

# Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de ciencias computacionales.

Ingeniería en computación



Seminario de Solución de Problemas de Sistemas Operativos - D01.

Violeta del Rocío Becerra Velázquez

“Programa 3. Algoritmo de planificación  
FCFS.”

2023A

17/02/2023

Flores Estrada Abraham Miguel Angel

Olguín Hernández Jair Benjamín

Índice.

Índice.	2
Objetivo.	3
Lenguaje utilizado.	3
Video del programa:	10

## Objetivo.


El algoritmo de planificación FCFS (First Come, First Served) es uno de los métodos más simples y antiguos para la planificación de procesos en un sistema operativo. Como su nombre lo indica, este algoritmo asigna la CPU al primer proceso que llega y lo ejecuta hasta que completa su tiempo de ejecución, momento en el cual se pasa al siguiente proceso en la cola. Este método de planificación es fácil de entender e implementar, ya que no requiere un seguimiento complejo del estado del proceso. Sin embargo, puede llevar a una ineficiencia en la utilización de la CPU, especialmente si los procesos largos llegan primero y ocupan la CPU durante mucho tiempo, lo que hace que los procesos más cortos esperen en la cola. Por lo que buscamos crear un programa que pueda simular el algoritmo de FCFS, donde podemos aprovechar del programa de simulación por lotes con multiprogramación pero ahora aplicando el algoritmo para que tenga una lista de procesos listos sin importar el lote y así mismo los procesos que puedan estar bloqueados.

## Lenguaje utilizado.

Continuamos con el uso de python, debido a que esta actividad se puede desarrollar sobre el código pasado, donde podemos aprovechar gran parte solamente modificando algunas partes y agregando lista de bloqueados, así mismo podemos sacar provecho del contador que hacemos con las funciones de python para poder utilizar y obtener todos los tiempos necesarios.

## Desarrollo del programa.

En este caso debido a los requerimientos del programa la segunda interfaz cambio para agregar una lista de bloqueados que veremos más detalladamente adelante, pero primero es necesario explicar como es el proceso que hacemos en código. En este caso seguimos utilizando nuestra lista principal que es donde guardamos todos nuestros procesos que se generan de manera automática. Después de esto recordemos que al aplicar el algoritmo de FCFS tendremos que tener una lista de 4 procesos listos para entrar al CPU, por lo que nosotros hacemos nuestra lista de "listos", que como su nombre indica iremos guardando los procesos pero asegurándonos que no pasen de 4 procesos, esto lo hacemos gracias a 2 contadores, uno que nos ayuda a saber cuál fue el último proceso agregado y otro para saber cuantos procesos tenemos en nuestra lista.




```

1  while (act)<numProc and i<5:
2      listos.append(self.procesos[act])
3      print(act)
4      act+=1;
5      i+=1

```

Con esto nos aseguramos que siempre vamos a tener cuatro procesos en nuestra cola de “listos”, después de esto empezaremos a iterar hasta que nuestra lista de terminados sea igual al número de procesos que tenemos. Una vez con esto lo primero que hacemos es verificar que nuestra cola de listos no sea 0, con esto empezar a mandarlos a procesar y tener control de sus tiempos. Después de esto vamos a empezar a ejecutar el primer proceso en la cola, donde vamos a verificar que el tiempo total sea 0, esto para poder darles un tiempo de respuesta. Y una vez que actualizamos esto lo sacamos de la cola y actualizamos nuestra cola de listos con el siguiente proceso.



```

1  if (len(listos)>0):
2      flag=True;
3      j=listos[0]
4      if j.Tt == 0:
5          j.tRespuesta = self.tg - j.tLlegada
6      listos.pop(0)
7      self.actualizarListos(listos)

```

Con esto ya tenemos nuestra cola de listos con cuatro procesos asegurados, y tenemos en ejecución un proceso, podemos tener tres casos principales, que al igual que el programa anterior hacemos uso de las teclas para saber que caso puede pasar, donde podemos pausar todo el programa, mandar el proceso en ejecución finalizado con error o tener una interrupción, donde ahora tendremos que mandarlo a bloqueados con una espera de 8 segundos. En este caso lo que hacemos es iterar sobre el tiempo de proceso del proceso que se encuentra en ejecución, este el tiempo que se genera en automático desde el inicio, donde en

cada momento del programa se verifica que tecla se presiona, pero a diferencia del programa anterior tenemos un cambio en la parte de interrupción, donde ahora no solo se manda al final de la cola, sino que tiene que pasar por la lista de bloqueados, por lo que hacemos uso de una nueva lista auxiliar llamada bloq, donde esta irá guardando todo aquel proceso que haya tenido una interrupción.

```
1  for k in range(tp):
2      self.actualizarListos(listos)
3      if self.FPausa:
4          while True:
5              if self.FPausa == False:
6                  break;
7      if self.FError:
8          j.res="ERROR";
9          self.FError=False;
10         break;
11     if self.FInterrupcion:
12         j.Tb = 8
13         bloq.append(j)
14         self.FInterrupcion=False;
15         flag=False;
16         break;
```

