# Program 3 SPAM

Keith Schmitt

CIS 452-10

Program 3

Professor Dulimarta

Language and Environment Design Decisions:

I wrote this program using C++ and the Qt library. I created this project using a MVC design pattern using underlying data structures to interact with a user level graphical interface that Qt provides. I split the logic and functions into two different objects: page\_table and memory. Page\_table is used to represent the page table for a process. This will take input from the mainwindow class which just acts as the interface.

Data Structures:

The way I organized the structure of this gui application is via different objects of a page\_table for each instantiation of a “process” and a memory class that will simulate the frames of actual physical memory. Each page\_table has a list (std::vector) of the pages of code and data sections that they can put into memory. When prompted they can execute their allocate() function which will place their pages into the frames if they are available, handled by the memory object. The memory simply just checks for open frames to put the processes into. It has a list (std::vector) of frames that it just checks f any of them are open, if not then it will return an “error” which is just a string saying that there is no more frames for the process to be allocated into.

mainwindow:

The mainwindow class utilizes the Qt environment that is full of useful tools such as QTextEdits, QStrings, QLists, Buttons and so on that make gui application development in C++ much more approachable. So the main purpose of this class is to communicate data from the user to the underlying data structures that will handle the algorithms. This class also checks for valid inputs and valid terminations for the simulation. This class also has a memory object and a std::vector of page\_table objects to allow for multiprogramming. Termination of the program is handled just by hitting the red “x” on the application, which will appropriately call the destructor and delete the correct objects.

page\_table:

The page\_table class has the responsibility to use a PAGE\_SIZE to divide the inputted code size and data size into the correct number of allocate to memory calls. Will eventually return back a string so the user can see whether and where the code or data fit into memory.

memory:

The memory class acts as physical memory, has an allocate and deallocate function. So this class has a vector corresponding to the number of frames (std::vector<int> frames), who the owner of that frame (std::vector<int> owned\_by) and a vector to allocated whether a frame is The allocate function is called from the page\_table which will send a frame and whether it is code or data.

Input:

So input is structured how Professor Dulimarta has it in his SPAM assignment specification:

pid cod\_size data\_size

or

pid -1 (to terminate)

So the user just has to type these into the file text box in order to communicate with the program. On success the program should visually update the frames with correct values and say which process is in which frame.

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "memory.h"

#include <iostream>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Constructor used to initialize ui, memory and page\_table

\* objects

\* @author Keith Schmitt

\* @version 1.0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

**ui**(new Ui::MainWindow){

**ui**->setupUi(this);

setup\_frames();

**ui**->**verticalLayout**->setSpacing(0);

**my\_memory** = new memory;

this->**frameSize** = 512;

updateFrames();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* method to update the gui frames

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MainWindow::**updateFrames**(){

for(int i = 0; i < **frame\_list**.count(); ++i){

// underlying storage has things that are free stored as -1

if (**my\_memory**->getFrame(i) == -1){

**frame\_list**.at(i)->setText(QString("Free"));

}

//we want to print out the pid and whether it is a code or data section

else{

**frame\_list**.at(i)->setText("P" + QString::number(**my\_memory**->getFrameOwner(i))

+": " + "-- " + QString

::fromStdString(**my\_memory**->getFrameType(i)));

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* called in constructor to setup gui frames

\* to be put into the framelist vector

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MainWindow::**setup\_frames**(){

**frame\_list**.push\_back(**ui**->**textEdit\_1**);

**frame\_list**.push\_back(**ui**->**textEdit\_2**);

**frame\_list**.push\_back(**ui**->**textEdit\_3**);

**frame\_list**.push\_back(**ui**->**textEdit\_4**);

**frame\_list**.push\_back(**ui**->**textEdit\_5**);

**frame\_list**.push\_back(**ui**->**textEdit\_6**);

**frame\_list**.push\_back(**ui**->**textEdit\_7**);

**frame\_list**.push\_back(**ui**->**textEdit\_8**);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* destructor, make sure to delete new objects

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

MainWindow::~***MainWindow***(){

delete **my\_memory**;

delete **ui**;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* allocate all button logic

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MainWindow::**on\_pushButton\_clicked**(){

QStringList lines;

lines = **ui**->**textEdit**->toPlainText().split('\n', QString::SkipEmptyParts);

for (QStringList::iterator iter = lines.begin(); iter != lines.end(); ++iter){

**ui**->**plainTextEdit**->insertPlainText( \*iter +'\n');

QString line = \*iter;

QStringList arguments;

arguments = line.split(' ');

//checking if it is in correct format

if (arguments.count() == 3 ){

int process = arguments[0].toInt();

int code = arguments[1].toInt();

int data = arguments[2].toInt();

if (arguments[0] < 0 || arguments[1] < 0 || arguments[2] < 0 ){

**ui**->**plainTextEdit**->clear();

**ui**->**plainTextEdit**->insertPlainText(QString::fromStdString(

"Sorry error at input: "+

std::to\_string(iter - lines.begin())+

"\nTry a valid input:\npid code\_size data\_size\n"

"or to terminate\nprocess -1\n"));

