Computación Tolerante a Fallas

Sección D06

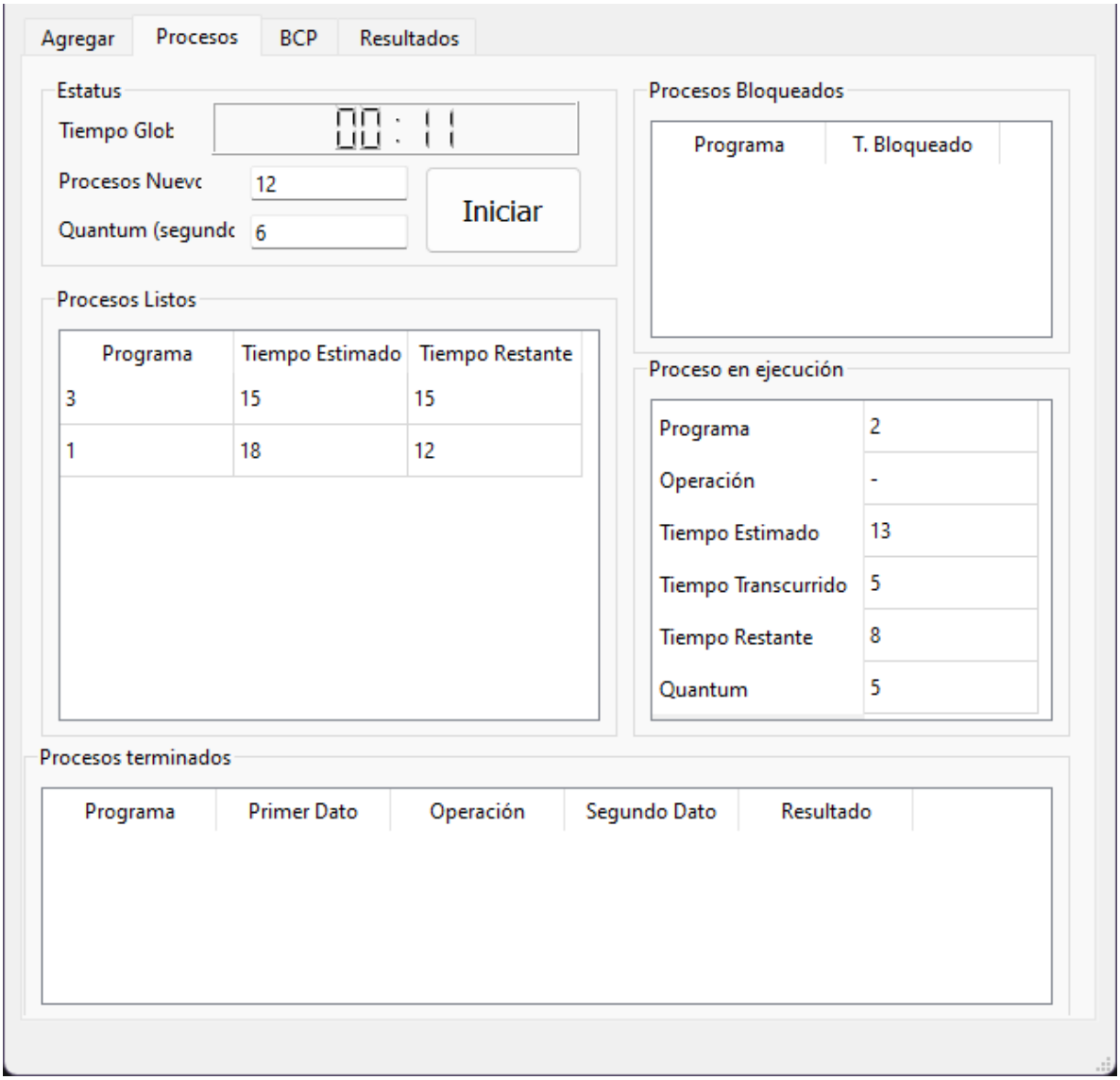
An Introduction to Scaling Distributed Python Applications

Fernández Venegas David Guadalupe

Código: 216437352

Profesor: López Franco Michel Emanuel

Para esta actividad se realizó un programa para un simulador de un planificador, el cual está hecho para poder observar con detalle el funcionamiento de un planificador con el algoritmo Round Robin. Dicho programa cuenta con una interfaz gráfica desarrollada con Qt para poder observar el progreso y la rotación de los procesos en el planificador.



