UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

INFORME TÉCNICO
"Planificación de CPU y gestión de memoria"

Curso: Base de datos II

Docente: Msc. Ing. Hugo Manuel Barraza Vizcarra

Podestá Condori, Richard Podestá (2022075743)

Tacna – Perú
2025

ÍNDICE

ÍNDICE DE FIGURAS

Informe Técnico

TEMA: Planificación de CPU y gestión de memoria

I. Introducción

Problema:

➢ En los sistemas operativos, la correcta gestión de procesos y memoria es esencial para garantizar un uso eficiente de los recursos de hardware. Sin embargo, comprender estos conceptos únicamente de forma teórica puede ser abstracto. Surge la necesidad de un simulador simple que permita observar de manera controlada el comportamiento de procesos, la planificación de CPU y la asignación de memoria.

Objetivos:

➢ Diseñar e implementar un simulador de sistema operativo en C++ que permita gestionar procesos y memoria.

➢ Implementar los algoritmos de planificación FCFS, SPN y Round Robin para comparar su desempeño.

➢ Calcular métricas de rendimiento como tiempo de respuesta, espera, retorno y throughput.

➢ Simular un esquema de gestión de memoria con particiones variables usando First-Fit y Best-Fit.

Alcance:

➢ El simulador procesa una lista de procesos definida por el usuario (PID, llegada, servicio), ejecuta los algoritmos de planificación seleccionados y calcula métricas de desempeño. Además, simula la asignación de bloques de memoria para cada proceso y muestra el estado de la memoria después de cada solicitud.

Supuestos:

➢ Todos los tiempos de llegada y servicio son enteros y conocidos.

➢ El quantum para Round Robin es fijo y ≥ 2.

➢ La memoria es lineal y de tamaño configurable; se asigna en el momento de llegada del proceso.

II. Marco conceptual

Procesos y Estados

➢ Un proceso es un programa en ejecución, compuesto por su código, datos y el contexto de CPU necesario para continuar su ejecución. Durante su ciclo de vida, un proceso puede encontrarse en distintos estados: nuevo, listo, ejecutando, bloqueado o terminado (Silberschatz, Galvin y Gagne, 2019).

Planificación de CPU

➢ La planificación es la decisión de cuál proceso se ejecutará cuando la CPU esté disponible.

- FCFS (First Come, First Served): Atiende procesos en orden de llegada y no expropia la CPU. Puede generar el efecto convoy, donde procesos cortos esperan demasiado si hay procesos largos delante.
- SPN (Shortest Process Next, no expropiativo): Atiende primero al proceso con menor tiempo de servicio conocido. Minimiza el tiempo promedio de espera, pero requiere conocer de antemano la duración del proceso, lo que no siempre es posible en sistemas reales.
- Round Robin: Asigna la CPU por intervalos de tiempo (quantum) iguales. Si el proceso no termina, se vuelve a encolar. Evita la inanición, pero si el quantum es demasiado pequeño puede generar sobrecarga de cambio de contexto.

Gestión de Memoria

➢ Se utiliza un esquema de particiones variables: la memoria comienza como un único bloque libre y se divide dinámicamente a medida que los procesos solicitan espacio.

- First-Fit: Asigna el primer bloque suficientemente grande que encuentre.
- Best-Fit: Asigna el bloque que deje la menor cantidad de memoria libre sin usar (menor fragmentación interna).

III. Diseño del simulador

Estructuras de datos:

- Almacena toda la información de un proceso (PID, llegada, servicio, inicio, fin, tiempo restante).
- Tiene métodos internos (respuesta(), espera(), retorno()) que calculan métricas de rendimiento de forma automática.

Figura 1

Estructura proceso con variables y métodos.

```
// ==== Estructura de proceso ====
// Representa cada proceso con sus tiempos y cálculos
struct Proceso {
    int pid;        // Identificador único del proceso
    int llegada;    // Momento en que llega al sistema
    int servicio;   // Tiempo total que requiere de CPU
    int inicio;     // Momento en que empieza su ejecución
    int fin;        // Momento en que termina su ejecución
    int restante;   // Tiempo de servicio restante (usado en Round Robin)

    // Métodos para calcular métricas
    int respuesta() const { return inicio - llegada; }          // Tiempo de respuesta
    int espera() const { return fin - llegada - servicio; }     // Tiempo de espera
    int retorno() const { return fin - llegada; }               // Tiempo de retorno
};
```

Nota. Elaboración Propia.

