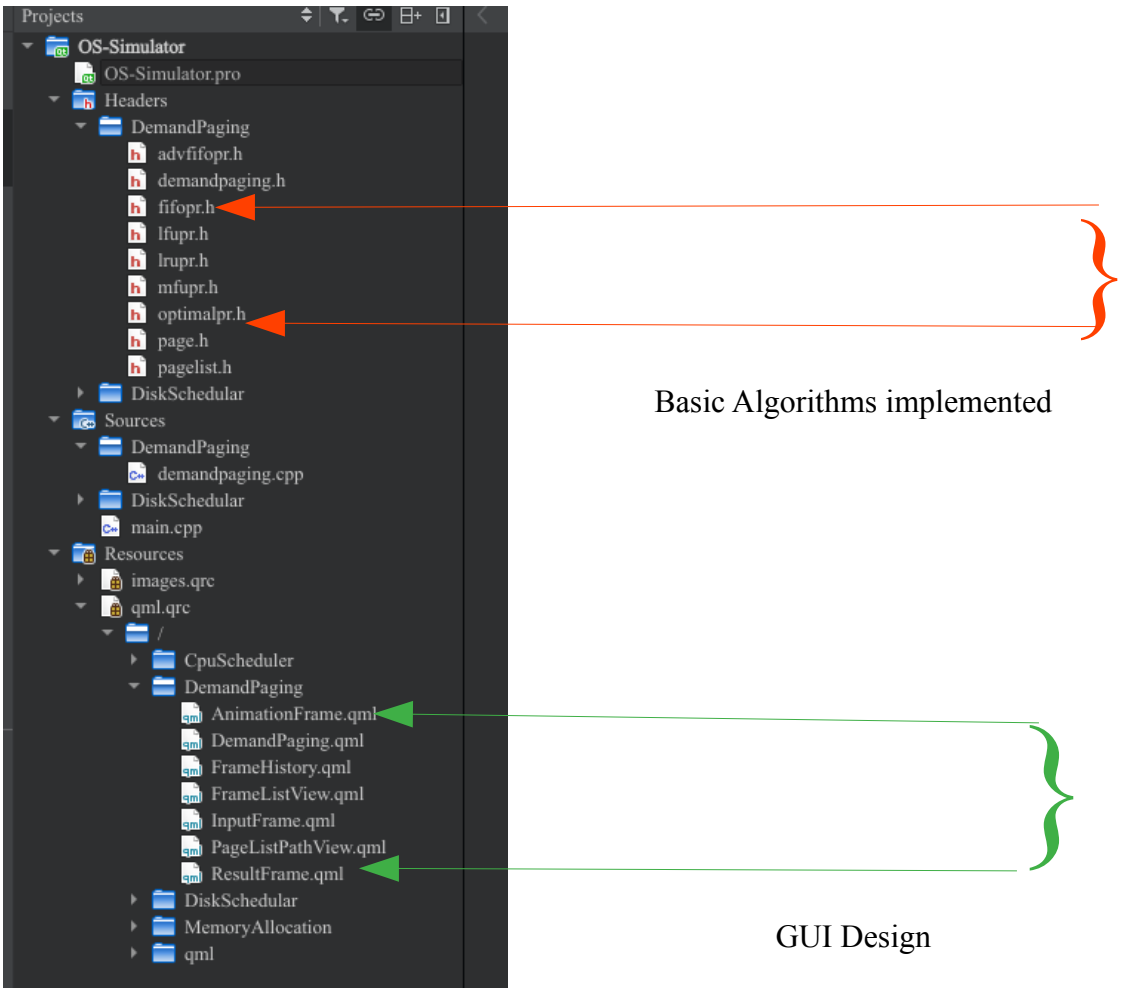
OS Mini Project
on
Demand Paging/ Page Replacement Algorithms

By
Rajiv Singh (SE Comps B 58)
Sushrut Kuchik (SE Comps B 62)
Vivek Vaishya (SE Comps B 66)

Abstract

Demand Paging is a method of virtual memory management. In a system that uses demand paging, the operating system copies a disk page into physical memory only if an attempt is made to access it and that page is not already in memory. It follows that a process begins execution with none of its pages in physical memory and many page faults will occur until most of a process's working set of pages are located in physical memory. Make a GUI application to simulate the mostly employed Page Replacement policies by Operating Systems.

