UNIVERSIDAD NACIONAL

SAN AGUSTIN DE AREQUIPA

Facultad de Ingeniería, Producción y Servicios

Escuela Profesional de Ciencia de la Computación

Laboratorio 7

Presentado por:

Parizaca Mozo, Paul Antony

CUI:

20210686

Curso:

Sistemas Operativos - Grupo A

Docente:

Yessenia Yari Ramos

Arequipa, Perú

2023

Github:
https://github.com/PaulParizacaMozo/SistemasOperativos/tree/main/Laboratorio_07

1.- Algoritmo Round Robin

round_robin.c

```C/C++
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Estructura para representar un proceso
typedef struct {
    char id[20];            // Identificador del proceso
    int tiempoServico;      // Tiempo total de ejecucion(CPU) necesario para el proceso
    int tiempoRestante;     // Tiempo de CPU restante para el proceso
} Proceso;

// Función del algoritmo Round Robin
void roundRobin(Proceso procesos[], int n_procesos, int quantum) {
    int tiempoTotal = 0;

    //  Bucle para hallar el tiempo de ejecucion de totos los procesos
    for (int i = 0; i < n_procesos; i++) {
        tiempoTotal += procesos[i].tiempoServico;
        procesos[i].tiempoRestante = procesos[i].tiempoServico;
    }

    printf("Tiempo\t\tProceso\n");
    // Inicio del algoritmo Round Robin
    int tiempoActual = 0;
    while (tiempoActual < tiempoTotal) { //Ejecucion hasta que cumpla todos los procesos
        for (int i = 0; i < n_procesos; i++) { // Recorrido de proceso en proceso
            if (procesos[i].tiempoRestante > 0) { // Comprueba si el procesos esta completado
                // Calcula el tiempo restante para el procesos actual
                int tiempoServico = (procesos[i].tiempoRestante < quantum) ? procesos[i].tiempoRestante : quantum; // Calculo del valor a restar al tiempo restante
                procesos[i].tiempoRestante -= tiempoServico; //Actualizacion de tiempo restante
                tiempoActual += tiempoServico; // Actualizacion del tiempo actual recorrido
```

```
                // Imprime el proceso actual y la informacion del tiempo
recorrido
                printf("%d -> %d\t\t%s\n", tiempoActual - tiempoServico,
tiempoActual, procesos[i].id);
                // Si un proceso termina mostrar el tiempo que demora en
completarse.
                if (procesos[i].tiempoRestante == 0) {
                    printf(">>> Proceso %s completado. Tiempo de total de
ejecucion(retorno): %d\n\n", procesos[i].id, tiempoActual);
                }
            }
        }
    }
}

int main() {

    //Ejemplo 1
    printf("**** EJEMPLO 1 ****\n");
    // Cantidad de procesos y valor del quantum
    int n_procesos = 4;
    int quantum = 3;

    // Arreglo con los datos del ejemplo 1
    Proceso procesos[] = {
        {"A", 8, 0},
        {"B", 4, 0},
        {"C", 9, 0},
        {"D", 5, 0}
    };

    // Función Round Robin
    roundRobin(procesos, n_procesos, quantum);

    printf("**** EJEMPLO 2 ****\n");
    // Cantidad de procesos y valor del quantum
    int n_procesos2 = 5;
    int quantum2 = 2;

    // Arreglo con los datos del ejemplo 2
    Proceso procesos2[] = {
        {"P1", 5, 0},
        {"P2", 3, 0},
        {"P3", 1, 0},
        {"P4", 2, 0},
        {"P5", 3, 0}
    };
```

```
    // Funcion Round Robin
    roundRobin(procesos2, n_procesos2, quantum2);

    return 0;
}
```

Función principal:

El algoritmo Round Robin funciona asignando un intervalo de tiempo fijo, conocido como "quantum".

```
int quantum = 3;
```

Inicialización:
Se crea una lista de procesos listos para ejecutarse.

