

Simulador de cambio de contexto

SISTEMAS OPERATIVOS

KADIHA NAHIR MUHAMAD ORTA, SAHIAN SALOME GUTIÉRREZ OSSA, MARIAMNY RAMIREZ

Estructura del programa:

El programa está dividido en las siguientes secciones:

proceso.h: tiene la estructura del proceso con sus variables: pid, pc, quantum, los registros ax, bx y cx, la lista de instrucciones, el estado y un array de repeticiones máximas para el JUMP que evitará que este quede en un bucle infinito. Asimismo, está el constructor donde todos los valores están en cero y el estado como “Listo”.

cargador.h: carga procesos. h; la estructura de datos map para almacenar pares clave-valor; la biblioteca regex para trabajar con las expresiones regulares de las instrucciones y los procesos; y declara la función cargarProcesos con la estructura de map para que con cada PID se asocie el proceso correspondiente.

proceso.h

```
C proceso.h
1  #ifndef PROCESO_H
2  #define PROCESO_H
3
4  #include <string>
5  #include <vector>
6
7  using namespace std;
8
9  struct Proceso {
10    int pid;          // ID del proceso
11    int pc;           // Contador de programa
12    int ax, bx, cx; // Registros
13    int quantum;     // Tiempo asignado
14    int memoria_utilizada; // Memoria utilizada (NUEVO CAMPO)
15    string estado;   // Estado actual
16    vector<string> instrucciones; // Lista de instrucciones
17    int repeticionesMaxJMP[50] = {0}; // Guarda cuántas veces se saltó a cada línea
18
19    // Constructor
20    Proceso() : pid(0), pc(0), ax(0), bx(0), cx(0), quantum(0), memoria_utilizada(0), estado("Listo") {}
21};
```

cargador.h

```
C cargador.h
1  #ifndef CARGADOR_H
2  #define CARGADOR_H
3
4  #include "proceso.h"
5  #include <map>
6  #include <string>
7
8  // Carga procesos desde un archivo en formato especificado
9  std::map<int, Proceso> cargarProcesos(const std::string &nombreArchivo);
10
11 #endif
```

cargador.cpp: implementación de cargarProcesos, iniciamos verificando que el nombre del archivo ingresado por el usuario es correcto. De lo contrario se despliega un mensaje de error. Después declaramos el string línea y los procesosRechazados para mediante la creación de un patrón verificar que la línea cumple con la estructura de un proceso siendo imprescindible que tenga el PID y el Quantum ya que de no estar los registros serán guardados con el valor de cero. Se borran los espacios extra y se van obteniendo las líneas verificando que coincidan con el patrón para capturar los valores de estos, si no se agregan al número de procesosRechazados y se indica cuál fue la línea descartada. Además, se verifica que no hayan PID duplicados y que el quantum sea mayor a cero. Finalmente se cargan las instrucciones con la función cargarInstrucciones y si el archivo que las contiene no está vacío se agregan.

instrucciones.h: carga vector y string y declara la función booleana esInstruccionValida y cargarInstrucciones.

cargador.cpp

```
cargador.cpp X
cargador.cpp
1 #include "cargador.h"
2 #include "instrucciones.h"
3 #include <fstream>
4 #include <iostream>
5 #include <map>
6 #include <regex>
7
8 using namespace std;
9
10 std::map<int, Proceso> cargarProcesos(const std::string &nombreArchivo) {
11     std::map<int, Proceso> procesos;
12     std::ifstream archivo(nombreArchivo);
13
14     if (!archivo.is_open()) {
15         std::cout << "Error: No se encontró el archivo '" << nombreArchivo << "'\n";
16         return procesos;
17     }
18
19     std::string linea;
20     int procesosRechazados = 0;
21
22     std::regex patron(
23         R"^(PID\s*:\s*(\d+)(?:\s*,\s*AX\s*=\s*(\d+))?(?:\s*,\s*BX\s*=\s*(\d+))?(?:\s*,\s*CX\s*=\s*(\d+))?\s*,\s*Quantum\s*=\s*"
24     );
25
26     while (getline(archivo, linea)) {
27         linea.erase(0, linea.find_first_not_of(" \t"));
28         linea.erase(linea.find_last_not_of(" \t") + 1);
29
30         if (linea.empty()) continue;
31
32         std::smatch match;
33         if (!std::regex_match(linea, match, patron)) {
34             std::cout << "Proceso rechazado: formato inválido -> " << linea << "\n";
35             procesosRechazados++;
36             continue;
37         }
38     }
39 }
```