Source Code Flow



Source Code

//demandpaging.h

```
#ifndef DEMANDPAGING_H
#define DEMANDPAGING_H

#include <QObject>
#include <QThread>

class DemandPaging : public QObject
{
	Q_OBJECT
	Q_PROPERTY(int FrameSize READ FrameSize WRITE setFrameSize NOTIFY FrameSizeChanged)
	Q_PROPERTY(QList<QObject*> FrameList READ FrameList NOTIFY FrameListChanged)
	Q_PROPERTY(QList<QObject*> PageList READ PageList NOTIFY PageListChanged)
	Q_PROPERTY(QList<QObject*> FrameHistory READ FrameHistory NOTIFY FrameHistoryChanged)
	Q_PROPERTY(QObject* FirstPage READ FirstPage NOTIFY FirstPageChanged)
	Q_PROPERTY(int PageFaults READ PageFaults NOTIFY PageFaultsChanged)
	Q_PROPERTY(int PageHits READ PageHits NOTIFY PageHitsChanged)
	Q_PROPERTY(int TotalPages READ TotalPages NOTIFY TotalPagesChanged)
	Q_PROPERTY(QString ExtraTip READ ExtraTip NOTIFY ExtraTipChanged)

public:
	explicit DemandPaging(QObject *parent = nullptr);

	int FrameSize() const{ return frameSize;}
	void setFrameSize(const int FrameSize);

	QList<QObject*> FrameList() const{ return frameList;}

	QList<QObject*> PageList() const{ return pageList;}
	QObject* FirstPage() const{ return pageList.first();}

	QList<QObject*> FrameHistory() const{ return frameHistory;}

	int PageFaults() const{ return pageFaults;}
	int PageHits() const{ return pageHits;}
	int TotalPages() const{ return totalPages;}
	QString ExtraTip() const{ return extraTip;}

signals:
	void FrameSizeChanged();
	void FrameListChanged();
	void PageListChanged();
	void FirstPageChanged();
	void FrameHistoryChanged();

	void startThreadCalc();
	void moveNextPage(int frameIndex);

	void PageFaultsChanged();
	void PageHitsChanged();
```

```
        void TotalPagesChanged();
        void ExtraTipChanged();

public slots:
        void addPage(int page);
        void startCalculation(int typeOfAlgorithm);
        void reset();

        void updatePageList(int changedFrameNo);
        void updateFrameList(int changedFrameNo, int changedValue);
        void updateResult(int faults, int hits);

private:
        int frameSize;
        QList<QObject*> frameList;
        QList<QObject*> pageList;
        QList<QObject*> frameHistory;
        QThread calcThread;
        int pageFaults;
        int pageHits;
        int totalPages;
        QString extraTip;
};

#endif // DEMANDPAGING_H
```