- struct BloqueMemoria: representa un bloque de memoria (inicio, tamaño, estado).

Figura 2

Estructura BloqueMemoria con variables.

```
// ==== Estructura de bloque de memoria ====
// Representa un segmento de memoria (libre u ocupado)
struct BloqueMemoria {
    int id;             // Número identificador del bloque
    int inicio;         // Dirección inicial del bloque en memoria
    int tamano;         // Tamaño del bloque
    bool libre;         // true = está disponible, false = está ocupado
    int pid_asignado;   // PID del proceso que ocupa el bloque (-1 si está libre)
};
```

Nota. Elaboración Propia.

Colas y vectores:

- vector<Proceso> : almacena procesos pendientes y permite ordenarlos (sort) según llegada o tiempo de servicio.

Figura 3

Vector Proceso.

```
// ==== Programa Principal ====
int main() {
    cout << "=== Simulador de Sistema Operativo ===\n";
    cout << "Seleccione el algoritmo de planificacion:\n1) FCFS\n2) SPN\n3) Round Robin\n";

    int opcion; cin >> opcion;
    vector<Proceso> procesos;
```

Nota. Elaboración Propia.

- queue<Proceso> : en Round Robin, se usa para rotar procesos según el quantum (es decir, simula la cola circular del CPU).

Figura 4

Queue Proceso.

```
// ==== Algoritmo Round Robin ====
void planificar_RR(vector<Proceso>& procesos, int quantum) {
    queue<Proceso> cola;        // Cola circular de procesos
    vector<Proceso> resultado;  // Procesos completados
    int tiempoActual = 0;
    // Inicializar tiempo restante
    for (auto &p : procesos) {
        p.restante = p.servicio;
    }
    while (!procesos.empty() || !cola.empty()) {
        // Mover procesos que ya llegaron a la cola
        for (auto it = procesos.begin(); it != procesos.end();) {
            if (it->llegada <= tiempoActual) {
                cola.push(*it);
                it = procesos.erase(it);
            } else {
                ++it;
            }
        }
        if (!cola.empty()) {
            Proceso p = cola.front(); cola.pop();
```

Nota. Elaboración Propia.

Manejo de llegada y servicio:

Los procesos se ordenan por llegada y se despachan según el algoritmo.

En RR, el campo restante indica cuánto tiempo de CPU le queda.

Figura 5

Algoritmo Round Robin

```
// ==== Algoritmo Round Robin ====
void planificar_RR(vector<Proceso>& procesos, int quantum) {
    queue<Proceso> cola;         // Cola circular de procesos
    vector<Proceso> resultado;  // Procesos completados
    int tiempoActual = 0;
    // Inicializar tiempo restante
    for (auto &p : procesos) {
        p.restante = p.servicio;
    }
    while (!procesos.empty() || !cola.empty()) {
        // Mover procesos que ya llegaron a la cola
        for (auto it = procesos.begin(); it != procesos.end();) {
            if (it->llegada <= tiempoActual) {
                cola.push(*it);
                it = procesos.erase(it);
            } else {
                ++it;
            }
        }
        if (!cola.empty()) {
            Proceso p = cola.front(); cola.pop();
            if (p.inicio == -1) p.inicio = tiempoActual;
            int ejecucion = min(quantum, p.restante);
            p.restante -= ejecucion;
            tiempoActual += ejecucion;
            if (p.restante > 0) {
                cola.push(p);  // Vuelve a la cola si no terminó
            } else {
                p.fin = tiempoActual;
                resultado.push_back(p);
            }
        } else {
            tiempoActual++;
        }
    }
    mostrarTabla(resultado);
}
```

Nota. Elaboración Propia.

Módulo de memoria:

Busca bloques libres, asigna el espacio y divide si sobra.

Muestra visualmente el estado de la memoria tras cada asignación.

