



Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de Ciencias Computacionales.
Ingeniería en Computación



Actividad 7.
Programa 3. Algoritmo de Planificación
FCFS (First Come First Server)

Materia: Sistemas Operativos

Sección: D04

Profesor: Becerra Velázquez Violeta del Rocío.

Equipo:

- Carriola Navarro Aldo Fernando | 221978167
- Cervantes Araujo Maria Dolores | 217782452

Fecha: 08/10/2023

Programa 3. Algoritmo de Planificación FCFS

Índice

Objetivos _____	4
Notas Sobre el Lenguaje _____	4
TDA o Estructuras Utilizadas en el Programa _____	5
Funcionamiento _____	9
Errores y Soluciones _____	12
Conclusiones _____	15

Tabla de Imágenes

Ilustración 1. Función nueva.....	5
Ilustración 2. función main cambios 1	5
Ilustración 3. función main cambios 2	6
Ilustración 4. función main cambios 3	7
Ilustración 5. función procesos.	8
Ilustración 6. Inicio del programa.....	9
Ilustración 7. Procesos en tablas.	9
Ilustración 8. Interrupción por error	9
Ilustración 9. Interrupción a bloqueados.....	10
Ilustración 10. Paso de bloqueados a listos.....	10
Ilustración 11. Datos genéricos por proceso	11
Ilustración 12. Error 1	12
Ilustración 13.Causa del Error 1	12
Ilustración 14. Solución Error 1	13
Ilustración 15. Error 2.....	13
Ilustración 16. Causa del Error 2	14
Ilustración 17. Solución Error 2.....	14

Objetivos

En los anteriores dos programas se ha creado un procesamiento por lotes de 5 procesos a la vez, siendo que hasta que finalizará dicho lote entraban un nuevo lote a nuestro sistema. Además de contar con interrupciones, errores y pausa en el programa. Sin embargo, para este tercer programa se buscan nuevos objetivos, teniendo en cuenta lo siguiente por realizar:

- ✓ Ahora las interrupciones ocasionadas al presionar la tecla 'i', pondrá a nuestro proceso en estado bloqueado por un tiempo exacto (ocho segundos), donde una vez finalizado su tiempo de bloqueo se encolará en la lista de procesos listos.
- ✓ Ya no existen los lotes, sin embargo, se deberá seguir la regla de que siempre deberá haber máximo 5 procesos en el sistema, es decir, contando procesos listos, en ejecución y bloqueados. Entonces una vez un proceso salga a terminados, se encolará un proceso de nuevo a listo (en caso de que si exista uno).
- ✓ Como se puede notar en los puntos anteriores, es generar un algoritmo First Come First Server, donde se vayan encolando los procesos a listos conforme lleguen, sin importar si viene de estado bloqueado o nuevo.
- ✓ Ahora una vez terminados todos nuestros procesos se mostrará una tabla final que incluya ID, operación, resultado (si el proceso finalizó por error) y los siguientes tiempos: estimado, transcurrido, restante, bloqueado, llegada, servicio, espera, respuesta, retorno, final.

Notas Sobre el Lenguaje

El lenguaje utilizado para llevar a cabo el programa fue *Python*, dado que nos permite experimentar con diversas funciones que apoyan en la creación del programa, permitiéndonos que sea orientado a objetos, estructural, con interfaz, a consola, con color e incluso animación.

Dado que es un lenguaje de nuestro total conocimiento y dominio para ambos, optamos por crear el programa de forma estructural, es decir, mediante funciones. Buscando siempre mediante acuerdos, que ambos nos sintiéramos familiarizados con el lenguaje y lógica del programa, de esta manera optimizamos tiempos y nos fue más sencillo modelar y visualizar el programar conforme lo esperado.

TDA o Estructuras Utilizadas en el Programa

Todo el código de nuestro programa se implementa en un solo archivo con programación estructural o secuencial, se utilizó el TDA lista de listas, que nos permitió una mejor manipulación con los datos al momento de reflejarlos en las tablas.