//fifopr.h

```
#ifndef FIFOPR_H
#define FIFOPR_H

#include <QThread>
#include "page.h"

class FifoThread: public QObject{
        Q_OBJECT

public:
        FifoThread(QList<QObject*> PageList, int FrameSize, QObject *parent = nullptr)
                :QObject (parent){
                pageList.append(PageList);
                frameSize = FrameSize;
        }

        void calculate(){
                int faults = 0, hits = 0;
                for(int i = 0; i < frameSize; i++)
                        frameList.append(new Page(-1));

                int counter = 0;
                while(!pageList.isEmpty()){
```

```
                auto firstPage = qobject_cast<Page*>(pageList.takeFirst());
                emit updatePageList(counter);
                QThread::msleep(500);

                if(contains(firstPage)){
                        hits++;
                        emit updateFrameList(-1, firstPage->Id());
                }
                else{
                        frameList.replace(counter, firstPage);
                        emit updateFrameList(counter, firstPage->Id());
                        counter = (counter + 1)%frameSize;
                        faults++;
                }

                emit updateResult(faults, hits);
                QThread::msleep(250);
        }
    }

    bool contains(Page *page){
        foreach(QObject *obj, frameList){
            auto frame = qobject_cast<Page*>(obj);
            if(frame->Id() == page->Id())
                return true;
        }
        return false;
    }

signals:
    void updatePageList(int changedFrameNo);
    void updateFrameList(int changedFrameNo, int changedValue);
    void updateResult(int faults, int hits);

private:
    int frameSize;
    QList<QObject*> frameList;
    QList<QObject*> pageList;
};

#endif // FIFOPR_H
```

//lfupr.h

```
#ifndef LFUPR_H
#define LFUPR_H

#include <QThread>
#include "page.h"

class CountClass{
public:
```

```
        CountClass(int Id, int Count)
                :id(Id), count(Count){}
        int Id() const{return id;}
        int Count() const{ return count;}
        void incrementCount(){ count++; }
        void decrementCount(){ if(count != 0) count--;}
private:
        int id, count;
};

class LfuThread: public QObject{
        Q_OBJECT

public:
        LfuThread(QList<QObject*> PageList, int FrameSize, QObject *parent = nullptr)
                :QObject (parent){
                pageList.append(PageList);
                frameSize = FrameSize;
        }

        void calculate(){
                int faults = 0, hits = 0;
                for(int i = 0; i < frameSize; i++)
                        frameList.append(new Page(-1));

                foreach (QObject *obj, pageList){
                        auto id = qobject_cast<Page*>(obj)->Id();
                        int i;
                        for(i = 0; i < repeated.length() && repeated[i].Id() != id; i++);

                        if(i == repeated.length())
                                repeated.append(CountClass(id, 0));
                }

                int count = 0;
                while(count < frameSize){
                        auto firstPage = qobject_cast<Page*>(pageList.takeFirst());
                        incrementCount(firstPage->Id());
                        frameList.replace(count, firstPage);
                        emit updatePageList(count);
                        QThread::msleep(500);
                        faults++;
                        emit updateFrameList(count++, firstPage->Id());
                }

                while(!pageList.isEmpty()){
                        auto firstPage = qobject_cast<Page*>(pageList.takeFirst());

                        int position = contains(firstPage);
                        if(position > -1){
                                incrementCount(firstPage->Id());
                                emit updatePageList(position);
                                QThread::msleep(500);
                                hits++;
```

```
                emit updateFrameList(-1, firstPage->Id());
            }
            else{
                int l = leastCount();
                int con = contains(new Page(l));
                frameList.replace(con, firstPage);
                incrementCount(firstPage->Id());

                emit updatePageList(con);
                QThread::msleep(500);

                emit updateFrameList(con, firstPage->Id());
                faults++;
            }

            emit updateResult(faults, hits);
            QThread::msleep(250);
        }
    }

    int contains(Page *page){
        for(int i = 0 ; i < frameList.length(); i++){
            auto frame = qobject_cast<Page*>(frameList[i]);
            if(frame->Id() == page->Id())
                return i;
        }
        return -1;
    }

    void incrementCount(int Id){
        for(int i = 0; i < repeated.length(); i++)
            if(repeated[i].Id() == Id){
                repeated[i].incrementCount();
                repeated.move(i, frameSize - 1);
                break;
            }

    }

    int leastCount(){
        int least = 0;

        for(int i = 1; i < frameSize - 1; i++)
            if(repeated[i].Count() < repeated[least].Count())
                least = i;

        repeated[least].decrementCount();
        repeated.move(least, frameSize);
        return repeated[frameSize].Id();
    }

signals:
    void updatePageList(int changedFrameNo);
```

```
    void updateFrameList(int changedFrameNo, int changedValue);
    void updateResult(int faults, int hits);

private:
    int frameSize;
    QList<QObject*> frameList;
    QList<QObject*> pageList;
    QList<CountClass> repeated;
};

#endif // LFUPR_H
```

