

Programa 1 y Reporte 1

# **Simular el Proceso por Lotes**

---

Salvador Castañeda Andrade  
Sistemas Operativos

# Índice

<b>Índice.....</b>	<b>1</b>
<b>Introducción.....</b>	<b>2</b>
<b>Desarrollo.....</b>	<b>3</b>
Clase 'Process':.....	3
Función generate_data_process:.....	4
Función generate_operation_process:.....	4
Función generate_data_process:.....	5
Función generate_operation_process:.....	5
Función generate_process_txt:.....	6
Función clear_text_boxes:.....	7
Función update_GUI_process_lbl_txt_box:.....	7
Función update_running_process_text_box:.....	8
Función update_on_hold_process_text_box:.....	8
Función update_finished_process_text_box:.....	9
Función update_pending_batches_label:.....	9
Función calculate_current_batch_number:.....	10
Función calculate_current_batch_processes:.....	10
Función last_number_process_batch:.....	11
Función calculate_number_of_batches:.....	12
Función start_clock:.....	12
Función update_elapsed_time:.....	13
Función running_process:.....	13
Función generate_process:.....	14
Función generate_result_txt:.....	15
Función create_main_window:.....	16
<b>Conclusión.....</b>	<b>18</b>



## Introducción

El simulador de procesamiento por lotes es una herramienta diseñada en Python con la interfaz gráfica tkinter. Este programa simula el procesamiento de múltiples procesos en lotes, mostrando el estado de los procesos en ejecución, en espera y finalizados en tiempo real.

El código fuente proporciona una estructura modular y fácilmente comprensible, lo que permite a los desarrolladores entender y modificar el programa según sea necesario. A través de este simulador, los usuarios pueden experimentar y comprender mejor los conceptos de procesamiento por lotes y la gestión de procesos en un entorno controlado y visualmente intuitivo.

El simulador utiliza un conjunto de funciones para generar procesos aleatorios con diferentes operaciones matemáticas y tiempos de ejecución. Además, incluye funciones para actualizar la interfaz gráfica en función del estado de los procesos y para generar informes de resultados una vez que los procesos han sido completados.

## Desarrollo

### Clase 'Process':

Python

```
class Process:

    process_counter = 1

    def __init__(self, name, data, max_time, operation):

        self.process_number = Process.process_counter

        Process.process_counter += 1

        self.name = name

        self.data = data

        self.max_time = int(max_time)

        self.operation = operation

        self.result = None

    def __str__(self):

        return f"{self.process_number}. {self.name}\n{self.data}\nTME: {self.max_time}\n"
```

- Esta clase define un objeto **Process**, que representa un proceso en el sistema.
- Tiene los siguientes atributos:
  - **process\_number**: El número del proceso, que se incrementa cada vez que se crea un nuevo proceso.
  - **name**: El nombre del programador asignado al proceso.
  - **data**: Los datos del proceso, que pueden ser una operación matemática.
  - **max\_time**: El tiempo máximo de ejecución del proceso.
  - **operation**: La operación matemática a realizar.
  - **result**: El resultado de la operación, inicializado como **None**.
- El método **\_\_str\_\_** devuelve una representación legible del proceso, incluyendo su número, nombre del programador, datos y tiempo máximo de ejecución.

## Función generate\_data\_process:

Python

```
def generate_data_process():  
    name = random.choice(programmers)  
    max_time = random.randint(6, 12)  
    return name, max_time
```

- Esta función genera aleatoriamente un nombre de programador y un tiempo máximo de ejecución para un proceso.
- Utiliza la biblioteca **random** de Python para seleccionar aleatoriamente un nombre de la lista **programmers** y un número entre 6 y 12 para el tiempo máximo de ejecución.

