

**Centro Universitario de Ciencias
Exactas e Ingenierías**

Universidad de Guadalajara

REPORTE 02

Simulador Procesamiento por Lotes con Multiprogramación

Alumnos:

Carbajal Armenta Yessenia Paola
Sánchez Lozano Jonathan

Códigos:

220286482
215768126

Profesora:

Becerra Velázquez Violeta del Rocío

Materia:

Seminario de Soluciones de
Problemas de Sistemas Operativos

Departamento:

Ciencias Computacionales

Carrera:

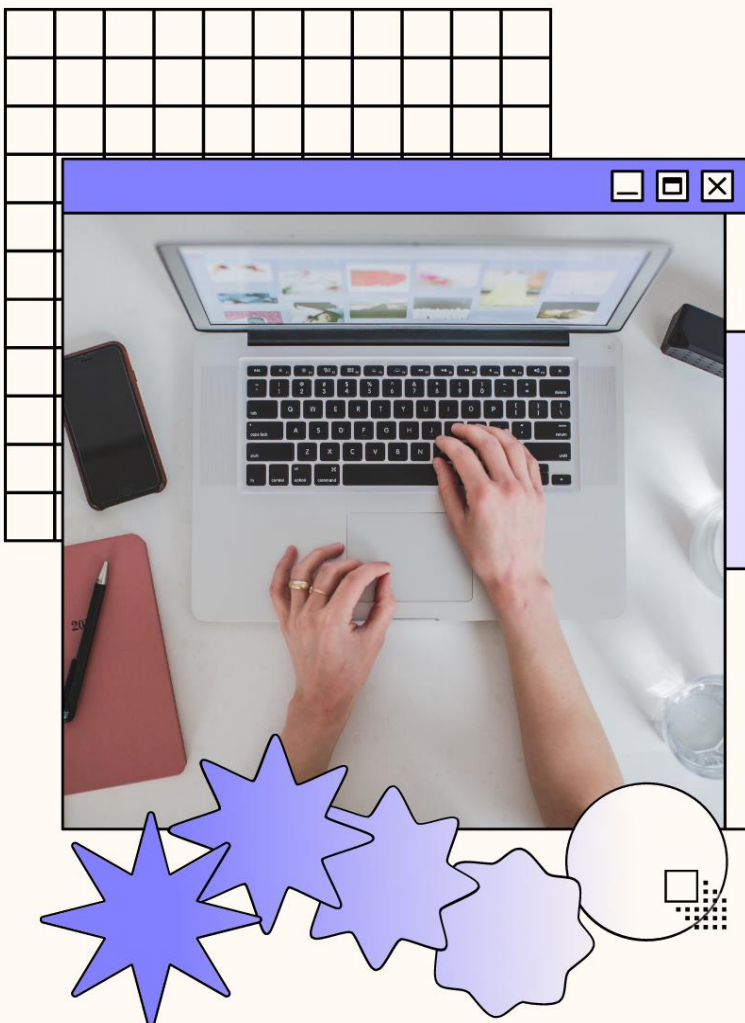
Ingeniería en Computación

NRC:

103844

Sección:

D01



Guadalajara, Jal. a 11 de septiembre del 2022

Actividad No. 4

Simular el Procesamiento por Lotes con Multiprogramación

Introducir desde teclado N procesos, estos serán los que conformen los lotes, la capacidad máxima de un lote es de 3. Simular el procesamiento por lotes con Multiprogramación.

Objetivo

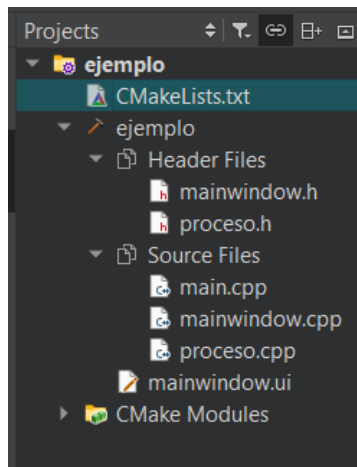
En la actividad presente se busca simular el procesamiento por lotes que ocurren dentro de los sistemas operativos. Como bien se sabe, antes de que existiera la multitarea los sistemas operativos realizaban un proceso obsoleto donde iban ejecutando los distintos procesos uno a uno y por lotes, es decir, de montón en montón hasta haber terminado todos y cada uno de los procesos listados.

