



Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías

Departamento de Ciencias Computacionales.
Ingeniería en Computación



Actividad 5.
Programa 2. Simular el Procesamiento por
Lotes con Multiprogramación

Materia: Sistemas Operativos

Sección: D04

Profesor: Becerra Velázquez Violeta del Rocío.

Equipo:

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

Fecha: 24/09/2023

Simular el Procesamiento por Lotes con Multiprogramación

Índice

Objetivos _____	4
Notas Sobre el Lenguaje _____	4
TDA o Estructuras Utilizadas en el Programa _____	5
Funcionamiento _____	6
Errores y Soluciones _____	10
Conclusiones _____	13

Tabla de Imágenes

Ilustración 1. Función nueva.....	5
Ilustración 2. Modificación en iteración while.	5
Ilustración 3. Captura de n procesos.....	6
Ilustración 4. Resultado en tablas.	6
Ilustración 5. Muestra de interrupción por error.....	6
Ilustración 6. Interrupción 1.....	7
Ilustración 7. Interrupción 2.....	7
Ilustración 8. Interrupción 3.....	8
Ilustración 9. Interrupción 4.....	8
Ilustración 10. Resultado final de los 5 lotes.	9
Ilustración 11. Causa del Error 1	10
Ilustración 12. Error 1 Tecla ‘Interrupción’	10
Ilustración 13. Error 1 Tecla ‘Error’	11
Ilustración 14. Solución 1	11
Ilustración 15. Error 2.....	12
Ilustración 16. Causa del Error 2	12
Ilustración 17. Solución Error 2.....	12

Objetivos

Anteriormente en el programa 1, se logró comprender e implementar el funcionamiento del procesamiento por lotes a grandes rasgos y de forma básica, es decir, donde entraban los procesos por lote (máximo cinco por lote), y que de forma secuencial conforme cada proceso llegaba a su tiempo estimado, finalizaba para que entrará el siguiente proceso en la fila de pendientes del lote actual.

Sin embargo, para este segundo programa lo que se busca es implementar interrupciones, errores y pausas, es decir:

1. Cuando el programa entre en estado de ‘pausa’, este se congelará junto con el contador, y solamente seguirá en ejecución cuando el usuario desea continuar con el mismo.
2. Cuando se este ejecutando un proceso, este ahora también podrá finalizar por error cuando el usuario lo indiqué, omitiendo su tiempo restante, para pasar directamente a lista de procesos terminados y que en el resultado se muestre un mensaje de ‘Error’.
3. Cuando un proceso se encuentre en ejecución, este podrá ser interrumpido cuando el usuario lo deseé, pasando de está manera con su tiempo transcurrido actual nuevamente a la fila de procesos pendientes del lote actual, esperando que sea su turno de ejecución. Es importante, tener en cuenta que cada proceso se podrá interrumpir cuantas veces queramos, sin embargo, una vez alcanzado correctamente el tiempo estimado, este pasará a la lista de finalizado como normalmente sucedía en el primer programa.

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.

Con las mejoras en el programa, se sumó una nueva función para la detección de teclas.

```
def intoTecla():
    global tecla
    tecla='q'
    if mv.kbhit():
        tecla = mv.getch().decode('utf-8')
        if tecla == 'p':
            os.system('cls')
            imprimir()
            print("Este programa se encuentra pausado...")
            while tecla != 'c':
                try:
                    tecla = mv.getch().decode('utf-8')
                    if tecla == 'c':
                        print("Reanudando Programa...")
                        t.sleep(1)
                        os.system('cls')
                except UnicodeDecodeError:
                    pass
```

Si el sistema detecta que el usuario presiona la tecla “p” pausará el programa y no importa que tecla presione el programa, solo podrá reanudarse una vez que presione la tecla “c”.

Ilustración 1. Función nueva.

```
while True:
    os.system('cls')
    contTiempo += 1
    contadorGeneral += 1
    imprimir()
    t.sleep(1)
    intoTecla()

    if tiempo_restante == 0:
        t.sleep(1)
        finalizado = proceso_actual.pop(0)
        procesos_terminados.append(finalizado)
        contProcess += 1
        break

    if tecla == 'i':
        interrumpido = proceso_actual.pop(0)
        interrumpido[4] = contTiempo
        lote_actual.append(interrumpido)
        break

    elif tecla == 'e':
        error = proceso_actual.pop(0)
        error[2] = 'Error'
        procesos_terminados.append(error)
        contProcess += 1
        break
```

Dentro de la función principal que es la encargada de mandar a imprimir los datos en las tablas por medio de la iteración *while*, se añadieron dos condicionales más para poder hacer que el programa detecte la tecla “i” y “e”, estas nos permitirán dar interrupciones a lo largo de la ejecución de los procesos por lotes, en caso de ser la “i”, el algoritmo hace que el proceso se vaya a la cola del lote actual.

