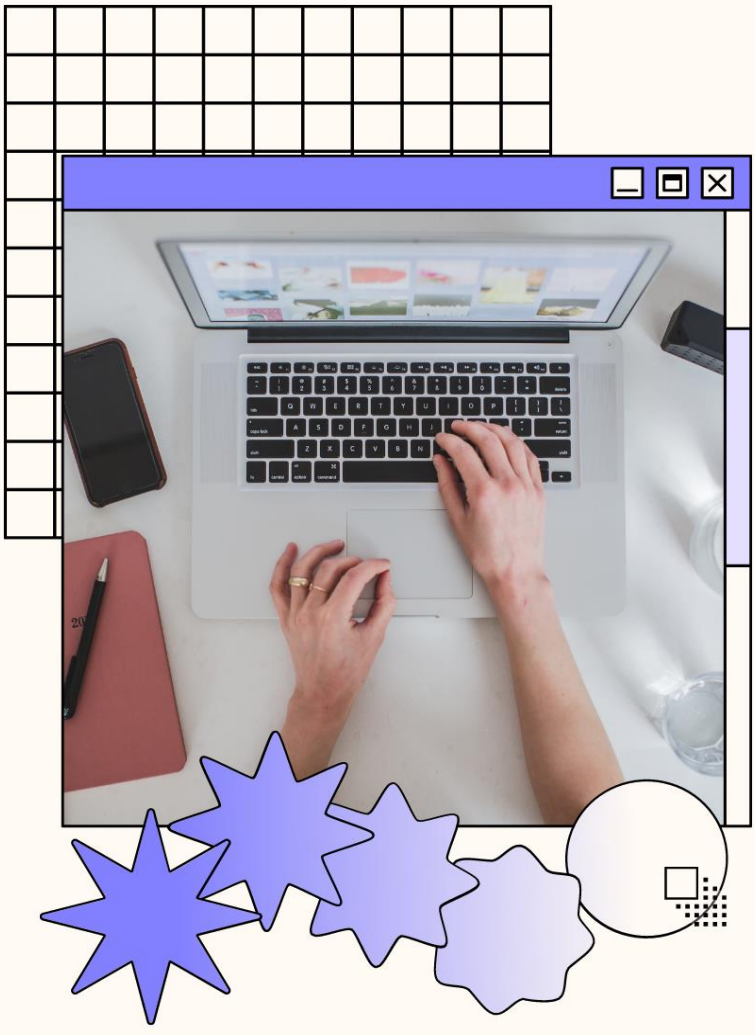
Centro Universitario de Ciencias Exactas e Ingenierías

Universidad de Guadalajara

REPORTE 01

Simular el Procesamiento por Lotes



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 30 de agosto del 2022

Actividad No. 2

Simular el Procesamiento por Lotes

Introducir desde teclado N procesos, estos serán los que conformen los lotes, la capacidad máxima de un lote es de 3, si el número de procesos que se introduce excede esta cantidad, se conformará otro lote. Al terminar un lote automáticamente se ejecuta el siguiente lote en espera.

Objetivo

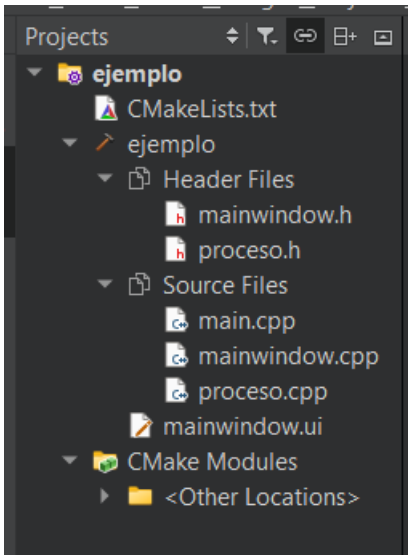
En la actividad presente se busca simular el procesamiento por lotes que ocurren dentro de los sistemas operativos. Como bien se sabe, antes de que existiera la multitarea los sistemas operativos realizaban un proceso obsoleto donde iban ejecutando los distintos procesos uno a uno y por lotes, es decir,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



A continuación, podemos observar las clases que se crearon para el programa la cuales serían el main, con declaraciones publicas y privadas de los voids y ciertas variables, así como la clase de proceso con las declaraciones de las variables que se necesitan en todo el programa, así como constructores, etc.