## Función generate\_operation\_process:

Python

```
def generate_operation_process():  
    operation = random.choice(operations)  
    num_A = random.randint(0, 10)  
    num_B = random.randint(0, 10)  
    if operation == "/" and num_B == 0:  
        num_B = random.randint(1, 10)  
    data = f"{str(num_A)} {operation} {str(num_B)}"  
    result = eval(data)  
    return operation, data, result
```

- Esta función genera una operación matemática aleatoria, junto con sus operandos y su resultado.
- Selecciona aleatoriamente una operación de la lista **operations**.
- Genera dos números enteros aleatorios entre 0 y 10 para ser los operandos de la operación.

- Verifica si la operación es división ("/") y el segundo operando es 0. En ese caso, genera un segundo operando aleatorio entre 1 y 10 para evitar divisiones por cero.
- Crea una cadena de texto **data** que representa la operación.
- Utiliza la función **eval()** para calcular el resultado de la operación.
- Devuelve la operación, los datos y el resultado.

### Función generate\_data\_process:

Python

```
def generate_data_process():
    name = random.choice(programmers)
    max_time = random.randint(6, 12)
    return name, max_time
```

- Esta función genera de forma aleatoria un nombre de programador y un tiempo máximo de ejecución para un proceso.
- Utiliza la función **random.choice** para seleccionar un nombre aleatorio de la lista **programmers**.
- Utiliza la función **random.randint** para generar un tiempo máximo de ejecución entre 6 y 12 segundos.
- Devuelve una tupla con el nombre del programador y el tiempo máximo de ejecución.

### Función generate\_operation\_process:

Python

```
def generate_operation_process():
    operation = random.choice(operations)
    num_A = random.randint(0, 10)
    num_B = random.randint(0, 10)
    if operation == "/" and num_B == 0:
        num_B = random.randint(1, 10)
    data = f"{str(num_A)} {operation} {str(num_B)}"
```

```
result = eval(data)

return operation, data, result
```

- Esta función genera de forma aleatoria una operación matemática, calcula su resultado y devuelve la operación, los datos de la operación y el resultado.
- Utiliza la función **random.choice** para seleccionar una operación aleatoria de la lista **operations**.
- Utiliza la función **random.randint** para generar dos números aleatorios entre 0 y 10 para realizar la operación.
- Verifica si la operación es una división y el segundo número es 0, en cuyo caso genera un nuevo segundo número entre 1 y 10 para evitar la división por cero.
- Crea una cadena de texto **data** que representa la operación matemática.
- Utiliza la función **eval** para evaluar la operación y obtener el resultado.
- Devuelve una tupla con la operación, los datos de la operación y el resultado.

### Función generate\_process\_txt:

Python

```
def generate_process_txt(processes):

    with open("data.txt", "w") as file:

        current_batch = 1

        for i, process in enumerate(processes, start=1):

            if i % 5 == 1:

                file.write(f"Batch {current_batch}\n")

                current_batch += 1

            file.write(f"{i}. {process}\n")

            file.write("\n")

        file.write("\n")
```

- Esta función genera un archivo de texto llamado "data.txt" que contiene la lista de procesos pendientes.

- Utiliza un bucle **for** para iterar sobre la lista de procesos.
- Verifica si el índice del proceso es múltiplo de 5 para determinar si se debe iniciar un nuevo lote.
- Escribe en el archivo el número del proceso, el nombre del programador, los datos del proceso y el tiempo máximo de ejecución.
- Al finalizar, escribe un salto de línea para separar los lotes en el archivo.

### Función `clear_text_boxes`:

Python

```
def clear_text_boxes():
    text_box_running.delete("1.0", tk.END)
    text_box_on_hold.delete("1.0", tk.END)
    text_box_finished.delete("1.0", tk.END)
```

- Esta función borra el contenido de los cuadros de texto de la interfaz gráfica para los procesos en ejecución, en espera y finalizados.