instrucciones.h

```
instrucciones.h
1 #ifndef INSTRUCCIONES_H
2 #define INSTRUCCIONES_H
3
4 #include <vector>
5 #include <string>
6
7 using namespace std;
8
9 bool esInstruccionValida(const string& instruccion);
10 vector<string> cargarInstrucciones(int pid);
11
12#endif
```

instrucciones.cpp: incluimos instrucciones.h y creamos la función expresionRegularInst con la cual creamos el patrón a verificar para las instrucciones, siendo que con aquellas como ADD, SUB, MUL se revisa cual de los registros es el que sigue y después de este primer registro si sigue otro registro o un dígito y en caso de instrucciones especiales como INC, JMP o Nop se declara lo que sigue después de la instrucción si aplica. Lo siguiente es que con icase se permite que las instrucciones sean escritas tanto en mayúsculas o minúsculas.

Por otro lado, se implementa la función cargarInstrucciones donde se crea una lista de instruccionesVálidas y un string para el nombre del archivo que está definido como la conversión en string del PID más un .txt. Se comprueba si el archivo cumple el nombre establecido. Y para comprobar las líneas se obtienen y si no siguen el patrón definido no son agregados a instruccionesValidas.

roundRobin.h: carga proceso.h y declara las funciones de convertirMayusculas, mostrarTablaContexto, ejecutarRoundRobin y la función booleana procesarInstrucción.

roundRobin.cpp: Se importa cctype para clasificar y transformar caracteres, queue para el manejo de la Cola FIFO (First In, First Out), sstream para la creación de objetos istringstream, algorithm para utilizar remove, sort, find, e iomanip para los manipuladores de formato como setw y left/right

Función convertirMayúsculas: recibe un string como parámetro y mediante un ciclo for recorre cada carácter utilizando toupper() de cctype para convertirlo a mayúscula, devolviendo el string en mayúsculas para evitar diferencias.

Función procesarInstrucción: recibe el proceso por referencia, la línea de instrucción y un booleano avanzarPC por referencia. Crea un objeto istringstream del string de la instrucción para extraer la operación mediante el operador >>, la convierte a mayúsculas y obtiene el resto de la línea con getline. Utiliza algorithm para eliminar las comas con remove y crea otro istringstream para separar los parámetros.

```
string convertirMayusculas(const string& texto) {
    string resultado = texto;
    for (int i = 0; i < resultado.size(); i++) {
        resultado[i] = toupper(resultado[i]);
    }
    return resultado;
}

string procesarInstruccion(Proceso& proceso, const std::string& lineaInstruccion, bool& avanzarPC) {
    istringstream flujoTexto(lineaInstruccion);
    string operacion;
    flujoTexto >> operacion;
    operacion = convertirMayusculas(operacion);

    string primerParametro, segundoParametro;
    string restoLinea;
    getline(flujoTexto, restoLinea);

    restoLinea.erase(remove(restoLinea.begin(), restoLinea.end(), ','), restoLinea.end());
    istringstream flujoParametros(restoLinea);
    flujoParametros >> primerParametro >> segundoParametro;

    string resultadoOperacion = "";
    avanzarPC = true;

    if (operacion == "ADD" || operacion == "SUB" || operacion == "MUL") {
        int valor = 0;

        if (convertirMayusculas(segundoParametro) == "AX") valor = proceso.ax;
        else if (convertirMayusculas(segundoParametro) == "BX") valor = proceso.bx;
        else if (convertirMayusculas(segundoParametro) == "CX") valor = proceso.cx;
        else if (!segundoParametro.empty()) valor = stoi(segundoParametro);

        string reg = convertirMayusculas(primerParametro);
        int antes = 0;
        int despues = 0;

        if (reg == "AX") {
            antes = proceso.ax;
            if (operacion == "ADD") proceso.ax += valor;
            else if (operacion == "SUB") proceso.ax -= valor;
        }
    }
}
```

Para las operaciones aritméticas (ADD, SUB, MUL) determina si el segundo parámetro es un registro o un número, realiza la operación correspondiente en el registro destino y construye el string del resultado mostrando la operación realizada. Para INC incrementa el registro especificado. Para JMP verifica que la línea de destino sea válida, incrementa el contador de repeticiones en el array repeticionesMaxJMP y si no excede el límite de 2 saltos cambia el PC y establece avanzarPC como false, caso contrario bloquea el salto. Para NOP simplemente no hace nada.

Función mostrarTablaContexto: despliega el estado actual de todos los procesos en formato de tabla usando iomanip para el formato, mostrando PID, PC, registros, quantum y estado de manera organizada.