En caso de que algún proceso tenga alguna interrupción y sea mandado a bloqueados, lo que hacemos es un bucle que irá recorriendo la lista de bloqueados, acudiendo proceso por proceso e irá restando el tiempo bloqueado, donde anteriormente asignamos el valor de 8, y restaura en 1 en 1 este valor, una vez que un proceso de la lista tenga su tiempo bloqueado en 0, se mandará a la cola de listos y se elimina de la lista de bloqueados.

```
1  if len(bloq)!=0:
2      cont = 0
3      toDel=False;
4      while cont < len(bloq):
5          row = bloq[cont]
6          row.Tb -= 1
7          if row.Tb == 0:
8              Del = row;
9              toDel = True
10             cont += 1
11     if toDel:
12         listos.append(Del)
13         bloq.remove(Del)
```

Por último tenemos que calcular los tiempos faltantes para cuando el proceso termine, donde simplemente hacemos resta de los tiempos que tenemos para sacar el tiempo de finalización, tiempo de retorno, y el tiempo de espera; y agregar el proceso que estaba en ejecución a la lista de procesos terminados.

```
1  if flag:
2      j.tFinali = self.tg
3      j.tRetorno = j.tFinali-j.tLlegada;
4      j.tEspera = j.tFinali - j.tLlegada - j.Tt
5      self.done.append(j)
```

Recordemos que estamos trabajando cada proceso como un objeto, por lo cual cada proceso tiene su propio tiempo de cada cosa.

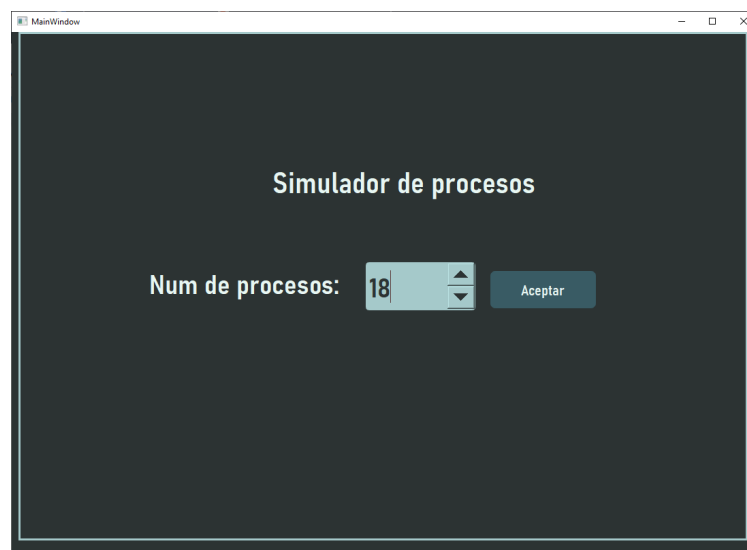
```
1  class proceso(object):
2      def __init__(self,Id,operacion,res,tiEs):
3          self.id = Id
4          self.operacion = operacion
5          self.res = str(res)
6          self.tiEs = tiEs
7          self.tLlegada = 0;
8          self.tFinali = 0;
9          self.tRetorno = 0;
10         self.tRespuesta = 0;
11         self.tEspera =0;
12         self.Tservicio = 0;
13         self.Tt = 0
14         self.Tr = self.tiEs
15         self.Tb = 0
```

- Tiempo de llegada: fue asignado cuando se generaron, por lo que los primeros cuatro procesos en estar listos inician en 0 y los demás es al valor del contador global en cuanto entran a la cola de listos.
- Tiempo de finalización: es asignado cuando el proceso termina, por lo que es igual al tiempo global del programa.
- Tiempo de retorno: es asignado cuando el proceso termina, y es igual al tiempo de finalización menos el tiempo de llegada.

- Tiempo de respuesta: es asignado cuando el proceso entra a la cola de listos y es igual al tiempo global de ese momento menos el tiempo de llegada.
- Tiempo de espera: es asignado cuando el proceso termina, y es igual al tiempo de finalización menos el tiempo de llegada menos el tiempo total del mismo proceso.
- Tiempo de servicio: es asignado cuando entra a ser procesado y es igual al tiempo transcurrido.

Veamos el programa en ejecución:

Inicia con pidiendonos el número de procesos:



Al iniciar el programa tendremos un proceso en ejecución, y 3 procesos más en la cola de listos, nuestra lista de bloqueados en 0 y los nuevos procesos (que en este caso serán 14, debido a que se restan los 3 de la cola de listos y el que está en ejecución).



Si hacemos que los procesos tengan una interrupción se mandan procesos a la lista de bloqueados, cada uno de estos tendrá su contador del tiempo bloqueado y la cola de listos estará en espera, debido a que la lista de bloqueados debe ser 0 para mandarnos nuevos procesos.