Para el uso de este programa hace falta el uso de varios hilos debido a que mientras cada proceso va siendo ejecutado, necesitamos observar mediante la interfaz gráfica dicho proceso. No solo eso, si no que el programa también maneja entradas de teclado para realizar diversas acciones durante la simulación (agregar un proceso nuevo, pausar y reanudar la simulación, mostrar tabla de datos, interrumpir un proceso, etc).

Durante este programa, mientras el algoritmo trabaja con los procesos al mismo tiempo actualiza los campos de la interfaz gráfica para observar los cambios que están sucediendo. Sin embargo, sin el uso de hilos, la interfaz solo se actualizaría hasta el final de la simulación. Para evitar esto, hacemos uso de la siguiente clase:

```
class SimulacionThread(QThread):
    finalizada = Signal()

    def __init__(self, simulador):
        super().__init__()
        self.simulador = simulador

    def run(self):
        self.simulador.ejecutar()
        self.finalizada.emit()
```

Esta clase se hereda de QThread para representar el hilo que ejecuta la simulación. Con el método init se inicializa el hilo junto con la ejecución de la simulación. El método run es llamado cuando se inicia el hilo. En este método, se ejecuta la simulación a través del simulador y luego se emite la señal finalizada al terminar.

De igual manera, se declara otra clase la cual representa un hilo que escucha las teclas presionadas en el teclado.

```
class TecladoThread(QThread):
    tecla_presionada = Signal(str)
    finalizada = Signal()

    def __init__(self):
        super().__init__()

    def run(self):
        keyboard.on_press(self.manejar_tecla_presionada)
        self.exec_()

    def manejar_tecla_presionada(self, event):
        key = event.name
        if key in ('e','w','p','c','n','b'):
            self.tecla_presionada.emit(key)

    @Slot()
    def detener(self):
        self.quit()
```

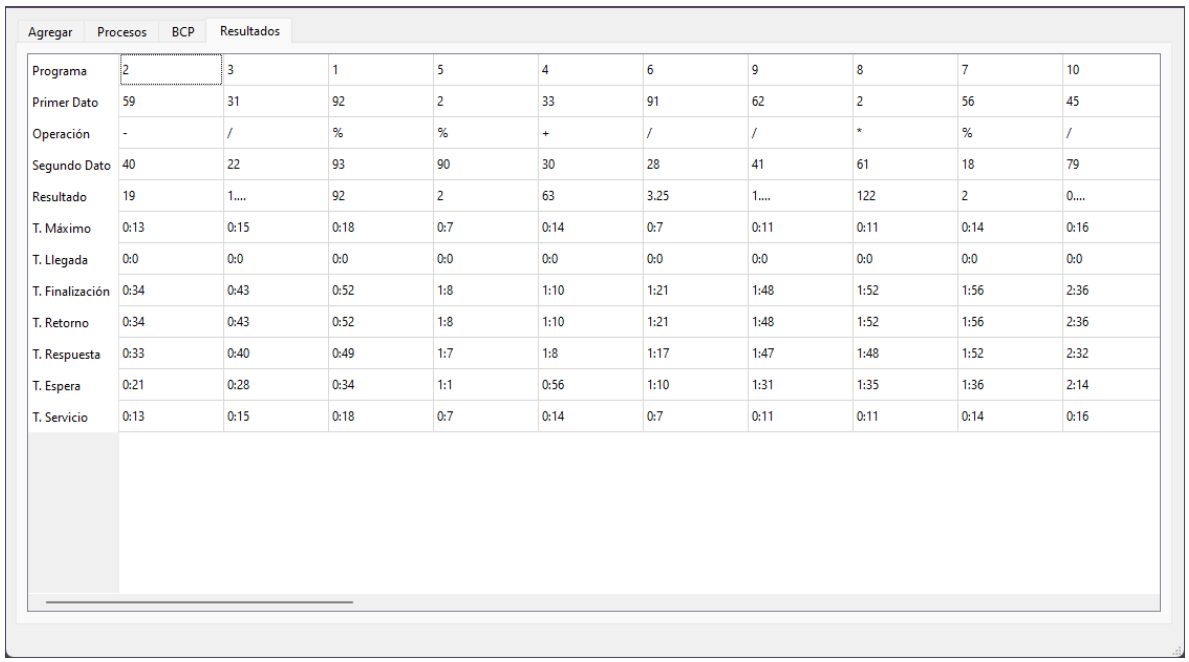
Teniendo así dos hilos: uno para ejecutar una simulación y otro para escuchar las teclas presionadas en el teclado. Cada hilo ejecuta su tarea en paralelo y emite señales para comunicar su progreso o eventos importantes.