En caso de que el usuario digite “e”, el proceso se ira a procesos terminados, con la leyenda de “ERROR” en el campo de resultado.

Ilustración 2. Modificación en iteración while.

Simular el Procesamiento por Lotes con Multiprogramación

Funcionamiento

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

1. Primero el programa pregunta las n veces que capturará datos:

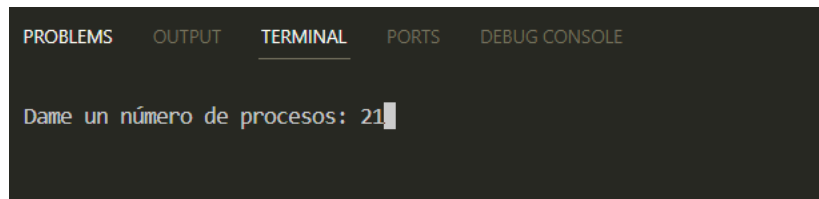


Ilustración 3. Captura de n procesos

2. Internamente se llenan los datos correspondientes y comienza el procesamiento por lotes con multiprogramación:



The screenshot displays the simulation state with the following data:

Lote Actual: 1		
ID	TME	TT
2	16	0
3	14	0
4	17	0
5	11	0

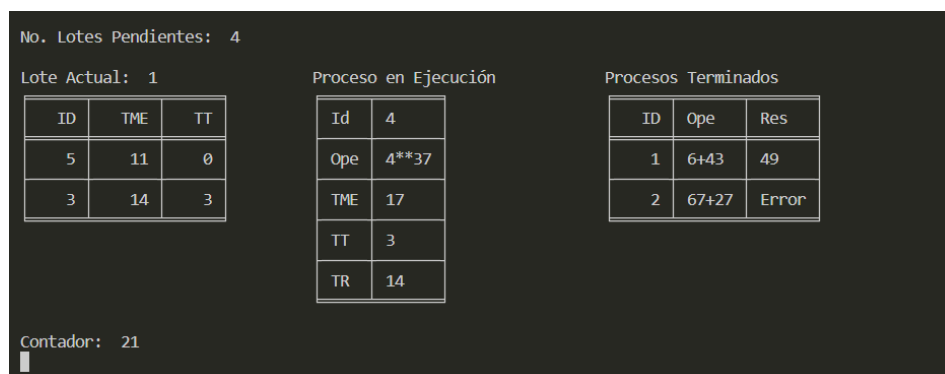
Proceso en Ejecución	
Id	1
Ope	6+43
TME	12
TT	2
TR	10

Procesos Terminados		
ID	Ope	Res

Contador: 2

Ilustración 4. Resultado en tablas.

3. Con la actualización el programa es capaz de interrumpir el proceso y que pueda terminar ya no solo porque el tiempo termino, sino también por producir un "ERROR" mostrando esta interrupción en la tabla de procesos terminados:



The screenshot displays the simulation state after an error interruption with the following data:

Lote Actual: 1		
ID	TME	TT
5	11	0
3	14	3

Proceso en Ejecución	
Id	4
Ope	4**37
TME	17
TT	3
TR	14

Procesos Terminados		
ID	Ope	Res
1	6+43	49
2	67+27	Error

Contador: 21

Ilustración 5. Muestra de interrupción por error.

4. Otra Interrupción en esta modificación es que el programa interrumpe un proceso, regresando el proceso actual a la cola del lote actual registrando el tiempo en que fue interrumpido y cuando vuelva a ejecutarse, reanudar el proceso a partir del tiempo transcurrido que tenía previamente:

No. Lotes Pendientes: 4

Lote Actual: 1

ID	TME	TT
4	17	15
5	11	2

Proceso en Ejecución

Id	3
Ope	11+47
TME	14
TT	7
TR	7

Procesos Terminados

ID	Ope	Res
1	6+43	49
2	67+27	Error

Contador: 39

Ilustración 6. Interrupción 1.