Figura 6

Asignación de memoria (First fit-Best fit).

```
// Asignación de memoria con First-Fit o Best-Fit
bool asignarMemoria(vector<BloqueMemoria>& memoria, int pid, int tamano, int estrategia) {
    int idx = -1;                    // Índice del bloque elegido
    int mejorSobra = INT_MAX;        // Diferencia mínima para Best-Fit

    for (int i = 0; i < memoria.size(); i++) {
        if (memoria[i].libre && memoria[i].tamano >= tamano) {
            if (estrategia == 1) { idx = i; break; }  // First-Fit: toma el primero que encuentra
            else {
                int sobra = memoria[i].tamano - tamano;
                if (sobra < mejorSobra) { mejorSobra = sobra; idx = i; }
            }
        }
    }
    if (idx == -1) return false; // No hay bloque que quepa

    BloqueMemoria &b = memoria[idx];
    int inicioAsignado = b.inicio;
    b.libre = false;
    b.pid_asignado = pid;
    int espacioRestante = b.tamano - tamano;
    b.tamano = tamano;

    // Si sobró espacio, crear un nuevo bloque libre
    if (espacioRestante > 0) {
        BloqueMemoria nuevo = { (int)memoria.size()+1, inicioAsignado + tamano, espacioRestante, true, -1 };
        memoria.insert(memoria.begin() + idx + 1, nuevo);
    }
    return true;
}
```

Nota. Elaboración Propia.

IV. Metodología de experimentos

Casos de prueba:

Conjunto de 3 a 6 procesos con llegadas escalonadas y tiempos de servicio variados.

Parámetros:

Tamaño de memoria: 1 MiB.

Quantum: 4 unidades de tiempo.

Estrategias de asignación: First-Fit y Best-Fit.

Repetibilidad:

Se utilizan los mismos datos de entrada para comparar los tres algoritmos y ambas estrategias de memoria.

V. Resultados

Datos de prueba:

Procesos: 3

Servicio 1: 12

Servicio 2: 8

Servicio 3: 5

Quantum (RR): 4 unidades de tiempo.

Tamaño de memoria: 10 Mib

Prueba en planificación FCFS:

Figura 7

Planificación FCFS.

```
=== Simulador de Sistema Operativo ===
Seleccione el algoritmo de planificacion:
1) FCFS
2) SPN
3) Round Robin
1
Ingrese cantidad de procesos: 3

Proceso ID #1 | Llegada: 0
Ingrese Servicio: 12

Proceso ID #2 | Llegada: 1
Ingrese Servicio: 8

Proceso ID #3 | Llegada: 2
Ingrese Servicio: 5

PID | Llegada | Servicio | Inicio | Fin | Respuesta | Espera | Retorno
--- | ------- | -------- | ------ | --- | --------- | ------ | -------
1   | 0       | 12       | 0      | 12  | 0         | 0      | 12
2   | 1       | 8        | 12     | 20  | 11        | 11     | 19
3   | 2       | 5        | 20     | 25  | 18        | 18     | 23

Promedio Respuesta: 9.66667
Promedio Espera: 9.66667
Promedio Retorno: 18
Throughput: 0.12 procesos/unidad de tiempo
```

Nota. Elaboración Propia.

Prueba en planificación SPN:

Figura 8

Planificación SPN.

```
=== Simulador de Sistema Operativo ===
Seleccione el algoritmo de planificacion:
1) FCFS
2) SPN
3) Round Robin
2
Ingrese cantidad de procesos: 3

Proceso ID #1 | Llegada: 0
Ingrese Servicio: 12

Proceso ID #2 | Llegada: 1
Ingrese Servicio: 8

Proceso ID #3 | Llegada: 2
Ingrese Servicio: 5

PID | Llegada | Servicio | Inicio | Fin | Respuesta | Espera | Retorno
--- | ------- | -------- | ------ | --- | --------- | ------ | -------
1   | 0       | 12       | 0      | 12  | 0         | 0      | 12
3   | 2       | 5        | 12     | 17  | 10        | 10     | 15
2   | 1       | 8        | 17     | 25  | 16        | 16     | 24

Promedio Respuesta: 8.66667
Promedio Espera: 8.66667
Promedio Retorno: 17
Throughput: 0.12 procesos/unidad de tiempo
```

Nota. Elaboración Propia.

