**Ultima 2.0**

CS-435 Operating System

Phase 5 - Implementation and testing

By:

Kyle Lucas: [kylucas@iu.edu](mailto:kylucas@iu.edu)

Joshua Carter: [cartejos@iu.edu](mailto:cartejos@iu.edu)

Jordan Winkler: [jomawink@iu.edu](mailto:jomawink@iu.edu)

Team: Unix

4/27/20

**Table of Contents**

[**Abstract**](#_bvbugy5f4o3t) **4**

[**Phase Description**](#_a2zu7r79v62w) **5**

[**Design Diagrams**](#_38fz94llgr6x) **7**

[**Gantt chart**](#_raukei9cutw7) **7**

[**UML Models**](#_juxkev8u26ow) **8**

[**Ultima Overview**](#_ipl6cm3r2k7n) **8**

[**Program Execution**](#_klrcvhyrkrj8) **9**

[**The File System**](#_qeotgzale19l) **10**

[**Friendship Classes**](#_1qm210j2kqph) **11**

[**Class Descriptions**](#_ghmkuqwttzss) **12**

[**Main**](#_3sq1rqjs6xxl) **12**

[**OS**](#_seovizsii174) **13**

[**Task\_block**](#_vpzo8wltkk3n) **14**

[**Window**](#_jn9ex4gtvt88) **15**

[**Phase I classes**](#_uempizu0cnqm) **16**

[**Semaphore**](#_t9kjw6l1rl7a) **16**

[**Scheduler**](#_m6nbplfsxun6) **17**

[**Phase II classes**](#_wv1dzmrexxrq) **18**

[**Messenger**](#_6l0wsujnfguv) **18**

[**Message**](#_cd6tlfpbrt0k) **19**

[**Phase III classes**](#_qbey6swh6ixf) **20**

[**Memory**](#_t2zs4inac7b1) **20**

[**Page**](#_23vjdpyrq0vq) **21**

[**Phase IV classes**](#_i98pjay1cf1f) **22**

[**File\_system**](#_niavscey4bts) **22**

[**i\_node**](#_5btgy9mmaxlj) **23**

[**Source Code**](#_fxojpv1dvbja) **24**

[**README**](#_tt6szeka9hry) **24**

[**Main.cpp**](#_t9e563igfiwj) **26**

[**General.cpp**](#_qbt0mfnh74p1) **27**

[**Test.cpp**](#_qonpk3ii7gxa) **99**

[**Data Structures**](#_drffoye7nm30) **122**

[**Class: Binary\_search\_tree**](#_b6kc64ejvqaj) **122**

[**Class: Stack**](#_v27u72s78cb) **130**

[**Class: Queue**](#_v6rjai3sga8w) **137**

[**Makefile**](#_8gm5f9y9keee) **145**

[**Output**](#_yz65gk7nu733) **147**

[**Screen Dumps**](#_ebbfamdlaro) **147**

[**$ make**](#_qjwcfm8ab9ak) **147**

[**User creation & logon**](#_fkdee7mehvn0) **147**

[**Testing the file system**](#_xuod4g6qtuac) **148**

[**$ make test**](#_b7pu7t6yqkim) **149**

[**Needed Fixes/Future Features:**](#_tn325jxt4nhi) **151**

[**Test File**](#_kygtvql538jv) **151**

[**Scheduler Class**](#_hsr8iedwlwsc) **151**

[**Messenger Class**](#_r4zh5k1ggvf2) **151**

[**Memory Class**](#_fuv27von39xb) **151**

[**File System Class**](#_t322s6xu42ud) **151**

[**References:**](#_vbx9vj1qsmi3) **152**

# **Abstract**

In Phase V of the ULTIMA 2.0 operating system the goal for this section is to test and refine the code we have already written.

Much like the previous phase, phase IV depends on the completion and workability of the three previous phases. Without having this baseline our system would not be able to function correctly. Since memory is a critical portion of a system we need to implement a memory semaphore, implemented in phase I. This will complete our final phase of the project

# 

# **Phase Description**

Ultima File System(UFS): [[1]](#footnote-0)

Review and refine all documentation from phase I to phase IV. Expand and explain concepts, include drawing showing the inter and intra relationship between various OS components. Review and refine all code components submitted. Review and refine your test plans to show how each phase is working independently and as part of the integrated ULTIMA operating system. Make sure that every member of the group has similar responsibility in producing this phase and equally participates in the presentation.

# **Design Diagrams**

## Gantt chart

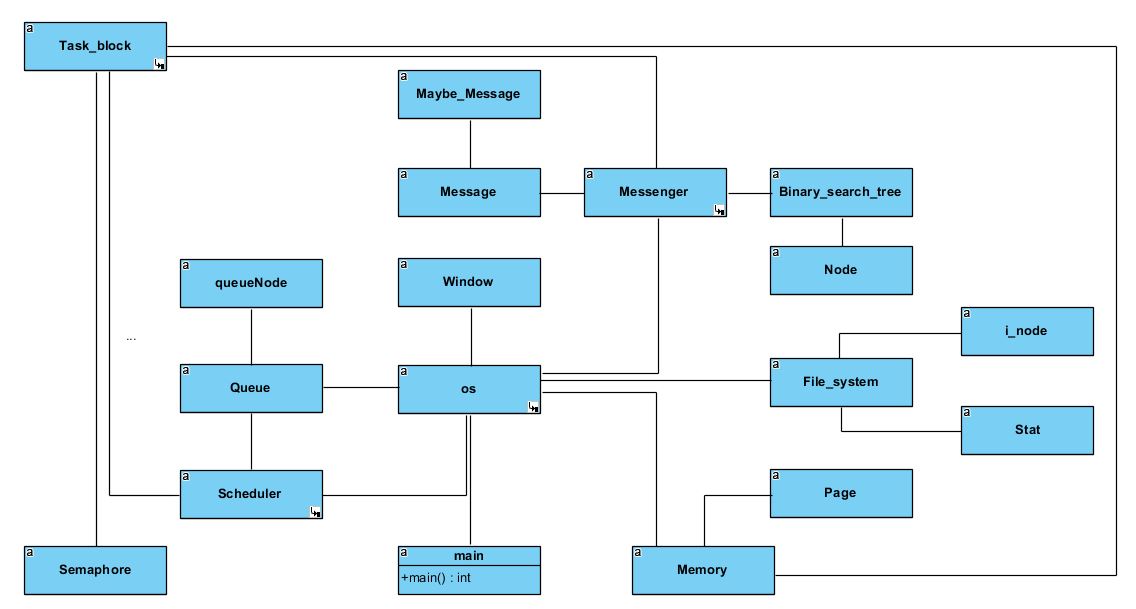
## 

## UML Models

These diagrams illustrate the use of the ULTIMA 2.0 operating system.

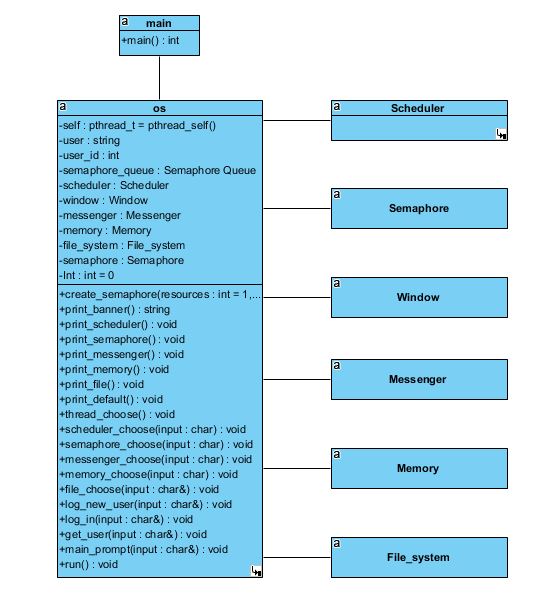
### Ultima Overview

Our Program executes when main() is called in main.cpp.

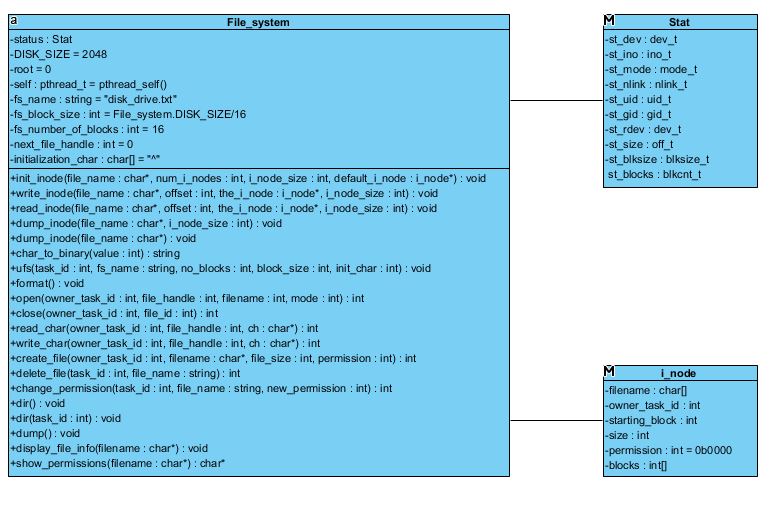


### Program Execution

The program executes when main() is called. Main() starts the OS class which creates a queue to hold the semaphores, and starts the Scheduler, Window, Messenger, Memory, and File\_system classes. In addition to this, the OS class begins the program by asking the user to log into the program and presents the user with a UI.

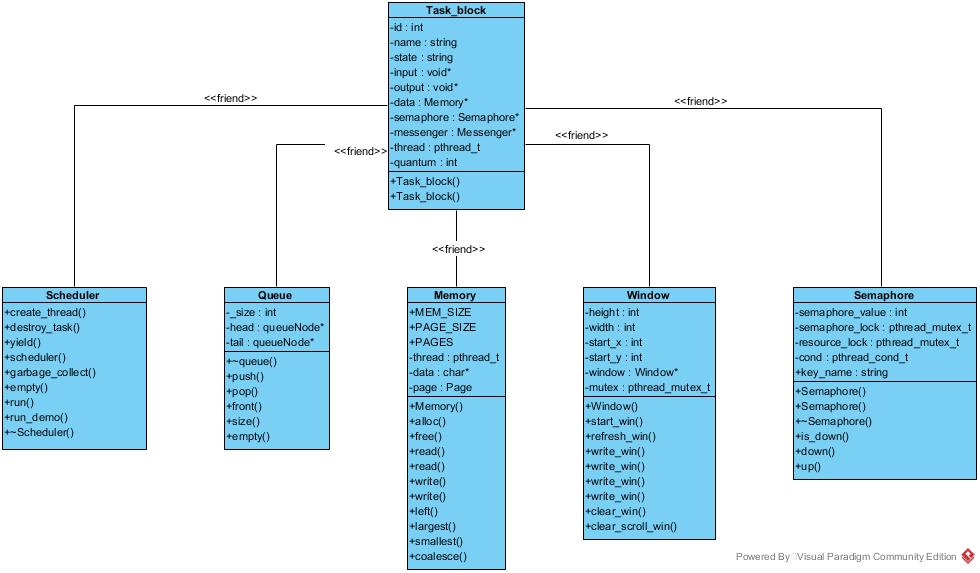


### The File System

The File System is implemented using a file management system that uses a 2K disk drive (2048 bytes). The file system works as a traditional file management system that uses functions such as read, write, open, close, or check the directory of the file system.

### Friendship Classes

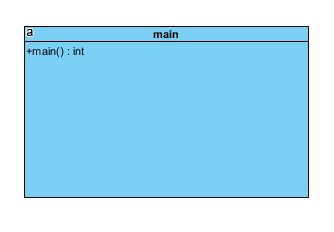
These relationships need to be made so classes can call methods of other classes.



## Class Descriptions

### Main

Main.cpp executes the program.



### 

### 

### OS

The OS class is responsible for managing aspects of the operating system which keep it running properly. The OS class creates a queue to hold the semaphores, and starts the Scheduler, Window, Messenger, Memory, and File\_system classes. In addition to this, the OS class begins the program by asking the user to log into the program and presents the user with a UI.



### Task\_block

Task\_block is the class which manages most of the operations of Ultima 2.0. It is responsible for managing the memory, the messages and the semaphores.

### Window

The Window class is responsible for outputting information to the screen.

### 

### Phase I classes

Phase I of the ULTIMA 2.0 operating system is the inception of the project and created a task scheduler and a semaphore class. Each of these classes were needed to form the foundation of the operating system, and both of these classes would be expanded upon in later phases.

#### Semaphore

A semaphore is simply a variable which is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment. Semaphores are of two types, binary and counting. ULTIMA 2.0 makes use of a binary semaphore in the form of a mutex, which is simply an integer which can have only two values: 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.

Semaphores are created within this class and managed by Task\_block. In ULTIMA 2.0, semaphores are used to manage communication between threads, and other purposes when the critical region is being accessed.

### 

#### Scheduler

The scheduler is responsible for creating threads, destroying threads, and garbage collection.

### 

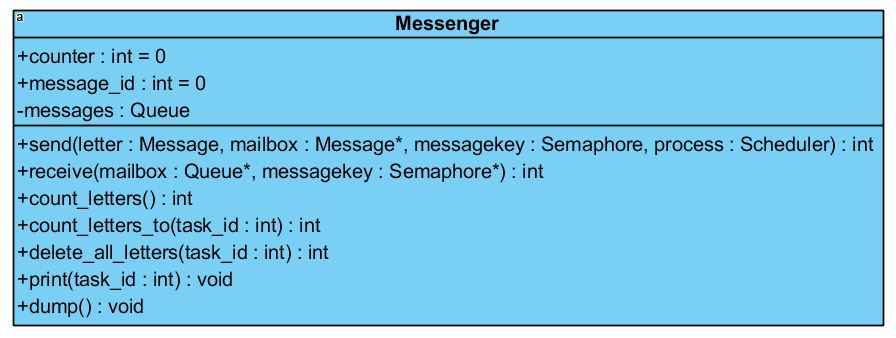
#### 

### Phase II classes

Phase II of the project adds inter-process communication to the operating system, and to handle this the Messenger and Message classes are added to the program. These classes, in addition with the Task\_block and semaphore classes, handle IPC between threads.

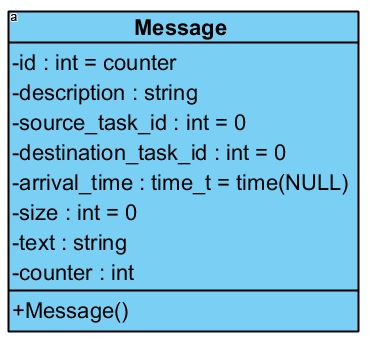
#### Messenger

The Messenger class is responsible for sending and receiving messages. Messages are stored in a queue, and the messenger can destroy all or some of the messages. The messenger is managed through the Task\_block class which solves race conditions during IPC.



#### Message

The Message class is responsible for storing all of the information about messages which are passed by the Task\_block (IPC).



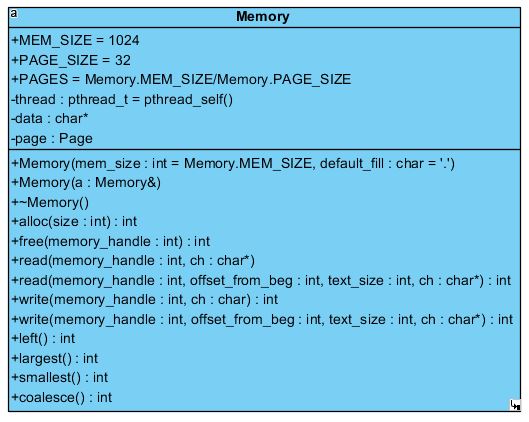
### 

### Phase III classes

Phase III adds memory management to the program and Memory and Page classes are added to create the memory, and the Task\_block class will manage the memory.

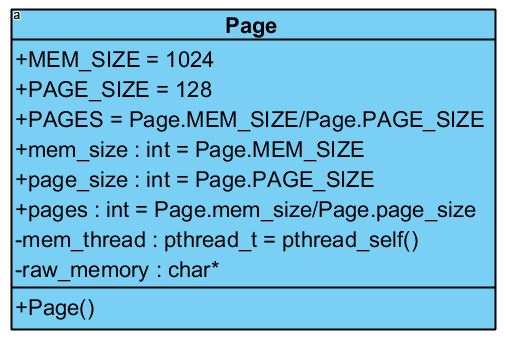
#### Memory

Memory is responsible for managing all of the memory requests, as well as memory allocation, memory freeing, and coalescing of open pages. The default page size is 32 bytes, with a total memory allocation of 1024 bytes.



#### Page

The Page class is responsible for managing the pages associated with the memory allocation class, Memory.



### Phase IV classes

Phase IV of the program adds the ability to manage a file system and File\_system and i\_node classes are added to support this. The file system has functions to perform standard operations such as creating a file, deleting one, or changing permissions.