Desarrollo

Para el desarrollo de la práctica decidimos usar el lenguaje de programación C++ en complemento con el framework QT. Seleccionamos este lenguaje ya que, además de ser con el que más cómodo nos sentimos, nos da un campo más amplio a la hora de manejar con instrucciones de nivel un poco más bajo, en comparación de Python u otros lenguajes de nivel superior que tienden a obviar algunas partes.

Se desarrolló el programa con programación orientada a objetos y con la funcionalidad de una interfaz gráfica para una mayor comodidad, se crearon instancias para cada para cada proceso y así poder almacenarlos por lotes para facilitar su proceso y así poder simularlo de una manera más real.

Se terminaron creando los siguientes archivos:



Comenzamos mostrando un poco de las clases mainwindow.h y proceso.h donde simplemente se hacen declaraciones privadas y publicas de variables y métodos para el uso de ellos en sus respectivos cpp.

A screenshot of the Qt Creator editor showing the code for mainwindow.h. The left sidebar shows the project structure with mainwindow.h selected. The main editor area displays the C++ code for the MainWindow class, which inherits from QMainWindow. The code includes public methods for generating processes, creating operations, starting processes, showing working lot, running processes, showing finished processes, interrupting processes, and a delay method. It also includes protected methods for key press events and private slots for process number changes and button clicks. The private section includes a pointer to the UI, a pause variable, and process counters.

