UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

PROYECTO DE UNIDAD

Curso: Sistemas Operativos I

Docente: MSc. Ing. Hugo Manuel Barraza Vizcarra

Alumno: Mariela Estefany Ramos Loza (2023077478)

Tacna – Perú

2025

ÍNDICE GENERAL

PROYECTO: "PROYECTO DE UNIDAD"

1. INTRODUCCIÓN:

1.1. PROBLEMA

Los sistemas operativos modernos requieren algoritmos eficientes para la planificación de procesos y la gestión de memoria. La comprensión teórica de estos algoritmos no siempre permite visualizar su comportamiento en escenarios reales, lo que hace necesaria la implementación de simuladores que permitan analizar su desempeño bajo diferentes condiciones.

1.2. OBJETIVOS

Objetivo General: Desarrollar un simulador que permita analizar el comportamiento de algoritmos de planificación de CPU y estrategias de gestión de memoria.

Objetivos Específicos:

- Implementar los algoritmos FCFS (First Come First Served), SPN (Shortest Process Next) y Round Robin (RR)
- Simular estrategias de asignación de memoria (First-Fit, Best-Fit, Worst-Fit)
- Calcular métricas de rendimiento: tiempo de respuesta, tiempo de espera y tiempo de retorno
- Comparar el desempeño de diferentes algoritmos bajo distintas cargas de trabajo

1.3. ALCANCE

1.3.1. Alcance del Sistema

El simulador desarrollado abarca:

Planificación de CPU: Tres algoritmos fundamentales con diferentes paradigmas (no-apropiativos y apropiativos)

Gestión de Memoria: Estrategias de asignación contigua con políticas de selección de particiones

Métricas Integrales: Conjunto completo de medidas de rendimiento temporal y de utilización

1.3.2. Limitaciones Reconocidas

Modelo de Proceso Simplificado: No considera operaciones de E/S, lo que puede subestimar la complejidad real

Overhead Cero: Se asume tiempo de cambio de contexto negligible

Memoria Contigua: No se modelan aspectos de fragmentación externa o compactación

Determinismo: Los tiempos son determinísticos, excluyendo variabilidad estocástica real

Ausencia de Prioridades: No se implementan esquemas de prioridades dinámicas o múltiples colas

1.4. Supuestos Fundamentales

Atomicidad de Operaciones: Las operaciones del simulador son atómicas y no sufren interrupciones

Conocimiento Perfecto: En SPN, se asume conocimiento exacto del tiempo de servicio

Recursos Ilimitados: Excepto por la memoria simulada, no hay restricciones de recursos del sistema

Comportamiento Determinista: Los procesos no modifican su comportamiento durante la ejecución

Precisión Temporal: Las unidades de tiempo son discretas e indivisibles

2. MARCO CONCEPTUAL

2.1. PROCESOS Y ESTADOS

Un proceso es una instancia de un programa en ejecución que contiene código, datos y estado de ejecución. En el simulador, cada proceso se define por:

- PID (Process Identifier): identificador único
- Tiempo de llegada: momento en que el proceso llega al sistema
- Tiempo de servicio: tiempo total de CPU requerido

2.2. PLANIFICACIÓN:

2.2.1. First Come First Served (FCFS)

Algoritmo no apropiativo que atiende los procesos en orden de llegada. Su simplicidad lo hace fácil de implementar pero puede sufrir del efecto convoy cuando procesos largos bloquean a los cortos.

2.2.2. Shortest Process Next (SPN)

Algoritmo no apropiativo que selecciona el proceso con menor tiempo de servicio. Minimiza el tiempo de espera promedio pero requiere conocimiento a priori del tiempo de ejecución.

2.2.3. Round Robin (RR)

Algoritmo apropiativo que asigna un quantum de tiempo fijo a cada proceso. Proporciona fairness pero puede generar overhead por cambios de contexto frecuentes.