Función ejecutarRoundRobin: implementa el algoritmo Round Robin creando una cola de enteros para almacenar los PIDs. Llena la cola con todos los procesos disponibles y mientras la cola no esté vacía, extrae el primer proceso con front() y lo remueve con pop(). Cambia el estado del proceso a "Ejecutando" e inicia un ciclo que ejecuta instrucciones mientras no se agote el quantum y queden instrucciones por procesar. Para cada instrucción llama a procesarInstruccion, muestra el resultado y avanza el PC si no hubo salto. Al finalizar el quantum o las instrucciones, verifica si el proceso terminó completamente o debe regresar a la cola, actualiza su estado correspondiente y muestra el cambio de contexto con la tabla.

```

void mostrarTablaContexto(const map<int, Proceso>& procesos, std :: ofstream* flog) {
    cout << "\n Cambio de contexto...\n";
    if (flog) (*flog) << "\nCambio de Contexto...\n";

    cout << "\n Estado actual de los procesos:\n";
    if (flog) (*flog) << "\nEstado actual de los procesos:\n";
    cout << " PID || PC || AX || BX || CX || Quantum || Estado\n";
    cout << "-----\n";
    const int ANCHO_ESTADO = 12;

    for (const auto& par : procesos) {
        const Proceso& p = par.second;

        // Mostrar "Bloqueado(n)" cuando aplica
        string estadoMostrar = p.estado;
        if (p.estado == "Bloqueado") {
            estadoMostrar += "(" + to_string(p.tiempoBloqueo) + ")";
        }

        cout << "|| " << setw(4) << right << p.pid
            << " || " << setw(4) << right << p.pc
            << " || " << setw(4) << right << p.ax
            << " || " << setw(4) << right << p.bx
            << " || " << setw(4) << right << p.cx
            << " || " << setw(8) << right << p.quantum
            << " || " << setw(ANCHO_ESTADO) << left << corta(estadoMostrar, ANCHO_ESTADO)
            << " ||\n";
    }

    cout << "-----\n";
}

```

```

void ejecutarRoundRobin(map<int, Proceso>& listaProcesos) {
    std::ofstream log("Seguimiento.log"); //crea o sobreescribe el archivo
    queue<int> colaEjecucion;
    srand(time(0));

    for (auto& elemento : listaProcesos) {
        colaEjecucion.push(elemento.first);
    }

    cout << "\nIniciando ejecución con Round Robin...\n";
    bool limpiadoBuffer = false; // para limpiar el \n solo una vez

    while (!colaEjecucion.empty()) {
        int idProceso = colaEjecucion.front();
        colaEjecucion.pop();

        Proceso& procesoActual = listaProcesos[idProceso];

        if(procesoActual.estado == "Bloqueado"){
            procesoActual.tiempoBloqueo--;
            if(procesoActual.tiempoBloqueo > 0){
                colaEjecucion.push(idProceso);
                mostrarTablaContexto(listaProcesos, &log);
                continue;
            }
            //Se desbloquea
            procesoActual.estado = "Listo";
        }

        procesoActual.estado = "Ejecutando";

        log <<"\nProceso PID: " << procesoActual.pid
             << " (Quantum = " << procesoActual.quantum
             << ", Estado: " << procesoActual.estado << ")\n";
    }
}

```

main.cpp: Se incluye cargador.h y roundRobin.h para acceder a las funciones principales

Función mostrarProcesosEnTabla: recibe un map de procesos como parámetro constante y despliega una tabla. Utiliza un ciclo for con auto para iterar sobre cada par clave-valor, extrayendo el proceso con par.second. Usa setw() para establecer el ancho de cada columna, right para alinear números a la derecha y left para el estado y así muestra la tabla.

Función mostrarInstrucciones: itera sobre el map de procesos y para cada proceso muestra su PID y lista todas sus instrucciones numeradas con un índice entre paréntesis. Si el vector de instrucciones está vacío despliega un mensaje indicando que no hay instrucciones cargadas.

Función main: solicita al usuario el nombre del archivo de procesos mediante cin, llama a cargarProcesos para obtener los procesos cargados y muestra la cantidad total. Si se cargaron procesos válidos, despliega la tabla con mostrarProcesosEnTabla, muestra las instrucciones con mostrarInstrucciones, ejecuta el algoritmo Round Robin con ejecutarRoundRobin y finalmente muestra el estado final de todos los procesos. Si no se cargaron procesos válidos despliega un mensaje de error. La función retorna 0 indicando ejecución exitosa.