//lrupr.h

```
#ifndef LRUPR_H
#define LRUPR_H

#include <QThread>
#include "page.h"

class LruThread: public QObject{
    Q_OBJECT

public:
    LruThread(QList<QObject*> PageList, int FrameSize, QObject *parent = nullptr)
        :QObject (parent){
        pageList.append(PageList);
        frameSize = FrameSize;
    }

    void calculate(){
        int faults = 0, hits = 0;
        for(int i = 0; i < frameSize; i++)
            frameList.append(new Page(-1));

        int lIndex = 0;
        while(!pageList.isEmpty()){
            auto firstPage = qobject_cast<Page*>(pageList.takeFirst());

            int oindex = indexOf(firstPage);
            if(oindex > -1){
                displace(firstPage);
                emit updatePageList(oindex);
                QThread::msleep(500);
                hits++;
                emit updateFrameList(-1, firstPage->Id());
            }
            else{
                if(!tempFrameList.isEmpty() && tempFrameList.length() >= frameList.length())
                    lIndex = indexOf(qobject_cast<Page*>(tempFrameList.takeFirst()));

                frameList.replace(lIndex, firstPage);
                tempFrameList.append(firstPage);
```

```
                emit updatePageList(lIndex);
                QThread::msleep(500);

                emit updateFrameList(lIndex, firstPage->Id());
                lIndex++;
                faults++;
            }

            emit updateResult(faults, hits);
            QThread::msleep(250);
        }
    }

    void displace(Page *page){
        for(int i = 0; i < tempFrameList.length(); i++){
            if(page->Id() == (qobject_cast<Page*>(tempFrameList[i]))->Id()){
                tempFrameList.removeAt(i);
                break;
            }
        }
        tempFrameList.append(page);
    }

    int indexOf(Page *page){
        for(int i = 0 ; i < frameList.length(); i++){
            auto frame = qobject_cast<Page*>(frameList[i]);
            if(frame->Id() == page->Id())
                return i;
        }
        return -1;
    }

signals:
    void updatePageList(int changedFrameNo);
    void updateFrameList(int changedFrameNo, int changedValue);
    void updateResult(int faults, int hits);

private:
    int frameSize;
    QList<QObject*> frameList;
    QList<QObject*> tempFrameList;
    QList<QObject*> pageList;
};

#endif // LRUPR_H
```

//mfupr.h

```
#ifndef MFUPR_H
#define MFUPR_H
```

```qt
#include <QThread>
#include "page.h"

class MfuThread: public QObject{
	Q_OBJECT

public:
	MfuThread(QList<QObject*> PageList, int FrameSize, QObject *parent = nullptr)
		:QObject (parent){
		pageList.append(PageList);
		frameSize = FrameSize;
	}

	void calculate(){
		int faults = 0, hits = 0;
		for(int i = 0; i < frameSize; i++)
			frameList.append(new Page(-1));

		for(int i = 0; i < frameSize; i++){
			auto firstPage = qobject_cast<Page*>(pageList.takeFirst());
			emit updatePageList(i);
			QThread::msleep(500);

			frameList.replace(i, firstPage);
			emit updateFrameList(i, firstPage->Id());

			emit updateResult(++faults, hits);
			QThread::msleep(250);
		}

		int counter = frameSize - 1;
		while(!pageList.isEmpty()){
			auto firstPage = qobject_cast<Page*>(pageList.takeFirst());
			emit updatePageList(counter);
			QThread::msleep(500);

			int con = contains(firstPage);
			if(con > -1){
				hits++;
				emit updateFrameList(-1, firstPage->Id());
				counter = con;
			}
			else{
				frameList.replace(counter, firstPage);
				emit updateFrameList(counter, firstPage->Id());
				faults++;
			}

			emit updateResult(faults, hits);
			QThread::msleep(250);
		}
	}
```

```
    int contains(Page *page){
        for(int i = 0 ; i < frameList.length(); i++){
            auto frame = qobject_cast<Page*>(frameList[i]);
            if(frame->Id() == page->Id())
                return i;
        }
        return -1;
    }

signals:
    void updatePageList(int changedFrameNo);
    void updateFrameList(int changedFrameNo, int changedValue);
    void updateResult(int faults, int hits);

private:
    int frameSize;
    QList<QObject*> frameList;
    QList<QObject*> pageList;
};#endif // MFUPR_H
```