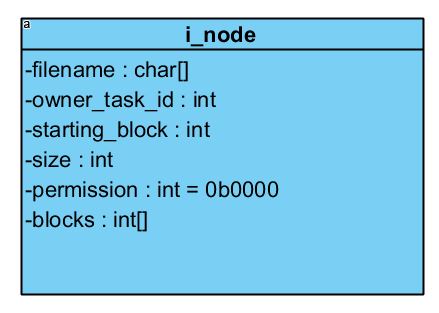
#### File\_system

The File\_system class is responsible for handling the files in ULTIMA 2.0. It uses the struct Stat and an i\_node system to keep track of the files and permissions, and which user is accessing the file. This should work with two different files, one of which is a 2K filesystem of 128 byte blocks and the other an I-node structure. Our file system has filesystem operations, file operations, and directory operations. Output of the filesystem will display a filehandle(file pointer), file name, blocks used, size, starting block, status, permissions of the file, and the owner ID.

### 

#### i\_node

i\_node is a subclass of the File\_system class and stores information about files such as owner, file name, location, size, and permissions.



### 

# **Source Code**

## README

//Overview

# Jordan Winkler, Joshua Carter, Kyle Lucas

# ULTIMA 2.0

# Welcome to our semester long group project dubbed ULTIMA 2.0

# During this project we have split into four different phases:

# Phase 1 - Semaphore and Scheduler

# .Design a scheduler and semaphore to help with process management

# Phase 2 - IPC (Inter process communication)

# .Create a mailbox system to pass message/communications between tasks

# Phase 3 - Memory Management

# .Set up a 1k memory system

# Phase 4 - File System

# .Make a 2k disk drive to work as a filesystem

# All these phases will lead into phase 5 which will be a demo of the project

//README

ULTIMA 2.0 is a simulate operating system that we have design to act like a

real operating system would. This OS comes with a semaphore and scheduler to assist

in creating and working with tasks. A inter process communication system that we have

decided to name, mailbox, that can send and receive messages from other tasks in the

system. A memory management system that has 1k of space available to give tasks/file/etc.

We have also tried to design a disk drive that is user friendly and allows you to open,

read, write, delete, see,...., for different files.

Like most user designed tools/objects they come with some bugs so please share any found

bugs with the users so we can attempt to patch them up.

//INSTALLATION

Please use the provided source files to run the code. Simply add the files to your

desired directory then compile the code using a makefile, also provided.

//COMPILATION

Once all the files have been added to the current directory you can run the program

as such:

user$: make

//run the main program

//follow on screen menus to run through different parts

user$: make test

//run all of the various test cases that comprise test.cpp

user$:make clean

//creates clean versions of the files and removes old ones

//ERROR?

If any errors occur during compilation some additional packages may need to be

installed by the users. If errors still occur follow this step please refer to

the contact section.

//CONTACT

If you have any problems, questions, or comments please contact use directly

at the email address of the authors listed above.

## 

## Main.cpp

/\*

Jordan Winkler, Kyle Lucas, Joshua Carter

Sun Apr 12 10:41:52 EDT 2020

Phase 1: Scheduler and Semaphore

Phase 2: interprocess communication

Phase 3: memory

Phase 4: file system

\*/

#include "general.cpp" // code file

int main()

{

// make some processes

os.scheduler.create\_task("bob", stub);

os.scheduler.create\_task("Givin", stub);

// show what processes running and dying looks like

os.run();

pthread\_exit(NULL); // used instead

//return 0;

}

/\*

\*/

## General.cpp

/\*

Jordan Winkler, Joshua Carter, Kyle Lucas

Fri Apr 24 14:00:04 EDT 2020

A project designed to demonstrate operating systems concepts.

Runs on WSL windows 10, Ubuntu, Raspian, Debian

make run #compiles then runs the program

make test #compiles then runs all of the tests

Phase 1: Scheduler and Semaphore

Phase 2: interprocess communication

Phase 3: memory

Phase 4: file system

Phase 5: OS (window, demo)

\*/

// pre-compiler toggles and global variables

//{

// optional toggle for self hosting design

// else allow for heap memory calls and other possible OS

// dependent calls

// Note: not completely done, so the system is mostly self hosting

// making this with c++ makes it harder to track memory use

#define SELF\_HOSTING 1

#undef SELF\_HOSTING

// for gnu c++ specific constructs like pthread\_yield

#ifndef \_\_GNUC\_\_

// pthread\_yield is not posix compliant, but temporary pauses are

// through scheduler that comes with posix

#include <sched.h>

int pthread\_yield()

{

return sched\_yield();

}

#endif

//}

// libraries used

//{

// ^- set fold method to {,} to get a better look at code

#include <ctime>

#include <cmath>

#include <cstdio>

// sleep

#ifdef \_WIN32

#include <Windows.h>

#else

#include <unistd.h>

#endif

// used in windows class, c library

#include <ncurses.h>

// used to multithread in scheduler and semaphore, c library

#include <pthread.h>

// used for testing

#include <assert.h> // used to do basic testing, c library

#include <iostream> // the stream classes are easy to use for testing

#include <fstream> // file streams

#include <sstream> // conversion of x to string class

#include <string.h> // c string functions

#include <string> // string class

// added to handle file operations

#include <sys/sysmacros.h>

#include <sys/types.h>

#include <sys/stat.h>

//--- not in use yet ---//

//#include <editline/readline.h>

//#include <editline/history.h>

// more compliant c++ wrapper that is part of the c++ standard libs

//#include <thread>

// Quality of life improvement over function pointers

//#include <functional> // makes function variables look nicer

// Standard namespaces are tedious to type, and will not be an issue in this project.

using namespace std;

//}

// make a string of any registered object

// note: loops twice in gdb?

string str(auto& a)

{

stringstream ss;

ss << a;

return ss.str();

}

// c++ is odd about this method for primitives

string str(int a)

{

stringstream ss;

ss << a;

return ss.str();

}

/\*{===========================================================

||

|| Class: Window

||

|| Purpose: an api over WINDOW from ncurses

||

|| Attributes:

|| height

|| width

|| x

|| y

|| \*window

|| mutex

||

|| Functions:

|| constructor(set defaults)

|| start\_win()

|| write\_win(window, y, x, string)

|| write\_win(window, string)

|| write\_win(window, y, x, int)

|| write\_win(window, int)

|| task\_win()

|| task\_win()

|| clear\_win()

|| clear\_window()

|| refresh\_win()

\\*=========================================================}\*/

class Window

{

public:

// super user is creator

pthread\_t self = pthread\_self();

int height;

int width ;

int x;

int y;

WINDOW\* window; // to avoid copy construction

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

friend ostream& operator<<(ostream& os, const Window& w)

{

os << "(Window)";

return os;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Window (constructor)

|

| Purpose: set defualts

|

| Precodition: nothing

| Postcondition: windows object created

|

| Parameters:

|

| IN: hieght, width, x, y

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void create(int pH = 0, int pW = 0, int pY = 0, int pX = 0)

{

pthread\_mutex\_lock(&mutex);

// ---Entering a critical region in code---

height = pH;

width = pW;

y = 1;//pY;

x = 1;//pX;

window = newwin(height, width, pY, pX);

box(window, 0, 0);

scrollok(window, true);

// ---Leaving a critical region---

pthread\_mutex\_unlock(&mutex);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: refresh\_win

|

| Purpose: refreshes the screen to update information displayed

|

| Precodition: window needs to exist

| Postcondition: nonthing

|

| Parameters:

|

| IN: window wanting to refresh

| Out: nothing

|

| Returns: nothing, but screen gets refreshed

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void refresh()

{

pthread\_mutex\_lock(&mutex);

// ---Entering a critical region in code---

wrefresh(window);

// ---Leaving a critical region---

pthread\_mutex\_unlock(&mutex);

}

// scroll the text on the window

// scroll is allowed to be a macro in some implementations

// but those are C implementations. Overloading works instead.

#undef scroll

void scroll(int n = 1)

{

pthread\_mutex\_lock(&mutex);

// ---Entering a critical region in code---

wscrl(window, n);

box(window, 0, 0);

// ---Leaving a critical region---

pthread\_mutex\_unlock(&mutex);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: write\_win

|

| Purpose: write the passed string to specific window using x/y

| coordinates to determine placement

|

| Precodition: window needs to exist

| Postcondition: new display to the screen with passed text

|

| Parameters:

|

| IN: window trying to write to, x placement on screen, y

| placement on screen, string to display

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void print(string str\_text, int y, int x)

{

pthread\_mutex\_lock(&mutex);

// ---Entering a critical region in the code---

const char\* text = str\_text.c\_str();

mvwprintw(window, y, x, text);

box(window, 0, 0);

wrefresh(window);

// ---Leaving a critical region---

pthread\_mutex\_unlock(&mutex);

}

void print(string

str\_text) // c++ has some warts, can't default value member values

{

//int y = y + 1, x = x + 1;

print(str\_text, y, x);

}

void println(string str\_text)

{

print(str\_text, y, x);

if(y < LINES)

{

y++;

}

else

{

scroll();

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: write\_win

|

| Purpose: write to a window on the screen given x,y coordinates

| and display an integer, converted to a string

|

| Precodition: window needs to exist to write to it

| Postcondition: newly displayed window appears to the screen

| with passed integer converted to string format

|

| Parameters:

|

| IN: x coordinate, y coordinate for placement, number(integer)

| to be displayed, and window to write too

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void print(int pNum, int y, int x)

{

pthread\_mutex\_lock(&mutex);

// ---Entering a critical region in code---

string str\_text = to\_string(pNum);

const char\* text = str\_text.c\_str();

mvwprintw(window, y, x, text);

box(window, 0, 0);

//wrefresh(window);

// ---Leaving a critical region---

pthread\_mutex\_unlock(&mutex);

}

void print(int pNum)

{

//int y = y + 1, x = x + 1;

print(pNum, y, x);

}

void println(int pNum)

{

//int y = y + 1, x = x + 1;

print(pNum, y, x);

if(y < LINES)

{

y++;

}

else

{

scroll();

}

}

void operator<<(auto& a)

{

stringstream ss;

ss << a;

print(ss.str());

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: clear\_win

|

| Purpose: Pass a window that you want to clear out the information

| from and refresh the window so it is cleared

|

| Precodition: window needs to exist to clear it

| Postcondition: cleared out window will now display

|

| Parameters:

|

| IN: window you are wanting to 'clear' information from

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void clear()

{

pthread\_mutex\_lock(&mutex);

// ---Entering a critical region in code---

wclear(window);

box(window, 0, 0);

wrefresh(window);

// reset these guys

x = 1;

y = 1;

// ---Leaving a critical region---

pthread\_mutex\_unlock(&mutex);

}

//friend class MCB;

}; // end of window class

// global instances

#define THREAD\_COUNT 3

Window top\_window;

Window thread[THREAD\_COUNT];

Window terminal;

Window dump\_screen;

// pushed to the top of code for self hosting memory option

/\*{===========================================================

||

|| Class: Memory

||

|| Purpose: give task control blocks some memory for their

|| functions/threads

||

|| Note: For now memory constructs then sleeps. To prevent

|| races, use semaphores and the scheduler

||

|| Attributes:

|| memory block

|| data

|| constructor

|| destructor

|| << (as a friend)

|| queue of memory blocks

||

|| Functions:

|| Memory

|| Memory

|| ~Memory

|| alloc

|| free

|| read

|| read

|| write

|| write

|| left

|| largest

|| smallest

|| coalesce

|| << (as a friend)

||

\\*=========================================================}\*/

class Memory // mem\_mgr

{

public:

#ifndef SELF\_HOSTING

#define MEM\_SIZE 1024

#define PAGE\_SIZE 128

#define PAGES (MEM\_SIZE/PAGE\_SIZE)

int mem\_size = MEM\_SIZE;

int page\_size = PAGE\_SIZE;

int pages = (mem\_size / page\_size);

#else

#define MEM\_SIZE 1024

#define PAGE\_SIZE 128

#define PAGES (MEM\_SIZE/PAGE\_SIZE)

#endif

//private:

// initializer owns it

pthread\_t mem\_thread = pthread\_self();

#ifndef SELF\_HOSTING

char\* raw\_memory; // pointer to raw memory

#else

char raw\_memory[MEM\_SIZE];

#endif

/\*{===========================================================

||

|| Class: Page

||

|| Purpose: create a 'page' structure to hold chunks of memory

||

|| Note: Can change the size of pages to have different memory

|| chunk sizes, must be done in above statements

||

|| Attributes:

|| name

|| end

|| free\_size

||

|| Functions:

|| Page

\\*=========================================================}\*/

class Page // a structure overlying memory

{

public:

// only name on first node is read

#ifndef SELF\_HOSTING

pthread\_t\* name = new pthread\_t[PAGES];

int\* end = new int[PAGES];

// meta data, size of free block

int\* free\_size = new int[PAGES];

#else

pthread\_t name[PAGES];

int end[PAGES];

// meta data, size of free block

int free\_size[PAGES];

#endif

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Page

|

| Purpose: Assign pages to Memory so it can give that to

| tasks or functions

|

| Precondition: nothing

| Postcondition: pages can now be used and assigned to

| tasks

|

| Parameters:

|

| IN: unsigned pages

| OUT: pages no assigned a size and able to use

|

| Returns: pages of memory

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Page()

{

// label all pages, then give a size value

int i = 0;

for(; i < PAGES ; i++)

{

// meaning is owned by OS or Memory

name[i] = pthread\_self();

end[i] = 0; // default

}

for(int j = 0 ; j < PAGES ; j++, i--)

{

free\_size[j] = i;

}

}

};

Page page;

// functions

//{

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Memory

|

| Purpose: default constructor that makes memory the right size

| and blanks out the bits. Uses heap memory by request

|

| Precondition: nothing

| Postcondition: Memory has been initialized to the right size

| and is now ready to be used

|

| Parameters:

|

| IN: size of the memory, character to fill memory block with

| OUT: a chunk of memory the size of memory passed

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Memory(int mem\_size = MEM\_SIZE, char default\_fill = '.')

{

#ifndef SELF\_HOSTING

raw\_memory = new char[mem\_size];

pages = (mem\_size / PAGE\_SIZE);

#endif

for(int i = 0 ; i < mem\_size ; i++)

{

raw\_memory[i] = default\_fill;

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Memory

|

| Purpose: copy constructor for the memory function

| Note: currently not needed but made just in case

|

| Precodition: nothing

| Postcondition: memory is copied

|

| Parameters:

|

| IN: Memory reference

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// copy constructor and destructor currently not needed

Memory(const Memory& a)

{

#ifndef SELF\_HOSTING

raw\_memory = new char[a.mem\_size];

#endif

for(int i = 0 ; i < MEM\_SIZE ; i++)

{

raw\_memory[i] = a.raw\_memory[i];

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: ~Memory

|

| Purpose: must be defined so data structures are released

| Note: currently not needed or used but implemented

|

| Precodition: nothing

| Postcondition: deallocates the memory chunk

|

| Parameters:

|

| IN: nothing

| Out: chunk of memory that had been set aside is now gone

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

~Memory()

{

#ifndef SELF\_HOSTING

delete[](raw\_memory);

raw\_memory = NULL;

#endif

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: alloc

|

| Purpose: allocate the number of pages, or chunks of memory

| requested by the caller. size determined by passed

| 'size' variable

|

| Precodition: memory needs to exist to allocate too

| Postcondition: if the size requested is available, the task

| will get that size of memory back

|

| Parameters:

|

| IN: size of memory requesting, relates to page size which are

| set to be 128 byte blocks

| Out: size of memory requested if enough is free

|

| Returns: allocated memory size

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// similar to C version of malloc

int alloc(int size)

{

// allocate pages

int blocks\_to\_alloc = ceil(((double) size) / PAGE\_SIZE);

// get threads name

pthread\_t process\_thread = pthread\_self();

//pause calling function

// Direct page allocation, go

int mem\_dist = 0; // amount of free memory

int mem\_handle\_top = 0; // top of memory

int mem\_handle\_bot = 0; // bottom of memory

for(; mem\_handle\_top < blocks\_to\_alloc ; mem\_handle\_top++)

{

if(pthread\_equal(page.name[mem\_handle\_top], mem\_thread))

{

mem\_dist++;

mem\_handle\_top++;

}

if(mem\_dist == size)

{

for(mem\_handle\_bot = mem\_handle\_top ;

mem\_handle\_top - mem\_handle\_bot < mem\_dist ;

mem\_handle\_bot--)

{

page.name[mem\_handle\_bot - 1] = process\_thread;

page.free\_size[mem\_handle\_bot] = 0;

}

break;

}

}

// unpause calling function

// no space

if(mem\_dist == 0)

{

return -1;

}

// give a top pointer on allocation

page.end[mem\_handle\_bot] = mem\_dist;

return mem\_handle\_bot;

// make space for error or memory handle

//int error\_or\_memory\_handle;

// block thread

// start up os

// look for that many blocks

// if there is space, add threads name to all those blocks

// else make handle -1

// wake up process

// return index to first block, or error code

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: free

|

| Purpose: Take a page pointer, memory\_handle, to point to in use

| memory then deallocates that memory and free's it

|

| Precodition: the block of memory your trying to free needs to

| already exist and be allocated

| Postcondition: the page, or block of memory, has been freed

|

| Parameters:

|

| IN: memory smart pointer to where a memory block is allocated

| Out: memory was freed and '#' was put in the block of memory

|

| Returns: Returns true when the memory has been freed and filled

| with '#' in the freed memory

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// similar to free for C language

int free(int memory\_handle)

{

// pause calling function

for(int i = memory\_handle ;

i < memory\_handle + page.end[memory\_handle]; i++)

{

page.name[i] = mem\_thread;

page.end[i] = 0;