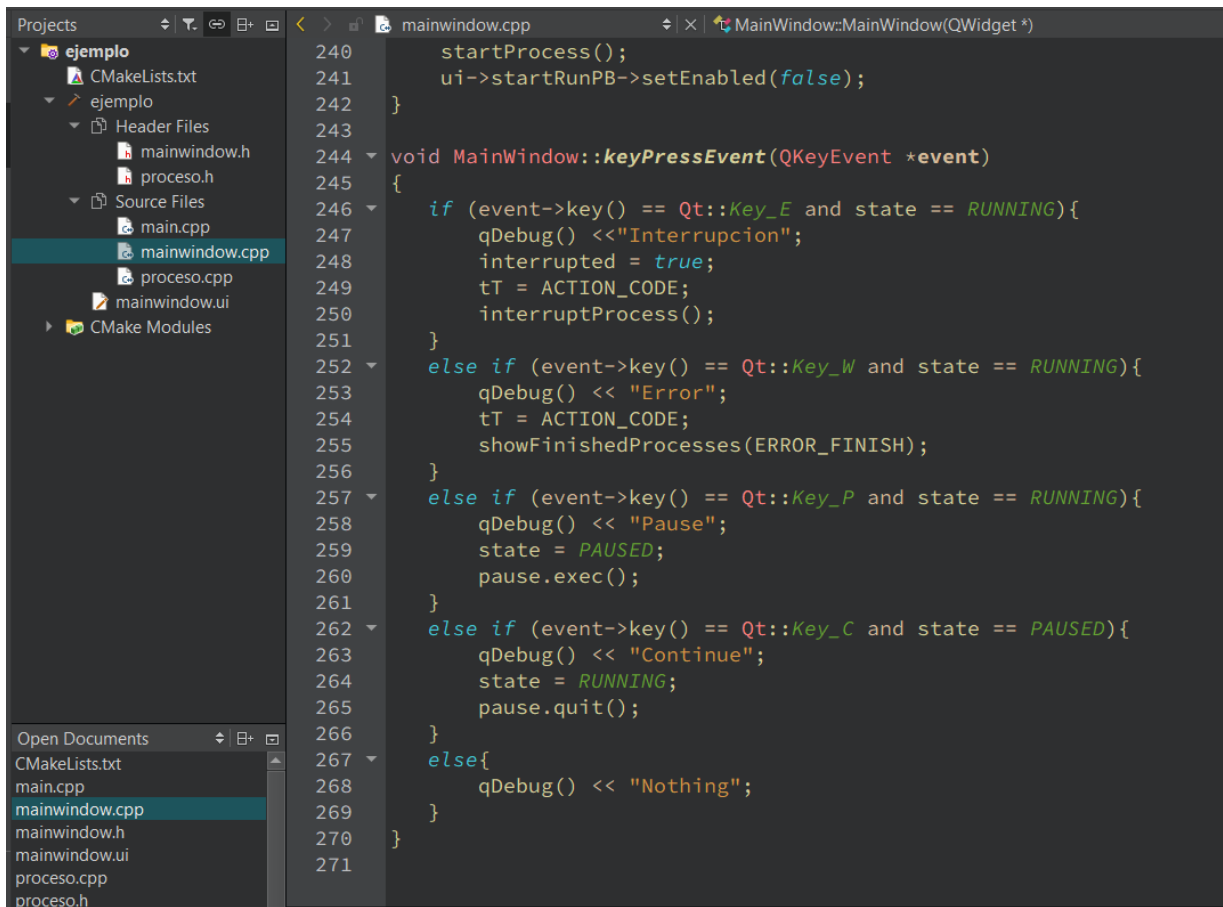
```
39 class MainWindow : public QMainWindow
40 {
41     Q_OBJECT
42
43 public:
44     MainWindow(QWidget *parent = nullptr);
45     ~MainWindow();
46     void generateProcesses();
47     void createOperation(Proceso &p, int num1, int num2, int signOperator);
48     void startProcess();
49     void showWorkingLot();
50     void runProcess();
51     void showFinishedProcesses(bool finishedType);
52     void interruptProcess();
53     void delay();
54
55 protected:
56     void keyPressEvent(QKeyEvent *event);
57
58 private slots:
59
60     void on_processNumberSP_valueChanged(int arg1);
61
62     void on_processAccountPB_clicked();
63
64     void on_startRunPB_clicked();
65
66 private:
67     Ui::MainWindow *ui;
68     QEventLoop pause;
69
70     int capturedProcess = 0;
71     int totalProcess;
```

```
11
12 class Proceso
13 {
14 private:
15     string operacion;
16     int id;
17     int tiempoEstimado;
18     int tiempoTranscurrido;
19     int resultadoOperacion;
20     bool finalizacion;
21
22 public:
23     Proceso();
24     void setNombre(const string &value);
25     int getId() const;
26     void setId(int value);
27     int getTiempoEstimado() const;
28     void setTiempoEstimado(int value);
29     int getTiempoTranscurrido() const;
30     void setTiempoTranscurrido(int value);
31     int getResultadoOperacion() const;
32     void setResultadoOperacion(int value);
33     string getOperacion() const;
34     void setOperacion(const string &value);
35     bool getFinalizacion() const;
36     void setFinalizacion(bool value);
37
38     Proceso &operator=(const Proceso& p);
39 };
40
41 #endif // PROCESO_H
42
```

Después se encuentran los archivos de código cpp donde contamos con diversos procesos como en el main.cpp donde solo se hace la inicialización del programa.

