

# Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de Ciencias Computacionales

Sistemas Operativos



Profesora: Becerra Velázquez Violeta del Rocío

Alumno: Monjaraz Briseño Luis Fernando

Código: 218520958

Carrera: Ingeniería en Computación

Sección: D04

Actividad 7

Fecha: 08/10/2023

Algoritmo de planificación FCFS (First Come First Server).

## **Índice**

<b>Índice</b>	<b>2</b>
<b>Tabla de imágenes</b>	<b>3</b>
<b>Datos personales</b>	<b>4</b>
<b>Datos de la materia</b>	<b>4</b>
<b>Número de actividad</b>	<b>4</b>
<b>Objetivo de la actividad</b>	<b>4</b>
<b>Notas acerca del lenguaje</b>	<b>4</b>
<b>Link del VIDEO</b>	<b>16</b>
<b>Conclusión</b>	<b>16</b>

**Tabla de imágenes**

<i>Ilustración 1 Struct</i>	5
<i>Ilustración 2 Para contar los procesos listos y nuevos.</i>	5
<i>Ilustración 3 Rand ejemplo.</i>	6
<i>Ilustración 4 Interrupción.</i>	6
<i>Ilustración 5 Bloqueados.</i>	7
<i>Ilustración 6 Tabla de tiempos.</i>	8
<i>Ilustración 7 ValidaNumerosEnteros.</i>	9
<i>Ilustración 8 Gotoxy</i>	9
<i>Ilustración 9 IWillHaveOrder</i>	10
<i>Ilustración 10 ThisIsOrder</i>	11
<i>Ilustración 11 datosLotes (cabe aclarar que el nombre es el mismo que el de las anteriores actividades, sin embargo, este ya no son lotes).</i>	12
<i>Ilustración 12 ImprimirData.</i>	13
<i>Ilustración 13 Main</i>	13
<i>Ilustración 14 Ejemplo 1.</i>	14
<i>Ilustración 15 Ejemplo 2.</i>	14
<i>Ilustración 16 Ejemplo 3.</i>	15
<i>Ilustración 17 Ejemplo 4.</i>	15
<i>Ilustración 18 Ejemplo 5, tabla de tiempos. (Es otra ejecución, por lo que no son los mismos procesos del ejemplo anterior)</i>	16

**Datos personales**

Nombre: Monjaraz Briseño Luis Fernando

Código: 218520958

Correo: [luis.monjaraz5209@alumnos.udg.mx](mailto:luis.monjaraz5209@alumnos.udg.mx)

**Datos de la materia**

Materia: Sistemas Operativos

Sección: D04

Horario: Martes, Jueves, Sábado. 11:00 a 12:55

NRC: 204880

Clave: IL366

**Número de actividad**

Programa 3. Algoritmo de planificación FCFS (First Come First Server).

**Objetivo de la actividad**

El objetivo de esta actividad es recrear el funcionamiento de un algoritmo de planificación FCFS. Este mismo para el procesamiento se basa en el “Diagrama de 5 Estados”, en este caso se manejan 5 procesos en listos y el resto estarán en nuevos hasta que el número de procesos disminuya. Esta actividad ayudara a comprender el funcionamiento y como se programaría un sistema basado en FCFS, esto mientras observamos como las diversas opciones que implementamos se llevan a cabo, junto al final habrá una tabla de tiempos donde se enmarcan los diversos tiempos que fueron solicitados.

En resumen, esta actividad es sumamente útil para comprender la teoría del FCFS y el manejo de los Tiempos.

**Notas acerca del lenguaje**

Lenguaje usado: C++

Motivo: Cabe resaltar que este código es una modificación bastante grande al código del programa 2 por lo que estaba anclado a reutilizar C++, aunque he de ser sincero que en gran parte del desarrollo de este código me planteé reiniciar desde 0 el código o incluso cambiar el lenguaje a Python. Cabe resaltar que para las impresiones utilice gotoxy y que realmente la parte más difícil de este código “Bloqueo” pues esta me tomo alrededor de 16 horas en programar (para que cuando terminara me haya dado cuenta de que era mucho más sencillo de como lo pensaba).