Resumen

El sistema con la estructura de proceso y de cada una de las clases a utilizar como cargador.h, instrucciones. h, roundRobin.h y sus respectivos cpp. Primero carga los procesos haciendo las verificaciones correspondiente para que no hayan datos erróneos que no sigan la estructura dictada; posteriormente hace lo mismo con las instrucciones poniendo especial énfasis en el nombre de los archivos y finalmente con el roundRobind maneja la ejecución de estas instrucciones y los cambios que experimentan los registros con las mismas avisando mediante una tabla la actualidad de cada uno de ellos y todos estas funciones y clases importantes son llamadas en el main.

Ejecución

```
Ingrese nombre del archivo de procesos: procesos.txt
⚠No se pudo abrir el archivo '5.txt' para el Proceso 5
Proceso 5 descartado: sin instrucciones.
Proceso rechazado: formato inválido -> jjjj
Total procesos rechazados: 2
```

Procesos Cargados (4)

| PID | PC | AX | BX | CX | Quantum | Estado |
|-----|----|----|----|----|---------|--------|
| 1 | 0 | 10 | 20 | 5 | 4 | Listo |
| 2 | 0 | 0 | 0 | 0 | 6 | Listo |
| 3 | 0 | 15 | 0 | 0 | 3 | Listo |
| 4 | 0 | 0 | 7 | 9 | 5 | Listo |

Instrucciones del PID 1:

- (0) ADD AX, BX
- (1) SUB BX, 5
- (2) MUL CX, 2
- (3) INC AX
- (4) NOP

Instrucciones del PID 2:

- (0) INC BX
- (1) ADD AX, 3
- (2) NOP
- (3) JMP 2

Instrucciones del PID 3:

- (0) ADD AX, 10
- (1) INC AX
- (2) MUL AX, 2
- (3) NOP

Instrucciones del PID 4:

- (0) SUB BX, CX
- (1) INC BX
- (2) ADD CX, 4
- (3) JMP 7
- (4) NOP

Iniciando ejecución con Round Robin...

[Proceso PID: 1 | Quantum: 4 | Memoria: 50 KB]

PC: 0 | AX: 10 | BX: 20 | CX: 5

Instrucción: ADD AX, BX

Resultado: AX = 10 ADD 20 = 30

PC: 1 | AX: 30 | BX: 20 | CX: 5

Instrucción: SUB BX, 5

Resultado: BX = 20 SUB 5 = 15

PC: 2 | AX: 30 | BX: 15 | CX: 5

Instrucción: MUL CX, 2

Resultado: CX = 5 MUL 2 = 10

PC: 3 | AX: 30 | BX: 15 | CX: 10

Instrucción: INC AX

Resultado: AX = 30 + 1 = 31

Estado actual de los procesos:

| PID | PC | AX | BX | CX | Quantum | Memoria | Estado |
|-----|----|----|----|----|---------|---------|--------|
| 1 | 4 | 31 | 15 | 10 | 4 | 62 | Listo |
| 2 | 0 | 0 | 0 | 0 | 6 | 40 | Listo |
| 3 | 0 | 15 | 0 | 0 | 3 | 40 | Listo |
| 4 | 0 | 0 | 7 | 9 | 5 | 50 | Listo |

[Proceso PID: 2 | Quantum: 6 | Memoria: 40 KB]

PC: 0 | AX: 0 | BX: 0 | CX: 0

Instrucción: INC BX

Resultado: BX = 0 + 1 = 1

PC: 1 | AX: 0 | BX: 1 | CX: 0

Instrucción: ADD AX, 3

Resultado: AX = 0 ADD 3 = 3

PC: 2 | AX: 3 | BX: 1 | CX: 0

Instrucción: NOP

No operation

PC: 3 | AX: 3 | BX: 1 | CX: 0

Instrucción: JMP 2

Saltando a la línea 2

PC: 2 | AX: 3 | BX: 1 | CX: 0

[Proceso PID: 3 | Quantum: 3 | Memoria: 52 KB]

PC: 3 | AX: 52 | BX: 0 | CX: 0

Instrucción: NOP

No operation

Proceso 3 finalizado.

Estado actual de los procesos:

| PID | PC | AX | BX | CX | Quantum | Memoria | Estado |
|-----|----|----|----|----|---------|---------|-----------|
| 1 | 5 | 31 | 15 | 10 | 4 | 62 | Terminado |
| 2 | 4 | 3 | 1 | 0 | 6 | 52 | Terminado |
| 3 | 4 | 52 | 0 | 0 | 3 | 52 | Terminado |
| 4 | 5 | 0 | -1 | 13 | 5 | 62 | Terminado |

Todos los procesos han finalizado.