Ambos hilos se inicializan y se corren al momento de presionar el botón de inicio dentro de la interfaz gráfica, permitiéndonos de esta forma la sincronización en el programa.

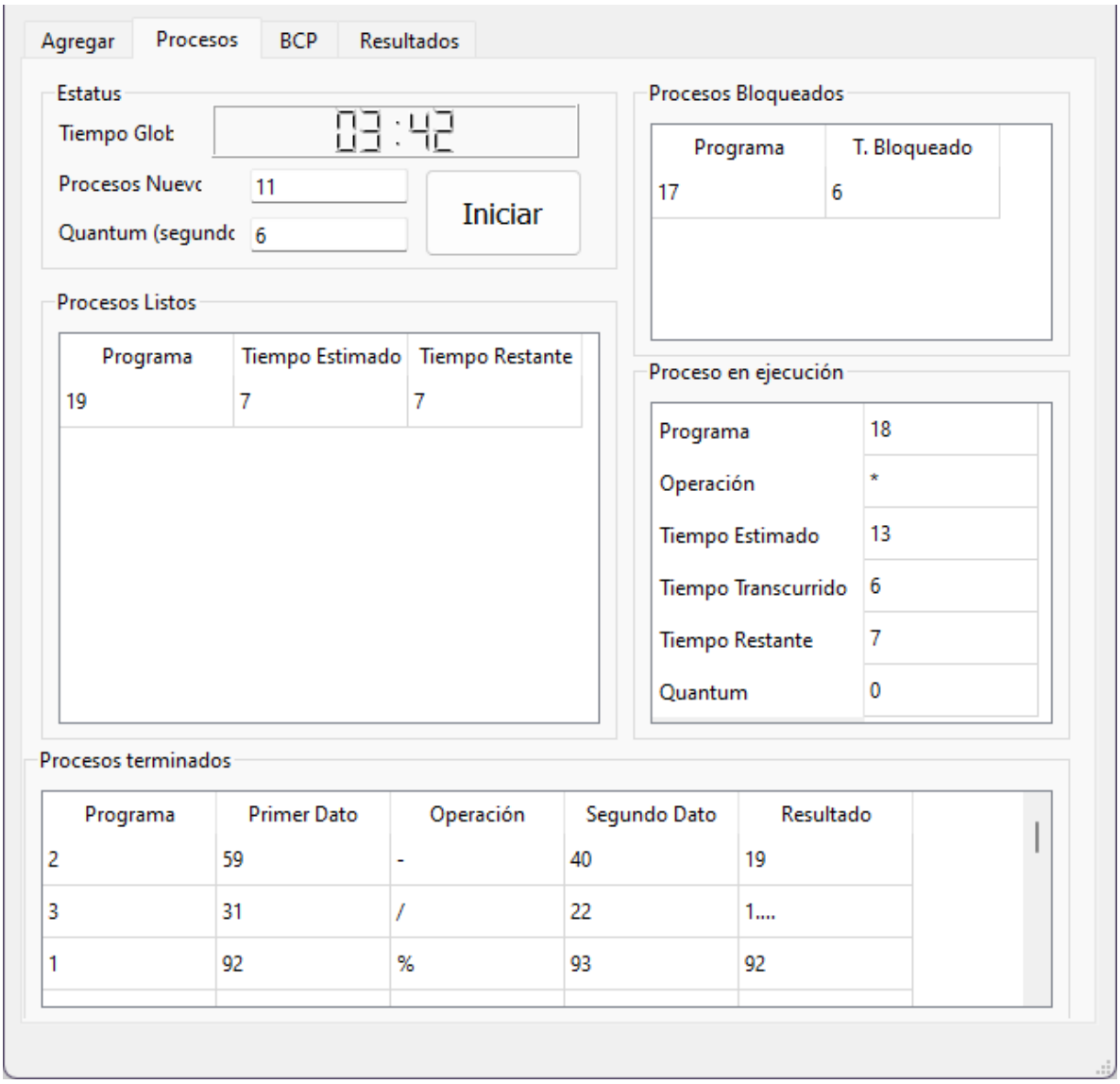
```
    @Slot()
    def iniciar(self):
        if not self.simulacion_thread or not
self.simulacion_thread.isRunning():
            self.simulacion_thread = SimulacionThread(self.planificador)
            self.simulacion_thread.finalizada.connect(self.simulacion_finali
zada)
            self.simulacion_thread.start()

        if not self.teclado_thread or not self.teclado_thread.isRunning():
            self.teclado_thread = TecladoThread()
            self.teclado_thread.tecla_presionada.connect(self.actualizar_tec
la_presionada)
            self.teclado_thread.start()
```

