

Ejercicios Prácticos: Algoritmos de Planificación de Procesos

A continuación, se presentan 6 escenarios prácticos. Para cada uno de ellos, debes:

1. **Dibujar el diagrama de Gantt** correspondiente.
2. **Calcular el Tiempo Promedio de Espera (Tpe)** o el **Tiempo Promedio de Retorno (Trp)** según se solicite.

Nota: Para los ejercicios de prioridad, un número MENOR indica una MAYOR prioridad (ej. Prioridad 1 es más importante que Prioridad 3).

Ejercicio 1: FCFS (First Come First Served)

Instrucción: Dibuja el diagrama de Gantt y calcula el **Tiempo Promedio de Espera (Tpe)**.

Proceso	Tiempo de Llegada	Ráfaga de CPU (Duración)
P1	0 ms	5 ms
P2	1 ms	3 ms
P3	2 ms	4 ms

Ejercicio 2: SJF (Shortest Job First) - NO Expulsivo

Instrucción: Dibuja el diagrama de Gantt y calcula el **Tiempo Promedio de Espera (Tpe)**. Recuerda que, al no ser expulsivo, una vez un proceso entra a la CPU, no sale hasta terminar.

Proceso	Tiempo de Llegada	Ráfaga de CPU (Duración)
P1	0 ms	6 ms
P2	1 ms	2 ms
P3	2 ms	4 ms

Ejercicio 3: SJF - Expulsivo (Apropiativo)

Instrucción: Dibuja el diagrama de Gantt y calcula el **Tiempo Promedio de Espera (Tpe)**. Atención: si llega un proceso nuevo con una ráfaga *menor* al tiempo que le queda al proceso actual, ocurre un cambio de contexto.

Proceso	Tiempo de Llegada	Ráfaga de CPU (Duración)
P1	0 ms	6 ms
P2	2 ms	2 ms
P3	4 ms	1 ms

Ejercicio 4: PRIORIDAD - NO Expulsivo

Instrucción: Dibuja el diagrama de Gantt y calcula el **Tiempo Promedio de Retorno (Trp)**. (Recuerda: $Trp = \text{Tiempo de finalización} - \text{Tiempo de llegada}$).

Proceso	Tiempo de Llegada	Ráfaga de CPU	Prioridad
P1	0 ms	4 ms	2
P2	1 ms	3 ms	1
P3	2 ms	2 ms	3

Ejercicio 5: PRIORIDAD - Expulsivo

Instrucción: Dibuja el diagrama de Gantt y calcula el **Tiempo Promedio de Espera (Tpe)** usando los mismos datos del ejercicio anterior.

Proceso	Tiempo de Llegada	Ráfaga de CPU	Prioridad
---------	-------------------	---------------	-----------

P1	0 ms	4 ms	2
P2	1 ms	3 ms	1
P3	2 ms	2 ms	3

Ejercicio 6: ROUND ROBIN (Quantum = 2 ms)

Instrucción: Dibuja el diagrama de Gantt y calcula el **Tiempo Promedio de Retorno (Trp)**. Asume que todos llegan al mismo tiempo en el orden P1, P2, P3.

Proceso	Tiempo de Llegada	Ráfaga de CPU
P1	0 ms	5 ms
P2	0 ms	3 ms
P3	0 ms	4 ms

Ejercicio 7: Analizar y ejecutar

El algoritmo Round Robin es siempre expulsivo. Si el proceso agota su quantum de tiempo y no ha terminado, el sistema lo interrumpe y lo envía al final de la cola de listos.

A continuación, vamos a simular en gcc el mismo ejercicio visto en la teoría (Quantum = 4). Utilizaremos la estructura `std::queue` (Cola FIFO) de la librería estándar de gcc para representar la Cola de Listos.

round_robin.c

```
#include <stdio.h>

#include <unistd.h> // Para la función sleep()

// 1. Estructura que representa el Bloque de Control de Proceso (BCP) struct
Proceso { int id; int rafaga_cpu;    // Tiempo total que requiere el proceso int
tiempo_restante; // Tiempo que le falta para terminar };

// 2. Implementación básica de una Cola (FIFO) en C #define MAX_COLA
100 struct Proceso cola_listos[MAX_COLA]; int frente = 0; int final_cola = 0;

void encolar(struct Proceso p) { cola_listos[final_cola++] = p; }
```

```

struct Proceso desencolar() { return cola_listos[frente++]; }

int cola_vacia() { return frente == final_cola; }

int main() { // 3. Configuramos el Quantum int quantum = 4;

// 4. Cargamos los procesos basados en el ejercicio de clase
// {ID, Ráfaga de CPU, Tiempo Restante inicial}
struct Proceso p1 = {1, 10, 10};
struct Proceso p2 = {2, 5, 5};
struct Proceso p3 = {3, 8, 8};
struct Proceso p4 = {4, 3, 3};

// Ingresamos todos los procesos a la cola en orden de llegada
encolar(p1);
encolar(p2);
encolar(p3);
encolar(p4);

printf("=====\n");
printf(" Iniciando Planificador ROUND ROBIN (PCP) \n");
printf(" Quantum establecido: %d ms\n", quantum);
printf("=====\n\n");

int tiempo_actual = 0;

// 5. Bucle principal: El procesador trabaja mientras haya procesos en la cola
while (!cola_vacia()) {
    // El PCP saca al primer proceso de la fila
    struct Proceso p_actual = desencolar();

    printf("[Tiempo %d ms] -> Entra a CPU el Proceso P%d (Faltan: %d ms)\n",
        tiempo_actual, p_actual.id, p_actual.tiempo_restante);

    // El proceso se ejecutará por el Quantum, o por el tiempo que le quede si es
    menor
    int tiempo_ejecucion = (p_actual.tiempo_restante < quantum) ?
p_actual.tiempo_restante : quantum;

    // Simulamos el procesamiento en la CPU
    sleep(1);

    // Avanzamos el reloj del sistema y restamos el tiempo al proceso
    tiempo_actual += tiempo_ejecucion;
    p_actual.tiempo_restante -= tiempo_ejecucion;

// 6. Verificamos si el proceso terminó o si hay que expulsarlo

```

```

    if (p_actual.tiempo_restante > 0) {
        // El proceso no ha terminado, se le acabó el Quantum. ¡EXPULSIÓN!
        printf("  [!] P%d agoto su Quantum. Es EXPULSADO y vuelve al final de la
cola.\n\n", p_actual.id);
        encolar(p_actual); // Se inserta al final de la fila
    } else {
        // El proceso completó su ráfaga al 100%
        printf("  [V] P%d HA TERMINADO su ejecucion en el tiempo %d ms.\n\n",
p_actual.id, tiempo_actual);
    }
}

printf("=====\n");
printf("Todos los procesos han finalizado.\n");
printf("Tiempo de finalizacion global: %d ms.\n", tiempo_actual);
printf("=====\n");

return 0;
}

```