### Función `update_GUI_process_lbl_txt_box`:

Python

```
def update_GUI_process_lbl_txt_box():
    update_running_process_text_box()
    update_on_hold_process_text_box()
    update_finished_process_text_box()
    update_pending_batches_label()
    lbl_global_clock.after(1000, update_elapsed_time)
```

- Esta función actualiza todos los elementos de la interfaz gráfica relacionados con los procesos, incluyendo los cuadros de texto y las etiquetas.
- Llama a otras funciones para actualizar los cuadros de texto de los procesos en ejecución, en espera y finalizados, así como la etiqueta con el número de lotes pendientes.



- Utiliza `lbl_global_clock.after(1000, update_elapsed_time)` para actualizar el tiempo transcurrido cada segundo.

### Función `update_running_process_text_box`:

Python

```
def update_running_process_text_box():
    text_box_running.delete("1.0", tk.END)

    if current_process:
        text_box_running.insert("1.0", f"{current_process}\n")
```

- Esta función actualiza el cuadro de texto de la interfaz gráfica para mostrar el proceso en ejecución.
- Borra el contenido actual del cuadro de texto.
- Si hay un proceso en ejecución (**current\_process**), inserta la información del proceso en el cuadro de texto.

### Función `update_on_hold_process_text_box`:

Python

```
def update_on_hold_process_text_box():
    text_box_on_hold.delete("1.0", tk.END)

    if pending_processes:
        next_process = pending_processes[0]

        text_box_on_hold.insert(tk.END, f"{next_process}\n")

        current_batch = calculate_current_batch_number(current_process_number)
        current_batch_processes = calculate_current_batch_processes()

        text_box_on_hold.insert(tk.END, f"Pending processes:
{current_batch_processes}")
```

- Esta función actualiza el cuadro de texto de la interfaz gráfica para mostrar los procesos en espera.
- Borra el contenido actual del cuadro de texto.

- Si hay procesos en espera (**pending\_processes**), muestra el próximo proceso en la lista de procesos en espera.
- Calcula el número de procesos en el lote actual y lo muestra en el cuadro de texto.

### Función `update_finished_process_text_box`:

Python

```
def update_finished_process_text_box():
    text_box_finished.delete("1.0", tk.END)

    global finished_processes

    text_box_finished.delete("1.0", tk.END)

    for i, process in enumerate(finished_processes, start=1):
        result = eval(process.data)

        text_box_finished.insert(tk.END, f"{i}. {process.name}\n{process.data} = {result}\n\n")
```

- Esta función actualiza el cuadro de texto de la interfaz gráfica para mostrar los procesos finalizados.
- Borra el contenido actual del cuadro de texto.
- Utiliza un bucle **for** para iterar sobre la lista de procesos finalizados (**finished\_processes**) y muestra el nombre del programador, los datos del proceso y el resultado de la operación.

### Función `update_pending_batches_label`:

Python

```
def update_pending_batches_label():
    T_batches = calculate_number_of_batches()

    current_batch = calculate_current_batch_number(current_process_number)

    n_batches = T_batches - current_batch

    if n_batches == 0:
```

```

        text = "Number of pending batches: 0"
    else:
        text = f"Number of pending batches: {n_batches}"
    lbl_pending_batches.config(text=text)

```

- Esta función actualiza la etiqueta que muestra el número de lotes pendientes.
- Calcula el número total de lotes (**T\_batches**), el número de lote actual (**current\_batch**) y el número de lotes pendientes (**n\_batches**).
- Actualiza el texto de la etiqueta según el número de lotes pendientes.

### Función `calculate_current_batch_number`:

Python

```

def calculate_current_batch_number(current_process_number):
    current_batch = math.ceil(current_process_number / 5)
    return current_batch

```

- Esta función calcula el número de lote actual en base al número de proceso actual.
- Utiliza la función **math.ceil** para redondear hacia arriba el resultado de la división del número de proceso actual entre 5, ya que cada lote contiene 5 procesos.