Estructuras: Utilice un “Struct” llamado “Process” en esta se almacenan los datos que se utilizaran para los procesos, la estructura es una cola estática (limitada a 5), para que siempre haya solamente 5 procesos en el estado de “Listo”, cabe resaltar que también los demás procesos están en Colas de 5, sin embargo, mediante un while este va mandándolos a la cola 0 que es la que se maneja en todo el código.

```

34 struct Process {
35     string operation;
36     int number1;
37     int number2;
38     string result;
39     int estimatedTime;
40     int programNumber;
41     int currentQueue;
42     int tiempoTranscurrido; // no se aplica en todos los casos
43     int tiempoRestante; // no se aplica en todos los casos
44
45     int tiempoBloqueado;
46     int tiempoLlegada;
47     int tiempoFinalizacion;
48     int tiempoRetorno;
49     int tiempoRespuesta;
50     int tiempoEspera;
51     int tiempoServicio; // este sera igual al tiempoTranscurrido o al estimatedTime
52 };

```

Ilustración 1 Struct

```

307 queue<Process> cpc = queues[0];
308 while(!cpc.empty()){
309     Process temporal = cpc.front();
310     cpc.pop();
311     contadordeprocesoslistos = contadordeprocesoslistos + 1;
312     contadordeprocesosnuevos = contadordeprocesosnuevos - 1;
313 }

```

Ilustración 2 Para contar los procesos listos y nuevos.

#### Funciones:

- ValidaNumerosEnteros: Esta función valida que lo que el usuario ingrese sea un número entero, esto mediante un char para facilitar su comparación.
- Gotoxy: Esta me ayuda a hacer el aspecto visual del programa.
- IWillHaveOrder: Es la función que grafica las líneas visuales de los datos.
- ThisIsOrder: Es la función que grafica las líneas visuales del programa.
- DatosLotes: Es la función encargada de solicitar y hacer los cálculos con los datos de los procesos, al igual que validarlos.
- ImprimirDatos: Es la función encargada de imprimir toda la información, por lo que también es la que imprime el tiempo.
- Main: Es la función principal que llama a las demás y realiza limpieza en la pantalla.

Al ser una “actualización bastante grande” del programa anterior la estructura es prácticamente la misma, o sea, se utilizó una cola estática, donde el usuario ingresa la cantidad de procesos y el sistema elegirá todos los demás valores de forma aleatoria, en este caso use “Rand()”, este no es completamente aleatorio pues este se considera “pseudoaleatorio” pero a final de cuentas ni yo ni el usuario determina los valores. Ejemplo del uso de “Rand()”:

```
226     int operationN1 = rand() % 101;
227     int operationN2 = rand() % 101;
228     newProcess.number1 = operationN1;
229     newProcess.number2 = operationN2;
230
231     if(newProcess.operation == "/"){
232         while(newProcess.number2 == 0){
233             int operationN2 = rand() % 101;
234             newProcess.number2 = operationN2;
235         }
236     }
```

Ilustración 3 Rand ejemplo.

Para dar la “ilusión” de que se tarda utilice Sleep y gotoxy para que se muestre una vez que terminara el tiempo de ejecución, en general, eso aplico para toda la impresión, pues con la ayuda de Sleep controle los tiempo para que una vez que pasara el tiempo máximo estimado este mostrara los resultados, al igual que en estos tiempos se pueden realizar las interrupciones, pausas y errores, y con la ayuda de gotoxy le di forma a todo el código, es verdad que esto trae problemas pero nada que no se solucione calculando correctamente las coordenadas al igual que borrar correctamente las mismas. Realmente no tiene mucha ciencia el código, o sea, si tengo muchas declaraciones, pero estas son para controlar en su mayoría el tiempo o la cantidad de procesos. Esto también aplica para los bloqueos, pues se manejan dentro del bucle de tiempo.

```
447     if (_kbhit() && process.tiempotranscurrido != process.estimatedTime) {
448         char key = _getch();
449         if (key == 'I' || key == 'i') { // interrumpir
450             pulsar = 'i';
451             interruptedProcess.tiempotranscurrido = process.tiempotranscurrido; // Guardar el tiempo transcurrido
452             interruptedProcess.tiemporestante = process.tiemporestante; // Guardar el tiempo total
453             queues[i].pop(); // Sacar el proceso de la cola actual
454             bloqueados.push(interruptedProcess);
455             gotoxy(18,15);
456             cout << " ";
457             timecontador = timecontador - 1;
458             // Limpiar la cola temporal
459             while (!tempQueue.empty()) {
460                 tempQueue.pop();
461             }
462             // Volver a llenar la cola temporal con los procesos restantes de la cola original
463             queue<Process> originalQueue = queues[i];
464             Process tprocess = originalQueue.front();
465             y = 7;
466             while (!originalQueue.empty()) {
467                 tempQueue.push(originalQueue.front());
468                 gotoxy(3,5);
469                 cout << "ID";
470                 gotoxy(3,y);
471                 cout << tprocess.programNumber;
```