No. Lotes Pendientes: 3

Lote Actual: 2

ID	TME	TT
7	12	1
9	11	1

Proceso en Ejecución

Id	10
Ope	79%42
TME	17
TT	2
TR	15

Procesos Terminados

ID	Ope	Res
1	6+43	49
2	67+27	Error
3	11+47	58
4	4**37	Error
5	89%54	35
-	-	-
6	2**86	Error
8	83/77	Error

Contador: 71

Ilustración 7. Interrupción 2.

Simular el Procesamiento por Lotes con Multiprogramación

No. Lotes Pendientes: 3

Lote Actual: 2

ID	TME	TT
7	12	2

Proceso en Ejecución

Id	10
Ope	79%42
TME	17
TT	16
TR	1

Procesos Terminados

ID	Ope	Res
1	6+43	49
2	67+27	Error
3	11+47	58
4	4**37	Error
5	89%54	35
-	-	-
6	2**86	Error
8	83/77	Error
9	100-92	8

Contador: 96

Ilustración 8. Interrupción 3.

No. Lotes Pendientes: 2

Lote Actual: 3

ID	TME	TT
14	14	1
11	16	9

Proceso en Ejecución

Id	12
Ope	44+51
TME	9
TT	5
TR	4

Procesos Terminados

ID	Ope	Res
1	6+43	49
2	67+27	Error
3	11+47	58
4	4**37	Error
5	89%54	35
-	-	-
6	2**86	Error
8	83/77	Error
9	100-92	8
10	79%42	37
7	71*9	639
-	-	-
13	4+63	Error
15	48*68	Error

Contador: 127

Ilustración 9. Interrupción 4.