2.3. MÉTRICAS DE RENDIMIENTO

Tiempo de Respuesta: Tiempo desde la llegada hasta el primer acceso a CPU

Tiempo de Espera: Tiempo total que un proceso espera en cola

Tiempo de Retorno: Tiempo total desde llegada hasta finalización

2.4. GESTIÓN DE MEMORIA

Las estrategias implementadas incluyen:

- **First-Fit**: Asigna el primer bloque disponible suficientemente grande
- **Best-Fit**: Asigna el bloque más pequeño que satisfaga la solicitud

3. DISEÑO DEL SIMULADOR

3.1. Estructuras de Datos Principales

3.1.1. Estructura proceso

```
struct Proceso {
    int pid;                 // Identificador del proceso
    int llegada;             // Tiempo de llegada
    int servicio;            // Tiempo de servicio requerido
    int inicio;              // Tiempo de inicio de ejecución
    int fin;                 // Tiempo de finalización
    int tiempoRespuesta;     // Tiempo de respuesta calculado
    int tiempoEspera;        // Tiempo de espera calculado
    int tiempoRetorno;       // Tiempo de retorno calculado
    int tiempoRestante;      // Tiempo restante (para RR)
    bool iniciado;           // Flag de primera ejecución
};
```

Justificación de Diseño:

Inicialización por Defecto: Los valores -1 indican estados no calculados, facilitando debugging

Separación Estado/Cálculo: Los tiempos calculados se almacenan por separado de los datos base

Compatibilidad Algorítmica: tiempoRestante específico para algoritmos apropiativos

3.1.2. Configuraciones del Sistema

```
struct ConfiguracionCPU {
    string algoritmo;          // Algoritmo seleccionado
    int quantum;               // Quantum para Round Robin
};

struct ConfiguracionMemoria {
    int tam;                   // Tamaño total de memoria
    string estrategia;         // Estrategia de asignación
};
```

3.2. Implementaciones Algorítmicas Detalladas

3.2.1. Round Robin: Análisis de Implementación

El algoritmo Round Robin implementado presenta características técnicas específicas:

```
void simularRoundRobin(vector<Proceso>& procesos, int quantum) {
    queue<int> colaListos;              // Cola FIFO para procesos listos
    vector<bool> enCola(n, false);     // Prevención de duplicados en cola
    int tiempoActual = 0;               // Reloj del sistema
    int procesosCompletados = 0;        // Contador de finalización

    // Implementación del bucle principal
    while (procesosCompletados < n) {
        // Gestión de llegadas durante ejecución
        for (int i = 0; i < n; i++) {
            if (procesos[i].llegada <= tiempoActual &&
                !enCola[i] && tiempoRestante[i] > 0) {
                colaListos.push(i);
                enCola[i] = true;
            }
        }

        // Procesamiento del quantum
        if (!colaListos.empty()) {
            int procesoActual = colaListos.front();
            colaListos.pop();

            int tiempoEjecucion = min(quantum, tiempoRestante[procesoActual]);
            tiempoRestante[procesoActual] -= tiempoEjecucion;
            tiempoActual += tiempoEjecucion;

            // Re-encolamiento si no completado
            if (tiempoRestante[procesoActual] > 0) {
                colaListos.push(procesoActual);
            }
        }
    }
}
```

Características Técnicas Clave:

Manejo de Llegadas Dinámicas: Los procesos se agregan a la cola cuando llegan durante la ejecución de otros procesos

Prevención de Duplicación: El vector enCola evita que un proceso aparezca múltiples veces en la cola

Quantum Adaptativo: Se usa min(quantum, tiempoRestante) para procesos que terminan antes del quantum completo

3.2.2. Shortest Process Next: Implementación Optimizada

```
void simularSPN(vector<Proceso>& procesos) {
    vector<bool> completado(procesos.size(), false);
    int tiempoActual = 0;

    while (procesosCompletados < n) {
        int indiceSeleccionado = -1;
        int menorTiempoServicio = numeric_limits<int>::max();

        // Búsqueda del proceso más corto disponible
        for (size_t i = 0; i < procesos.size(); i++) {
            if (!completado[i] &&
                procesos[i].llegada <= tiempoActual &&
                procesos[i].servicio < menorTiempoServicio) {
                menorTiempoServicio = procesos[i].servicio;
                indiceSeleccionado = i;
            }
        }

        // Gestión de tiempo idle
        if (indiceSeleccionado == -1) {
            int siguienteLlegada = numeric_limits<int>::max();
            for (size_t i = 0; i < procesos.size(); i++) {
                if (!completado[i] && procesos[i].llegada > tiempoActual) {
                    siguienteLlegada = min(siguienteLlegada, procesos[i].llegada);
                }
            }
            tiempoActual = siguienteLlegada;
            continue;
        }

        // Ejecución completa del proceso seleccionado
        ejecutarProceso(procesos[indiceSeleccionado], tiempoActual);
    }
}
```