Ilustración 4 Interrupción.

```

554 by = 19;
555 if (!bloqueados.empty()) {
556     queue<Process> tempbloq = bloqueados;
557     while (!tempbloq.empty()) {
558         Process temporal = tempbloq.front();
559         tempbloq.pop();
560         temporal.tiempobloqueado = temporal.tiempobloqueado - 1;
561         // Aquí se imprimirán los datos de cada proceso en la cola de bloqueados
562         gotoxy(3, by);
563         cout << temporal.programNumber;
564         gotoxy(6, by);
565         cout << temporal.tiempobloqueado; // tiempo bloqueado
566         by = by + 1;
567         bloqueados.pop();
568         bloqueados.push(temporal);
569         if (temporal.tiempobloqueado <= 0) {
570             temporal.tiempobloqueado = 8;
571             queues[i].push(temporal);
572             bloqueados.pop();
573         }
574         queue<Process> tempQueue = queues[i]; // Copia temporal de la cola
575         y = 7;
576         ay = 7;
577         int contadorded = 5;
578         while(contadorded >=0)
579         {
580             gotoxy(3,y);
581             cout << "          ";
582             y = y + 1;
583             contadorded--;
584         }
585         v = 7;

```

PROBLEMAS    SALIDA    CONSOLA DE DEPURACIÓN    TERMINAL    PUERTOS

Ilustración 5 Bloqueados.

```
908     while(!totalProcessesQueue->empty()){
909         Process process = totalProcessesQueue->front();
910         totalProcessesQueue->pop();
911         gotoxy(3,ffty);
912         cout << process.programNumber;
913         gotoxy(7,ffty);
914         cout << process.tiempollegada;
915         gotoxy(11,ffty);
916         cout << process.tiempofinalizacion;
917         gotoxy(15,ffty);
918         cout << process.tiemporetorno;
919         gotoxy(19,ffty);
920         cout << process.tiemporespuesta;
921         gotoxy(25,ffty);
922         cout << process.tiempoespera;
923         gotoxy(31,ffty);
924         cout << process.tiemposervicio;
925         gotoxy(37,ffty);
926         cout << process.estimatedTime;
927         ffty = ffty + 1;
928     }
929 }
```

*Ilustración 6 Tabla de tiempos.*

Importante: En cuanto habrá el programa (importante que sea desde el .exe) dele al botón de “Maximizar pantalla”, o sea, el cuadradito que está en medio de minimizar y cerrar, esto porque se emplea Gotoxy y este ocasiona problemas si la pantalla no es lo suficientemente grande, de igual forma el Gotoxy está adaptado a mi pantalla (14 pulgadas) por lo que, si su pantalla es menor no se verá bien, si esta es mayor si se verá bien. De todas formas, implemente unas líneas de código que vuelven la pestaña más grande de lo normal. Nota: La cantidad de librerías es porque son de “colección” de librerías, por lo que no se usaron todas.

Como se puede apreciar la mayoría de los if cambian un valor de “pulsar” este es para que al momento de la impresión se seleccione la forma correcta dependiendo de la opción.