//optimalpr.h

```
#ifndef OPTIMALPR_H
#define OPTIMALPR_H

#include <QThread>
#include "page.h"

class OptimalThread: public QObject{
    Q_OBJECT

public:
    OptimalThread(QList<QObject*> PageList, int FrameSize, QObject *parent = nullptr)
        :QObject (parent){
        pageList.append(PageList);
        frameSize = FrameSize;
    }

    void calculate(){
        int faults = 0, hits = 0;
        for(int i = 0; i < frameSize; i++)
            frameList.append(new Page(-1));

        int counter = 0;
        while(!pageList.isEmpty()){
            auto firstPage = qobject_cast<Page*>(pageList.takeFirst());

            int lIndex = contains(firstPage);
            if(lIndex > -1){
                hits++;
                emit updatePageList(lIndex);
                QThread::msleep(500);
                emit updateFrameList(-1, firstPage->Id());
```

```
                }
                else{
                        lIndex = (counter < frameSize) ? counter++ : forLookFuture();
                        frameList.replace(lIndex, firstPage);
                        faults++;
                        emit updatePageList(lIndex);
                        QThread::msleep(500);
                        emit updateFrameList(lIndex, firstPage->Id());
                }

                emit updateResult(faults, hits);
                QThread::msleep(250);
        }
}

int contains(Page *page){
        for(int i = 0 ; i < frameList.length(); i++){
                auto frame = qobject_cast<Page*>(frameList[i]);
                if(frame->Id() == page->Id())
                        return i;
        }
        return -1;
}

int forLookFuture(){
        //signed int nearOccurence = -1;
        int index = 0;
        for(int j = 0, nearOccurence = 0; j < frameSize; j++){
                auto frame = qobject_cast<Page*>(frameList[j]);
                if(frame->Id() != -1){
                        int i;
                        for(i = 0; i < pageList.length(); i++){
                                auto page = qobject_cast<Page*>(pageList[i]);
                                if(page->Id() == frame->Id()){
                                        if(nearOccurence < i){
                                                nearOccurence = i;
                                                index = j;
                                        }
                                        break;
                                }
                        }
                        if(i == pageList.length()){
                                nearOccurence = i;
                                index = j;
                        }
                }
        }
        return index;
}

signals:
        void updatePageList(int changedFrameNo);
        void updateFrameList(int changedFrameNo, int changedValue);
        void updateResult(int faults, int hits);
```