Tipo de procesador y cantidad de memoria RAM utilizada

Procesador: AMD Ryzen 5 7520U with Radeon Graphics

Memoria RAM total: 3635428 kB

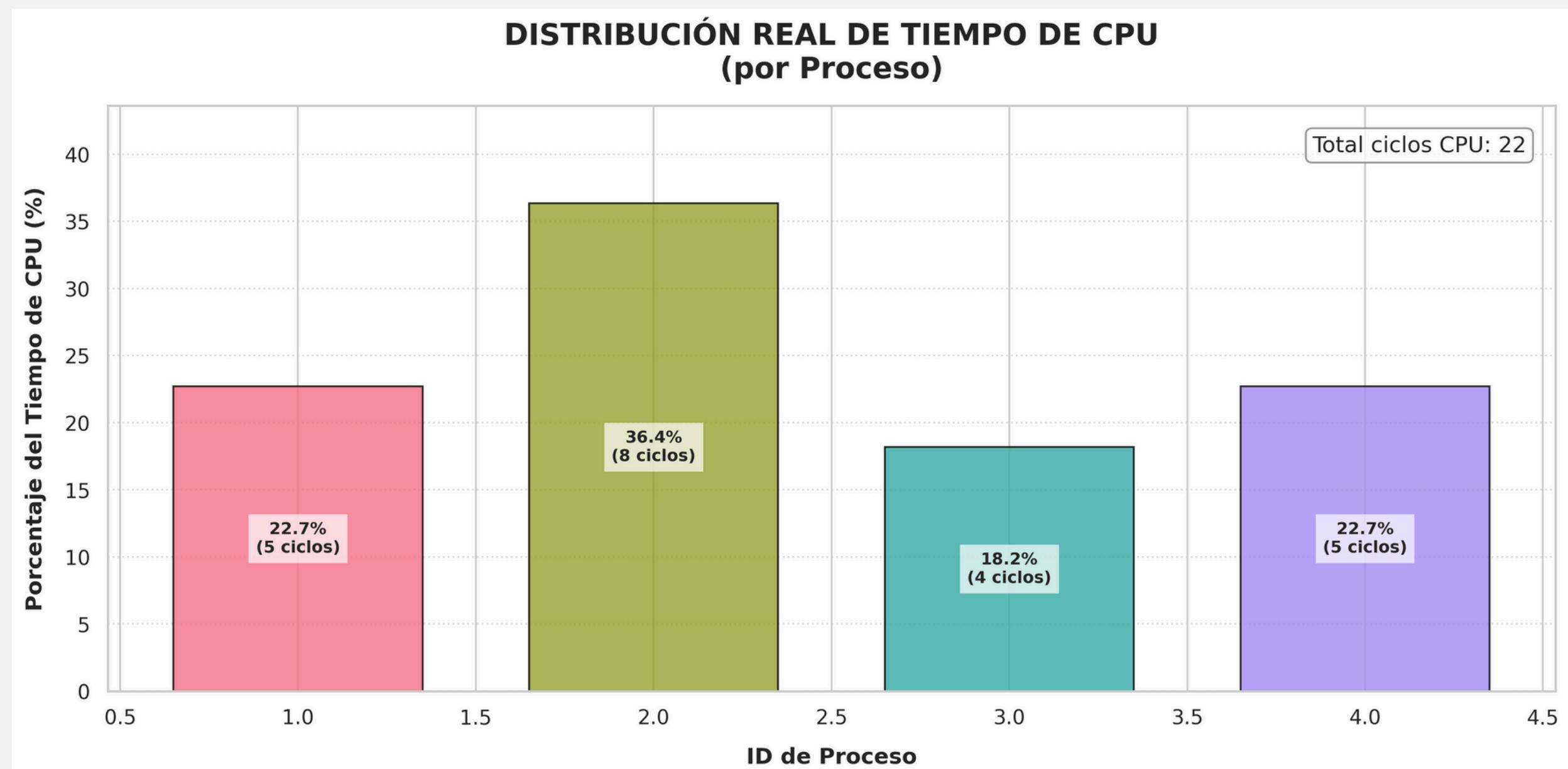
Memoria RAM usada por el programa: 3328 kB

Consumo real del simulador, round robin y carga de archivos

La gráfica representa la cantidad de ciclos de CPU en los que cada proceso estuvo en estado Ejecutando, es decir, cuando usó el procesador. Aunque cada proceso tiene un quantum distinto, el algoritmo Round Robin asigna el CPU de forma equitativa por turnos. Esto permite que todos los procesos avancen y puedan finalizar.