Funciones:



```

54 // Función que valida que los datos ingresados por el usuario sean números enteros
55 bool ValidaNumerosEnteros(char *dato){
56     bool ban = true;
57     int i = 0;
58     if (*dato == '-' || *dato == '+') {
59         i++;
60     }
61     while (*(dato + i) != '\0') {
62         if (*(dato + i) < '0' || *(dato + i) > '9') {
63             ban = false;
64             break;
65         }
66         i++;
67     }
68     return ban;
69 }
70

```

Ilustración 7 ValidaNumerosEnteros.

```

71 void gotoxy(int x,int y){
72     HANDLE hcon;
73     hcon = GetStdHandle(STD_OUTPUT_HANDLE);
74     COORD dwPos;
75     dwPos.X = x;
76     dwPos.Y= y;
77     SetConsoleCursorPosition(hcon,dwPos);
78 }
79

```

Ilustración 8 Gotoxy

```
88  √ void IWillHaveOrder(){
89      int x = 1, y = 1;
90      gotoxy(0,0);
91      printf("%c", 201); // 
92      gotoxy(132,0);
93      printf("%c", 187); // 
94      gotoxy(0,31);
95      printf("%c", 200); // 
96      gotoxy(132,31);
97      printf("%c", 188); // 
98  √  while (y<=30)
99      {
100          gotoxy(0,y);
101          printf("%c", 186); // 
102          gotoxy(132,y);
103          printf("%c", 186); // 
104          y++;
105      }
106  √  while (x<=131)
107      {
108          gotoxy(x,0);
109          printf("%c", 205); // 
110          gotoxy(x,31);
```

*Ilustración 9 IWillHaveOrder*

```

void ThisIsOrder(){
    int x = 1, y = 1;
    gotoxy(0,0);
    printf("%c", 201); // 
    gotoxy(132,0);
    printf("%c", 187); // 
    gotoxy(0,31);
    printf("%c", 200); // 
    gotoxy(132,31);
    printf("%c", 188); // 
    while (y<=30)
    {
        gotoxy(0,y);
        printf("%c", 186); // 
        gotoxy(132,y);
        printf("%c", 186); // 
        y++;
    }
    while (x<=131)
    {
        gotoxy(x,0);
        printf("%c", 205); // 
        gotoxy(x,31);
        printf("%c", 205); // 
        x++;
    }
}

```

Ilustración 10 ThisIsOrder

Algoritmo de planificación FCFS (First Come First Server).

```
184 void datosLotes(){
185     char totalProcessesc[100];
186     gotoxy(1,1);
187     cout << "Ingrese el numero de procesos: ";
188     cin >> totalProcessesc;
189     while(!ValidaNumerosEnteros(totalProcessesc)){
190         gotoxy(1,1);
191         cout << "
192         gotoxy(1,1);
193         cout << "Ingrese el numero de procesos de nuevo: ";
194         cin >> totalProcessesc;
195     }
196     totalProcesses = atoi(totalProcessesc);
197
198     int currentQueue = 0;
199     for (int i = 1; i <= totalProcesses; ++i) {
200         Process newProcess;
201         newProcess.currentQueue = currentQueue+1;
202         int operationIndex = rand() % 6;
203         switch (operationIndex)
204         {
205             case 0:
206                 newProcess.operation = "+";
207                 break;
208             case 1:
209                 newProcess.operation = "-";
210                 break;
211             case 2:
212                 newProcess.operation = "*";
213                 break;
214             case 3:
215                 newProcess.operation = "/";
```

*Ilustración 11 datosLotes (cabe aclarar que el nombre es el mismo que el de las anteriores actividades, sin embargo, este ya no son lotes).*

```

292 void imprimirdata() {
293     int procesosTotales = totalProcesses;
294     int contadordeprocesosnuevos = totalProcesses;
295     int contadordeprocesoslistos = 0;
296     int ffy = 7;
297     char pulsar = ' ';
298     int a = 1;
299     gotoxy(80, 13);
300     cout << "I = Interrumpir";
301     gotoxy(80, 15);
302     cout << "E = Error";
303     gotoxy(80, 17);
304     cout << "P = Pausar";
305     gotoxy(80, 19);
306     cout << "C = Continuar";
307     queue<Process> cpc = queues[0];
308     while(!cpc.empty()){
309         Process temporal = cpc.front();
310         cpc.pop();
311         contadordeprocesoslistos = contadordeprocesoslistos + 1;
312         contadordeprocesosnuevos = contadordeprocesosnuevos - 1;
313     }
314     queue<Process> totalProcessesQueue[1]; // En esta cola se almacenaran todos los datos finalizados
315     for (int i = 0; i < maxQueues; i++) {
316         int tiemposervicios = 0;
317         gotoxy(1,1);
318         cout << "Procesos Nuevos: " << contadordeprocesosnuevos; // Modi
319         int acomodainterumpir = 11;
320         int ay = 7;
321         int y = 7;
322         if (!queues[i].empty()) {
323             gotoxy(3,3);
324             cout << "P Listos #" << contadordeprocesoslistos << endl; // Modi
325             queue<Process> tempQueue = queues[i]; // Copia temporal de la cola