Para el nuevo programa, se agregó una nueva función que nos ayuda a imprimir todos los valores que se manejan a lo largo de cada ejecución de los procesos:

```
def impresion_final():
    print("\n\nProcesos")
    headersB = ["ID", "Ope", "Res", "TME", "TI", "TR", "TB", "TLI", "TS", "TE", "TRes", "TRet", "TF"]
    procesosGen = [[dato[0], dato[1], dato[2], dato[3], dato[4], dato[5], dato[14], dato[7], dato[8], dato[9], dato[10], dato[11], dato[12]] for dato in procesos_terminados]
    print(tabulate(procesosGen, headers=headersB, tablefmt="double_outline"))
```

Ilustración 1. Función nueva

Debido a que previamente ya guardamos cada dato en las listas, el mandar llamar todo ya es más sencillo haciéndolo mediante la función previa. A su vez realizamos cambios en la estructura de nuestra función principal, es decir, la encargada de capturar y mostrar todos los procesos que se van ejecutando a lo largo del programa, mostrándose a continuación:

```
def main():
    global contProcess
    global procesosMax
    global procesos_actuales
    global proceso_ejecucion
    global procesos_terminados
    global contListos
    global contPendiente
    global contadorGeneral
    global procesos_bloqueados

    procesosMax = 5
    contProcess = 0
    contListos = 0
    contadorGeneral = 0

    procesos()
    procesos_terminados = []
    procesos_actuales = []
    procesos_bloqueados = []
    proceso_ejecucion = []

    while contProcess != numProcess:

        long1 = len(procesos_actuales)
        long2 = len(proceso_ejecucion)
        long3 = len(procesos_bloqueados)
        long = long1+long2+long3

        for _ in range(procesosMax-long):
            if not procesos_pendientes:
                break #rompe el programa

            proceso = procesos_pendientes.pop(0)
            proceso[7] = contadorGeneral
            procesos_actuales.append(proceso)
```

Ilustración 2. función main cambios 1

Programa 3. Algoritmo de Planificación FCFS

En esta primera parte de la función *main* podemos notar que, a comparación de las entregas anteriores, se eliminaron variables que para esta entrega nos complicaban la ejecución cuando estábamos manipulando los tiempos entre cada proceso, también dado que ya no se maneja por lotes, cambiamos la excepción en la iteración for para poder calcular la longitud de la lista de procesos nuevos o procesos pendientes, para posteriormente pasarlos a procesos listos o actuales.

```
if procesos_actuales:
    actual = procesos_actuales.pop(0)
    if actual[13] == 0:
        actual[13] = 1
        contResp = contadorGeneral-actual[7]
        actual[10] = contResp
    proceso_ejecucion.append(actual)

while True:
    contPendiente = len(procesos_pendientes)
    contListos = len(procesos_actuales)
    os.system("cls")
    if proceso_ejecucion:
        proceso_ejecucion[0][4] += 1
        imprimir()
        contadorGeneral += 1
        t.sleep(1)
        intoTecla()

    for bloqueados in procesos_bloqueados:
        bloqueado = bloqueados[6]

        if bloqueado < 8:
            bloqueados[6] += 1
            bloqueados[14] += 1

        elif bloqueado == 8:
            bloqueados[6] = 0
            bloqueados[14] += 1
            desbloqueado = procesos_bloqueados.pop(0)
            procesos_actuales.append(desbloqueado)
            if not proceso_ejecucion:
                actual = procesos_actuales.pop(0)
                proceso_ejecucion.append(actual)

    if proceso_ejecucion:
        if proceso_ejecucion[0][5] == 0:
            t.sleep(1)
            finalizado = proceso_ejecucion.pop(0)
            finalizado[12] = contadorGeneral
            finalizado[8] = finalizado[4]
            finalizado[11] = finalizado[12]-finalizado[7]
            finalizado[9] = finalizado[11]-finalizado[8]
            procesos_terminados.append(finalizado)
            contProcess += 1
            break
```

Ilustración 3. función main cambios 2

Posteriormente, realizamos cambios para ya no manejar con variables los tiempos, sino que trabajar directamente con el dato posicionado en la lista que estamos guardando constantemente como es el caso del tiempo restante y tiempo transcurrido, de esta manera evitamos caer en números negativos en caso de bloquear muchas veces un mismo proceso o en caso de que el proceso termine por interrupción *ERROR* que se guarde en lista el tiempo que le resta al proceso para poder terminar.