5. Al termino de todos los procesos se muestran en la tabla de terminados por cada lote que termina:

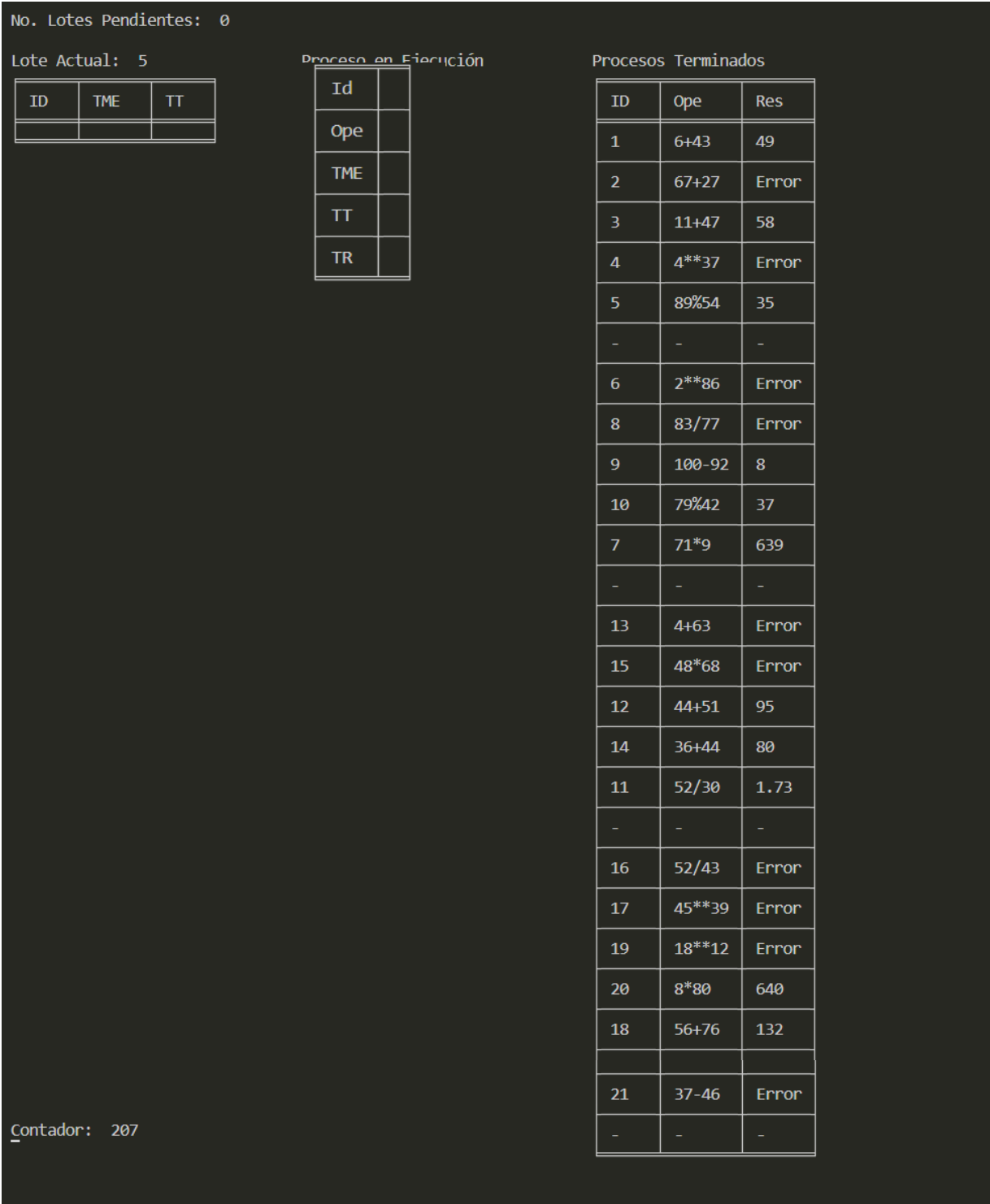


Ilustración 10. Resultado final de los 5 lotes.

Errores y Soluciones

Implementación de función para las teclas: Inicialmente para generar la funcionalidad deseada para este programa, optamos por crear una función donde se encontrará todas las excepciones para las teclas ‘p’, ‘c’, ‘i’ y ‘e’, como se puede observar en la Ilustración 11. Para mandarla a llamar en nuestro programa principal cada vez que se hacía la impresión de pantalla cada segundo transcurrido.

```

117 def intoTecla():
118     global tecla
119     tecla='q'
120     if mv.kbhit():
121         tecla = mv.getch().decode('utf-8')
122         if tecla == 'p':
123             os.system('cls')
124             imprimir()
125             print("Este programa se encuentra pausado...")
126             while tecla != 'c':
127                 try:
128                     tecla = mv.getch().decode('utf-8')
129                     if tecla == 'c':
130                         print("Reanudando Programa...")
131                         t.sleep(1)
132                         os.system('cls')
133                 except UnicodeDecodeError:
134                     pass
135
136         if tecla == 'i':
137             interrumpido = proceso_actual.pop(0)
138             interrumpido[4] = contTiempo
139             lote_actual.append(interrumpido)
140
141         elif tecla == 'e':
142             error = proceso_actual.pop(0)
143             error[2] = 'Error'
144             procesos_terminados.append(error)
145             contProcess += 1

```

Ilustración 11. Causa del Error 1

Sin embargo, al ejecutar el programa y probar la funcionalidad de las teclas, en el caso de ‘pausar’ y ‘continuar’ operaba correctamente, sin embargo, para las teclas de ‘interrupción’ y ‘error’, irónicamente nos generó errores, ya que, nos indicaba que al intentar acceder a lista del proceso actual y a la variable ‘contProcess’ no podía hacerlo correctamente, como se muestra en la Ilustración 12 y 13 respectivamente.

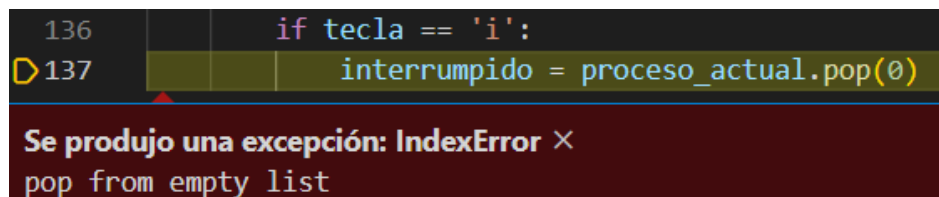


Ilustración 12. Error 1 Tecla ‘Interrupción’

```

141         elif tecla == 'e':
142             error = proceso_actual.pop(0)
143             error[2] = 'Error'
144             procesos_terminados.append(error)
145             contProcess += 1

```

Se produjo una excepción: UnboundLocalError ×
 local variable 'contProcess' referenced before assignment

Ilustración 13. Error 1 Tecla 'Error'