// by spec, prevent sensitive info leaks

for(int j = i \* PAGE\_SIZE ; j < (i + 1)\*PAGE\_SIZE ; j++)

{

raw\_memory[j] = '#';

}

}

// unpause calling function

return true;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: read

|

| Purpose: takes a pointer to a memory block and reads the memory

|

| Precodition: memory to read information from needs to be created

| Postcondition: the memory was read or returned an error saying

| the delete failed.

|

| Parameters:

|

| IN: memory pointer to what to read and a pointer to read the data

| Out: reads the information in the memory

|

| Returns: true if it was able to read the memory or -1 if an error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int read(int memory\_handle, char\* ch)

{

if(page.name[memory\_handle] == pthread\_self())

{

\*ch = raw\_memory[memory\_handle \* PAGE\_SIZE];

return true;

}

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: read

|

| Purpose: same as previous read but will check ownership first

|

| Precodition: memory block being read needs to be created

| Postcondition: memory was read and returned what it was

|

| Parameters:

|

| IN: memory pointer to what is to be read, page reference for

| how many are left, size of text, and pointer to read the data

| Out: reads the information in memory if it is the task

|

| Returns: true if it was able to read the memory or -1 if an error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int read(int memory\_handle, int offset\_from\_beg,

int text\_size, char\* ch)

{

// rename the computations

int begin\_page = memory\_handle;

int end\_page = ceil((memory\_handle \* PAGE\_SIZE +

offset\_from\_beg + text\_size) / PAGE\_SIZE);

int begin\_raw = memory\_handle \* PAGE\_SIZE + offset\_from\_beg;

int end\_raw = memory\_handle \* PAGE\_SIZE +

offset\_from\_beg + text\_size;

// check if that memory belongs to the process

for(int i = begin\_page; i < end\_page; i++)

{

if(page.name[i] != pthread\_self())

{

return -1; // another process owns a page

}

}

// allow change

for(int i = begin\_raw ; i < end\_raw; i++)

{

ch[i - begin\_raw] = raw\_memory[i];

}

return true;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: write

|

| Purpose: write to the memeory when a task says it wants to

| save that information to memory

|

| Precodition: have a task request that it would like to write

| to memory

| Postcondition: memory block now has information written to it

|

| Parameters:

|

| IN: memory pointer to where to write and pointer of what to write

| Out: written information to memory

|

| Returns: true if it was able to write to memory or -1 if error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int write(int memory\_handle, char ch)

{

if(page.name[memory\_handle] == pthread\_self())

{

raw\_memory[memory\_handle \* PAGE\_SIZE] = ch;

return true;

}

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: write

|

| Purpose: write to the memory when a task says it wants to write

| to memory, checks if the task owns that memory

|

| Precodition: have a task request that it would like to write

| to memory

| Postcondition: memory block now has information written to it

|

| Parameters:

|

| IN: memory pointer to where to write, page reference for how

| many are left, size of text, pointer to write the data

| Out: written information to memory

|

| Returns: true if it was able to write to memory or -1 if error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int write(int memory\_handle, int offset\_from\_beg,

int text\_size, char\* ch)

{

// rename the computations

int begin\_page = memory\_handle;

int end\_page = ceil((memory\_handle \* PAGE\_SIZE +

offset\_from\_beg + text\_size) / PAGE\_SIZE);

int begin\_raw = memory\_handle \* PAGE\_SIZE + offset\_from\_beg;

int end\_raw = memory\_handle \* PAGE\_SIZE +

offset\_from\_beg + text\_size;

// check if that memory belongs to the process

for(int i = begin\_page; i < end\_page; i++)

{

if(page.name[i] != pthread\_self())

{

return -1; // another process owns a page

}

}

// allow change

for(int i = begin\_raw ; i < end\_raw; i++)

{

raw\_memory[i] = ch[i - begin\_raw];

}

return true;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: left

|

| Purpose: checks the number of pages and says how much memory

| is left

|

| Precodition: have memory allocated

| Postcondition: says how much memory is still available, determined

| by pages

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: top variable which is the number of pages, memory, left

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int left()

{

// checking the raw memory is the same complexity

// as checking the list

int mem\_left = 0;

for(int i = 0 ; i < PAGES ; i++)

{

if(page.name[i] == mem\_thread)

{

mem\_left++;

}

}

return mem\_left;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: largest

|

| Purpose: which the memory to see how many pages are available

| to determine the largest size of memory available

|

| Precodition: have memory allocated

| Postcondition: determines how many pages are used and how many

| are left to find the available memory

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: top variable which holds the available memory size

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int largest()

{

// limit infimum method

int top = 0;

int temp\_mem\_block = 0;

for(int i = 0 ; i < PAGES ; i++)

{

if(page.name[i] == mem\_thread)

{

temp\_mem\_block++;

}

else

{

temp\_mem\_block = 0;

}

if(temp\_mem\_block > top)

{

top = temp\_mem\_block;

}

}

return top;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: smallest

|

| Purpose: return the smallest block of memory that is available

|

| Precodition: have memory allocated

| Postcondition: determines what the smallest block of memory that

| can be allocated is

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: lessers variable which holds the smallest amount available

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int smallest()

{

int lesser = PAGES;

int temp\_mem\_block = 0;

for(int i = 0 ; i < PAGES ; i++)

{

if(page.name[i] == mem\_thread)

{

temp\_mem\_block++;

}

else

{

if(lesser > temp\_mem\_block and temp\_mem\_block != 0)

{

lesser = temp\_mem\_block;

}

temp\_mem\_block = 0;

}

}

if(lesser > temp\_mem\_block and temp\_mem\_block != 0)

{

lesser = temp\_mem\_block;

}

return lesser;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: coalesce

|

| Purpose: combine meta data on free blocks of memory

| does not burp memory, given our page block model

|

|

| Precodition: have memory allocated

| Postcondition: memory blocks are now 'joined'

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: true is returned if completed

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int coalesce()

{

for(int i = PAGES - 1, j = 0 ; j < PAGES ; j++, i--)

{

if(page.name[i] == pthread\_self())

{

page.free\_size[j] = i;

}

}

return true;

}

//}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: <<

|

| Purpose: allow this class to be printed and converted into

| a string for testing

|

| Precodition: nothing

| Postcondition: the memory object was added to the stream

|

| Parameters:

|

| IN: stream mechanism, memory block

| Out: stream mechanism, memory block

|

| Returns: stream mechanism

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

friend ostream& operator<<(ostream& os,

const Memory& mem)

{

os << "(Memory ";

for(int i = 0 ; i < MEM\_SIZE ; i++)

{

os << mem.raw\_memory[i];

}

os << ')';

return os;

}

void dump()

{

dump\_screen.println("Status | Memory | Starting | Ending | Size/ | Current | Task-ID");

dump\_screen.println(" | Handle | Location | Location | Bytes | Location | ");

string line;

if (not pthread\_equal(mem\_thread, pthread\_self()))

line += "Free |";

else

line += "Used |";

// TODO: calculate all values and put them in

line += " nil | nil | nil | nil | nil | nil ";

dump\_screen.println(line);

}

void core\_dump()

{

// TODO: make this fit on screen

string raw;

for(int i = 0 ; i < MEM\_SIZE ; i++)

{

raw += raw\_memory[i];

}

dump\_screen.println(raw);

}

public:

// MCB is for testing programs

friend class MCB;

// let these pass to test block as global

//#undef MEM\_SIZE

//#undef PAGE\_SIZE

//#undef PAGES

};

// past this point memory still heap allocates

#undef SELF\_HOSTING

// data structures

//{

/\*{===============================================================

||

|| Class: binary search tree

||

|| Purpose: make message class faster

||

|| Attributes:

|| Nodes (elements of tree)

|| datum

|| left, right pointers

|| root (location of base)

||

|| Functions:

|| initializer

|| deconstructor

|| insert

|| delete\_branch

|| search

|| minimum

|| maximum

|| preorder

|| inorder

|| postorder

|| print (default inorder)

||

|| Friends with:

|| Test\_suite

||

\\*=============================================================}\*/

template<class type> class Binary\_search\_tree

{

private:

struct Node

{

type datum;

Node\* left, \*right;

};

Node\* root;

public:

Binary\_search\_tree()

{

root = NULL;

}

~Binary\_search\_tree()

{

delete\_branch(root);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: insert

|

| Precodition: nothing

| Postcondition: tree has new leaf

|

| Parameters:

|

| IN: datum of the declare binary tree type

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void insert(type datum)

{

// Ask for memory

#ifndef SELF\_HOSTING

Node\* new\_node = new Node;

#else

Node\* new\_node = (Node\*) memory.alloc(sizeof(

Node) / PAGE\_SIZE);

#endif

new\_node->datum = datum;

new\_node->right = NULL;

new\_node->left = NULL;

// start at the base, get into position

Node\* p = root;

Node\* trail\_p = NULL;

while(p != NULL)

{

trail\_p = p;

if(new\_node->datum < p->datum)

{

p = p->left;

}

else

{

p = p->right;

}

}

// Assign the data

if(trail\_p == NULL) // tree was empty

{

root = new\_node;

root->datum = new\_node->datum;

}

else if(new\_node->datum < trail\_p->datum)

{

trail\_p->left = new\_node;

}

else

{

trail\_p->right = new\_node;

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: delete\_branch

|

| Precodition: nothing

| Postcondition: section of binary search tree is removed

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void delete\_branch(Node\* node)

{

if(node == NULL)

{

return;

}

delete\_branch(node->right);

delete\_branch(node->left);

delete node;

node = NULL;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: search

|

| Precodition: nothing

| Postcondition: location of data, if it is on the tree

|

| Parameters:

|

| IN: node of binary search tree, datum of search

| Out: nothing

|

| Returns: location of datum or NULL

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Node\* search(Node\* node, type key)

{

if(node == NULL or key == node->datum)

{

return node;

}

if(key < node->datum)

{

return search(node->left, key);

}

else

{

return search(node->right, key);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: minimum

|

| Precodition: nothing

| Postcondition: location of minimum datum data has been found

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: location of min datum or NULL

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Node\* minimum(Node\* node)

{

while(node->left != NULL)

{

node = node->left;

}

return node;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: maximum

|

| Precodition: nothing

| Postcondition: location of maximum datum data has been found

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: location of max datum or NULL

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Node\* maximum(Node\* node)

{

while(node->right != NULL)

{

node = node->right;

}

return node;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: preorder

|

| Precodition: nothing

| Postcondition: preorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void preorder(Node\* node)

{

if(node != NULL)

{

dump\_screen.println(str(node->datum));

preorder(node->left);

preorder(node->right);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: inorder

|

| Precodition: nothing

| Postcondition: inorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void inorder(Node\* node)

{

if(node != NULL)

{

inorder(node->left);

dump\_screen.println(str(node->datum));

inorder(node->right);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: postorder

|

| Precodition: nothing

| Postcondition: postorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void postorder(Node\* node)

{

if(node != NULL)

{

postorder(node->left);

postorder(node->right);

dump\_screen.println(str(node->datum));

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: print

|

| Precodition: nothing

| Postcondition: inorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void print()

{

inorder(root);

}

friend class Test\_suite;

};

/\*{===========================================================

||

|| Class: Stack<datum\_type>

||

|| Purpose: A handcrafted textbook Stack, easy data

|| structure to work with.

||

|| Attributes:

|| Node

|| TYPE Element

|| Node\* next

|| Top

|| stack\_size

||

|| Functions:

|| Stack

|| ~Stack

|| reset

|| empty

|| push

|| pop

|| peek

|| print

||

\\*=========================================================}\*/

template <class TYPE> class Stack

{

struct Node // each node holds an element of type TYPE

{

TYPE datum;

Node\* next;

};

Node\* top;

int \_size;

public:

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Stack

|

| Summary: This is a simple constructor

|

| Precodition: none

| Postcondition: stack is created

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Stack()

{

top = NULL;

\_size = 0;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: ~Stack

|

| Summary: This is a simple deconstructor

|

| Precodition: there is a Stack

| Postcondition: that Stack memory is freed

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

~Stack()

{

reset();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: push

|

| Summary: take a value and push/add it to the stack

|

| Precodition: there is a stack

| Postcondition: the value has been added to the stack

|

| Parameters:

|

| IN: type and value to add to the stack

| Out: stack size increases

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void push(TYPE& value)

{

Node\* Temp\_node = new Node;

if(Temp\_node == NULL)

{

dump\_screen.println("Push: memory allocation error");

return; //exit(1);

}

Temp\_node->datum = value;

Temp\_node->next = top;

top = Temp\_node;

\_size++;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: pop

|

| Summary: if there is an element pop it from the stack

|

| Precodition: there is a stack

| Postcondition: the element is popped from the stack

|

| Parameters:

|

| IN: nothing

| Out: popped element

|

| Returns: element was popped from the stack and returned

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

TYPE pop()

{

TYPE Return\_datum;

if(top != NULL)

{

Node\* Temp\_node = top;

Return\_datum = top->datum;

top = top->next;

delete Temp\_node;

\_size--;

}

return(Return\_datum);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: size

|

| Summary: return the size of the stack

|

|

| Precodition: there is a stack

| Postcondition: the size of the stack is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: size of stack

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int size()

{

return \_size;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: reset

|

| Summary: pops all the elements of the stack to empty it

|

| Precodition: there is a stack

| Postcondition: the stack is now empty

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void reset()

{

while(empty() != TRUE)

{

pop();

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: empty

|

| Summary: checks to see if any elements are in the stack

|

| Precodition: there is a stack

| Postcondition: if the stack is empty

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: true or false depending if anything is in the stack

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int empty()

{

if(\_size == 0)

{

return(true);

}

else

{

return(false);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: peek

|

| Summary: if there is an element return what the top elements

| value is

|

| Precodition: there is a stack

| Postcondition: the top elements value is returned

|

| Parameters:

|

| IN: nothing

| Out: top elements value

|

| Returns: element at the tops of the stacks value

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

TYPE peek()

{

TYPE Return\_datum;

if(top != NULL)

{

Return\_datum = top->datum;

}

return (Return\_datum);

}

// needs to be made to conform to rest of code

friend ostream& operator<<(ostream& os, Stack<TYPE>& q)

{

Stack::Node\* temp = q.top;

if(q.\_size == 0)

{

os << "(Stack)";

}

else

{

os << "(Stack ";

while(temp->next != NULL)

{

os << (temp->datum) << ' ';

temp = temp->next;

}

os << temp->datum;

os << ")";

}

return os;

}

};

/\*{===========================================================

||

|| Class: Queue<datum\_type>

||

|| Purpose: A handcrafted textbook Queue, easy data

|| structure to work with.

||

|| Attributes:

|| Node

|| Q datum

|| Node\* next

|| \_size

|| head

|| tail

||

|| Functions:

|| ~Queue

|| push

|| pop

|| front

|| size

|| empty

|| << (is a friend)

|| >> (is a friend)

|| (friend) MCB

||

\\*=========================================================}\*/

template<class Q> class Queue

{

private:

// block of Queue

struct Node

{

Q datum;

Node\* next = NULL;

};

// Queue knows the number of Nodes it has

int \_size = 0;

// For speed of access Queue's always know of its head

// and tail

Node\* head = NULL;

Node\* tail = NULL;

public:

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: ~Queue

|

| Summary: This is a simple deconstructor

|

| Precodition: there is a Queue

| Postcondition: that Queue memory is freed

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

~Queue()

{

Node\* temp = head;

while(head != NULL)

{

head = head->next;

#ifdef SELF\_HOSTING

memory.free(temp);

#else

delete temp;

#endif

temp = head;

}

// for safety

tail = NULL;

head = NULL;

//while(empty() == false)

// pop();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: push

|

| Purpose: Push values into the queue

|

| Precodition: queue was declared

| Postcondition: queue as another element in the back

|

| Parameters:

|

| IN: value of queues datatype

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void push(Q value)

{

Node\* temp;

if(tail != NULL)

{

#ifdef SELF\_HOSTING

temp = (Node\*) memory.alloc(sizeof(Node));

#else

temp = new Node;

#endif

temp->datum = value;

temp->next = NULL;

tail->next = temp;

tail = temp;

}

else

{

#ifdef SELF\_HOSTING

tail = (Node\*) memory.alloc(sizeof(Node));

#else

tail = new Node;

#endif

tail->datum = value;

tail->next = NULL;

head = tail;

}

\_size++;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: pop

|

| Purpose: remove element at the front of the q, return value

|

| Precodition: queue exists, has element

| Postcondition: element removed from front

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: item at front of queue

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Q pop()

{

Q pop\_value;

if(head == NULL)

{

dump\_screen.println("Error: push on empty Queue\n");

exit(1);

}

else if(\_size != 1)

{

Node\* temp = head;;

pop\_value = head->datum;

head = head->next;

delete(temp);

temp = NULL;

\_size--;

}

else

{

pop\_value = head->datum;

delete(head);

head = NULL;

tail = NULL;

\_size = 0;

}

return pop\_value;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: front

|

| Purpose: see the element at the front without removal

|

| Precodition: queue exists, has element

| Postcondition: head datum is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: copy of element at the front

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Q front()

{

return head->datum;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: size (accessor)

|

| Purpose: get the size of the queue

|

| Precodition: queue exists

| Postcondition: size of queue is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: size of queue

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int size()