### Función `calculate_current_batch_processes`:

Python

```

def calculate_current_batch_processes():
    current_batch = 0
    number_batches = calculate_number_of_batches()
    n_processes = int(entry_num_processes.get())
    current_batch = calculate_current_batch_number(current_process_number)
    complement_number_process_batch = last_number_process_batch()

```

```

if n_processes % 5 == 0:
    current_batch_processes = ((current_process_number - (current_batch * 5)) * -1)
    return current_batch_processes
elif n_processes % 5 != 0:
    current_batch_processes = ((current_process_number - (current_batch * 5)) * -1)
    if current_batch == number_batches:
        current_batch_processes = ((current_process_number - (current_batch * 5)) * -1)
        - (complement_number_process_batch)
    return current_batch_processes

```

- Esta función calcula el número de procesos en el lote actual.
- Utiliza el número total de procesos (**n\_processes**), el número de lotes (**number\_batches**), el número de proceso actual (**current\_process\_number**) y el número de procesos en el último lote incompleto (**complement\_number\_process\_batch**) para determinar cuántos procesos quedan en el lote actual.

### Función last\_number\_process\_batch:

```

C/C++

def last_number_process_batch():
    n_processes = int(entry_num_processes.get())
    n_batches = calculate_number_of_batches()
    number_process_batch = ((((((n_processes - (n_batches * 5)) * -1) - 5) * -1) - 5) *
-1))
    return number_process_batch

```

- Esta función calcula el número de procesos en el último lote.
- Utiliza el número total de procesos (**n\_processes**) y el número de lotes (**n\_batches**) para determinar cuántos procesos quedan en el último lote.

## Función `calculate_number_of_batches`:

Python

```
def calculate_number_of_batches():  
    number_batches = 0  
  
    n_processes = int(entry_num_processes.get())  
  
    number_batches = math.ceil(n_processes / 5)  
  
    return number_batches
```

- Esta función calcula el número total de lotes.
- Utiliza el número total de procesos (**n\_processes**) y la división entera para determinar cuántos lotes se necesitan, redondeando hacia arriba con **math.ceil** ya que cada lote contiene 5 procesos.

## Función `start_clock`:

Python

```
def start_clock():  
    global start_time, elapsed_seconds  
  
    start_time = datetime.now()  
  
    elapsed_seconds = 0  
  
    update_elapsed_time()
```

- Esta función inicia el contador de tiempo global (**elapsed\_seconds**) y guarda la fecha y hora de inicio en **start\_time**.
- Llama a la función **update\_elapsed\_time** para comenzar a actualizar el tiempo transcurrido.

## Función update\_elapsed\_time:

Python

```
def update_elapsed_time():  
    global elapsed_seconds, start_time  
    if start_time:  
        elapsed_time = datetime.now() - start_time  
        elapsed_seconds = elapsed_time.seconds  
    else:  
        elapsed_seconds = 0  
    lbl_global_clock.config(text=f"Global Clock: {elapsed_seconds} seconds")  
    lbl_global_clock.after(1000, update_elapsed_time)
```

- Esta función actualiza el tiempo transcurrido en la etiqueta de la GUI que muestra el reloj global.
- Calcula la diferencia entre el tiempo actual y el tiempo de inicio para obtener el tiempo transcurrido en segundos.
- Actualiza la etiqueta con el tiempo transcurrido y utiliza **after** para programar la actualización cada segundo.

## Función running\_process:

Python

```
def running_process():  
    global pending_processes, current_process, finished_processes,  
    current_process_number  
    if not current_process and pending_processes:  
        current_process = pending_processes.pop(0)  
        current_process_number = current_process.process_number  
    if current_process:
```

```

if current_process.max_time > 0:
    current_process.max_time -= 1
else:
    finished_processes.append(current_process)
    current_process = None
update_GUI_process_lbl_txt_box()
lbl_global_clock.after(1000, running_process)

```

- Esta función simula el proceso de ejecución de los procesos.
- Verifica si no hay un proceso en ejecución (**current\_process**) y si hay procesos pendientes (**pending\_processes**), y luego saca el próximo proceso de la lista y lo asigna como el proceso actual.
- Si hay un proceso en ejecución, reduce el tiempo restante de ejecución en 1 segundo. Si el tiempo restante llega a 0, el proceso se mueve a la lista de procesos finalizados (**finished\_processes**).
- Llama a la función **update\_GUI\_process\_lbl\_txt\_box** para actualizar la interfaz gráfica.
- Utiliza **after** para programar la función para que se ejecute cada segundo.