}

else{

**the\_pages**.push\_back(new page\_table(process, code, data, **my\_memory**, **frameSize**));

**ui**->**plainTextEdit**->insertPlainText(QString::fromStdString(

**the\_pages**[**the\_pages**.size() -1]->allocate(process, code, data)));

}

}

//logic for two inputs on the line; here we wanna see if its termination

else if( arguments.count() == 2){

int process = arguments[0].toInt();

int code = arguments[1].toInt();

if (code == -1){

**ui**->**plainTextEdit**->insertPlainText(QString::fromStdString(

**my\_memory**->deallocate(process)));

}

else{

**ui**->**plainTextEdit**->clear();

**ui**->**plainTextEdit**->insertPlainText(QString::fromStdString(

"Sorry error at input: "+std::to\_string(

iter - lines.begin())+"\nTry a valid input:"+

"\npid code\_size data\_size\n"+

"or to terminate\nprocess -1\n"));

}

}

else{

//incorrect input

**ui**->**plainTextEdit**->clear();

**ui**->**plainTextEdit**->insertPlainText(

QString::fromStdString("Sorry error at input: "+std::to\_string(

iter - lines.begin())+"\nTry a valid input:\npid code\_size data\_size"+

"\ni.e 0 100 200\n"));

}

**ui**->**plainTextEdit**->verticalScrollBar()->setValue(**ui**->**plainTextEdit**->

verticalScrollBar()->maximum());

updateFrames();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* clear button

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MainWindow::**on\_pushButton\_2\_clicked**(){

**ui**->**textEdit**->clear();

**ui**->**plainTextEdit**->clear();

//delete old memory

delete **my\_memory**;

//create new memory

**my\_memory** = new memory();

//update the ui

**the\_pages**.clear();

updateFrames();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*clear input button

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MainWindow::**on\_pushButton\_4\_clicked**(){

**ui**->**textEdit**->clear();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* change frame size

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void MainWindow::**on\_pushButton\_3\_clicked**(){

QString input = **ui**->**lineEdit**->text();

int i;

if ((i = input.toInt()) < 0) {

**ui**->**plainTextEdit**->insertPlainText("Please enter something into the text box\n");

}

else if (i < 0 || i >30 ){

**ui**->**plainTextEdit**->insertPlainText("Invalid frame size, try a positive int and less than 30\n");

}

else{

//ui->verticalLayout->addWidget(new QTextEdit);

**frameSize** = 4096/i;

delete **my\_memory**;

**the\_pages**.clear();

**frame\_list**.clear();

**my\_memory** = new memory;

**my\_memory**->setSize(**frameSize**);

recreateFrames();

**ui**->**label\_4**->setText(QString::fromStdString("Frame Size: " + std::to\_string(**my\_memory**->getFrameSize()) + " KB"));

**ui**->**plainTextEdit**->setPlainText(QString::fromStdString("Hey so you now have " + std::to\_string(**frame\_list**.size()) + " frames of size: "+ std::to\_string(**my\_memory**->getFrameSize())));

}

}

void MainWindow::**recreateFrames**(){

clearLayout(**ui**->**verticalLayout**);

int RAMoverFrame = 4096 / **my\_memory**->getFrameSize();

**ui**->**verticalLayout**->activate();

**ui**->**plainTextEdit**->setPlainText(QString::fromStdString("RoverF: "+ std::to\_string(RAMoverFrame)));

for (int i = 0; i < RAMoverFrame; ++i ){

**frame\_list**.push\_back(new QTextEdit);

**ui**->**verticalLayout**->addWidget(**frame\_list**[i]);

if (**my\_memory**->getFrame(i) == -1){

**frame\_list**.at(i)->setText(QString("Free"));

}

else{

**frame\_list**.at(i)->setText("P" + QString::number(**my\_memory**->getFrameOwner(i))

+": " + "-- " + QString

::fromStdString(**my\_memory**->getFrameType(i)));

}

}

}

void MainWindow::**clearLayout**(QLayout\* layout){

QLayoutItem\* old\_text;

while((old\_text = **ui**->**verticalLayout**->*takeAt*(0)) !=NULL){

if (old\_text->*layout*()){

clearLayout(old\_text->*layout*());

delete old\_text->*layout*();

}

if (old\_text->*widget*()){

delete old\_text->*widget*();

}

delete old\_text;

}

}

page\_table.cpp

#include "page\_table.h"

#include "memory.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* constructor for a page table that has in its parameters a reference to a mem

\* object that it will interact with.

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

page\_table::**page\_table**(int process, int code\_size, int data\_size, memory\* mem){

**my\_memory** = mem;

this->**process** = process;

this->**code\_size** = code\_size;

this->**data\_size** = data\_size;

}

std::vector<int> \*page\_table::**getPages**(){

return &**pages**;

}

bool page\_table::**isEmpty**(){

return (**pages**.size() == 0);

}

std::string page\_table::**allocate**(int process, int code\_size, int data\_size){

std::string output;

//limit the number of code\_pages

int code\_pages = code\_size/**PAGE\_SIZE**;

for (int i = 0; i <= code\_pages; ++i){

if (code\_size-**PAGE\_SIZE**\*i != 0){

**pages**.push\_back(code\_size- **PAGE\_SIZE**\*i);

output += "code for P" + std::to\_string(process) + ": " +**my\_memory**->allocate(process, 0)+ "\n";

}

}

//limit it by the defined page code\_size

//initially 512 Kb

int data\_pages = data\_size/**PAGE\_SIZE**;

for (int i = 0; i <= data\_pages; ++i ){

if (data\_size - **PAGE\_SIZE**\*i != 0){

**pages**.push\_back(data\_size - **PAGE\_SIZE**\*i);

output += "data for P" + std::to\_string(process) + ": " +**my\_memory**->allocate(process, 1)+"\n";

}

}

return output;

}

int page\_table::**getProcess**(){

return **process**;

}

void page\_table::**setPage**(int x){

this->**PAGE\_SIZE** = x;

}