Con esto, además de poder visualizar el progreso de nuestro planificador, podemos hacer uso del programa para interrumpir procesos o ver la tabla de progreso.



Programa	2	3	1	5	4	6	9	8	7	10
Primer Dato	59	31	92	2	33	91	62	2	56	45
Operación	-	/	%	%	+	/	/	*	%	/
Segundo Dato	40	22	93	90	30	28	41	61	18	79
Resultado	19	1....	92	2	63	3.25	1....	122	2	0....
T. Máximo	0:13	0:15	0:18	0:7	0:14	0:7	0:11	0:11	0:14	0:16
T. Llegada	0:0	0:0	0:0	0:0	0:0	0:0	0:0	0:0	0:0	0:0
T. Finalización	0:34	0:43	0:52	1:8	1:10	1:21	1:48	1:52	1:56	2:36
T. Retorno	0:34	0:43	0:52	1:8	1:10	1:21	1:48	1:52	1:56	2:36
T. Respuesta	0:33	0:40	0:49	1:7	1:8	1:17	1:47	1:48	1:52	2:32
T. Espera	0:21	0:28	0:34	1:1	0:56	1:10	1:31	1:35	1:36	2:14
T. Servicio	0:13	0:15	0:18	0:7	0:14	0:7	0:11	0:11	0:14	0:16


Agregar
Procesos
BCP
Resultados
Estatus
Tiempo Glob
03 : 42
Procesos Nuevc
11
Quantum (segundc
6
Iniciar
Procesos Bloqueados
Programa
T. Bloqueado
17
6
Procesos Listos
Programa
Tiempo Estimado
Tiempo Restante
19
7
7
Proceso en ejecución
Programa
18
Operación
*
Tiempo Estimado
13
Tiempo Transcurrido
6
Tiempo Restante
7
Quantum
0
Procesos terminados
Programa
Primer Dato
Operación
Segundo Dato
Resultado
2
59
-
40
19
3
31
/
22
1....
1
92
%
93
92

Conclusión:

El uso de hilos en la programación es una técnica fundamental para mejorar la eficiencia y la capacidad de respuesta de las aplicaciones, especialmente en situaciones donde se requiere realizar múltiples tareas concurrentemente. Al emplear hilos, es posible ejecutar diferentes partes del código de manera independiente y paralela, lo que permite aprovechar al máximo los recursos del sistema y mejorar el rendimiento general de la aplicación.

Esta actividad además ha sido útil para comprender la diferencia entre los procesos y los hilos: Un proceso es una instancia independiente de un programa en ejecución que tiene su propio espacio de memoria, mientras que un hilo es una unidad de ejecución más pequeña que existe dentro de un proceso y comparte el mismo espacio de memoria. Los procesos son más pesados en términos de recursos del sistema, ya que requieren su propio espacio de memoria, mientras que los hilos son más livianos y comparten recursos con otros hilos dentro del mismo proceso.