## Función generate\_process:

Python

```

def generate_process():
    global pending_processes, finished_processes
    n_processes = int(entry_num_processes.get())
    pending_processes = []
    finished_processes = []
    Process.process_counter = 1
    for i in range(n_processes):
        name, max_time = generate_data_process()

```

```

    operation, data, result = generate_operation_process()

    p = Process(name, data, max_time, operation)

    p.result = result

    pending_processes.append(p)

clear_text_boxes()

generate_process_txt(pending_processes)

update_GUI_process_lbl_txt_box()

start_clock()

lbl_global_clock.after(1000, running_process)

```

- Esta función genera la lista de procesos pendientes.
- Obtiene el número total de procesos desde la entrada de la interfaz gráfica.
- Reinicia las listas **pending\_processes** y **finished\_processes** y el contador de procesos (**Process.process\_counter**).
- Utiliza un bucle **for** para generar cada proceso y añadirlo a la lista de procesos pendientes.
- Llama a las funciones **clear\_text\_boxes**, **generate\_process\_txt**, **update\_GUI\_process\_lbl\_txt\_box** y **start\_clock** para actualizar la interfaz gráfica y comenzar el contador de tiempo global.

### Función generate\_result\_txt:

Python

```

def generate_result_txt():

    global finished_processes

    if finished_processes:

        with open("results.txt", "w") as file:

            current_batch = 1

            for i, process in enumerate(finished_processes, start=1):

```



```

if i % 5 == 1 and i != 1:
    file.write("\n")

if i % 5 == 1:
    file.write(f"Batch {current_batch}\n")
    current_batch += 1

file.write(f"{i}. {process.name}\n{process.data} = {process.result}\n")
file.write("\n")

file.write("\n")

```

- Esta función genera un archivo de texto (**results.txt**) que contiene los resultados de los procesos finalizados.
- Itera sobre la lista de procesos finalizados y escribe en el archivo el nombre del programador, los datos de la operación y el resultado de cada proceso, agrupados por lotes.

## Función create\_main\_window:

Python

```

def create_main_window():

    global entry_num_processes, text_box_running, text_box_on_hold,
    lbl_pending_batches, lbl_finished, lbl_global_clock, text_box_finished

    window = tk.Tk()


    window.title("Batch Processing")

    window.geometry("700x400")

    # Código de configuración de la interfaz gráfica omitido por brevedad

    window.mainloop()

```

- 
- Esta función crea la ventana principal de la interfaz gráfica.
  - Define la estructura y disposición de los elementos de la interfaz gráfica, como etiquetas, cuadros de texto y botones.
  - Configura el título de la ventana, su tamaño y llama al método **mainloop** para iniciar el bucle principal de la interfaz gráfica.



## Conclusión

En resumen, el simulador de procesamiento por lotes proporciona una herramienta efectiva para entender y visualizar el funcionamiento del procesamiento por lotes en un entorno controlado. A través de su interfaz gráfica intuitiva y sus funciones modulares, los usuarios pueden experimentar con diferentes configuraciones de procesos y comprender mejor los conceptos detrás de este tipo de procesamiento.

El código fuente está diseñado de manera que sea fácil de entender y modificar, lo que permite a los desarrolladores adaptar el simulador según sus necesidades específicas. Además, el programa incluye funciones para generar informes de resultados, lo que lo hace útil para analizar el rendimiento y la eficiencia de los procesos simulados.

En conclusión, el simulador de procesamiento por lotes es una herramienta valiosa para estudiantes, académicos y profesionales que deseen explorar y experimentar con los principios del procesamiento por lotes de manera práctica y educativa.