MainWindow

Nuevos procesos:

13

Contador Global:

18

Simulador de procesos

Procesos Terminados

	Id	Op	Res
1	4	76 * 301	22876

Proceso en Ejecución

	1
Id	3
Operacion	310 / 33
Tiempo	8
Tiempo Total	5
Tiempo Restante	3

Iniciar

Proceso bloqueados

	ID	TB
1	1	7
2	5	6
3	2	4

Ver tiempos

Volveremos a mandar procesos interrumpidos y dejaremos que algunos terminan normalmente o con error.

**Simulador de procesos**

Nuevos procesos: **5**

Procesos Terminados

	Id	Op	Res
1	4	76 * 301	22876
2	3	310 / 33	9.393939393939394
3	1	449 / 7_	ERROR
4	5	482 / 6_	ERROR
5	2	107 + 2_	307
6	6	963 - 1_	814
7	7	191 + 92	ERROR
8	8	835 % _	63
9	9	87 % 110	ERROR

Listos

	Id	TiEs	TT
1	10	7	1

Proceso en Ejecución

	1
Id	13
Operacion	454 / 601
Tiempo	13
Tiempo Total	2
Tiempo Restante	11

Iniciar

Proceso bloqueados

	ID	TB
1	11	8
2	12	7

Contador Global: **222**

Ver tiempos



Cuando tengamos el apartado de nuevos procesos en 0, es porque todos los procesos generados ya han pasado por la cola de listos, con esto podemos ver que cada proceso va a tener un tiempo de llegada diferente.

MainWindow

Simulador de procesos

Nuevos procesos: 0

Procesos Terminados

	Id	Op	Res
2	3	310 / 33	9.393939393939394
3	1	449 / 7...	ERROR
4	5	482 / 6...	ERROR
5	2	107 + 2...	307
6	6	963 - 1...	814
7	7	191 + 92	ERROR
8	8	835 % ...	63
9	9	87 % 110	ERROR
10	13	454 / 6...	0.7554076539101497
11	10	885 + 4...	1295
12	11	426 * 7...	ERROR
13	16	91 + 81	ERROR
14	12	534 * 5...	ERROR
15	17	878 / 8...	1.0305164319248827

Ver tiempos

Proceso en Ejecución

	1
Id	14
Operacion	326 % 717
Tiempo	10
Tiempo Total	5
Tiempo Restante	5

Proceso bloqueados

	ID	TB
1	18	7

Contador Global: 179

Inicio

Una vez terminado todos los procesos nos dará una tabla con todos los tiempos de cada uno de los procesos.

MainWindow

Tiempos

Volver a simulador

Id	TME	T.Llegada	T.Finalizaci	T.Retorno	T.Respuesta	T.Espera	T.Servicio	Operacion	Resultado
1	12	0	21	21	0	18	3	449 / 713	ERROR
2	12	0	84	84	2	72	12	107 + 200	307
3	8	0	21	21	3	13	8	310 / 33	9.393939393939394
4	6	0	10	10	4	4	6	76 * 301	22876
5	7	10	24	14	1	10	4	482 / 683	ERROR
6	12	21	91	70	3	58	12	963 - 149	814
7	6	21	92	71	6	66	5	191 + 92	ERROR
8	7	24	100	76	5	69	7	835 % 772	63
9	8	84	120	36	8	31	5	87 % 110	ERROR
10	7	91	139	48	6	41	7	885 + 410	1295
11	6	92	142	50	6	46	4	426 * 738	ERROR
12	11	100	163	63	42	56	7	534 * 518	ERROR
13	13	120	133	13	0	0	13	454 / 601	0.7554076539101497
14	10	133	184	51	13	41	10	326 % 717	326
15	16	139	198	59	8	43	16	173 % 110	63
16	14	142	149	7	6	6	1	91 + 81	ERROR
17	6	149	171	22	0	16	6	878 / 852	1.0305164319248827
18	11	163	207	44	3	33	11	134 % 17	15

## Conclusiones.

Al principio dudamos un poco del cómo podía ser el código del programa, ya que antes de la investigación e implementación desconocíamos totalmente del algoritmo, pero en cuanto buscamos de este supimos que era muy parecido al uso de la cola con el FIFO. A partir de esto encontramos varias ideas que nos podrían ayudar para realizar la simulación, a pesar de todo no fue complicado porque al igual que los programas anteriores se puede hacer uso de los programas anteriores y solo hacer modificaciones a algunas funciones, pero tuvimos que tener mucho cuidado con los tiempos, debido a que ahora el tiempo que transcurrían en procesamiento era el más importante de todos, haciendo que tuviéramos que tener algunas validaciones para este.

## Video del programa

En YouTube:

[https://youtu.be/56C\\_C10OJtA](https://youtu.be/56C_C10OJtA)

En drive:

<https://drive.google.com/file/d/18tpQjCMrW6-tSn7e6YpVhbp8Kcsj0qdj/view?usp=sharing>