proceso.h — Proceso

```
#ifndef PROCESO_H
#define PROCESO_H

#include <iostream>

class Proceso
{
private:
    std::string nombre;
    std::string operacion;
    int id;
    int tiempoEstimado;
    int resultadoOperacion;

public:
    Proceso();
    std::string getNombre() const;
    void setNombre(const std::string &value);
    int getId() const;
    void setId(int value);
    int getTiempoEstimado() const;
    void setTiempoEstimado(int value);
    int getResultadoOperacion() const;
    void setResultadoOperacion(int value);
    std::string getOperacion() const;
    void setOperacion(const std::string &value);

    //Operators
    Proceso &operator=(const Proceso& p);
};

#endif // PROCESO_H
```

mainwindow.h — MainWindow

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <iostream>
#include <vector>
#include <thread>
#include <windows.h>
#include <QTimer>
#include <QMessageBox>

#include "proceso.h"

#define CAPTURE_DATA 0
#define SHOW_LOTS 1
#define PROCESS_PER_LOT 3
#define NUMBER_TO_DELAY 500000000


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    void addProcess();
    bool operationValid();
    bool idExist(int id);
    void splitProcessInLots();
    void startProcess();
    void showWorkingLot();
    void runProcess();
    void delay();

private slots:

    void on_processNumberSP_valueChanged(int arg1);

    void on_processAcountPB_clicked();

    void enable_add_process_button();

    void on_nameLE_textChanged(const QString &arg1);
```

Después, nos encontramos con los archivos cpp, donde se encuentra la inicialización de la pantalla, los constructores y destructores, las impresiones, las validaciones, entre otros.

main.cpp*

```cpp
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

mainwindow.cpp — MainWindow::addProcess() -> void

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("Programa 1.  Simular el Procesamiento por Lotes");
    setFixedSize(1280,720);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::addProcess()
{
    Proceso p;
    std::string operation;
    int result,number1,number2;

    number1 = ui->number1SP->value();
    number2 = ui->number2SP->value();

    switch (ui->symbolsCB->currentIndex()){
    case 0:
        result = number1 + number2;
        operation = std::to_string(number1)+" + "+std::to_string(number2);
        break;
    case 1:
        result = number1 - number2;
        operation = std::to_string(number1)+" - "+std::to_string(number2);
        break;
    case 2:
        result = number1 * number2;
        operation = std::to_string(number1)+" * "+std::to_string(number2);
        break;
    case 3:
        result = number1 / number2;
        operation = std::to_string(number1)+" / "+std::to_string(number2);
        break;
    case 4:
        result = number1 % number2;
        operation = std::to_string(number1)+" % "+std::to_string(number2);
        break;
```

proceso.cpp — Proceso::setId(int) -> void

```cpp
#include "proceso.h"

std::string Proceso::getOperacion() const
{
    return operacion;
}

void Proceso::setOperacion(const std::string &value)
{
    operacion = value;
}

Proceso &Proceso::operator=(const Proceso &p)
{
    this->id = p.getId();
    this->nombre = p.getNombre();
    this->operacion = p.getOperacion();
    this->tiempoEstimado = p.getTiempoEstimado();
    this->resultadoOperacion = p.getResultadoOperacion();

    return *this;
}

Proceso::Proceso()
{

}

std::string Proceso::getNombre() const
{
    return nombre;
}

void Proceso::setNombre(const std::string &value)
{
    nombre = value;
}

int Proceso::getId() const
{
    return id;
}

void Proceso::setId(int value)
{
    id = value;
}
```

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.

Conclusiones

Carbajal Armenta Yessenia Paola:

Al llevar a cabo esta práctica nos percatamos de que no era tan sencillo como pensábamos, pues debimos investigar mucho para poder lograrlo, además que

batallamos mucho en la parte del diseño, sin embargo, nos percatamos y nos dimos una idea más clara sobre este procesamiento, el cual es la manera más eficaz de cargar un gran número de registros desde sistemas de origen.

Sánchez Lozano Jonathan:

Esta práctica al principio me pareció algo confusa y pensamos que iba a estar muy difícil pero la realidad fue otra, conforme fuimos investigando como hacer las cosas nos dimos cuenta de que no era difícil si no tardada. Al final pudimos resolver la práctica, aunque en ciertas ocasiones hacíamos pruebas y ver los fallos que teníamos para irlos resolviendo poco a poco, pero con la ayuda de mi compañera no fue una tarea difícil. Me quedo con la enseñanza de saber cómo trabajan los sistemas

Link del video:

https://drive.google.com/file/d/1imgzJtnupLeDYUHgxyr1qg7np6KzByEk/view