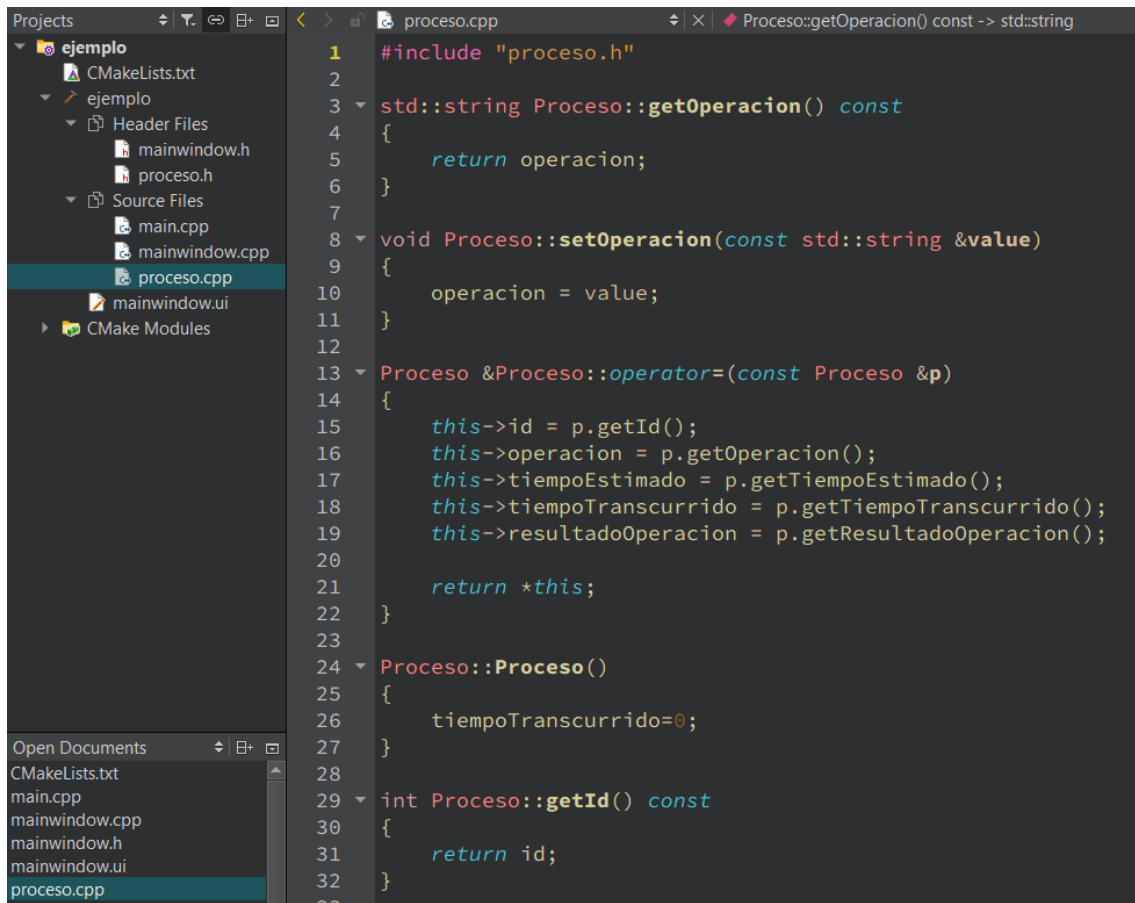
```
1 #include "mainwindow.h"
2
3 #include <QApplication>
4
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     MainWindow w;
9     w.show();
10    return a.exec();
11 }
12
```

El mainwindow.cpp donde se hacen los procesos más importantes y largos como los son todos los datos capturados, operaciones, IDs, números aleatorios de tiempos y las capturas de entrada por teclado que en este caso fueron las letras E, W, P y C como se observa a continuación.



```
240     startProcess();
241     ui->startRunPB->setEnabled(false);
242 }
243
244 void MainWindow::keyPressEvent(QKeyEvent *event)
245 {
246     if (event->key() == Qt::Key_E and state == RUNNING){
247         qDebug() <<"Interrupcion";
248         interrupted = true;
249         tT = ACTION_CODE;
250         interruptProcess();
251     }
252     else if (event->key() == Qt::Key_W and state == RUNNING){
253         qDebug() << "Error";
254         tT = ACTION_CODE;
255         showFinishedProcesses(ERROR_FINISH);
256     }
257     else if (event->key() == Qt::Key_P and state == RUNNING){
258         qDebug() << "Pause";
259         state = PAUSED;
260         pause.exec();
261     }
262     else if (event->key() == Qt::Key_C and state == PAUSED){
263         qDebug() << "Continue";
264         state = RUNNING;
265         pause.quit();
266     }
267     else{
268         qDebug() << "Nothing";
269     }
270 }
271
```

En seguida se encuentra nuestro proceso.cpp el cual cuenta con los setters y getters de nuestro programa.