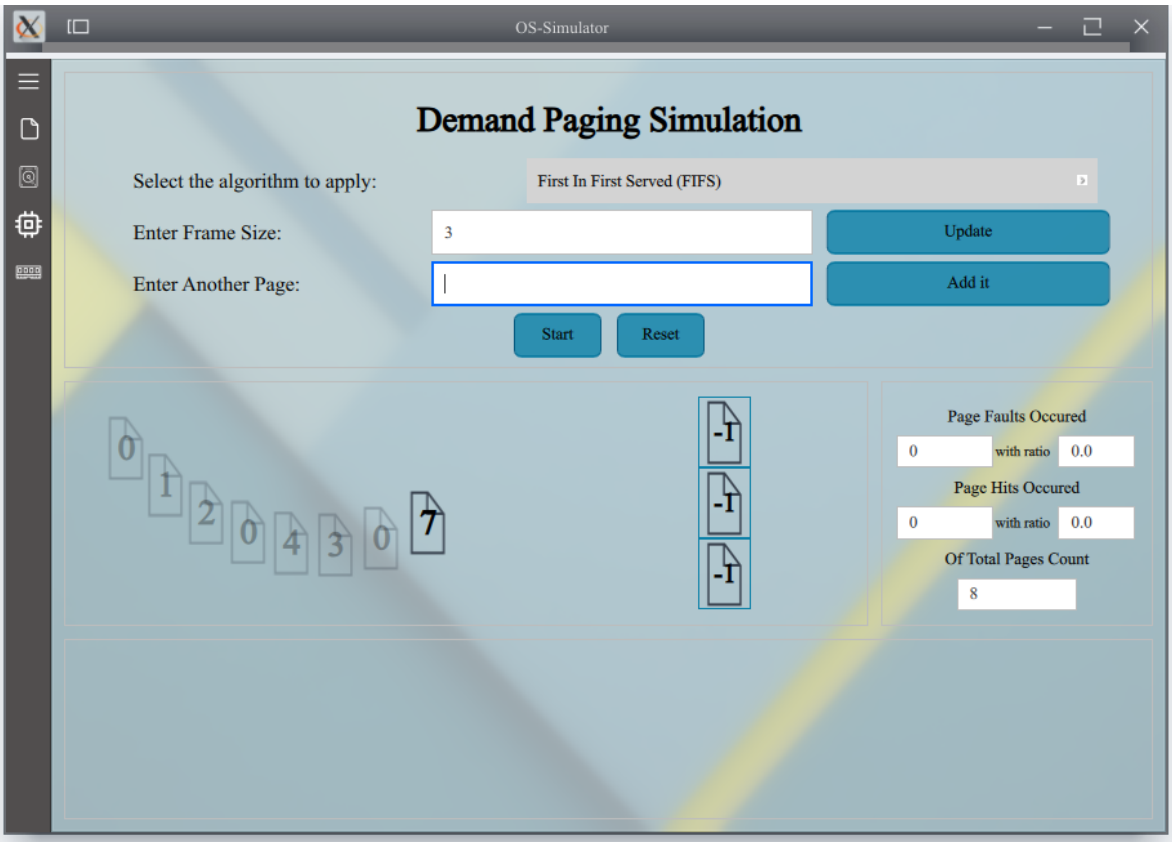
```
private:
    int frameSize;
    QList<QObject*> frameList;
    QList<QObject*> pageList;
};

#endif // OPTIMALPR_H
```

//DemandPaging.qml

```
import QtQuick 2.12
import QtQuick.Controls 2.5
import "../qml/CustomComponents"

Rectangle {
    id: demandPaging
    color: "transparent"

    Column{
        anchors.fill: parent
        spacing: 10
        anchors.margins: 10

        InputFrame{
            id: inputFrame
            width: parent.width
            anchors.horizontalCenter: parent.horizontalCenter
        }

        Row{
            width: parent.width
            height: (parent.height - inputFrame.height)*0.6 - 20
            spacing: 10

            AnimationFrame{
                id: animationFrame
                width: parent.width*0.75 - 10
                height: parent.height
            }

            ResultFrame{
                width: parent.width*0.25
                height: parent.height
            }
        }

        FrameHistory{
            id: frameList
            width: parent.width
            height: (parent.height - inputFrame.height)*0.4
            anchors.bottomMargin: 10
        }
    }
}
```

Screenshots

Before Solving



After Solving