Se observa cómo la carga de CPU se distribuye en función de la duración y complejidad de cada proceso, más allá de su quantum.

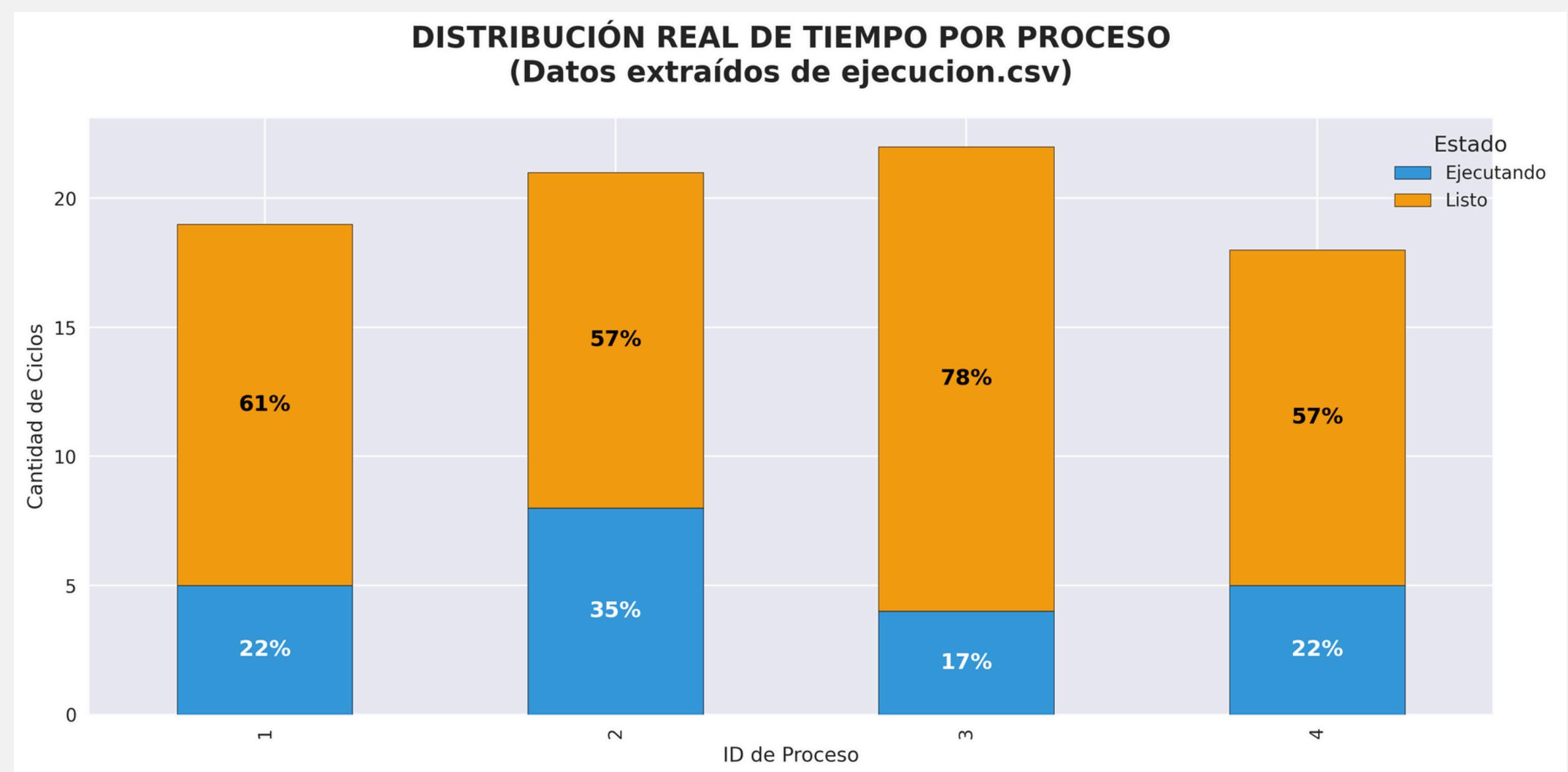
Distribución del tiempo de CPU por proceso