{

return \_size;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: empty

|

| Purpose: tells if a queue is empty, useful for going

| through all the elements in a queue via while loops

|

| Precodition: queue exists

| Postcondition: boolean of empty or not empty is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: whether the queue is empty or not

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

bool empty()

{

return (\_size == 0) ? true : false;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: <<

|

| Purpose: allow for use in any standard stream mechanism

|

| Precodition: nothing

| Postcondition: object is sent along a stream as a string

|

| Parameters:

|

| IN: stream mechanism, queue

| Out: stream mechanism, queue

|

| Returns: stream mechanism

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

friend ostream& operator<<(ostream& os, Queue<Q>& q)

{

Queue::Node\* temp = q.head;

if(q.\_size == 0)

{

os << "(Queue)";

}

else

{

os << "(Queue ";

while(temp->next != NULL)

{

os << (temp->datum) << ' ';

temp = temp->next;

}

os << temp->datum;

os << ")";

}

return os;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: >>

|

| Purpose: convert appropriate strings to queues

|

| Precodition: there is a queue declared

| Postcondition: queue is built from strong representation

|

| Parameters:

|

| IN: stream mechanism, queue

| Out: stream mechanism, queue

|

| Returns: stream mechanism

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

friend istream& operator>> (istream& in, Queue<Q>& queue)

{

char ch;

string keyword = "Queue";

// capture the chracters and see if it matches

// the string

in >> ch;

if(ch != '(')

{

dump\_screen.println("mismatched parens\n");

exit(-1);

}

for(unsigned int i = 0 ; i < keyword.length() ; i++)

{

in >> ch;

if(ch != keyword[i])

{

dump\_screen.println("wrong object\n");

exit(-1);

}

}

in >> ch; // for space, or end

if(ch == ')')

{

return in;

}

// to be continued

//Q element;

stringstream ss;

return in;

}

// MCB is used in testing software

friend class MCB;

};

/\*{

TODO: document and test }\*/

// what is n kyle?

struct n

{

int p;

int info;

struct n\* l;

};

class Priority\_Queue

{

private:

n\* f;

public:

Priority\_Queue()

{

f = NULL;

}

void insert(int i, int p)

{

n\* t, \*q;

t = new n;

t -> info = i;

t -> p = p;

if(f == NULL || p < f->p)

{

t -> l = f;

f = t;

}

else

{

q = f;

while(q -> l != NULL && q -> l -> p <= p)

{

q = q -> l;

}

t -> l = q -> l;

q -> l = t;

}

}

void del()

{

n\* t;

if(f == NULL)

{

//dump\_screen.print( "Queue Underflow\n";

}

else

{

t = f;

//dump\_screen.print( "Deleted item is: " +str( t->info) + str(endl);

f = f->l;

free(t);

}

}

void show()

{

n\* ptr;

ptr = f;

if(f == NULL)

{

//dump\_screen.print( "Queue is empty\n";

}

else

{

//dump\_screen.print( "Queue is :\n";

//dump\_screen.print( "Priority Item\n";

while(ptr != NULL)

{

//dump\_screen.print( ptr->p << " " << ptr->info << endl;

ptr = ptr->l;

}

}

}

};

//}

// co-dependency

class Semaphore;

class Scheduler;

/\*{===========================================================

||

|| Class: Semaphore

||

|| Purpose: A textbook semaphore is a flag for a resource.

|| Flags if a resource is available, such as:

|| a block of code, memory, or IO mechanism

|| Design choice:

|| semaphore is more than (int, P, V)

|| It has a queue, OOP stuff, can be named,

|| and needs pthread.h

||

|| Attributes:

|| semaphore value

|| pthread mutex

|| pthread conditional

|| name of the resource protected

||

|| Functions:

|| constructor(semaphore value = 1)

|| is\_down()

|| down()

|| down(thread)

|| up()

|| up(thread)

|| << (a friend)

||

\\*=========================================================}\*/

class Semaphore

{

public:

friend Scheduler; // because they are more or less the same total class now

// So resource keys can have meaningful names at runtime

string key\_name; // resource\_name

private:

//public:

int semaphore\_value = 1; // the lock

Queue<pthread\_t> waiting\_ids; // by request

// the actual lock, so the class semaphore

// is not interrupted

pthread\_mutex\_t semaphore\_lock;// = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t resource\_lock;// = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_cond\_t cond;//= PTHREAD\_COND\_INITIALIZER;

public:

// functions

//{

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Semaphore

|

| Purpose: constructor for the semaphore

| default to being called memory

| start up mutex and conditional

|

| Precodition: nothing

| Postcondition: semaphore made

|

| Parameters:

|

| IN: (optional) number of resources

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Semaphore(int resources = 1,

string name = "memory") // assume binary

{

key\_name = name;

semaphore\_value = resources;

// init locks and condtionals

if(pthread\_mutex\_init(&semaphore\_lock, NULL) != 0)

{

printf("error: mutex1 issue\n");

exit(-1);

}

if(pthread\_mutex\_init(&resource\_lock, NULL) != 0)

{

printf("error: mutex2 issue\n");

exit(-1);

}

if(pthread\_cond\_init(&cond, NULL) != 0)

{

printf("error: mutex conditional issue\n");

exit(-1);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Semaphore

|

| Purpose: copy constructor for the semaphore

| default to being called memory

| start up mutex and conditional

|

| Precodition: there is a semaphore

| Postcondition: semaphore copied

|

| Parameters:

|

| IN: semaphore

| Out: the same semaphore, unchanged

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Semaphore(const Semaphore& s)

{

semaphore\_value = s.semaphore\_value;

key\_name = s.key\_name;

semaphore\_lock = s.semaphore\_lock;

resource\_lock = s.resource\_lock;

cond = s.cond;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: Semaphore

|

| Purpose: deconstructor for the semaphore

|

| Precodition: there exists a semaphore

| Postcondition: semaphore memory is released

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

~Semaphore()

{

if(pthread\_mutex\_destroy(&semaphore\_lock) != 0)

{

printf("error: mutex destruction issue\n");

exit(-1);

}

if(pthread\_mutex\_destroy(&resource\_lock) != 0)

{

printf("error: mutex destruction issue\n");

exit(-1);

}

if(pthread\_cond\_destroy(&cond) != 0)

{

printf("error: cond destruction issue\n");

exit(-1);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: is\_down

|

| Purpose: soft down, let the scheduler know what is up

| Letting the accessor know of blocking is a secondary

| Utility since the actual pthread handles blocking

| (Without rewriting the posix thread library)

| Uses a mutex to complete this scan, but does not

| force a down after checking. This allows the scheduler

| to check if processes should be blocked.

|

| Precodition: semaphore exists

| Postcondition: information whether a resource is blocked

|

| Parameters:

|

| IN: the Task\_block of process to access the resource

| Out: nothing

|

| Returns: the Task\_block that went in, with a block

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

bool is\_down()

{

return semaphore\_value <= 0;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: down

|

| Purpose: pauses a process, and tracks paused processes

|

| Precodition: semaphore exists

| Postcondition: process is blocked if resource is busy,

| and given spot in line

|

| Parameters:

|

| IN: pthread to block and queue

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void down(pthread\_t id)

{

pthread\_mutex\_lock(&semaphore\_lock);

// Have not found a way for threads to block each other

if(pthread\_equal(pthread\_self(), id))

{

if(semaphore\_value <= 0)

{

waiting\_ids.push(id);

pthread\_cond\_wait(&cond, &resource\_lock);

}

semaphore\_value--;

}

pthread\_mutex\_unlock(&semaphore\_lock);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: down

|

| Purpose: For more scheduler control see

| down(pthread\_t)

|

| Precodition: semaphore exists

| Postcondition: process is blocked if resource is busy

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void down()

{

// contians 3 instructions that might be interupted

pthread\_mutex\_lock(&semaphore\_lock);

if(semaphore\_value <= 0)

{

pthread\_cond\_wait(&cond, &resource\_lock);

}

semaphore\_value--;

pthread\_mutex\_unlock(&semaphore\_lock);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: up

|

| Purpose: a version of up that does not rely on the built in scheduler

|

| Precodition: nothing

| Postcondition: a thread waiting on resource is unblocked

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void up(pthread\_t id)

{

pthread\_t temp; // going to search a queue

// handling a queue is a single process job

pthread\_mutex\_lock(&semaphore\_lock);

// only 1 process should be considered active

semaphore\_value++;

if(semaphore\_value >= 1)

{

do

{

// wakes up signals until valid one wakes

// this is the only unfreeze thread function, I believe

pthread\_cond\_signal(&cond);

// signal checks if it is its turn

temp = waiting\_ids.pop();

}

while((not pthread\_equal(id, pthread\_self())) and

// asked for a nonexistent thread to run

(not waiting\_ids.empty()));

// calling down on id != self

if((not pthread\_equal(id, pthread\_self())))

{

waiting\_ids.push(temp);

pthread\_cond\_wait(&cond, &resource\_lock);

}

}

pthread\_mutex\_unlock(&semaphore\_lock);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: up

|

| Purpose: a version of up that uses the built in scheduler

|

| Precodition: nothing

| Postcondition: a thread waiting on resource is unblocked

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void up()

{

pthread\_mutex\_lock(&semaphore\_lock);

semaphore\_value++;

if(semaphore\_value >= 1)

{

pthread\_cond\_signal(&cond);

}

pthread\_mutex\_unlock(&semaphore\_lock);

}

//}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: <<

|

| Purpose: allow for use of semaphore in standard streams

|

| Precodition: a stream library was included

| Postcondition: a semaphore was added to a stream

|

| Parameters:

|

| IN: stream mechanism, semaphore

| Out: stream mechanism, semaphore

|

| Returns: stream mechanism

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

friend ostream& operator<<(ostream& os,

const Semaphore& semaphore)

{

os << "(Semaphore "

<< semaphore.key\_name << ' '

<< semaphore.semaphore\_value //<< ' '

//<< semaphore.semaphore\_lock << ' '

//<< semaphore.resource\_lock << ' '

//<< semaphore.cond

<< ")";

return os;

}

// include some functions which will allow you to dump\_screen the contents of the semaphore in

// a readable format.

void dump(int level = 1)

{

dump\_screen.println("Semaphore Dump:");

dump\_screen.println("Resource: " + key\_name);

dump\_screen.println("Sema\_value: " + str(semaphore\_value));

// this one still lisp prints TODO: make it fit spec

dump\_screen.println("Sema\_queue: " + str(waiting\_ids));

}

// MCB is used in the testing code

friend class MCB;

};

/\*{===========================================================

||

|| Class: Scheduler

||

|| Purpose: schedule multithreaded processes

||

|| Attributes:

|| Task\_block (task control block)

|| \*function

|| id

|| name

|| state // running, blocked, ready

|| \*input

|| \*output

|| \*data

|| \*semaphore

|| \*thread

|| quantum

||

|| queue of Task\_block

|| id (incrementor)

|| quantum (default)

||

|| Functions:

|| create\_thread

|| destroy\_task

|| garbage collect

|| empty

|| run

|| run\_demo

|| <<

||

\\*=========================================================}\*/

class Scheduler

{

public:

// super user is creator

pthread\_t self = pthread\_self();

class Task\_block // TCB\* process\_table

{

public:

// main function

// they have to be of this type

// and have casts inside the call

void\* (\*function)(void\*) = NULL;

// properties of function

int id = 0;

string name; // shorthand name

string state; // running, blocked, ready

//void\* input = NULL; // for sake of abstraction

//void\* output = NULL; // also for sake of abstraction

void\* data = NULL; // pointer to some memory

//Semaphore\* semaphore = NULL; // lock for memory

//Messenger\* messenger = NULL;

pthread\_t thread = 0; // actual thread

//int quantum = 0; // set how much time each process takes

// needed so copy constructor works

Task\_block()

{

function = NULL;

id = 0;

//input = NULL;

//output = NULL;

data = NULL;

//semaphore = NULL;

//messenger = NULL;

thread = 0;

//quantum = 0; // don't run an empty block

}

Task\_block(const Task\_block& a)

{

function = a.function;

id = a.id;

name = a.name;

state = a.state;

//input = a.input;

//output = a.output;

data = a.data;

//semaphore = a.semaphore;

//messenger = a.messenger;

thread = a.thread;

//quantum = a.quantum;

}

// task control block

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: <<

|

| Purpose: allow this class to be printed and converted into

| a string for testing outputs

|

| Precodition: nothing

| Postcondition: the Task\_block object was added to the stream

|

| Parameters:

|

| IN: stream mechanism, Task\_block block

| Out: stream mechanism, Task\_block block

|

| Returns: stream mechanism

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

friend ostream& operator<<(ostream& os,

const Task\_block& task\_block)

{

os << "(Task\_block "

<< "function\* " // not a class so can't be printed

<< task\_block.id << " "

<< task\_block.name << " "

<< task\_block.state << " ";

// the next 4 are void pointers

/\*

if(task\_block.input == NULL)

{

os << "NULL ";

}

else

{

os << "input\*" << " ";

}

if(task\_block.output == NULL)

{

os << "NULL ";

}

else

{

os << "output\*" << " ";

}

\*/

if(task\_block.data == NULL)

{

os << "NULL ";

}

else

{

os << "data\*" << " ";

}

/\*

if(task\_block.semaphore == NULL)

{

os << "NULL ";

}

else

{

os << \*(task\_block.semaphore) << " ";

}

\*/

os << "thread" //<< " " // iostream leaks memory on threads

//<< task\_block.quantum

<< ")";

return os;

}

};

friend Semaphore; // because they are more or less the same total class now

Semaphore semaphore;

Queue<Task\_block> task\_block\_q;

//Task\_block current\_task\_block; // a global temp\_task\_block

int id = 0; // starts at 0 and is incremented

//int quantum = 1; // time each process gets

Scheduler()

{

semaphore.semaphore\_value = 0;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: create thread

|

| Purpose: to create task control blocks

|

| Precodition: nothing

| Postcondition: task control block is added to queue

|

| Parameters:

|

| IN: name and function pointer,

| (optionally function input location, output location

| location of memory source [ram, ssd, etc..]

| Out: a task control block is added to the queue

|

| Returns: the id of the created Task\_block

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int create\_task(string name,

void\* (\*instruction)(void\*),

void\* data = NULL)

//void\* input = NULL,

//void\* output = NULL,

//Semaphore\* semaphore = NULL,

//Messenger\* messenger = NULL,

/\*int quantum = 1\*/

{

Task\_block temp;

// make data

temp.function = instruction;

temp.id = id;

temp.name = name;

temp.state = "ready";

//temp.input = input;

//temp.output = output;

temp.data = data;

//temp.semaphore = semaphore;

//temp.messenger = messenger;

//temp.quantum = quantum;

task\_block\_q.push(temp);

id++;

return temp.id;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: destroy task

|

| Purpose: set task to dead by id

|

| Precodition: nothing

| Postcondition: thread is killed and marked as dead

|

| Parameters:

|

| IN: task id

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void destroy\_task(int id)

{

Task\_block temp\_task\_block;

for(int i = 0 ; i < task\_block\_q.size() ; i++)

{

temp\_task\_block = task\_block\_q.pop();

if(temp\_task\_block.id == id)

{

// If you don't care about leaking memory

pthread\_cancel(temp\_task\_block.thread);//kill it

// limits leaks but causes main to wait for thread

//pthread\_join(temp\_task\_block.thread,NULL);

temp\_task\_block.state = "dead";

}

task\_block\_q.push(temp\_task\_block);

}

}

// problem: there is no posix way to pause a thread

// lets process die, mark it as dead

void yield(int id)

{

// get to the requested id

Task\_block temp\_task\_block;

for(int i = 0 ; i < task\_block\_q.size() ; i++)

{

temp\_task\_block = task\_block\_q.pop();

if(pthread\_equal(temp\_task\_block.id, id))

{

// shut it down

if(temp\_task\_block.state == "running")

{

semaphore.down(

id); // empty semaphore, need to release this eventually

}

temp\_task\_block.state = "ready";

}

task\_block\_q.push(temp\_task\_block);

}

// run the next guy

temp\_task\_block = task\_block\_q.pop();

task\_block\_q.push(temp\_task\_block);

semaphore.up(temp\_task\_block.id);

temp\_task\_block.state == "running";

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: garbage collect

|

| Purpose: remove dead tasks

|

| Precodition: nothing

| Postcondition: dead tasks are removed from task\_block

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void garbage\_collect()

{

Queue<Task\_block> temp\_task\_block\_q;

Task\_block temp\_task\_block;

// go through the q and empty it of false

while(task\_block\_q.empty() == false)

{

temp\_task\_block = task\_block\_q.pop();

if(temp\_task\_block.state != "dead")

{

temp\_task\_block\_q.push(temp\_task\_block);

}

}

while(temp\_task\_block\_q.empty() == false)

{

task\_block\_q.push(temp\_task\_block\_q.pop());

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: empty

|

| Purpose: tells if the scheduler is empty

|

| Precodition: nothing

| Postcondition: it is known if the scheduler is empty

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: if the scheduler is empty

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

bool empty()

{

return (task\_block\_q.size() == 0) ? true : false;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: run

|

| Purpose: run the task control blocks

|

| Precodition: nothing

| Postcondition: multhreaded tasks are running

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void run()

{

// round robin run functions

for(int i = 0 ; i < task\_block\_q.size() ; i++)

{

auto temp = task\_block\_q.pop();

pthread\_create(&temp.thread, NULL, temp.function,

temp.data);

task\_block\_q.push(temp);