Ejecución en turnos:
Se selecciona el primer proceso.
Se le asigna un tiempo de ejecución igual al quantum.
Si el proceso no ha terminado se pasa al siguiente proceso, y se queda en espera.
Si el proceso ha terminado, continúa hasta terminar con los demás procesos.

Repetición:
Este proceso se repite hasta que todos los procesos hayan terminado de ejecutarse.

Ejecución con ejemplos dados
-Ejemplo 1

Teste 1 :

Proceso	Tiempo de llegada	Tiempo de servicio
A	0	8
B	1	4
C	2	9
D	3	5

Quantum 3 unit.

```
//Ejemplo 1
printf("**** EJEMPLO 1 ****\n"); format:
// Cantidad de procesos y valor del quantum
int n_procesos = 4;
int quantum = 3;

// Arreglo con los datos del ejemplo 1
Proceso procesos[] = {
    {"A", 8, 0},
    {"B", 4, 0},
    {"C", 9, 0},
    {"D", 5, 0}
};

// Función Round Robin
roundRobin(procesos, n_procesos, quantum);
```

Ejecución

```
❯ gcc round_robin.c -o eject && ./eject
**** EJEMPLO 1 ****
Tiempo          Proceso
0 -> 3          A
3 -> 6          B
6 -> 9          C
9 -> 12         D
12 -> 15                A
15 -> 16                B
>>> Proceso B completado. Tiempo de total de ejecucion(retorno): 16

16 -> 19                C
19 -> 21                D
>>> Proceso D completado. Tiempo de total de ejecucion(retorno): 21

21 -> 23                A
>>> Proceso A completado. Tiempo de total de ejecucion(retorno): 23

23 -> 26                C
>>> Proceso C completado. Tiempo de total de ejecucion(retorno): 26
```

Ejemplo 2:

Teste 2:

Proceso	Tiempo de llegada	Tiempo de servicio
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Quantum 2 unit

```
printf("**** EJEMPLO 2 ****\n"); format:
// Cantidad de procesos y valor del quantum
int n_procesos2 = 5;
int quantum2 = 2;

// Arreglo con los datos del ejemplo 2
Proceso procesos2[] = {
    {"P1", 5, 0},
    {"P2", 3, 0},
    {"P3", 1, 0},
    {"P4", 2, 0},
    {"P5", 3, 0}
};

// Funcion Round Robin
roundRobin(procesos2, n_procesos2, quantum2); procesos: n_procesos: quantum
```

```
**** EJEMPLO 2 ****
Tiempo          Proceso
0 -> 2          P1
2 -> 4          P2
4 -> 5          P3
>>> Proceso P3 completado. Tiempo de total de ejecucion(retorno): 5

5 -> 7          P4
>>> Proceso P4 completado. Tiempo de total de ejecucion(retorno): 7

7 -> 9          P5
9 -> 11         P1
11 -> 12                P2
>>> Proceso P2 completado. Tiempo de total de ejecucion(retorno): 12

12 -> 13                P5
>>> Proceso P5 completado. Tiempo de total de ejecucion(retorno): 13

13 -> 14                P1
>>> Proceso P1 completado. Tiempo de total de ejecucion(retorno): 14


~/U/SistemasOperativos/Laboratorio_07 on git main ?2
```