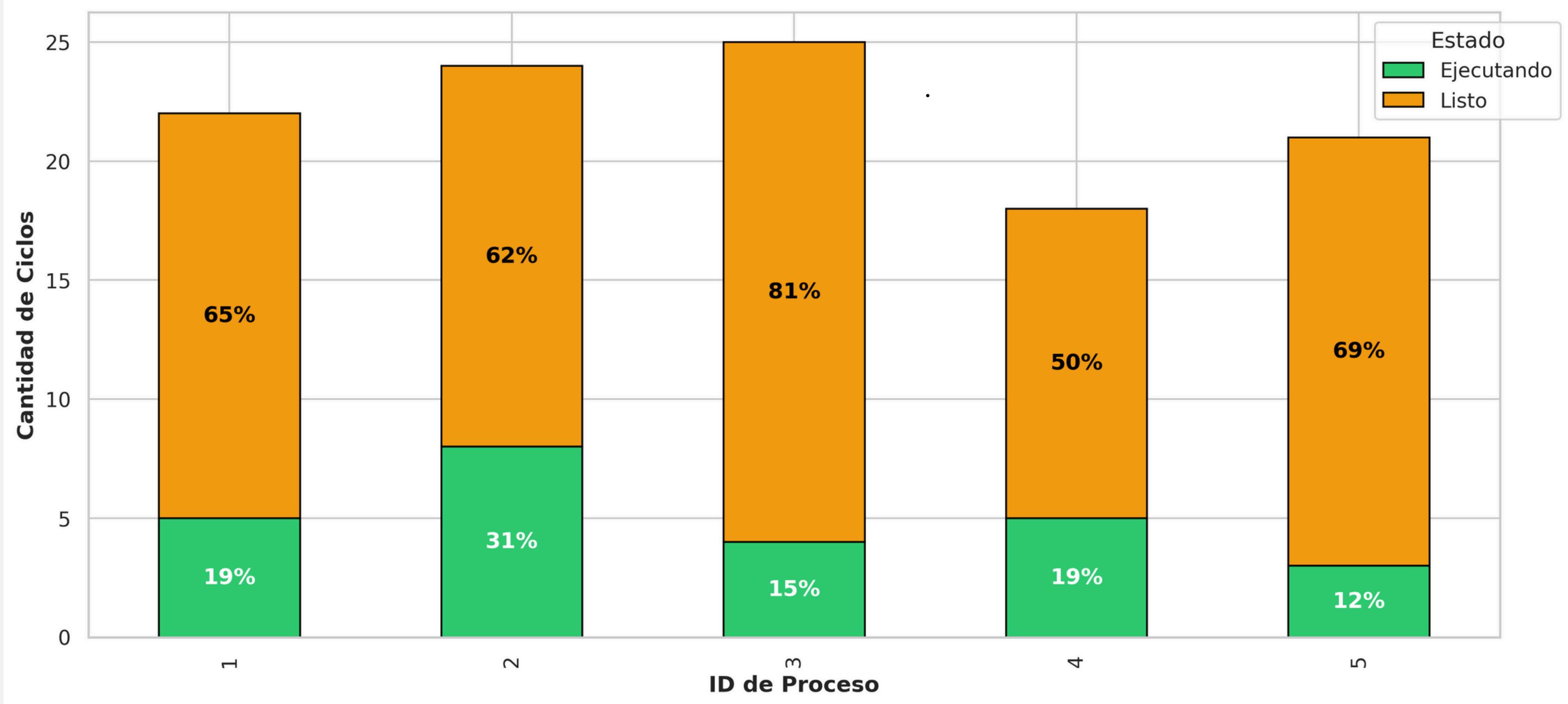
La gráfica muestra el porcentaje de tiempo que cada proceso pasó en los estados Ejecutando y Listo hasta su finalización. Aunque cada proceso tiene su propio quantum, la mayor parte del tiempo suele estar en estado Listo, esperando su turno para ejecutar. Esto se debe al cambio de contexto entre múltiples procesos.

Al aumentar la cantidad de procesos (4 o más), la diferencia de porcentajes entre Listo y Ejecutando se hace más evidente, mostrando cómo el CPU se reparte entre todos por turnos.