Prueba en planificación RR:

Figura 9

Planificación RR.

```
=== Simulador de Sistema Operativo ===
Seleccione el algoritmo de planificacion:
1) FCFS
2) SPN
3) Round Robin
3
Ingrese cantidad de procesos: 3

Proceso ID #1 | Llegada: 0
Ingrese Servicio: 12

Proceso ID #2 | Llegada: 1
Ingrese Servicio: 8

Proceso ID #3 | Llegada: 2
Ingrese Servicio: 5
Ingrese quantum: 4

PID | Llegada | Servicio | Inicio | Fin | Respuesta | Espera | Retorno
--- | ------- | -------- | ------ | --- | --------- | ------ | -------
1   | 0       | 12       | 0      | 20  | 0         | 8      | 20
2   | 1       | 8        | 8      | 24  | 7         | 15     | 23
3   | 2       | 5        | 12     | 25  | 10        | 18     | 23

Promedio Respuesta: 5.66667
Promedio Espera: 13.6667
Promedio Retorno: 22
Throughput: 0.12 procesos/unidad de tiempo
```

Nota. Elaboración Propia.

Prueba en gestión de memoria:

Figura 10

Gestión de memoria.

```
Ingrese tamano total de memoria: 10458
Estrategia de asignacion [(1) first-fit/(2) best-fit: ]1
Cantidad de solicitudes de memoria: 2
Solicitud #1 PID: 1
Tamano solicitado: 1200
Memoria asignada al PID 1 (1200 unidades)
Solicitud #2 PID: 2
Tamano solicitado: 3256
Memoria asignada al PID 2 (3256 unidades)

Estado de memoria:
ID | Inicio | Tamano | Libre | PID
1  | 0      | 1200   | NO    | 1
2  | 1200   | 3256   | NO    | 2
3  | 4456   | 6002   | SI    | 0

Simulacion finalizada.
```

Nota. Elaboración Propia.

VI. Discusión

Durante la simulación se pudieron observar las diferencias de desempeño entre los algoritmos de planificación.

El FCFS resultó ser el más simple de implementar, pero mostró el problema del convoy effect: si un proceso muy largo llega primero, los demás deben esperar a que termine, aumentando su tiempo de espera promedio.

El SPN (SJF no expropiativo) logró reducir significativamente el tiempo de espera promedio y el tiempo de retorno, ya que prioriza los procesos más cortos. Sin embargo, en un sistema real no siempre se conoce el tiempo de servicio exacto de cada proceso, lo que limita su uso práctico. Además, puede provocar inanición si llegan continuamente procesos cortos y un proceso largo nunca logra ejecutarse.

El Round Robin ofrecía mayor equidad entre procesos, ya que cada uno recibía tiempo de CPU de manera cíclica. Se observó que la elección del quantum es un parámetro crítico:

- *Con un quantum demasiado pequeño, el sistema incurre en exceso de cambios de contexto, desperdiciando tiempo en sobrecarga administrativa.*
- *Con un quantum demasiado grande, se comporta de forma similar a FCFS y pierde su ventaja en interactividad.*

En cuanto a la gestión de memoria, el uso de particiones variables con First-Fit y Best-Fit permitió ilustrar la asignación dinámica de bloques.

- *First-Fit es rápido ya que asigna el primer bloque disponible que encuentra, pero puede dejar fragmentación externa significativa.*
- *Best-Fit reduce la fragmentación externa inicial al elegir el bloque que más se ajusta al tamaño solicitado, pero requiere más tiempo de búsqueda y puede fragmentar la memoria en bloques demasiado pequeños para futuras solicitudes.*

Estos resultados evidencian que no existe un único algoritmo "óptimo"; la elección depende de los objetivos del sistema: equidad, tiempo de respuesta, o aprovechamiento de recursos.

VII. Conclusiones

- *El simulador permitió poner en práctica los conceptos de procesos, planificación de CPU y gestión de memoria de manera integrada y didáctica.*
- *Se comprobó que FCFS es sencillo pero puede perjudicar procesos cortos si uno muy largo ocupa primero el CPU.*
- *SPN optimiza el tiempo de espera, pero necesita conocimiento previo del tiempo de servicio y puede causar inanición.*
- *Round Robin es más justo y adecuado para sistemas interactivos, pero requiere seleccionar cuidadosamente el quantum para balancear sobrecarga y eficiencia.*

- *En memoria, First-Fit es más eficiente en tiempo de búsqueda, mientras que Best-Fit es más eficiente en espacio pero más costoso computacionalmente.*
- *La simulación es útil para experimentar con diferentes cargas de trabajo y entender las compensaciones (trade-offs) entre algoritmos.*