}

// tests need to be done before all of this implimentation

/\*

//int iterate\_lim = 1; // thread spam issues, will update

// do round robin on task blocks

//while((task\_block\_q.empty() != false)

// and (iterate < iterate\_lim))

{

iterate++;

// split off a process for processing

Task\_block temp = task\_block\_q.pop();

// quantum meaning times run, assuming functions don't loop forever

if(temp.quantum <= 0) // this never runs for some reason

{

destroy\_task(temp.id);

pthread\_join(temp.thread, NULL); // to be safe

temp.state = "dead";

}

else if(temp.quantum > 0)

{

task\_block\_q.push(temp);

temp.quantum--;

}

int error;

// If no state is messed with, run the thread without constraints

if(temp.state == "ready" and temp.semaphore == NULL)

{

temp.state = "running";

error = pthread\_create(&temp.thread, NULL, temp.function, &temp.input); // assume copy is made of input

if(error != 0)

{

exit(-1);

}

}

// there is at least a key

else if(temp.state == "ready")

{

temp.state = "running";

// otherwise 'blocks' would not show

if(temp.semaphore->is\_down())

{

temp.state = "blocked";

}

if(temp.state != "blocked")

{

//temp.semaphore->down();

temp.state = "running";

error = pthread\_create(&temp.thread, NULL, temp.function, &temp.input); // assume copy is made of input

if(error != 0)

{

exit(-1);

}

pthread\_join(temp.thread,NULL); // makes scheduler non-concurrent, for debug

temp.state = "ready";

//temp.semaphore->up();

}

}

garbage\_collect();

}

\*/

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: ~Scheduler

|

| Purpose: must be defined so data structures are released

| Note: any q created needs to be added here

|

| Precodition: nothing

| Postcondition: scheduler is gone

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

~Scheduler()

{

while(task\_block\_q.empty() == false)

{

Task\_block temp = task\_block\_q.pop();

pthread\_join(temp.thread,

NULL); // get all threads to exit first

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: <<

|

| Purpose: allows printing and string conversions on scheduler

|

| Precodition: nothing

| Postcondition: scheduler is added to a stream

|

| Parameters:

|

| IN: stream mechanism, scheduler

| Out: stream mechanism, scheduler

|

| Returns: stream mechanism

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

friend ostream& operator<<(ostream& os, Scheduler& s)

{

os << "(Scheduler "

<< s.task\_block\_q << " "

//<< s.semaphore\_q << " "

//<< "current\_task\_block: " << s.current\_task\_block << endl

<< s.id //<< " "

//<< "memory\_q: " << memory\_q << endl // this guy is void

//<< s.quantum

<< ')';

return os;

}

// debugging function with level indicating the verbosity of the dump\_screen include some

//functions which will allow you to dump\_screen the contents of the process table in a readable

//format.

void dump(int level)

{

dump\_screen.println("Task Name | Task ID | State");

for(int i = 0 ; i < task\_block\_q.size() ; i++)

{

auto temp = task\_block\_q.pop();

task\_block\_q.push(temp);

dump\_screen.println("Task" + str(temp.id) + "|" + str(

temp.id) + "|" + str(temp.state));

}

}

// MCB is a test suite

friend class MCB;

};

/\*{===========================================================

||

|| Class: Messenger (IPC interProcess communication)

||

|| Purpose: allow functions to talk to each other

||

|| Attributes:

|| Message

|| counter

|| id

|| description

|| source\_task\_id

|| destination\_task\_id

|| arrival\_time

|| size

|| text

|| Maybe\_message

|| Message message

|| exists

||

|| message\_id

|| queue messages

|| Functions:

|| send()

|| Maybe\_message receive()

|| count\_letters()

|| count\_letters\_to()

|| delete\_all\_letters()

|| <<

|| print()

|| dump\_screen()

||

\\*=========================================================}\*/

class Messenger // ipc

{

public:

// super user is creator

pthread\_t self = pthread\_self();

class Message

{

public:

// message type

//{

int id = 0;

string description; // default is empty string

int type; // 0=text, 1=service, 2=notification

//}

pthread\_t source\_task\_id = 0;

pthread\_t destination\_task\_id = 0;

time\_t arrival\_time = time(NULL);

int size = 0;

string text; // default is empty string

// functions

//{

Message()

{

// watch out for creating temp messages

//counter++; // keeps id's unique

}

Message(const Message& m)

{

id = m.id;

description = m.description;

source\_task\_id = m.source\_task\_id;

destination\_task\_id = m.destination\_task\_id;

arrival\_time = m.arrival\_time;

size = m.size;

text = m.text;

}

// for sake of string casts and easy printing

friend ostream& operator<<(ostream& os, const Message& m)

{

os << "(Message "

<< m.id << ' '

<< m.description << ' '

// pthread\_t is not always an integer, so printing is not portable

<< 0 << ' '// m.source\_task\_id << ' '

<< 0 << ' '//m.destination\_task\_id << ' '

//<< (double)m.arrival\_time << ' '

<< m.size << ' '

<< m.text << ')';

return os;

}

// So messages can be added to a binary search tree

friend bool operator==(Message a, Message b)

{

return a.id == b.id;

}

friend bool operator!=(Message a, Message b)

{

return a.id != b.id;

}

friend bool operator< (Message a, Message b)

{

return a.id < b.id;

}

friend bool operator> (Message a, Message b)

{

return a.id > b.id;

}

friend bool operator<=(Message a, Message b)

{

return a.id <= b.id;

}

friend bool operator>=(Message a, Message b)

{

return a.id >= b.id;

}

};

int counter = 0;

// functional programming monad, for returning a message or not

class Maybe\_message

{

public:

Message message;

bool exists = false; // default to avoid need for constructor

Maybe\_message()

{

exists = false; // default to avoid need for constructor

}

friend ostream& operator<<(ostream& os,

const Maybe\_message& m)

{

os << "(Maybe\_message "

<< m.message << ' '

<< m.exists << ')';

return os;

}

Maybe\_message(const Maybe\_message& m)

{

message = m.message;

exists = m.exists;

}

};

// data

int message\_id = 0;

Queue<Message> messages;

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: send

|

| Purpose: load a message onto the mailbox and send it to another

| task associated with an id

|

| Precodition: you need a mailbox to hold the message you want

| to send to a task

| Postcondition: as long as the destination exists it will send

| the message to that location

|

| Parameters:

|

| IN: letter holds the actual message wanting to be sent to a

| task, mailbox is the mailbox we want to add the message too,

| messagekey keeps the mailbox from being accessed by others,

| process identifies the location it needs to be sent too

| Out: nothing

|

| Returns: returns 0

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int send(

Message letter,

Queue<Messenger::Message>\* mailbox,

Semaphore\* messagekey

)

{

messagekey->down();

// construction of letter

letter.id = counter;

letter.source\_task\_id = pthread\_self();

counter++; // since message was made

mailbox->push(letter);

messagekey->up();

return 0;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: receive

|

| Purpose: lets the user know if the message it was trying to

| send to a task was successful or not

|

| Precodition: have a mailbox ready to receive the message

| Postcondition: identify whether the message made it to the

| correct location or not

|

| Parameters:

|

| IN: mailbox needed to check if the message made it, messagekey

| keeps the mailbox 'locked' to a specific task while it is down,

| process checks if it is the correct location being sent to

| Out: nothing

|

| Returns: returns the message if it was sent to the correct area

| or flags false if the message was not

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

Maybe\_message receive(Queue<Messenger::Message>\* mailbox,

Semaphore\* messagekey)

{

messagekey->down();

if(mailbox->size() == 0) // check size requirement

{

Maybe\_message nothing;

messagekey->up();

return nothing; // need to error check on the other end

}

for(int i = 0 ; i < mailbox->size() ; i++)

{

Message message = mailbox->pop();

mailbox->push(message);

// check if it is for the process

if(pthread\_equal(message.destination\_task\_id, pthread\_self()))

{

Maybe\_message yes\_message;

yes\_message.message = message;

yes\_message.exists = true;

messagekey->up();

return yes\_message;

}

Maybe\_message nothing;

messagekey->up();

return nothing;

}

Maybe\_message nothing;

return nothing;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: count\_letters

|

| Purpose: count the total number of messages sent between all

| tasks in the mailbox system

|

| Precodition: have a mailbox to sent messages too

| Postcondition: return the number of messages sent through the

| system

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: returns the number of messages sent within the mailbox

| system

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int count\_letters()

{

// return the message counter to show how many messages were sent

return messages.size();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: count\_letters\_to

|

| Purpose: This will count the number of messages that were sent

| to a specific task id

|

| Precodition: you need to have messages in the task to see

| Postcondition: if it has messages it will return how many messages

| are inside the mailbox

|

| Parameters:

|

| IN: task id to search message count for

| Out: nothing

|

| Returns: returns the number of messages in a tasks mailbox

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int count\_letters\_to(pthread\_t task\_id)

{

// display the number of messages sent to specific task based on the id

int count = 0;

for(int i = 0 ; i < messages.size() ; i++)

{

Message temp = messages.pop();

if(pthread\_equal(temp.destination\_task\_id, task\_id))

{

count++;

}

messages.push(temp);

}

return count;

}

// delete all sent and received?

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: delete\_all\_letters

|

| Purpose: delete all of the messages sent to a certain task

|

| Precodition: need messages to delete from the specific task

| Postcondition: if the task has messages they were deleted

|

| Parameters:

|

| IN: task id that you want to delete messages for

| Out: nothing

|

| Returns: returns 0

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int delete\_all\_letters(pthread\_t task\_id)

{

// delete the message of the task id passed

for(int i = 0 ; i < messages.size() ; i++)

{

Message temp = messages.pop();

// deleteing both sent and received

if(not pthread\_equal(temp.destination\_task\_id, task\_id) and

not pthread\_equal(temp.source\_task\_id, task\_id))

{

messages.push(temp);

}

}

return 0;

}

//}

// not much to print for a method block

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: <<

|

| Purpose: allow for use in any standard stream mechanism

|

| Precodition: nothing

| Postcondition: object is sent along a stream as a string

|

| Parameters:

|

| IN: stream mechanism, Messenger

| Out: stream mechanism, Messenger

|

| Returns: stream mechanism

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

friend ostream& operator<<(ostream& os, Messenger& m)

{

if(m.messages.size() == 0)

{

os << "(Messenger)";

return os;

}

os << "(Messenger";

for(int i = 0 ; i < m.messages.size() ; i++)

{

Message temp = m.messages.pop();

os << ' ' << temp;

m.messages.push(temp);

}

os << ')';

return os;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: print

|

| Purpose: Print out all the messages that were sent to a specific

| task id

|

| Precodition: have a messages send to a task

| Postcondition: print out the messages sent to a certain task

|

| Parameters:

|

| IN: task id that you want to print messages for

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void print(pthread\_t task\_id)

{

// print messages stored in the datatype(bin search tree?) from a certain task id

for(int i = 0 ; i < messages.size() ; i++)

{

Message temp = messages.pop();

// deleteing both sent and received

if(pthread\_equal(temp.destination\_task\_id, task\_id) or

pthread\_equal(temp.source\_task\_id, task\_id))

{

dump\_screen.print(str(temp) + ";");

}

messages.push(temp);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: dump\_screen

|

| Purpose: Display all the message from the ipc class

|

| Precodition: have messages to display on the screen

| Postcondition: messages will be displayed

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void dump()

{

dump\_screen.println("Task Number: -1 "); // root prints

dump\_screen.println("Message Count: " + str(messages.size()));

dump\_screen.println("Mail Box: ");

dump\_screen.println("Source | Destination | ""Message Content ""| Message | Message | Message ");

dump\_screen.println("Task-id | Task-id | "" ""| Size | Type | Arrival ");

dump\_screen.println(" | | "" ""| | | Time ");

for(int i = 0 ; i < messages.size() ; i++)

{

Message temp = messages.pop();

// deleteing both sent and received

dump\_screen.println(str(temp.source\_task\_id)

+ "|" +

str(temp.destination\_task\_id)

+ "|" +

str(temp.text)

+ "| |" +

str(temp.type) + "|" +

str(temp.arrival\_time));

messages.push(temp);

}

}

}; // end of Messenger class

/\*{===========================================================

\* ||

\* || Class: File\_system

\* ||

\* || Purpose: create a file-like system to create, read, write, and open files

\* ||

\* || Attributes:

\* || string fs\_name

\* || int fs\_block\_size

\* || int fs\_number\_of\_blocks

\* || int next\_file\_handle

\* || char initialization\_char

\* || I-node

\* || char filename[]

\* || int owner\_task\_id

\* || int starting\_block

\* || int size

\* || char permission[]

\* || unsigned int blocks[]

\* || Functions:

\* || ufs()

\* || open()

\* || close()

\* || read\_char()

\* || write\_char()

\* || create\_file()

\* || del\_file()

\* || change\_permission()

\* || dir()

\* || dir(task\_id)

\* || dump\_screen()

\* || display\_file\_info()

\* || show\_permissions()

\* ||

\* \\*=========================================================}\*/

class File\_system // ufc

{

public:

// file status storage

struct Stat

{

dev\_t st\_dev; //ID of device containg file

ino\_t st\_ino; //Inode number

mode\_t st\_mode; //File type and mode

nlink\_t st\_nlink; //Number of hard links

uid\_t st\_uid; //User ID of owner

gid\_t st\_gid; //Group Id of owner

dev\_t st\_rdev; //Device ID

off\_t st\_size; //Total size

blksize\_t st\_blksize; //Block size for filesystem I/O

blkcnt\_t st\_blocks; //Number of 512b blocks allocated

struct timespec st\_atim; //Time of last access

struct timespec st\_mtim; //Time of last modification

struct timespec st\_ctim; // Time of last status change

#define st\_atime st\_atim.tv\_sec //backward compatibility

#define st\_mtime st\_mtim.tv\_sec

#define st\_ctime st\_ctim.tv\_sec

};

Stat status;

#define DISK\_SIZE 2048

#define NUMBER\_OF\_BLOCKS 16

#define BLOCK\_SIZE (DISK\_SIZE/NUMBER\_OF\_BLOCKS)

#define root 0

/\*{===========================================================

||

|| Class: i\_node

||

|| Purpose: creates 'i\_node' structures that server as directory informtion

||

|| Note: For purpose of project it should have 16 i\_node structures each 40 bytes

||

|| Attributes:

|| char filename[]

|| int owner\_task\_id

|| int starting\_block

|| int size

|| char permission[]

|| unsigned int blocks[]

||

|| Functions:

|| init\_inode()

|| write\_inode()

|| read\_inode()

|| dump\_screen\_inode()

|| dump\_screen\_inode()

|| char\_to\_binary()

||

\\*=========================================================}\*/

class i\_node

{

public:

char filename[8]; // actually 7 char name, ends in 0

int owner\_task\_id; // root = 0, Jordan = 1, Josh = 2, Kyle = 3

int starting\_block; // block "pointer"

int size; // number of blocks \* size of blocks

int permission = 0b0000; // rwrw (owner, others)

int blocks[4]; // location of 4 blocks

};

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: init\_inode

|

| Purpose: used to initialize an i-node and start a file for

| the filesystem

|

| Precodition: have an i\_node struct available

| Postcondition: the i-node was initialized

|

| Parameters:

|

| IN: name of the filesystem, number of blocks for the files,

| the size of those blocks, and an initialization character

| Out: i-node started and file opened

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void init\_inode(char\* file\_name, int num\_i\_nodes,

int i\_node\_size, i\_node\* default\_i\_node)

{

fstream myfile(file\_name, ios::out);

myfile.seekp(0);// Place file pointer at the beginning

for(int i = 0; i < num\_i\_nodes ; i++)

{

myfile.write((char\*)default\_i\_node,

i\_node\_size); // write i-node

}

myfile.close();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: write\_inode

|

| Purpose: open a file for reading and writing so that we can

| write characters to the file

|

| Precodition: have an i\_node struct available

| Postcondition: the file has been written too

|

| Parameters:

|

| IN: name of the filesystem, offset, pointer to the i-node

| location that is being written too, size of the write

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void write\_inode(char\* file\_name, int offset,

i\_node\* the\_i\_node, int i\_node\_size)

{

fstream myfile(file\_name, ios::out | ios::in);

myfile.seekp(offset,

ios::beg); // Place the file pointer at the offset

myfile.write((char\*)the\_i\_node,

i\_node\_size); // write the i-node

myfile.close();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: read\_inode

|

| Purpose: open a file for reading and writing so that we can

| read what has been written to the file

|

| Precodition: have an i\_node struct available

| Postcondition: the file has been read

|

| Parameters:

|

| IN: name of the filesystem, offset, pointer to the i-node

| location that is being written too, size of the write

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void read\_inode(char\* file\_name, int offset,

i\_node\* the\_i\_node, int i\_node\_size)

{

fstream myfile(file\_name, ios::out | ios::in);

myfile.seekp(offset,

ios::beg); // Place the file pointer at offset

myfile.read((char\*)the\_i\_node,

i\_node\_size); // read the i-node

myfile.close();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: dump\_screen\_inode

|

| Purpose: dump\_screen all the contents of the file on the screen for

| the user to see the i-node information

|

| Precodition: have an i\_node struct available

| Postcondition: the file has been dump\_screened to the screen

|

| Parameters:

|

| IN: name of the filesystem, size of the write

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void dump\_screen\_inode(char\* file\_name, int i\_node\_size)