Distribución de estado por proceso



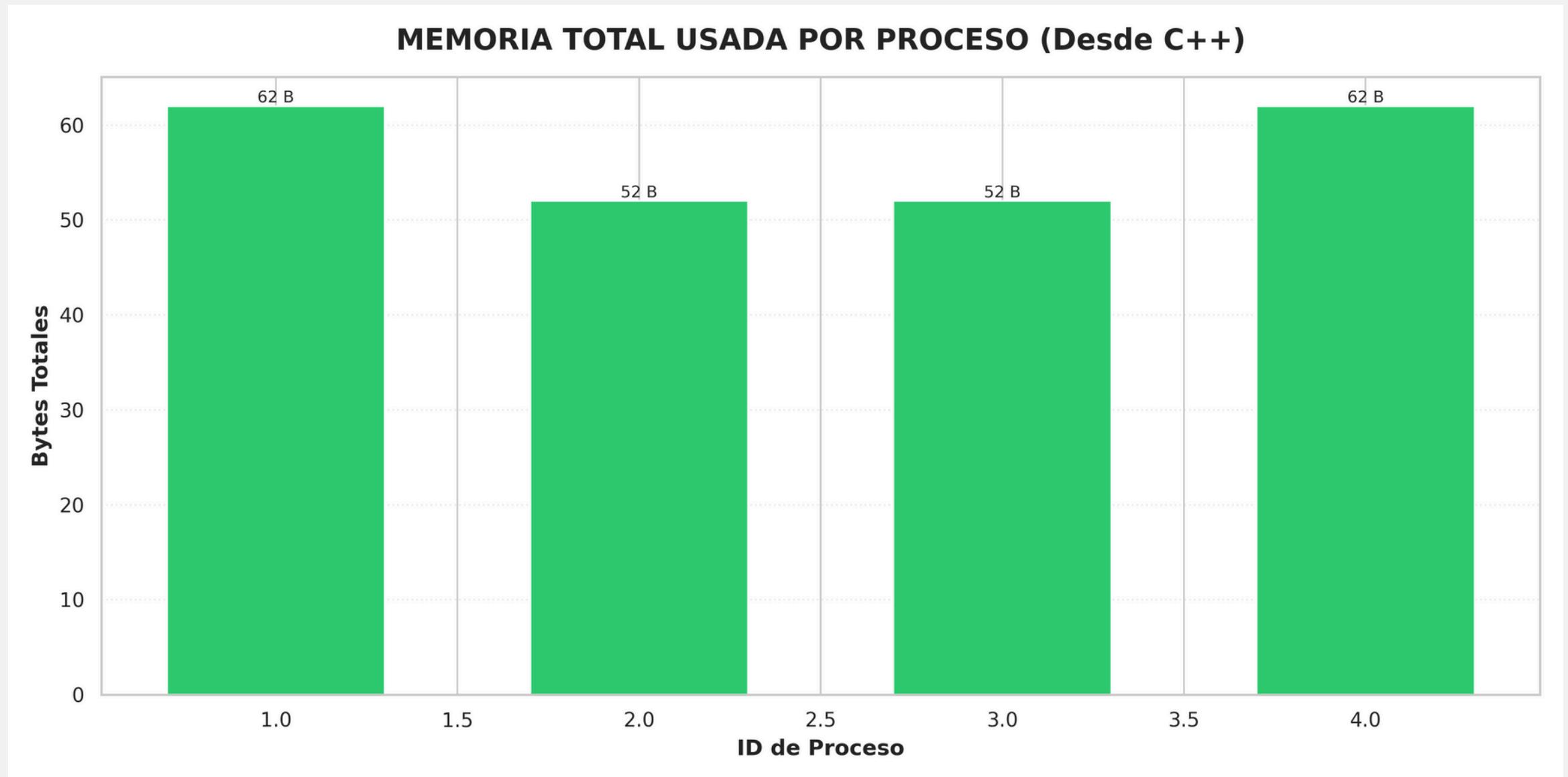
DISTRIBUCIÓN DE TIEMPO POR PROCESO



Memoria usada por proceso en Bytes

La gráfica muestra la memoria total usada por cada proceso.

Se asignan 12 bytes fijos al inicio por los registros AX, BX y CX (4 bytes cada uno), y se suman 10 bytes por cada instrucción ejecutada, simulando cómo un sistema utiliza la memoria a nivel de bytes.



Conclusiones

- El cambio de contexto es una base esencial en los sistemas operativos, ya que permite que múltiples procesos se ejecuten de forma ordenada. A través de este mecanismo, se garantiza que cada proceso tenga acceso al CPU de manera justa, aun cuando posea un quantum diferente.
- El algoritmo Round Robin demuestra su eficacia al distribuir el tiempo de CPU de forma equitativa, permitiendo que todos los procesos progresen gradualmente hasta su finalización. Su naturaleza cíclica asegura que ningún proceso sea favorecido o relegado indefinidamente.
- La memoria consumida por un proceso está directamente relacionada con su carga de trabajo, es decir, la cantidad de instrucciones ejecutadas. Esto resalta la importancia de la gestión de memoria en la planificación y ejecución eficiente de procesos dentro del sistema operativo.

Thank You