En adición, vemos que implementamos una iteración para un nuevo trabajo dentro del programa, que es la tabla de bloqueados, en esta tabla permanecerán los procesos que sean interrumpidos, un determinado tiempo, una vez que termine el lapso saldrán de esta lista y se integraran a la lista de procesos listos, todo se maneja por posiciones dentro de la lista.

```

        if tecla == 'i':
            if proceso_ejecucion:
                interrumpido = proceso_ejecucion.pop(0)
                interrumpido[6] = 1
                procesos_bloqueados.append(interrumpido)
            break

        elif tecla == 'e':
            if proceso_ejecucion:
                error = proceso_ejecucion.pop(0)
                error[2] = 'Error'
                error[8] = error[4]
                error[12] = contadorGeneral
                error[11] = error[12]-error[7]
                error[9] = error[11]-error[8]
                procesos_terminados.append(error)
                contProcess += 1
            break

    os.system("cls")
    imprimir()
    t.sleep(1)
    os.system("cls")
    impresion_final()
    #print("\nContador General:",contadorGeneral)
    mv.getch()

main()

```

Ilustración 4. función main cambios 3

Finalmente se presenta el algoritmo para las interrupciones, en este apartado solo se añadieron procesos dentro de la interrupción por error, para poder capturar los tiempos correspondientes, y al final imprimimos las tablas correspondientes limpiando en pantalla para reflejar mejor los cambios y que sea una vida limpia y eficiente para el usuario.

Programa 3. Algoritmo de Planificación FCFS

Un cambio más fue en la función de *procesos* que es la encargada de crear principalmente la lista de listas que captura cada información pertinente de cada proceso, añadiendo más valores a la lista e inicializándolos en 0, para posteriormente ser modificados en la ejecución del programa:

```
def procesos():
    global info
    global procesos_pendientes
    global totalProces
    global contTiempo
    global Tr

    contTiempo = 0
    Tr = 0
    tiempoBloq = 0
    tiempoLlegada = 0
    tiempoFinal = 0
    tiempoServicio = 0
    tiempoEspera = 0
    tiempoRetorno = 0
    tiempoRespuesta = 0
    bandera = 0
    tiempoTotalBloqueado = 0
    info = []
    procesos_pendientes = []

    os.system("cls")
    cantidad_procesos()
    totalProces = numProcess

    while totalProces != 0:
        os.system("cls")
        ID()
        isOperador()
        tiempo_estimado()
        info = [id, operaciones, resultado, TME, contTiempo, Tr, tiempoBloq, tiempoLlegada, tiempoServicio, tiempoEspera, tiempoRespuesta, tiempoRetorno, tiempoFinal,
                bandera, tiempoTotalBloqueado]
        procesos_pendientes.append(info)
        os.system("cls")
        totalProces -= 1
```

Ilustración 5. función *procesos*.

Funcionamiento

A continuación, se presentan algunas ilustraciones que reflejan el funcionamiento del programa de simulación con las mejoras:

1. Primero el programa pregunta los n procesos:

Dame un número de procesos: 12

Ilustración 6. Inicio del programa

2. Internamente se llenan los datos correspondientes y comienza el programa, evaluando que siempre sean 5 procesos entre la tabla de procesos listos, ejecutado y bloqueados:

No. Procesos Nuevos: 7		
Procesos Listos: 4		
ID	TME	TT
2	8	0
3	17	0
4	16	0
5	6	0
Proceso en Ejecución		
Id	1	
Ope	46%36	
TME	10	
TT	8	
TR	2	
Procesos Terminados		
ID	Ope	Res
Bloqueados		
ID	TB	
Contador: 7		

Ilustración 7. Procesos en tablas.