{

//dump\_screen.print( "----------------------" << endl;

fstream myfile(file\_name, ios::out | ios::in);

myfile.seekp(0,

ios::beg); // Place the file pointer at the beginning of the file

char c;

for(int i\_node = 0; i\_node < 8; i\_node++)

{

for(int size = 0; size < i\_node\_size; size++)

{

myfile.get(c);

//dump\_screen.print( str(c));

}

//dump\_screen.print( endl;

}

myfile.close();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: dump\_screen\_inode

|

| Purpose: dump\_screen all the contents of a specific file on the screen

| for the user to see the i-node information

|

| Precodition: have an i\_node struct available

| Postcondition: the file specified file has been dump\_screened to screen

|

| Parameters:

|

| IN: name of the filesystem

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void dump\_screen\_inode(char\* file\_name)

{

fstream myfile(file\_name, ios::out | ios::in);

myfile.seekp(0,

ios::beg); // Place the file pointer at the beginning of the file

i\_node my\_i\_node;

//dump\_screen.print( "==========================================" << endl;

//dump\_screen.print( "T-ID\tFileName\tFileSize\tBlocks" << endl;

for(int i\_node = 0; i\_node < 8; i\_node++)

{

read\_inode(file\_name, i\_node \* sizeof(my\_i\_node), &my\_i\_node,

sizeof(my\_i\_node));

//dump\_screen.print( my\_i\_node.owner\_task\_id << "\t";

//dump\_screen.print( my\_i\_node.filename << "\t\t";

//dump\_screen.print( my\_i\_node.size << "\t\t";

//dump\_screen.print( my\_i\_node.blocks[0] << ' '

// << my\_i\_node.blocks[1] << ' '

// << my\_i\_node.blocks[2] << ' '

// << my\_i\_node.blocks[3] << endl;

}

//dump\_screen.print( "==========================================" << endl;

myfile.close();

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: char\_to\_binary

|

| Purpose: convert a character to a binary value

|

| Precodition:

| Postcondition: the value is not in binary

|

| Parameters:

|

| IN: value to convert

| Out: binary result

|

| Returns: binary value is returned as a string

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

string char\_to\_binary(unsigned int value)

{

char theResult[128];

unsigned char mask = 0x80;

unsigned int i;

for(i = 0; i < sizeof(value) \* 8; i++)

{

if((value & mask) == 0)

{

theResult[i] = '0';

}

else

{

theResult[i] = '1';

}

mask >>= 1;

}

theResult[i] = '\0';

string result = theResult;

return(result);

}

// the max of i-nodes is defined

//i\_node i\_nodes[16];

// super user is creator

pthread\_t self = pthread\_self();

// what open() and close() work with

i\_node node\_file\_handle;

string fs\_name = "disk\_drive.txt";

string fs\_inode\_name = "inode.txt";

int fs\_block\_size = BLOCK\_SIZE;

int fs\_number\_of\_blocks = NUMBER\_OF\_BLOCKS;

int next\_file\_handle = 0;

char initialization\_char[2] = "^";

friend ostream& operator<<(ostream& out\_stream,

File\_system& f)

{

ifstream users("users.txt");

ifstream inode("inode.txt");

ifstream disk("disk\_drive.txt");

if(users.is\_open() and inode.is\_open() and disk.is\_open())

{

string buffer;

out\_stream << "(File\_system ";

out\_stream << "(users";

while(getline(users, buffer))

{

out\_stream << ' ';

out\_stream << buffer;

}

out\_stream << ")";

out\_stream << ' ';

out\_stream << "(inode";

while(getline(inode, buffer))

{

out\_stream << ' ';

out\_stream << buffer;

}

out\_stream << ")";

out\_stream << ' ';

out\_stream << "(disk";

while(getline(disk, buffer))

{

out\_stream << ' ';

out\_stream << buffer;

}

out\_stream << ")";

out\_stream << ")";

users.close();

inode.close();

disk.close();

}

return out\_stream;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: ufs

|

| Purpose: used by the 'superuser' to create the file system,

| format the virtual disk, and wipes the i\_nodes

|

| Precodition: have an i\_node struct available

| Postcondition: filesystem was created

|

| Parameters:

|

| IN: name of the filesystem, number of blocks for the files,

| the size of those blocks, and an initialization character

| Out: filesystem was created

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// should ask about the spec for this one. It sounds like is is designed to wipe the whole system each time the program is loaded

void ufs(int task\_id, string fs\_name, int no\_blocks,

int block\_size,

int init\_char)

{

// if not super user, leave

if(task\_id != root)

{

return;

}

//if(not pthread\_equal(pthread\_self(), self))

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: format

|

| Purpose: format the current file system similar to windows

|

| Precodition: need a filesystem in place to format

| Postcondition: the filesystem was formatted

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: filesystem has now been formatted

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void format()

{

// clean disk

fstream disk(fs\_name, ios::out | ios::in);

//fstream inode("inode.txt", ios::out | ios::in);

// Place the file pointer at the start

disk.seekp(0);

disk.write((char\*)initialization\_char, 1); // add file marker

char ok[] = ".";

// wipe the rest of the space

for(int i = 1 ; i < fs\_block\_size \* fs\_number\_of\_blocks ; i++)

{

disk.write((char\*)ok, 1);

}

disk.close();

// format i-nodes

//// NOTE: it may be smarter to have a clean size

// just remove and add a file

remove("inode.txt");

fclose(fopen("inode.txt", "w+"));

/\*

// set file name

char i\_node\_file[40\*16];

strcpy(i\_node\_file, "inode.txt");

// make a default node

i\_node node;

strcpy(node.filename, "inode");

node.owner\_task\_id = root;

node.starting\_block = 0;

node.size = 0;

node.permission = 0b0000;

node.blocks[0] = 0;

node.blocks[1] = 0;

node.blocks[2] = 0;

node.blocks[3] = 0;

// make 16 of these by default

init\_inode(i\_node\_file, 16, sizeof(i\_node), &node);

// wipe the rest of the space

\*/

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: open

|

| Purpose: open a file if it belongs ot the owner and file handle

| is a valid handle

|

| Precodition: have a file that belongs to the person trying to open

| Postcondition: file is now open

|

| Parameters:

|

| IN: task id of the person trying to open the file, the files

| pointer, the filename to open, if you want to open the file

| for read or write

| Out: a file is now opened

|

| Returns: allocated memory size

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int open(int owner\_task\_id, string filename, int mode)

// removed file\_handle, not needed to access file

{

/\*

the general design is to check permissions and if the inputted information is correct then return the exact location from disk

\*/

// root can open and read whatever, if it is physically possible

char inode\_file[] = "inode.txt";

i\_node node;

for(int i = 0 ; i < 16 ; i += sizeof(i\_node))

{

read\_inode(inode\_file, i, &node, sizeof(i\_node));

// all permissions check out

if(((owner\_task\_id == node.owner\_task\_id)

or (owner\_task\_id == root)) and

(node.filename == filename) and

(node.permission == mode))

{

strcpy(node\_file\_handle.filename, node.filename);

node\_file\_handle.owner\_task\_id = node.owner\_task\_id;

node\_file\_handle.starting\_block = node.starting\_block;

node\_file\_handle.size = node.size;

node\_file\_handle.permission = node.permission;

node\_file\_handle.blocks[0] = node.blocks[0];

node\_file\_handle.blocks[1] = node.blocks[1];

node\_file\_handle.blocks[2] = node.blocks[2];

node\_file\_handle.blocks[3] = node.blocks[3];

return node.starting\_block \* fs\_number\_of\_blocks;

}

}

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: close

|

| Purpose: close the file and return -1 if there is an error

|

| Precodition: have a file to close

| Postcondition: the file requested is now closed

|

| Parameters:

|

| IN: the id associated with the file, the task id of that

| goes with that file

| Out: return value if the file was closed and a closed file

|

| Returns: closed file with a return value

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int close(int owner\_task\_id, int file\_id)

{

if(((owner\_task\_id == root)

or (owner\_task\_id == node\_file\_handle.owner\_task\_id))

and (file\_id == node\_file\_handle.starting\_block))

{

strcpy(node\_file\_handle.filename, "inode");

node\_file\_handle.owner\_task\_id = 0;

node\_file\_handle.starting\_block = 0;

node\_file\_handle.size = 0;

node\_file\_handle.permission = 0b0000;

node\_file\_handle.blocks[0] = 0;

node\_file\_handle.blocks[1] = 0;

node\_file\_handle.blocks[2] = 0;

node\_file\_handle.blocks[3] = 0;

return 0;

}

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: read\_char

|

| Purpose: Read a character from the file and then return -1 if

| an error occurs while keeping track of current file

| location

|

| Precodition: have a file to read from

| Postcondition: the character has been read from the file

|

| Parameters:

|

| IN: task ID associated with the file, the file pointer for

| the file location, pointer for the current file location

| Out: return value if the file had an error

|

| Returns: character read from a file and value sent back if error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int read\_char(int owner\_task\_id, int file\_handle, char\* ch)

{

if(((owner\_task\_id == root) or

(owner\_task\_id == node\_file\_handle.owner\_task\_id))

and (

((file\_handle >= node\_file\_handle.blocks[0]) and

(file\_handle <= node\_file\_handle.blocks[0] + BLOCK\_SIZE))

and

((file\_handle >= node\_file\_handle.blocks[1]) and

(file\_handle <= node\_file\_handle.blocks[1] + BLOCK\_SIZE))

and

((file\_handle >= node\_file\_handle.blocks[2]) and

(file\_handle <= node\_file\_handle.blocks[2] + BLOCK\_SIZE))

and

((file\_handle >= node\_file\_handle.blocks[3]) and

(file\_handle <= node\_file\_handle.blocks[3] + BLOCK\_SIZE)))

)

{

ifstream file("inode.txt");

if(not file.is\_open())

{

return -1;

}

file.seekg(file\_handle, file.end);

file.get(\*ch);

return 0;

}

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: write\_char

|

| Purpose: write a character to a file and return -1 if an error

| occurs while keeping track of current file location

|

| Precodition: have a file to read to

| Postcondition: the characters have been written to the file

|

| Parameters:

|

| IN: task ID associated with the file, the file pointer for

| the file location, pointer for the current file location

| Out: return value if the file had an error

|

| Returns: character written to a file and value sent back if error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int write\_char(int owner\_task\_id, int file\_handle, char\* ch)

{

if(((owner\_task\_id == root) or

(owner\_task\_id == node\_file\_handle.owner\_task\_id))

and (

((file\_handle >= node\_file\_handle.blocks[0]) and

(file\_handle <= node\_file\_handle.blocks[0] + BLOCK\_SIZE))

and

((file\_handle >= node\_file\_handle.blocks[1]) and

(file\_handle <= node\_file\_handle.blocks[1] + BLOCK\_SIZE))

and

((file\_handle >= node\_file\_handle.blocks[2]) and

(file\_handle <= node\_file\_handle.blocks[2] + BLOCK\_SIZE))

and

((file\_handle >= node\_file\_handle.blocks[3]) and

(file\_handle <= node\_file\_handle.blocks[3] + BLOCK\_SIZE)))

)

{

ofstream file("inode.txt");

if(not file.is\_open())

{

return -1;

}

file.seekp(file\_handle, file.end);

file.put(\*ch);

return 0;

}

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: create\_file

|

| Purpose: creates the file used for the whole filesystem class

| to use and do operations with

|

| Precodition: nothing

| Postcondition: the file has been created

|

| Parameters:

|

| IN: task ID associated with the file, the file size needed to

| create the new file, permissions to create the new file with

| Out: new file created

|

| Returns: new file was created and return value if error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// directory options

// ufs\_Create\_file gets the next\_file\_handle

int create\_file(int owner\_task\_id, char\* filename,

int file\_size,

int permission)

{

// create an i-node

i\_node node;

// control for input size, truncate user input if too long

for(int i = 0 ; i < 8 ; i++)

{

node.filename[i] = filename[i];

}

node.owner\_task\_id = owner\_task\_id;

// look for free block

//node.starting\_block

if(file\_size > 4 \* NUMBER\_OF\_BLOCKS \* BLOCK\_SIZE)

{

return -1;

}

node.size = file\_size;

if(permission <= 16 and permission >= 0)

{

node.permission = permission;

}

//create a file and open it to write

ifstream myfile("inode.txt");

//check to see if the file exists

if(!myfile.is\_open())

{

return -1;

}

// write to inode file in the right spot

myfile.close();

//open file for both read and write

myfile.open(filename, ios::in | ios::out);

//file was opened for read and write and created if it didnt exist

//check to see if i\_node available then try to reserve the data blocks

//if enough space is not available return a-1

//return file\_handle if successful

//if(i\_node.isFull() == true) //this is not a real call just a way to think of what needs to happen

{

//cant create a file because no inodes available

return 0;

}

//else

{

//

}

return -1; //a - 1; // what is 'a'?

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: del\_file

|

| Purpose: delete the file by checking the files owner. then

| delete the file from the blocks writting $

|

| Precodition: have a file to delete

| Postcondition: delete the file and replace the block with $

|

| Parameters:

|

| IN: task ID of the files owner, the file name to delete

| Out: deleted file

|

| Returns: file is deleted and returns value if error

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

//only owner of the file can delete a file

int delete\_file(int task\_id, string file\_name)

{

//creat an i-node

i\_node node;

ofstream myfile(file\_name, ofstream::trunc);

if(node.owner\_task\_id != task\_id)

{

dump\_screen << "Not the correct owner\n";

return -1;

}

else

{

myfile.seekp(0, ios::end);

//it is the correct file so now fill it with $

//while(myfile.tellp()==0)

if(myfile.tellp() == 0)

{

if(not myfile.is\_open())

{

return -1; //file is not open so return -1

}

//fill the file with $ as specified by the phase description

myfile.put('$');

return 0;

}

}

/\*

//check the files owner, delete the content of the file from the data blocks by writing $

//delete the file from the i\_node table

if(task\_id != file\_id)

{

//file is not the owner so exit/return

dump\_screen.print( "You are not the chosen one" << endl;

return 0;

}

else

{

while(myfile)//isempty != true

{

//write '$' to the file until it is full

write\_char(task\_id, file\_name, '$')

//perform operations to delete from i-node table

}

//do we need to return a value? can this be void?

return -1;

//

}

\*/

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: change\_permission

|

| Purpose: change the permissions access that a file has if you

| are the owner of that file

|

| Precodition: have a file that you can modify

| Postcondition: the permission of the file was changed

|

| Parameters:

|

| IN: task ID of the files owner,the name of the file you are

| trying to change, the new permissions to set

| Out: changed permissions access

|

| Returns: erturns a value to say if an error occured

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

int change\_permission(int task\_id, string file\_name,

int new\_permission)

{

//get what file you want to change with the ID

//what file you wish to change

//set the new permissions of this file

/\*

// let input be 0b0000 or some other binary format

if (new\_permission <= 16 and new\_permission >= 0)

{

permission = new\_permission;

}

\*/

//do we need to return anything here?

//we should be able to just change filepermission and leave

return -1;

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: dir

|

| Purpose: Show the directory, displaying all the file names and

| the permissions with each file

|

| Precodition: have a files to display for the directory

| Postcondition: the directory is displayed with permissions showed

|

| Parameters:

|

| IN: nothing

| Out: directory is displayed

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void dir()

{

//each attribute needs to be displayed

//dump\_screen.print( "File name: " << endl; //filename

//dump\_screen.print( "File size: " << endl; //size

//dump\_screen.print( "File ID: " << endl; //files ID

//dump\_screen.print( "File permissions: " << endl; //file permission

//display the file directory/information

//can use display file info function to display this info

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: dir

|

| Purpose: Show the directory, displaying all the files names

| and show permissions associated with the file

|

| Precodition: have files to display for the directory if you

| are the owner of that directory

| Postcondition: the directory is displayed with what permissions

| the files have if the onwer accesses it

|

| Parameters:

|

| IN: nothing

| Out: directory is displayed

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

//this function is the same as previous one but displays a certain tasks ID

void dir(int task\_id)

{

//same as above function but checks if the correct owner is making function call

// does not compile

/\*

if(task\_id != file\_id)

{

dump\_screen.print( "The cake is a lie, you are not the one I seek" <<

endl; //this would only occur if the file you want to display does not exist

//return 0;

}

else

{

//display the file information of the task

}

\*/

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: dump\_screen

|

| Purpose: Dump the ufs i\_nodes and data blocks to display

|

| Precodition: have a directory system to disaply information

| Postcondition: dump\_screened directory information

|

| Parameters:

|

| IN: nothing

| Out: directory is dump\_screened

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void dump() // as in dump\_screen overload

{

//dir functions kind of acts as a 'dump\_screen' function for the moment

dump\_screen.println(

"File | File | Blocks used by this file | Size | Starting | Status | Permission | Owner ID | Create Time | Modified Time");

dump\_screen.println(

"handle | Name | | | Block | | | | | ");

// TODO: fill all of these entries out.