2.- Algoritmo Shortest Job First

ShortestJobFirst.c

```C/C++
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Estructura para representar un proceso
typedef struct {
    char id[20];            // Identificador del proceso
    int tiempoServicio;     // Tiempo total de ejecucion(CPU) necesario para
el proceso
    int tiempoRestante;     // Tiempo de CPU restante para el proceso
} Proceso;

// Función del algoritmo Shortest Job First (SJF)
void sjf(Proceso procesos[], int n_procesos) {
    // Ordena los procesos por tiempo de servicio del menor a mayor
    for (int i = 0; i < n_procesos - 1; i++) {
        for (int j = 0; j < n_procesos - i - 1; j++) {
            if (procesos[j].tiempoServicio > procesos[j + 1].tiempoServicio)
{
                // Intercambio de procesos
                Proceso temp = procesos[j];
                procesos[j] = procesos[j + 1];
                procesos[j + 1] = temp;
            }
        }
    }

    printf("Tiempo\t\tProceso\n");
    int tiempoActual = 0;
    for (int i = 0; i < n_procesos; i++) {
        // Imprime el proceso actual y la información del tiempo recorrido
        printf("%d -> %d\t\t%s\n", tiempoActual, tiempoActual +
procesos[i].tiempoServicio, procesos[i].id);
        // Actualiza el tiempo actual
        tiempoActual += procesos[i].tiempoServicio;
        // Muestra el mensaje de proceso completado
        printf(">>> Proceso %s completado. Tiempo total de
ejecucion(retorno): %d\n", procesos[i].id, tiempoActual);
    }
}

int main() {
```

```
    // Ejemplo 1
    printf("**** EJEMPLO 1 ****\n");
    // Cantidad de procesos
    int n_procesos = 4;

    // Arreglo con los datos del ejemplo 1
    Proceso procesos[] = {
        {"A", 8, 0},
        {"B", 4, 0},
        {"C", 9, 0},
        {"D", 5, 0}
    };

    // Función SJF
    sjf(procesos, n_procesos);

    printf("\n**** EJEMPLO 2 ****\n");
    // Cantidad de procesos
    int n_procesos2 = 5;

    // Arreglo con los datos del ejemplo 2
    Proceso procesos2[] = {
        {"P1", 5, 0},
        {"P2", 3, 0},
        {"P3", 1, 0},
        {"P4", 2, 0},
        {"P5", 3, 0}
    };

    // Función SJF
    sjf(procesos2, n_procesos2);

    return 0;
}
```

Función principal:

Ordenación de procesos:
Ordenamos los procesos por su tiempo de servicio de menor a mayor. Esto significa que los procesos con el tiempo de ejecución más corto se colocarán al principio del arreglo.

Ejecución de procesos:
Después de ordenar los procesos, recorremos el arreglo ejecutando cada proceso.
Imprime el tiempo de inicio y fin de ejecución para cada proceso.
Actualiza el tiempo actual después de ejecutar cada proceso.
Muestra un mensaje indicando que el proceso ha sido completado, junto con el tiempo total de ejecución (retorno).

```c
// Función del algoritmo Shortest Job First (SJF)
void sjf(Proceso procesos[], int n_procesos) {
    // Ordena los procesos por tiempo de servicio del menor a mayor
    for (int i = 0; i < n_procesos - 1; i++) {
        for (int j = 0; j < n_procesos - i - 1; j++) {
            if (procesos[j].tiempoServicio > procesos[j + 1].tiempoServicio) {
                // Intercambio de procesos
                Proceso temp = procesos[j];
                procesos[j] = procesos[j + 1];
                procesos[j + 1] = temp;
            }
        }
    }

    printf("Tiempo\t\tProceso\n"); format:
    int tiempoActual = 0;
    for (int i = 0; i < n_procesos; i++) {
        // Imprime el proceso actual y la información del tiempo recorrido
        printf("%d -> %d\t\t%s\n", tiempoActual, tiempoActual + procesos[i].tiempoServicio, procesos[i].id); format:
        // Actualiza el tiempo actual
        tiempoActual += procesos[i].tiempoServicio;
        // Muestra el mensaje de proceso completado
        printf(">>> Proceso %s completado. Tiempo total de ejecucion(retorno): %d\n", procesos[i].id, tiempoActual); format:
    }
}
```

Ejecución con ejemplos dados
-Ejemplo 1

Teste 1 :

Proceso	Tiempo de llegada	Tiempo de servicio
A	0	8
B	1	4
C	2	9
D	3	5

Quantum 3 unit.

```c
// Ejemplo 1
printf("**** EJEMPLO 1 ****\n"); format:
// Cantidad de procesos
int n_procesos = 4;

// Arreglo con los datos del ejemplo 1
Proceso procesos[] = {
    {"A", 8, 0},
    {"B", 4, 0},
    {"C", 9, 0},
    {"D", 5, 0}
};

// Función SJF
sjf(procesos, n_procesos);
```