3. El programa es capaz de interrumpir el proceso por error, terminando con dicho proceso:

No. Procesos Nuevos: 5		
Procesos Listos: 4		
ID	TME	TT
4	16	0
5	6	0
6	7	0
7	7	0
Proceso en Ejecución		
Id	3	
Ope	53+26	
TME	17	
TT	3	
TR	14	
Procesos Terminados		
ID	Ope	Res
1	46%36	10
2	66%38	Error
Bloqueados		
ID	TB	
Contador: 17		

Ilustración 8. Interrupción por error

Programa 3. Algoritmo de Planificación FCFS

- También puede realizar una interrupción por lapso de 8 segundos, mandando el proceso a la tabla de bloqueados:

No. Procesos Nuevos: 5

ID	TME	TT
5	6	0
6	7	0
7	7	0

Id	4
Ope	30%4
TME	16
TT	7
TR	9

ID	Ope	Res
1	46%36	10
2	66%38	Error

ID	TB
3	7

Contador: 29

Ilustración 9. Interrupción a bloqueados.

- Una vez que termina el tiempo de bloqueados, el proceso se coloca a la cola de la tabla de procesos listos, se puede observar que la sumatoria de las 3 tablas (listos, ejecutado y bloqueados) siempre es 5:

No. Procesos Nuevos: 5

ID	TME	TT
7	7	0
3	17	8
4	16	15

Id	6
Ope	61%72
TME	7
TT	6
TR	1

ID	Ope	Res
1	46%36	10
2	66%38	Error

ID	TB
5	5

Contador: 46

Ilustración 10. Paso de bloqueados a listos

6. Al finalizar todos los procesos se muestra una tabla genérica con todos los tiempos capturados de dichos procesos:

Procesos

ID	Ope	Res	TME	TT	TR	TB	TL1	TS	TE	TRes	TRet	TF
1	46%36	10	10	10	0	0	0	10	0	0	10	10
2	66%38	Error	8	5	3	0	0	5	10	10	15	15
6	61%72	61	7	7	0	0	10	7	31	31	38	48
7	57%8	1	7	7	0	0	15	7	33	33	40	55
3	53+26	Error	17	12	5	8	0	12	47	15	59	59
4	30%4	2	16	16	0	8	0	16	44	23	60	60
5	94-31	63	6	6	0	8	0	6	58	38	64	64
10	98**26	Error	16	2	14	0	59	2	6	6	8	67
12	57-61	Error	11	10	1	0	64	10	7	7	17	81
8	45+24	69	10	10	0	8	48	10	32	12	42	90
9	43/20	2.15	12	12	0	8	55	12	34	9	46	101
11	10-38	-28	17	17	0	8	60	17	37	7	54	114

Ilustración 11. Datos genéricos por proceso

Errores y Soluciones

Tiempo de bloqueo en proceso interrumpido: En este caso más que mostrarse un mensaje de error por programar algo de forma incorrecta, lo que sucedía es que, a la hora de implementar la lógica de los procesos bloqueados, estos no estaban contando correctamente sus 8 segundos dentro de este estado, además, solamente su contador aumentaba a la hora de bloquear otro proceso. En adición, a la hora de pasar el proceso de ejecución a bloqueado, estaba llegando con un tiempo inicial transcurrido de bloqueo en '2', siendo que debería iniciar en '1'. Así como se muestra en la Ilustración 12.