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: display\_file\_info

|

| Purpose: display the information of the file in regards to

| directory information of the file

|

| Precodition: have a directory or files to display information

| Postcondition: the directory information is displayed

|

| Parameters:

|

| IN: a pointer to the filename/directory to show information

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

//taken from lab 10 can be used as the 'dir' function as it displays all the file information

void display\_file\_info(const char\* filename)

{

// does not compile

/\*

dump\_screen.print( "Use lstat() to get file information for: " <<

filename << endl;

struct stat st;

if(lstat(filename, &st) == -1)

{

perror("lstat");

exit(EXIT\_FAILURE);

}

printf("ID of containing device: [%lx,%lx]\n",

(long) major(st.st\_dev), (long) minor(st.st\_dev));

printf("File type: ");

// extract the file-type by ANDing the st\_mode by bitmask S\_IFMT

switch(status.st\_mode & S\_IFMT)

{

case S\_IFBLK:

printf("block device\n");

break;

case S\_IFCHR:

printf("character device\n");

break;

case S\_IFDIR:

printf("directory\n");

break;

case S\_IFIFO:

printf("FIFO/pipe\n");

break;

case S\_IFLNK:

printf("symlink\n");

break;

case S\_IFREG:

printf("regular file\n");

break;

case S\_IFSOCK:

printf("socket\n");

break;

default:

printf("unknown file type\n");

break;

}

printf("I-node number: %ld\n", (long) st.st\_ino);

printf("Mode: %lo (octal)\n", (unsigned long) st.st\_mode);

printf("Permission:\n");

if((st.st\_mode & S\_ISUID) == S\_ISUID)

{

printf("\t\t\t Set-User-ID bit is set\n");

}

if((st.st\_mode & S\_ISGID) == S\_ISGID)

{

printf("\t\t\t Set-Group-ID bit is set\n");

}

if((st.st\_mode & S\_ISVTX) == S\_ISVTX)

{

printf("\t\t\t Sticky bit is set\n");

}

dump\_screen.print( "\t\t\t " << show\_permissions(filename) << endl;

printf("Link count: %ld\n", (long) st.st\_nlink);

printf("Ownership: UID=%ld GID=%ld\n", (long) st.st\_uid,

(long) st.st\_gid);

printf("Preferred I/O block size: %ld bytes\n",

(long) st.st\_blksize);

printf("File size: %lld bytes\n", (long long) st.st\_size);

printf("Blocks allocated: %lld\n", (long long) st.st\_blocks);

printf("Last status change: %s", ctime(&st.st\_ctime));

printf("Last file access: %s", ctime(&st.st\_atime));

printf("Last file modification: %s", ctime(&st.st\_mtime));

\*/

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: show\_permissions

|

| Purpose: displays information in regards to the permissions a

| file has set to it

|

| Precodition: have a file to read the permissions of

| Postcondition: the permissions are displayed

|

| Parameters:

|

| IN: file pointer to point to the file in which we are displaying

| the permission rights

| Out: nothing

|

| Returns: returns which mode the file has, which are what permissions

| the file has

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

char\* show\_permissions(const char\* filename)

{

// does not compile

/\*

char\* mode = new char(32);

if(stat(filename, &st) == 0)

{

mode\_t perm = st.st\_mode;

mode[0] = (perm & S\_IRUSR) ? 'r' : '-';

mode[1] = (perm & S\_IWUSR) ? 'w' : '-';

mode[2] = (perm & S\_IXUSR) ? 'x' : '-';

mode[3] = (perm & S\_IRGRP) ? 'r' : '-';

mode[4] = (perm & S\_IWGRP) ? 'w' : '-';

mode[5] = (perm & S\_IXGRP) ? 'x' : '-';

mode[6] = (perm & S\_IROTH) ? 'r' : '-';

mode[7] = (perm & S\_IWOTH) ? 'w' : '-';

mode[8] = (perm & S\_IXOTH) ? 'x' : '-';

mode[9] = '\0';

return mode;

}

else

{

return(NULL);

}

\*/

return(NULL);

}

};

// OS class code section, informally

//{

/\* Used for method passing purposes \*/

void\* stub(void\* a)

{

//pthread\_detach(pthread\_self());

// I guess I function

pthread\_exit(NULL);

//return NULL;

}

/\*{===========================================================

||

|| Class: OS

||

|| Purpose: It is the main operating system class. This is what

|| should be interacted with by main.

|| OS acts as a direct access to the different classes.

|| It uses object references from classes to make function

|| calls from said classes

||

|| Attributes:

|| scheduler

|| window

|| ipc

|| memory

|| semaphore\_q

||

|| Functions:

|| --none at the moment--

||

\\*=========================================================}\*/

class OS

{

public:

// constants

//{

const int banner\_size = 8;

const int thread\_win\_h = 5;

//const int thread\_win\_w = (COLS/THREAD\_COUNT); // macro COLS needs context

//}

/\* At the moment this is direct access

But in the future an extra layer of

system calls could be added for control \*/

// super user is creator

pthread\_t self = pthread\_self();

string user; // single user system for now

int user\_id; // determined by user\_log order

// instances of all classes made so far

//{

Queue<Semaphore> semaphore\_queue;

Scheduler scheduler; // controls processes

Messenger messenger; // inter process communication

Memory memory; // memory blocks

File\_system file\_system; // gives long term memory

// window system is going to be global

//#define THREAD\_COUNT 3

//Window top\_window;

//Window thread[THREAD\_COUNT];

//Window terminal;

//Window dump\_screen;

//}

// stuff for testing

//{

Semaphore semaphore; // simple unnamed semaphore to play with

int Int = 0;

friend ostream& operator<<(ostream& out\_stream, OS& os)

{

out\_stream << "(OS "

<< os.semaphore\_queue << ' '

<< os.scheduler << ' '

// << os.windows << ' '

<< os.messenger << ' '

<< os.memory << ' '

<< os.file\_system

<< ')';

return out\_stream;

}

//}

void create\_semaphore(int resources = 1,

string name = "memory")

{

Semaphore s(resources, name);

semaphore\_queue.push(s);

}

// pretty print functions

//{

string banner =

" \_ \_ \_ \_\_\_\_\_ \_ \_\_\_\_\_ \_\_ \_\_ \n"

" ( ) ( )( ) (\_ \_)(\_)/'\\\_/`\\( \_ ) /'\_\_`\\ /' \_`\\ \n"

" | | | || | | | | || || (\_) | (\_) ) ) | ( ) | \n"

" | | | || | \_ | | | || (\_) || \_ | /' / | | | | \n"

" | (\_) || |\_( )| | | || | | || | | | /' /( ) \_ | (\_) | \n"

" (\_\_\_\_\_)(\_\_\_\_/\'(\_) (\_)(\_) (\_)(\_) (\_) (\_\_\_\_\_/\'(\_)`\\\_\_\_/'";

string print\_banner()

{

top\_window.print(banner);

return banner;

}

void print\_scheduler()

{

dump\_screen.print(

"======================================================="

"scheduler: functional\n"

" =======================================================\n"

);

}

void print\_semaphore()

{

dump\_screen.print(

"======================================================="

"semaphore: non-functional\n"

" =======================================================\n"

);

}

void print\_messenger()

{

dump\_screen.print(

"======================================================="

"messenger (ipc): non-functional\n"

" =======================================================\n"

);

}

void print\_memory()

{

dump\_screen.print(

"======================================================="

"memory: non-functional\n"

" =======================================================\n"

);

}

void print\_file()

{

dump\_screen.print(

"======================================================="

"file system: non-functional\n"

" =======================================================\n"

);

}

void print\_default()

{

dump\_screen.print(

"======================================================="

"\n"

" =======================================================\n"

);

}

//}

// decision tree

//{

void thread\_choose()

{

}

void scheduler\_choose(char& input)

{

string temp\_n;

//int thread\_id;

//string program\_name;

while(true)

{

terminal << "[p]print, "

"\n [c]create thread, "

"\n [d]destroy thread, "

"\n [r]run scheduler, "

"\n [y]yield, "

"\n [g]collect garbage, "

"\n [e]check if scheduler is empty, "

"\n [b]go back"

"\n [q]quit\n";

terminal.y += 9;

cin >> input;

switch(input)

{

case 'p':

dump\_screen.print(str(scheduler)); // TODO: pretty print

break;

case 'c':

terminal.println("Creating a thread");

terminal.println("What would you like to name this thread?");

cin >> temp\_n;

scheduler.create\_task(temp\_n, stub);

terminal << scheduler ;

break;

case 'd':

dump\_screen.println("Which task would you like to destroy?\n");

dump\_screen.println(str(scheduler)) ;

int temp\_d;

cin >> temp\_d;

scheduler.destroy\_task(temp\_d);

dump\_screen.println("Task " + str(temp\_d) +

" has been destroyed.\n");

break;

case 'r':

scheduler.run();

dump\_screen.println("The scheduler is currently running;\n" +

str(

scheduler));

break;

case 'y':

dump\_screen.println("Which task would you like to yield?\n" +

str(

scheduler));

int temp\_y;

cin >> temp\_y;

scheduler.yield(temp\_y);

dump\_screen.println("Task " + str(temp\_y) +

" has been halted.\n");

dump\_screen.print(str(scheduler));

break;

case 'g':

dump\_screen.print(str(scheduler));

dump\_screen.println("\nTime to collect the garbage.\n");

scheduler.garbage\_collect();

dump\_screen.print(str(scheduler));

break;

case 'e':

scheduler.empty();

break;

case 'b':

return;

break;

default:

endwin();

exit(0);

}

}

}

void semaphore\_choose(char& input)

{

}

void messenger\_choose(char& input)

{

}

void memory\_choose(char& input)

{

}

void file\_choose(char& input)

{

char\* buffer = new char[64]; // used for user input and output

while(true)

{

print\_file();

terminal <<

" [F]format file "

"\n [o]open file, "

"\n [c]close file, "

"\n [r]read char, "

"\n [w]write char, "

"\n [C]create file, "

"\n [D]delete file, "

"\n [P]change permission, "

"\n [d]dir, "

"\n [p]print, "

"\n [f]file info, "

"\n [s]scheme print, "

"\n [b]back, "

"\n [q]quit\n ";

terminal.y += 14; // move printer pointer

cin >> input;

switch(input)

{

case 'F':

file\_system.format();

break;

case 'o':

dump\_screen.println("open(file\_handle, file\_name, mode)\n");

file\_system.open(user\_id, "inode", 0b1111);

break;

case 'c':

file\_system.close(user\_id, 0);

break;

case 'r':

file\_system.read\_char(user\_id, 0, buffer);

dump\_screen.print(str(buffer));

break;

case 'w':

dump\_screen.println("char: ");

cin >> buffer; // not sanatized

file\_system.write\_char(user\_id, 0, buffer);

break;

case 'C': // block scope, keep namespace cleaner

{

dump\_screen.println("file name: ");

cin >> buffer; // will chop of at 7 char

dump\_screen.println("file size: ");

int file\_size = 0;

cin >> file\_size;

dump\_screen.println("permission(rwrw, in binary): "); // as binary string

int permission;

cin >> permission;

if(cin.good())

{

file\_system.create\_file(user\_id, buffer, file\_size,

permission);

}

}

break;

case 'D':

dump\_screen.println("file name: ");

cin >> buffer;

file\_system.delete\_file(user\_id, buffer);

break;

case 'P':

{

dump\_screen.println("file name: ");

cin >> buffer;

dump\_screen.println("permission(rwrw, in binary): "); // as binary string

int permission;

cin >> permission;

file\_system.change\_permission(user\_id, buffer, permission);

}

break;

case 'd':

if(user\_id == 0)

{

file\_system.dir();

}

else

{

file\_system.dir(user\_id);

}

break;

case 'p':

file\_system.dump();

break;

case 'f':

dump\_screen.println("file name: ");

cin >> buffer;

file\_system.display\_file\_info(buffer);

break;

case 's':

dump\_screen.print(str(file\_system)) ;

break;

case 'b':

goto end;

default:

endwin();

exit(0);

}

}

end:

;

}

void log\_new\_user(char& input)

{

// append to file

fstream user\_log("users.txt", ios::app);

string username;

dump\_screen.println("Username: ");

cin >> username;

// could check if name is taken, ah well

user\_log << username;

user\_log << '\n';

user\_log.close();

}

bool log\_in(char& input)

{

print\_default();

string username;

terminal.println("Username: ");

cin >> username;

// no password

ifstream user\_log("users.txt");

if(user\_log.is\_open())

{

string file\_line;

user\_id = 0;

while(getline(user\_log, file\_line))

{

if(file\_line == username)

{

user = username;

dump\_screen.println("Welcome\n");

user\_log.close();

goto end;

}

user\_id++; // id is based on order in the user\_log file, root = 0

}

}

user\_log.close();

dump\_screen.println("User not found\n");

return false;

// found user

end:

return true;

}

void get\_user(char& input)

{

while(true)

{

print\_default();

terminal << "choose: "

"\n [n]new user, "

"\n [l]login\n ";

terminal.y += 3;

cin >> input;

switch(input)

{

case 'n':

log\_new\_user(input);

break;

case 'l':

if(log\_in(input) == true)

{

main\_prompt(input);

}

else

{

break;

}

default :

endwin();

exit(0);

}

}

}

void main\_prompt(char& input)

{

print\_default();

terminal << "choose: "

"\n [s]scheduler, "

"\n [S]semaphore, "

"\n [m]messenger, "

"\n [M]memory, "

"\n [f]file,"

"\n [l]logout,"

"\n [q]quit\n";

terminal.y += 8;

while(true)

{

cin >> input;

switch(input)

{

case 's':

print\_scheduler();

scheduler\_choose(input);

break;

case 'S':

semaphore\_choose(input);

break;

case 'm':

messenger\_choose(input);

break;

case 'M':

memory\_choose(input);

break;

case 'f':

file\_choose(input);

break;

case 'l':

goto end;

break;

default :

endwin();

exit(0);

}

}

end:

;

}

//}

void refresh\_all()

{

refresh(); // parent needs to..

top\_window.refresh();

for(int i = 0 ; i < THREAD\_COUNT ; i++)

{

thread[i].refresh();

}

terminal.refresh();

dump\_screen.refresh();

}

void run()

{

// start ncursing

initscr();

refresh(); // parent needs to refresh, probably

cbreak();

top\_window.create(banner\_size, COLS, 0, 0);

for(int i = 0 ; i < THREAD\_COUNT ; i++)

{

thread[i].create(thread\_win\_h, COLS / THREAD\_COUNT,

banner\_size, i \* (COLS / THREAD\_COUNT));

}

terminal.create(LINES - (banner\_size + thread\_win\_h),

COLS / 2, 13, 0);

dump\_screen.create(LINES - (banner\_size + thread\_win\_h),

COLS / 2, 13,

COLS / 2);

top\_window << banner;

refresh\_all();

char input = ' ';

print\_banner();

get\_user(input);

endwin();

}

}; // end of OS class

//}

/\*

Make a global object of OS

This gives OS the first thread, so other threads are not super user.

\*/

OS os;

/\*

\*/

## 

## Test.cpp

/\*

This file is a test suit for the ULTIMA 2.0 OS-like system

Jordan Winkler, Kyle Lucas, Joshua Carter

Mon Apr 13 21:55:05 EDT 2020

Phase 1: Scheduler and Semaphore

Phase 2: interprocess communication

Phase 3: memory

Phase 4: file system

\*/

// basic test harness

//{

#include <stdio.h>

#define literal(thing) #thing

// color macros for error print

#define red "\033[0;31m"

#define green "\033[0;32m"

#define clear\_color "\033[0m"

// counting arguments of a macro

#define arg\_11(a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, ...) a11

#define arg\_count(...) ELEVENTH\_ARGUMENT(dummy, ## \_\_VA\_ARGS\_\_, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)

#include "general.cpp" // code file

// incremented throughout testing

unsigned long int tests = 0;

unsigned long int good\_tests = 0;

unsigned long int bad\_tests = 0;

// Macros are made to avoid going out of scope

// end if test fails

#define require(predicate) \

tests++; \

if(!(predicate)) { \

printf("%s:%d:%s: ",\_\_FILE\_\_,\_\_LINE\_\_,\_\_func\_\_); \

printf(red "error: %s" clear\_color "\n\n", \

literal(predicate)); \

printf("tests passed: %lu/%lu\n",good\_tests,tests); \

exit(0); \

} \

else \

{ \

good\_tests++; \

}

// do not end if test fails, just print it

#define check(predicate,...) \

tests++; \

if(!(predicate)) { \

printf("%s:%d:%s: ",\_\_FILE\_\_,\_\_LINE\_\_,\_\_func\_\_); \

printf(red "error: %s\n\n" clear\_color, \

literal(predicate)); \

} \

else \

{ \

good\_tests++; \

}

#define print\_if\_there(...) \

if (arg\_count(\_\_VA\_ARGS\_\_) == 1) \

printf("%s\n",\_\_VA\_ARGS\_\_); \

// just print everything (lower random tests first)

#define verbose(predicate,...) \

tests++; \

if(!(predicate)) { \

printf("line[%d]:function[%s]:",\_\_LINE\_\_,\_\_func\_\_); \

printf("error[%s]\n", \

literal(predicate)); \

} \

else \

{ \

printf("line[%d]:function[%s]:",\_\_LINE\_\_,\_\_func\_\_); \

printf("pass[%s]\n", \

literal(predicate)); \

good\_tests++; \

}

// print the total amount of tests

#define end\_tests() \

if(good\_tests == tests) \

printf("\ntests passed:" green " %lu/%lu\n" \

clear\_color,good\_tests,tests); \

else \

printf("\ntests passed: %lu/%lu\n",\

good\_tests,tests);