```

Ilustración 12 ImprimirData.

```

1070 int main() {
1071     HWND consoleWindow = GetConsoleWindow();
1072     RECT desktop;
1073     GetWindowRect(GetDesktopWindow(), &desktop);
1074     MoveWindow(consoleWindow, desktop.left, desktop.top, desktop.right, desktop.bottom, TRUE);
1075     system("pause");
1076     ThisIsOrder();
1077     datosLotes();
1078     system("cls");
1079     IWillHaveOrder();
1080     imprimirdata();
1081     gotoxy(80,30);
1082     system("pause");
1083     system("cls");
1084     return 0;
1085 }

```

Ilustración 13 Main

Como se debería de ver:

## Algoritmo de planificación FCFS (First Come First Server).

D:\beto5\Descargas\Tareas\SO\Act7 SO\MonjarazLuisD04Act7\output\MonjarazLuisD04Act7.exe									
Procesos Nuevos: 3									
P	Listos	#4	Ejec	Terminados					
ID	TME	TT	Nn	ID	Ope	Res	Tanda	Guia	Procesos restantes
4	16	0	ID 3	2	69porcentaje	65	Error	1	8
5	8	0	Ope residuo						Tiempo total
6	16	0	TME 6						7
			TT 4						I = Interrumpir
			TR 2						E = Error
			Bloqueados						P = Pausar
			ID TB						C = Continuar
1	1								

Ilustración 14 Ejemplo 1.

D:\beto5\Descargas\Tareas\SO\Act7 SO\MonjarazLuisD04Act7\output\MonjarazLuisD04Act7.exe									
Procesos Nuevos: 2									
P	Listos	#5	Ejec	Terminados					
ID	TME	TT	Nn	ID	Ope	Res	Tanda	Guia	Procesos restantes
5	8	0	ID 4	2	69porcentaje	65	Error	1	7
6	16	0	Ope -	3	22residuo	49	22	1	Tiempo total
1	12	2	TME 16						14
7	15	0	TT 5						I = Interrumpir
			TR 11						E = Error
			Bloqueados						P = Pausar
			ID TB						C = Continuar
									Pausado

Ilustración 15 Ejemplo 2.

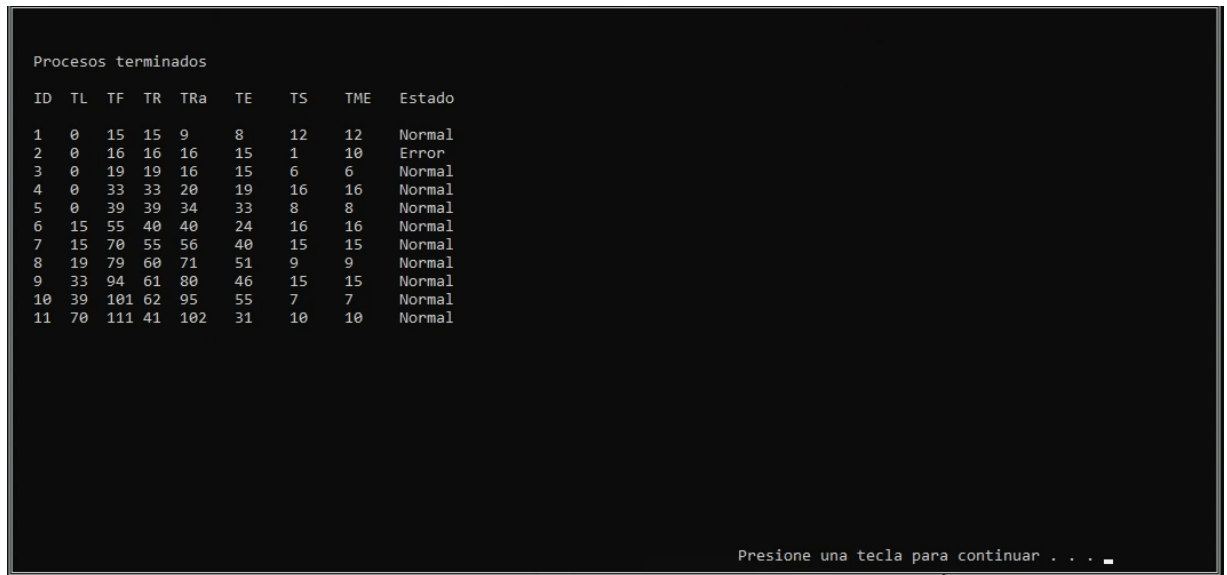
D:\beto5\Descargas\Tareas\SO\Act7 SO\MonjarazLuisD04Act7\output\MonjarazLuisD04Act7.exe									
Procesos Nuevos: 1									
P Listos #5		Ejec		Terminados				Procesos restantes	
ID	TME	TT	Nn	ID	Ope	Res	Tanda		
							Guia		
6	16	0	ID 5	2	69porcentaje65	Error	1	6	
1	12	2	Ope -	3	22residuo49	22	1		
7	15	0		4	61-63	Error	1		
8	9	0	TME 8						
			TT 3						
			TR 5						
Bloqueados									
ID TB									
								Tiempo total	
								20	
								I = Interrumpir	
								E = Error	
								P = Pausar	
								C = Continuar	