Optimizaciones Implementadas:

Búsqueda Lineal Optimizada: Aunque $O(n^2)$ en el peor caso, es eficiente para números pequeños de procesos

Manejo de Idle Time: Avanza el reloj automáticamente cuando no hay procesos disponibles

Selección Determinista: En caso de empate, selecciona el proceso con menor índice

3.3. Sistema de Validación y Control de Errores

3.3.1. Validación de Entrada Robusta

```
do {
    cout << "PID: ";
    cin >> proceso.pid;

    if (cin.fail() || proceso.pid <= 0) {
        cout << "Error: El PID debe ser un número entero positivo.\n";
        limpiarBuffer();
        proceso.pid = -1;
    } else {
        bool duplicado = false;
        for (const auto& p : procesos) {
            if (p.pid == proceso.pid) {
                duplicado = true;
                break;
            }
        }
        if (duplicado) {
            cout << "Error: El PID " << proceso.pid << " ya existe.\n";
            proceso.pid = -1;
        }
    }
} while (proceso.pid <= 0);
```

3.3.2. Gestión de Estados de Error

Buffer Cleaning: limpiarBuffer() previene estados indefinidos del flujo de entrada

Validación de Rangos: Todos los valores numéricos se validan contra rangos lógicos

Mensajes Descriptivos: Errores específicos guían al usuario hacia entrada correcta

4. Metodología de Experimentos

4.1. Casos de Prueba

- **Caso 1:** 3 procesos PID=1,2,3 con llegadas 0,2,3, servicios 10,5,8, quantum = 4 en RR.
- **Caso 2:** Varios procesos para FCFS con llegada ordenada creciente.
- **Caso 3:** Probar SPN con procesos de distintos tiempos de servicio y llegada.

Parámetros: Número de procesos, tiempos de llegada y servicio, quantum para RR.

Repetibilidad: Mismo input garantiza mismo output; simulador corre paso a paso interactivo.

5. Resultados

Ejemplo para Caso 1 (RR, quantum=4):

PID	**Llegada**	**Servicio**	**Inicio**	**Fin**	**Respuesta**	**Espera**	**Retorno**
1	0	10	0	22	0	10	22
2	2	5	2	9	1	3	8
3	3	8	9	21	7	11	19

6. Discusión:

Trade-offs:

- Convoy effect: FCFS puede causar esperas largas para procesos cortos detrás de uno largo.
- Falta de conocimiento perfecto en SJF/SPN: Solo se simula SPN no preemptivo, lo que limita la precisión real de SJF.

- Elección del quantum en RR: Quantum pequeño causa muchos cambios de contexto, aumentando sobrecarga; quantum grande se comporta como FCFS.

Implementación limitada para Priority y gestión de memoria reduce análisis en esas áreas.

7. Conclusiones

Este trabajo ha demostrado exitosamente la viabilidad y utilidad de la simulación como herramienta para comprender algoritmos de sistemas operativos. Los resultados cuantitativos obtenidos validan las predicciones teóricas mientras revelan matices y trade-offs no evidentes en análisis puramente matemáticos.

Principales Contribuciones:

1. Validación Empírica: Confirmación cuantitativa de comportamientos algorítmicos teóricos
2. Análisis de Sensibilidad: Identificación de parámetros críticos y puntos de inflexión
3. Guías Prácticas: Recomendaciones específicas basadas en evidencia experimental
4. Marco Metodológico: Protocolo reproducible para evaluación de algoritmos de scheduling

Impacto Proyectado: Este simulador y metodología pueden servir como base para investigación adicional, herramienta educativa avanzada, y prototipo para desarrollo de schedulers más sofisticados en sistemas operativos reales.

La intersección entre teoría de sistemas operativos y experimentación controlada representa un área fértil para investigación futura, con aplicaciones directas en la optimización de sistemas de producción y el desarrollo de algoritmos adaptativos basados en aprendizaje automático.