Así que, para solucionarlo simplemente dejamos dentro de esta función las teclas 'p' y 'c' que funcionaban correctamente, y las excepciones para error e interrupción, las eliminamos de esta función y las colocamos dentro del programa principal para que se hicieran las validaciones de que dichas teclas no fueron presionadas cada segundo procesado, y en caso de que sí hiciera la función esperada, pudiendo acceder y modificar sin problema las listas y variables necesarias, así como se expone en la Ilustración 14.

```

234     while True:
235         os.system("cls")
236         contTiempo += 1
237         contadorGeneral += 1
238         imprimir()
239         t.sleep(1)
240         intoTecla()
241
242         if tiempo_restante == 0:
243             t.sleep(1)
244             finalizado = proceso_actual.pop(0)
245             procesos_terminados.append(finalizado)
246             contProcess += 1
247             break
248
249         if tecla == 'i':
250             interrumpido = proceso_actual.pop(0)
251             interrumpido[4] = contTiempo
252             lote_actual.append(interrumpido)
253             break
254
255         elif tecla == 'e':
256             error = proceso_actual.pop(0)
257             error[2] = 'Error'
258             procesos_terminados.append(error)
259             contProcess += 1
260             break

```

Ilustración 14. Solución 1

Simular el Procesamiento por Lotes con Multiprogramación

Impresión en pantalla de todos los procesos terminados: Al tener programada la impresión de procesos pendientes con tabulaciones y solamente mostrar los terminados del lote actual, se contemplaba un máximo de 5 procesos en terminados a la vez, sin embargo, para este programa al eliminar la limpieza de nuestra lista de procesos terminados cada lote finalizado, a la hora de mostrar más de 5 procesos en pantalla, la tabla se imprimía de forma desperfecta.

No. Lotes Pendientes: 1

Lote Actual: 2

ID	TME	TT
9	7	0
10	16	0

Proceso en Ejecución

Id	8
Ope	76/77
TME	17
TT	2
TR	15

Procesos Terminados

ID	Ope	Res
1	9%50	9
2	3%90	Error
3	54%66	Error
4	38**61	Error
5	66/3	22.00
6	68*65	Error
7	86*91	Error

Contador: 34

Ilustración 15. Error 2

En adición, pudimos detectar que el error se encontraba en las tabulaciones a la hora de imprimir nuestras tablas finales en pantalla, así como se puede observar en la Ilustración 16.

```
181     for i in range(max_lines):
182         fila = lineas1[i] + "\t\t" + lineas2[i] + "\t\t" + lineas3[i]
183         tabla_final += fila + '\n'
```

Ilustración 16. Causa del Error 2

Resolviendo este problema sencillamente agregando una tabulación extra en nuestra variable 'fila' entre 'lineas2[i]' y 'lineas3[i]'. Tal como se observa en la Ilustración 17.

```
181     for i in range(max_lines):
182         fila = lineas1[i] + "\t\t" + lineas2[i] + "\t\t\t" + lineas3[i]
183         tabla_final += fila + '\n'
```

Ilustración 17. Solución Error 2

Conclusiones

Maria Dolores.

El procesamiento por lotes con multiprogramación nos permite realizar múltiples procesos al mismo tiempo, monitoreando el estado de dicho proceso y cambiando entre procesos cuando así se requiera. Esto nos lleva a la realización de este segundo programa en el cual podemos observar que se agregan interrupciones en el algoritmo; de esta manera mientras se ejecuta un proceso se puede producir un error y terminarlo o también se puede interrumpir provocando que el proceso se vuelva a regresar a la cola del lote.

Adicionalmente, para esta actividad solo fue necesario realizar algunas modificaciones al programa previo, por lo que, fue relativamente sencillo debido a que ya teníamos la lógica y una buena base para realizar lo solicitado en esta entrega.

En lo personal, dada la experiencia con el programa anterior, en esta ocasión organizarme con mi compañero no fue un problema, al contrario, considero que optimizamos muchísimo el tiempo, además de reducir el código para hacerlo más óptimo, logrando complementarnos en la realización del programa.

Aldo Fernando.

Tras los aprendizajes obtenidos en el primer programa y la práctica en la tarea de implementación Kbhit (Python en mi caso), siento que ha facilitado tener en claro cómo llevar a cabo las implementaciones requeridas para ese segundo programa de procesamiento por lotes con multiprogramación con sus respectivas excepciones de pausar, continuar, interrumpir y arrojar un resultado en error.

Por otro lado, fue interesante ver cómo puede tener un distinto funcionamiento unas líneas de código dependiendo si se encuentran dentro o fuera de una función anidada a otra función principal, como tal fue el caos del error 1. Sin embargo, gracias a ello puedo decir que me llevo los aprendizajes requeridos para este programa.

En adición, tras la experiencia de trabajar con mi compañera anteriormente para este programa, he notado que se ha logrado realizar las modificaciones correspondientes requeridas de una forma eficaz y eficiente, esperando siga siendo de esta forma para próximos programas.