Ilustración 16 Ejemplo 3.

Procesos Nuevos: 0									
P Listos #3		Ejec		Terminados				Procesos restantes	
ID	TME	TT	Nn	ID	Ope	Res	Tanda		
							Guia		
10	7	0	ID 9	1	85porcentaje72	61	1	3	
11	10	0	Ope porcentaje	2	69porcentaje65	Error	1		
				3	22residuo49	22	1		
			TME 15	4	61-63	-2	1		
			TT 5	5	24-80	-56	1		
			TR 10	6	60/64	0	2		
Bloqueados				7	60residuo49	11	2		
ID TB				8	99*94	9306	2		
								Tiempo total	
								85	
								I = Interrumpir	
								E = Error	
								P = Pausar	
								C = Continuar	

Ilustración 17 Ejemplo 4.

## Algoritmo de planificación FCFS (First Come First Server).



ID	TL	TF	TR	TRa	TE	TS	TME	Estado
1	0	15	15	9	8	12	12	Normal
2	0	16	16	16	15	1	10	Error
3	0	19	19	16	15	6	6	Normal
4	0	33	33	20	19	16	16	Normal
5	0	39	39	34	33	8	8	Normal
6	15	55	40	40	24	16	16	Normal
7	15	70	55	56	40	15	15	Normal
8	19	79	60	71	51	9	9	Normal
9	33	94	61	80	46	15	15	Normal
10	39	101	62	95	55	7	7	Normal
11	70	111	41	102	31	10	10	Normal

Presione una tecla para continuar . . .

Ilustración 18 Ejemplo 5, tabla de tiempos. (Es otra ejecución, por lo que no son los mismos procesos del ejemplo anterior)

Cabe resaltar que se realizaron las correcciones que me menciono en la clase.

Enlace de descarga (contenido):

[https://drive.google.com/drive/folders/1sKcTMp3yvUfvloHq49pC210uBmKAG43\\_?usp=sharing](https://drive.google.com/drive/folders/1sKcTMp3yvUfvloHq49pC210uBmKAG43_?usp=sharing)

**Link del VIDEO:**

Link (Youtube):

<https://youtu.be/Hn0hOyeQSAk>

Link (Drive):

<https://drive.google.com/file/d/1HPzDDk2Jwsvmggh6qAY4CFtmcR7AcQTq/view?usp=sharing>

### ***Conclusión***

En conclusión, esta actividad es una forma clara de entender y representar un FCFS, este con la implementación del diagrama de 5 estados, junto con kbhit. En lo personal esta actividad me ayudo mucho para entender el funcionamiento interno de un FCFS, tal vez el código no sea la forma correcta de realizar dicha actividad, sin embargo, fue la única forma de programarlo que se me ocurrió, debo decir que este código ha sido el más difícil de los que he realizado en este año, pues me tarde mucho mas de lo que planeaba gracias a Bloqueado. Regresando a la actividad cuenta con una impresión de los tiempos.