Ejecución

```
❯ gcc ShortestJobFirst.c -o eject && ./eject
**** EJEMPLO 1 ****
Tiempo          Proceso
0 -> 4          B
>>> Proceso B completado. Tiempo total de ejecucion(retorno): 4
4 -> 9          D
>>> Proceso D completado. Tiempo total de ejecucion(retorno): 9
9 -> 17         A
>>> Proceso A completado. Tiempo total de ejecucion(retorno): 17
17 -> 26                C
>>> Proceso C completado. Tiempo total de ejecucion(retorno): 26
```

Ejemplo 2:

Teste 2:

Proceso	Tiempo de llegada	Tiempo de servicio
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

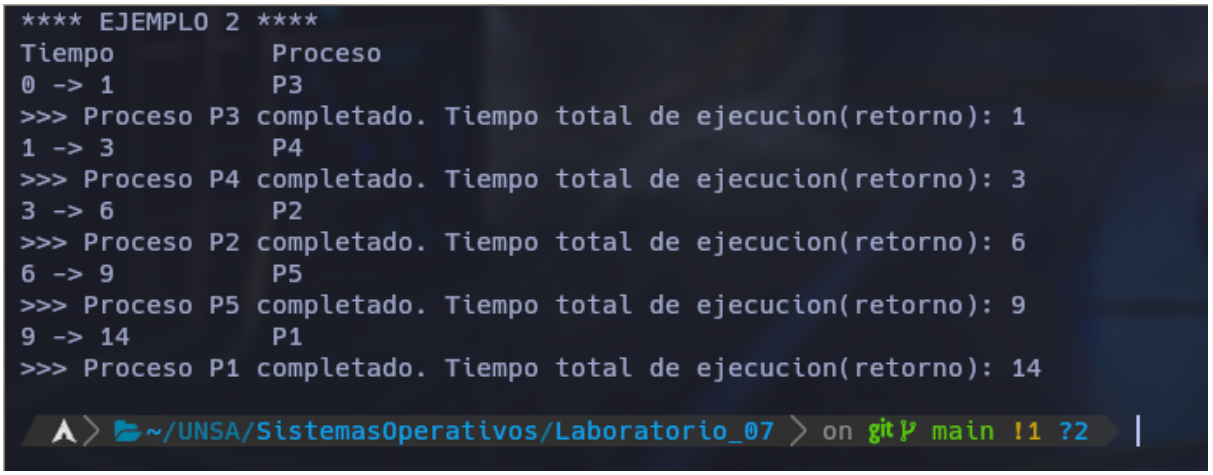
Quantum 2 unit

```c
printf("\n**** EJEMPLO 2 ****\n"); format:
// Cantidad de procesos
int n_procesos2 = 5;

// Arreglo con los datos del ejemplo 2
Proceso procesos2[] = {
    {"P1", 5, 0},
    {"P2", 3, 0},
    {"P3", 1, 0},
    {"P4", 2, 0},
    {"P5", 3, 0}
};

// Función SJF
sjf(procesos2, n_procesos2); procesos: n_procesos:
```

Ejecución:


**** EJEMPLO 2 ****
Tiempo Proceso
0 -> 1 P3
>>> Proceso P3 completado. Tiempo total de ejecucion(retorno): 1
1 -> 3 P4
>>> Proceso P4 completado. Tiempo total de ejecucion(retorno): 3
3 -> 6 P2
>>> Proceso P2 completado. Tiempo total de ejecucion(retorno): 6
6 -> 9 P5
>>> Proceso P5 completado. Tiempo total de ejecucion(retorno): 9
9 -> 14 P1
>>> Proceso P1 completado. Tiempo total de ejecucion(retorno): 14
~/UNSA/SistemasOperativos/Laboratorio_07 on git main !1 ?2