The screenshot shows a C++ IDE with a project named 'ejemplo'. The file explorer on the left lists the project files: CMakeLists.txt, ejemplo, Header Files (mainwindow.h, proceso.h), Source Files (main.cpp, mainwindow.cpp, proceso.cpp), mainwindow.ui, and CMake Modules. The 'proceso.cpp' file is selected and its code is displayed in the main editor. The code implements the 'Proceso' class with methods for getting and setting an operation, a static operator, and a constructor. The 'getOperacion()' method returns the 'operacion' member variable. The 'setOperacion()' method sets the 'operacion' member variable. The static operator 'operator=' is implemented to copy the state of one process to another. The constructor 'Proceso()' initializes 'tiempoTranscurrido' to 0. The 'getId()' method returns the 'id' member variable.

```
1 #include "proceso.h"
2
3 std::string Proceso::getOperacion() const
4 {
5     return operacion;
6 }
7
8 void Proceso::setOperacion(const std::string &value)
9 {
10     operacion = value;
11 }
12
13 Proceso &Proceso::operator=(const Proceso &p)
14 {
15     this->id = p.getId();
16     this->operacion = p.getOperacion();
17     this->tiempoEstimado = p.getTiempoEstimado();
18     this->tiempoTranscurrido = p.getTiempoTranscurrido();
19     this->resultadoOperacion = p.getResultadoOperacion();
20
21     return *this;
22 }
23
24 Proceso::Proceso()
25 {
26     tiempoTranscurrido=0;
27 }
28
29 int Proceso::getId() const
30 {
31     return id;
32 }
```

Finalmente, nos encontramos con la interfaz gráfica, donde se editaron los objetos y el diseño de estos para obtener una GUI visualmente amigable e intuitiva de usar y así obtener nuestras pantallas finales de ejecución.

Proceso por lotes con Multiprogramación

Seminario de sistemas operativos

Actividad de aprendizaje 4

Programa 2 Simulador al procesamiento por lotes de multiprogramación

Número de procesos:

0

Proceso por lotes con Multiprogramación

Lote en Ejecución:

ID	T. M. Estimac	T. Transcurrido
2	7	0
3	12	0

Proceso en Ejecución:

ID	1
Operación	-35*-69
T. M. Estimado	9
T. Transcurrido	1
T. Restante	8

Procesos Terminados:

ID	Operación	Resultado	N. de lote
LOTE 1			

Número de lotes pendientes

5

Contador global:

1

Simular procesamiento

Conclusiones

Carbajal Armenta Yessenia Paola:

Al llevar a cabo esta práctica me percaté de que no es tan difícil como pensaba, solo se tenían que hacer algunas modificaciones del programa pasado y añadir lo que es la generación aleatoria de los números ya sea para los ID, las operaciones y los tiempos, lo cual se logro con unas funciones además que la entrada por teclado para las interrupciones, pausas, errores y continuación me parece que es de gran ayuda para programas muy robustos.


Sánchez Lozano Jonathan:

Esta práctica solamente se agregaron ciertos factores de E/S al momento de ejecutar los procesos, fue relativamente sencillo agregar métodos para leer las E/S que fueron solicitadas para esta actividad, ya que en QT existen varias herramientas muy útiles para poder leer el teclado, además se añadió la generación aleatoria de operaciones, identificadores y también de tiempos estimados que se realizaron mediante ciertas funciones y herramientas.

Enlace del código

 <https://drive.google.com/drive/folders/1Nid--4PbJRYUYMYc7EtDqpSwdRoPDtC4?usp=sharing>

Enlace del vídeo

 https://drive.google.com/file/d/1ZnW9Z6x_lomNkBN7xRzRgkRjsgeR4ttw/view?usp=sharing