No. Procesos Nuevos: 2		
Procesos Listos: 0		
Proceso en Ejecución		
Procesos Terminados		
Bloqueados		
Contador: 17		

Ilustración 12.Error 1

Esto se debía a falta de excepciones, ya que todo este proceso se encontraba dentro de las excepciones de realizar alguna acción cada que el usuario presionará una tecla. Así como se observa en la Ilustración 13.

```

334         for bloqueados in procesos_bloqueados:
335             bloqueado = bloqueados[6]
336
337             if bloqueado < 8:
338                 bloqueados[6] += 1
339
340             elif bloqueado == 8:
341                 bloqueado = 0
342                 desbloqueado = procesos_bloqueados.pop(0)
343                 procesos_actuales.append(desbloqueado)

```

Ilustración 13.Causa del Error 1

Al final para poder solucionar este error y que se contabilizará en tiempo real inicializando correctamente el contador de cada proceso bloqueado, se generó una sentencia 'if', para que esto se realizará solamente si existen procesos bloqueados, además de colocarlo antes de las excepciones cada que el usuario presionará una tecla. Quedando programado y resuelto, así como se muestra en la Ilustración 14.

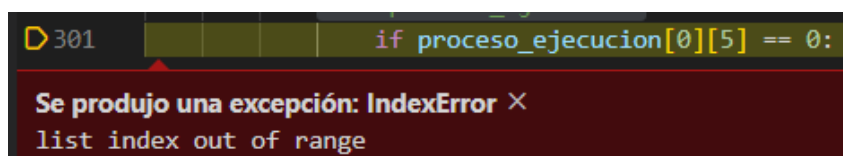
```

273         while True:
274             contPendiente = len(procesos_pendientes)
275             contListos = len(procesos_actuales)
276             os.system("cls")
277             if proceso_ejecucion:
278                 proceso_ejecucion[0][4] += 1
279             imprimir()
280             contadorGeneral += 1
281             t.sleep(1)
282             intoTecla()
283
284             for bloqueados in procesos_bloqueados:
285                 bloqueado = bloqueados[6]
286
287                 if bloqueado < 8:
288                     bloqueados[6] += 1
289                     bloqueados[14] += 1
290
291                 elif bloqueado == 8:
292                     bloqueados[6] = 0
293                     bloqueados[14] += 1
294                     desbloqueado = procesos_bloqueados.pop(0)
295                     procesos_actuales.append(desbloqueado)
296                     if not proceso_ejecucion:
297                         actual = procesos_actuales.pop(0)
298                         proceso_ejecucion.append(actual)
299
300             if proceso_ejecucion:

```

Ilustración 14. Solución Error 1

Datos fuera del rango de nuestras listas: Al implementar nuevas excepciones como las anteriores, nuestro programa se podía prestar para que existiera un caso que nunca habíamos considerado en nuestros programas anteriores, ya que, el estado de bloqueo no había sido incorporado. Es decir, que todos nuestros procesos se encuentren bloqueados, pero en procesos listos y en ejecución no existan procesos, siguiendo la regla de siempre tener máximo 5 procesos en el sistema. Así que, al solucionar el error 1, surgió el siguiente error a la hora de bloquear todos los procesos simultáneamente.



```

301         if proceso_ejecucion[0][5] == 0:

```

Se produjo una excepción: IndexError ×
list index out of range

Ilustración 15. Error 2

Así que, se podría decir que esto ocurría ya que siempre intuíamos que habría un proceso en ejecución, siendo que, viendo la lógica de los programas pasados, prácticamente el no tener procesos en ejecución significa que nuestro lote había finalizado o en su defecto todo el

