.

- **Resultados globales**
  - En el escenario de prueba, SPN logró la menor espera y retorno promedio.
  - RR destacó en tiempo de respuesta, demostrando su adecuación para sistemas interactivos.
  - FCFS, aunque simple, mostró el peor rendimiento cuando los procesos tienen duraciones muy diferentes.

- **Relevancia para la práctica profesional**

  Comprender estas diferencias es fundamental para seleccionar la política de planificación que mejor equilibre eficiencia, justicia e interactividad en sistemas operativos reales.