Programa 3. Algoritmo de Planificación FCFS

programa. Siendo así, que nuestro error se estaba generando con el código mostrado en la Ilustración 16.

```
301         if proceso_ejecucion[0][5] == 0:
302             t.sleep(1)
303             finalizado = proceso_ejecucion.pop(0)
304             finalizado[12] = contadorGeneral
305             finalizado[8] = finalizado[4]
306             finalizado[11] = finalizado[12]-finalizado[7]
307             finalizado[9] = finalizado[11]-finalizado[8]
308             procesos_terminados.append(finalizado)
309             contProcess += 1
310             break
311
312         if tecla == 'i':
313             if proceso_ejecucion:
314                 interrumpido = proceso_ejecucion.pop(0)
315                 #interrumpido[4] = contTiempo
316                 interrumpido[6] = 1
317                 procesos_bloqueados.append(interrumpido)
318                 break
319
320         elif tecla == 'e':
321             if proceso_ejecucion:
322                 error = proceso_ejecucion.pop(0)
323                 error[2] = 'Error'
324                 # error[4] = proceso_ejecucion[
325                 error[8] = error[4]
326                 error[12] = contadorGeneral
327                 error[11] = error[12]-error[7]
328                 error[9] = error[11]-error[8]
329                 procesos_terminados.append(error)
330                 contProcess += 1
331                 break
332
```

Ilustración 16.Causa del Error 2

Así que teniendo en cuenta la nueva lógica del programa, donde siempre se tendría que estar validando si existe un proceso en ejecución, el error fue solucionado simplemente implementando una sentencia 'if' para verificar si había o no un proceso en ejecución, así como se muestra en la Ilustración 17.

```
300         if proceso_ejecucion:
301             if proceso_ejecucion[0][5] == 0:
302                 t.sleep(1)
303                 finalizado = proceso_ejecucion.pop(0)
304                 finalizado[12] = contadorGeneral
305                 finalizado[8] = finalizado[4]
306                 finalizado[11] = finalizado[12]-finalizado[7]
307                 finalizado[9] = finalizado[11]-finalizado[8]
308                 procesos_terminados.append(finalizado)
309                 contProcess += 1
310                 break
```

Ilustración 17. Solución Error 2

Conclusiones

Maria Dolores.

Tras los aprendizajes obtenidos en los ejercicios de algoritmos de planificación, fue sencillo realizar los cambios pertinentes en este programa, dado que teníamos clara la idea y forma de implementar las operaciones necesarias para sacar los tiempos de cada proceso que iba entrando en ejecución.

Los problemas que surgieron fueron debido a que en la versión del programa 2 teníamos ciertas variables que para implementar en este algoritmo nos dificultaban la manipulación de datos o nos arrojaban números negativos, por lo que fue necesario cambiar un poco la logística del programa y controlar los datos directos desde la posición en lista del valor correspondiente.

En adición, considero que es interesante como se manejan los tiempos por cada proceso que se ejecuta en el procesador, y que internamente se puede extraer dicha información y cómo funcionan las interrupciones por bloqueados, dándonos una visión de cuando en el sistema de repente por algún problema se pausa un proceso de alguna aplicación y se espera cierto tiempo hasta que el proceso se logra reanudar correctamente.

Finalmente, con el termino de este programa logramos desarrollar y automatizar el código; así como seguir aprendiendo a trabajar en equipo de la mano de mi compañero, siendo buen complemento para poder resolver juntos cada error que surge con cada modificación.

Aldo Fernando.

Realmente me atrevería a decir que pese a que ya había conversado con mi compañera como realizaríamos los cambios en este programa, fue muy sorprendente darme cuenta de que en realidad no habíamos considerado la parte que al generar un nuevo estado: bloqueado. Varias partes de nuestro programa podría dejar de funcionar en situaciones muy exactas, como fue el caos del Error 2, además, que necesitaríamos eliminar diversos contadores con variables globales, si no, haciendo dichos contadores dentro de cada proceso.

Sin embargo, me atrevería a decir que no fue tan complejo resolver dichos errores y problemas presentados, sin embargo, en varios de ellos tuvimos que hacer diversas ‘posibles soluciones para ver cuál era más efectiva y nos requiera hacer el menor numero de cambios posibles a todo el programa.

Así que, afortunadamente en esta ocasión nuevamente he logrado adquirir los conocimientos esperados y me parece que hemos logrado crear un tercer programa efectivo y sobre todo claro a la hora de leer y entender el código.