// make a string of any registered object

// note: loops twice in gdb?

string str(auto& a)

{

stringstream ss;

ss << a;

return ss.str();

}

//} end of basic test code

// collection of void functions that implicitly part of MCB class

//{

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_semaphore\_function1

|

| Purpose: a basic test of the semaphore class making sure up

| and down are working properly when int is 1. also used

| in semaphore unit test and thread 1.

|

| Precodition: design a semaphore class

| Postcondition: semaphore class has additional tested working

| functionality

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void\* test\_semaphore\_function1(void\* input)

{

check(input != NULL);

OS\* os = (OS\*) input;

os->semaphore.down();

os->Int = 1;

os->semaphore.up();

pthread\_exit(NULL);

};

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_semaphore\_function2

|

| Purpose: a basic test of the semaphore class making sure up

| and down are working properly when int is 2. also used

| in semaphore unit test and thread 2.

|

| Precodition: design a semaphore class

| Postcondition: additional semaphore class testing

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void\* test\_semaphore\_function2(void\* input)

{

check(input != NULL);

OS\* os = (OS\*) input;

os->semaphore.down();

os->Int = 2;

os->semaphore.up();

pthread\_exit(NULL);

};

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_sema\_sched\_fun1

|

| Purpose: test setting memory during semaphore up/down so that

| this can be used later in integration tests and thread

| creating.

|

| Precodition: design a semaphore & scheduler class

| Postcondition: checked setting memory for scheduler testing

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void\* test\_sema\_sched\_fun1(void\* input)

{

check(input != NULL);

OS\* os = (OS\*) input;

os->semaphore.down();

os->memory.raw\_memory[0] = '1';

os->semaphore.up();

pthread\_exit(NULL);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_sema\_sched\_fun2

|

| Purpose: test setting memory during semaphore up/down so that

| this can be used later in integration tests and thread

| creating.

|

| Precodition: design a semaphore & scheduler class

| Postcondition: checked setting memory for scheduler testing

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void\* test\_sema\_sched\_fun2(void\* input)

{

check(input != NULL);

OS\* os = (OS\*) input;

os->semaphore.down();

os->memory.raw\_memory[0] = '2';

os->semaphore.up();

pthread\_exit(NULL);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: t\_sema2\_sched\_ipc\_fun1

|

| Purpose: test IPC processes with the scheduler while using

| semaphore to protect critical sections

|

| Precodition: design a semaphore & scheduler class

| Postcondition: tested IPC functionality

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void\* t\_sema2\_sched\_ipc\_fun1(void\* input)

{

check(input != NULL);

pthread\_exit(NULL);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_mail\_function1

|

| Purpose: test mailboxes with threads

|

| Precodition: design a message & scheduler class

| Postcondition: checked mailbox functionality

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void\* test\_mail\_function1(void\* input)

{

check(input != NULL);

//Safe\_messenger\* os = (Safe\_messenger\*) input;

pthread\_exit(NULL);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_mail\_function2

|

| Purpose: test mailboxes with threads

|

| Precodition: design a message & scheduler class

| Postcondition: checked mailbox functionality

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void\* test\_mail\_function2(void\* input)

{

check(input != NULL);

//Safe\_messenger\* os = (Safe\_messenger\*) input;

pthread\_exit(NULL);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_memory\_function

|

| Purpose: test the memory class by testing functions. allocate

| memory, write to it, read the memory, check that the

| pages are working and run basic memory functions.

|

| Precodition: design a semaphore & scheduler class

| Postcondition: checked setting memory for scheduler testing

|

| Parameters:

|

| IN: input pointer to a class in os

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// because set up checks memory constructor is not one of the functions using the class

void\* test\_memory\_function(void\* input)

{

check(input != NULL);

OS\* os = (OS\*) input;

int pointer = os->memory.alloc(1);

check(pointer != -1); // todo: change, magic\_numbers == bad

check(os->memory.write(pointer, 'c') != -1);

char\* ch = new char[PAGE\_SIZE];

for(int i = 0 ; i < PAGE\_SIZE ; i++)

{

ch[i] = 0;

}

check(os->memory.read(pointer, ch) != -1);

check(\*ch != 0);

check(\*ch == 'c');

for(int i = 0 ; i < PAGE\_SIZE ; i++)

{

ch[i] = 'a';

}

// basic block write

check(os->memory.write(pointer, 0, PAGE\_SIZE, ch) != -1);

// set ch to something else

for(int i = 0 ; i < PAGE\_SIZE ; i++)

{

ch[i] = 'b';

}

check(os->memory.read(pointer, 0, PAGE\_SIZE, ch) != -1);

for(int i = 0 ; i < PAGE\_SIZE ; i++)

{

check(ch[i] == 'a');

}

check(os->memory.left() == PAGES - 1);

check(os->memory.largest() == PAGES - 1);

check(os->memory.smallest() == PAGES - 1);

check(os->memory.free(pointer) != -1);

check(os->memory.left() == PAGES);

check(os->memory.largest() == PAGES);

check(os->memory.smallest() == PAGES);

pthread\_exit(NULL); // test continue in tester class

}

//}

/\*{===========================================================

||

|| Class: MCB

||

|| Purpose: It is the main operating system class. This is what

|| should be interacted with by main.

|| OS acts as a direct access to the different classes.

|| It uses object references from classes to make function

|| calls from said classes

||

|| Attributes:

|| test\_size

||

|| Functions:

|| test\_q\_int\_check()

|| test\_q\_int()

|| test\_stack\_int()

|| test\_binary\_tree()

|| test\_priority\_q()

|| test\_scheduler()

|| test\_semaphore()

|| test\_mailbox()

|| test\_memory()

|| test\_file()

|| test\_integration()

|| run\_tests()

||

\\*=========================================================}\*/

class MCB

{

const int test\_size = 50; // size of random tests

public:

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_q\_int\_check

|

| Purpose: test the queue class to ensure basic working functions.

| test that the constructor, pushing, counting the pushes,

| popping, reuse and ensure secondary functions work

|

| Note: This is an earlier version of testing that has since

| been updated to be automated and removed from core tests

|

| Precodition: design a template queue class

| Postcondition: checked working functionality of queue class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void test\_q\_int\_check()

{

Queue<int> int\_q;

//int errors = 0;

cout << "testing Queue<int>\n";

cout << "================================\n\n";

// test init

Queue<int> temp;

Queue<int> temp2;

cout << "construction\n";

cout << "--------------------------------\n";

cout << "temp: " << temp << endl;

cout << "temp2: " << temp2 << "\n\n";

// test pushing

for(int i = 0 ; i < 10 ; i++)

{

temp.push(i);

}

// test cout of pushing

cout << "pushing\n";

cout << "--------------------------------\n";

cout << "temp: " << temp << endl;

cout << "temp2: " << temp2 << "\n\n";

// test poping, empty, and multiple allocation

while(temp.empty() == false)

{

temp2.push(temp.pop());

}

cout << "transfer by push and pop\n";

cout << "--------------------------------\n";

cout << "temp: " << temp << endl;

cout << "temp2: " << temp2 << "\n\n";

cout << "re-transfer by push and pop\n";

cout << "--------------------------------\n";

// test reuse

while(temp2.empty() == false)

{

temp.push(temp2.pop());

}

cout << "temp: " << temp << endl;

cout << "temp2: " << temp2 << "\n\n";

// failed for now

// test deconstructor, copy constructors

//auto id = [](Queue<int> in) -> Queue<int> {return in;};

//cout << id(temp) << endl;

//cout << "deconstructor, copy\n";

//cout << "--------------------------------\n";

//cout << "temp: " << temp << endl;

//cout << "temp2: " << temp2 << "\n\n";

// test first

cout << "secondary functions\n";

cout << "--------------------------------\n";

cout << temp << endl;

cout << "front: " << temp.front() << endl;

cout << "size: " << temp.size() << endl;

cout << "empty: " << temp.empty() << endl;

//cout << "print: "; temp.print(); cout << "\n\n";

// test data

cout << "data: size and private info\n";

cout << "--------------------------------\n";

//cout << temp.\_size << endl;

cout << temp << endl;

cout << "size: " << sizeof(temp) << endl;

cout << "head: " << temp.head << endl;

cout << "tail: " << temp.tail << endl;

cout << "\_size: " << temp.\_size << "\n\n";

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_q\_int

|

| Purpose: stress test the template q class to see if it can

| handle everything. Tests overflow, if it is empty,

| array assertion and basic tests

|

| Precodition: design a template q class

| Postcondition: the q class is fully tested and working

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// unit tests of a template queue

void test\_q\_int()

{

Queue<int> int\_q;

srand(time(0));

auto q = int\_q;

//check(q == master.int\_q);

// basic tests

//check(str(q) != "(Queue)");

check(str(q) == "(Queue)");

check(q.empty() == true);

check(q.size() == 0);

auto object = 1;

q.push(object);

check(str(q) == "(Queue 1)"); // not general

check(q.size() == 1);

//check(str(q.front()) == str(object));

check(q.empty() == false);

object = q.pop();

check(str(object) == "1");

check(str(q) == "(Queue)");

check(q.empty() == true);

// test basic array of assertions

const string front = "(Queue";

const string end = ")";

string front\_mid;

string test\_str;

front\_mid = front;

int input;

for(int i = 0 ; i < test\_size ; i++)

{

//cout << i << ' ';

input = rand();

front\_mid += (' ' + str(input));

test\_str = front\_mid + end;

q.push(input);

check(test\_str == str(q));

}

// test empty

while(q.empty() == false)

{

q.pop();

}

check(str(q) == "(Queue)");

// test int overflow

front\_mid = front;

for(int i = 1 ; i > 0 ; i \*= 100)

{

front\_mid += ' ' + str(i);

test\_str = front\_mid + end;

q.push(i);

check(test\_str == str(q));

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_stack\_int

|

| Purpose: Stress test for the template stack class. Test to

| make sure it can handle different situations such

| as array assertion, if it is empty, overflow, and

| basic unit tests to ensure it is working

|

| Precodition: design a template stack class

| Postcondition: the stack class is fully tested and working

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// unit tests of a oslate stack

void test\_stack\_int()

{

Stack<int> int\_stack;

srand(time(0));

auto s = int\_stack;

//check(q == master.int\_q);

// basic tests

//check(str(q) != "(Queue)");

check(str(s) == "(Stack)");

check(s.empty() == true);

check(s.size() == 0);

auto object = 1;

s.push(object);

check(str(s) == "(Stack 1)"); // not general

check(s.size() == 1);

//check(str(q.front()) == str(object));

check(s.empty() == false);

object = s.pop();

check(str(object) == "1");

check(str(s) == "(Stack)");

check(s.empty() == true);

// test basic array of assertions

const string front = "(Stack";

const string end = ")";

string mid;

string test\_str;

int input;

for(int i = 0 ; i < test\_size ; i++)

{

input = rand();

mid = (' ' + str(input)) + mid;

test\_str = front + mid + end;

s.push(input);

//cout << input << endl;

//cout << s << endl;

check(test\_str == str(s));

}

// test empty

while(s.empty() == false)

{

s.pop();

}

check(str(s) == "(Stack)");

// test int overflow

mid = "";

for(int i = 1 ; i > 0 ; i \*= 100)

{

mid = ' ' + str(i) + mid;

test\_str = front + mid + end;

s.push(i);

check(test\_str == str(s));

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_binary\_tree

|

| Purpose: Stress test for a template binary tree class. Test to

| see if the template class is functional with basic

| test cases

|

| Precodition: design a template binary tree class

| Postcondition: have a fully tested and working binary tree class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// unit tests of a oslate stack

void test\_binary\_tree()

{

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_priority\_q

|

| Purpose: Stress test for a template priority queue class. Test

| to see if the template class is functional with basic

| test cases

|

| Precodition: design a template priority queue class

| Postcondition: have a fully tested and working priority queue

| class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// unit tests of a oslate stack

void test\_priority\_q()

{

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_scheduler

|

| Purpose: Stress test for the scheduler class. Test the class

| by creating 3 tasks to start and fill them. Let the threads

| run and slowly kill the tasks. Check correct states along

| the way to ensure each case is covered.

|

| Precodition: create a scheduler class and support functions

| Postcondition: have a fully tested and working scheduler class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// unit tests of scheduler

void test\_scheduler()

{

// set number of tasks

const int THREADS = 3;

// try to use only os calls

OS os;

// create some stub threads

for(int i = 0 ; i < THREADS ; i++)

{

os.scheduler.create\_task("test", stub);

}

// run them, and kill them off slowly

int kill\_id = 0;

while(os.scheduler.task\_block\_q.empty() == false

and kill\_id < THREADS)

{

// cycle task

Scheduler::Task\_block temp = os.scheduler.task\_block\_q.pop();

os.scheduler.task\_block\_q.push(temp);

os.scheduler.destroy\_task(kill\_id);

kill\_id++;

}

os.scheduler.garbage\_collect();

// TODO: show yielding works, and states all signal properly

// was scrapped on redesign

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_semaphore

|

| Purpose: Stress test for the semaphore class. Test the class

| by creating 2 task threads. Let the threads run and attempt

| to access the same resource at the same time. If a value

| returned is 3 or 0 then writting at the same time occured

| meaning the semaphore didn't work

|

| Precodition: create a semaphore class and support functions

| Postcondition: have a fully tested and working semaphore class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// unit tests of scheduler

void test\_semaphore()

{

// for sake of void\* casting

OS os;

int thread\_error\_code;

pthread\_t process1 = 0;

thread\_error\_code = pthread\_create(&process1, NULL,

&test\_semaphore\_function1,

(void\*) &os);

// max threads

check(thread\_error\_code != EAGAIN);

// invalid attributes

check(thread\_error\_code != EINVAL);

// lacks permission with the pthread

// library's scheduler

check(thread\_error\_code != EPERM);

pthread\_t process2 = 0;

thread\_error\_code = pthread\_create(&process2, NULL,

&test\_semaphore\_function2,

(void\*) &os);

// max threads

check(thread\_error\_code != EAGAIN);

// invalid attributes

check(thread\_error\_code != EINVAL);

// lacks permission with the pthread

// library's scheduler

check(thread\_error\_code != EPERM);

// wait for both threads to finish

pthread\_join(process1, NULL);

pthread\_join(process2, NULL);

// getting another value, ie 3 or 0, means writing happened at the same time

check((os.Int == 1) or (os.Int == 2))

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_mailbox

|

| Purpose: Stress test for the message class which is a mailbox.

| Start by creating a member of the class and loading information

| into the class, description and destination. Test if the

| message was received, sending messages, counting messages,

| and testing the thread/tasks for errors

|

| Precodition: create a message class and support functions

| Postcondition: have a fully tested and working message class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

// test for the ipc

void test\_mailbox()

{

OS os;

// created for first round of tests

Messenger::Message message;

Messenger::Maybe\_message maybe\_message;

message.description = "Hello";

//sending to self

message.destination\_task\_id = pthread\_self();

check(str(os.messenger) ==

"(Messenger)");

maybe\_message = os.messenger.receive(

&(os.messenger.messages),

&(os.semaphore));

// meaning message was not recieved

check(str(maybe\_message) ==

"(Maybe\_message (Message 0 0 0 0 ) 0)");

// would be good to shorten this

os.messenger.send(message,

&(os.messenger.messages),

&(os.semaphore));

check(str(os.messenger) ==

"(Messenger (Message 0 Hello 0 0 0 ))");

maybe\_message = os.messenger.receive(

&(os.messenger.messages),

&(os.semaphore));

check(str(maybe\_message) ==

"(Maybe\_message (Message 0 Hello 0 0 0 ) 1)");

// repeatability

maybe\_message = os.messenger.receive(

&(os.messenger.messages),

&(os.semaphore));

check(str(maybe\_message) ==

"(Maybe\_message (Message 0 Hello 0 0 0 ) 1)");

// count\_letters test

check(os.messenger.count\_letters() == 1);

check(os.messenger.count\_letters\_to(

pthread\_self()) == 1);

os.messenger.delete\_all\_letters(pthread\_self());

// needs to be updated

//check(str(os.messenger) == "(Messenger)");

int thread\_error\_code;

// testing all of the threading stuff

pthread\_t process1 = 0;

thread\_error\_code = pthread\_create(&process1, NULL, &test\_mail\_function1,

(void\*) &os);

// max threads

check(thread\_error\_code != EAGAIN);

// invalid attributes

check(thread\_error\_code != EINVAL);

// lacks permission with the pthread

// library's scheduler

check(thread\_error\_code != EPERM);

pthread\_t process2 = 0;

thread\_error\_code = pthread\_create(&process2, NULL, &test\_mail\_function2,

(void\*) &os);

// max threads

check(thread\_error\_code != EAGAIN);

// invalid attributes

check(thread\_error\_code != EINVAL);

// lacks permission with the pthread

// library's scheduler

check(thread\_error\_code != EPERM);

// wait for both threads to finish

pthread\_join(process1, NULL);

pthread\_join(process2, NULL);

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_memory

|

| Purpose: Stress test for the memory class. Test the class

| by filling the 1024 bytes. Test printing and copy

| constructors for the memory. Allocate a chunk of memory

| and check the page name system. Test that threads are

| properly working. Free up memory and test multiblocks.

|

| Precodition: create a memory class and support functions

| Postcondition: have a fully tested and working memory class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void test\_memory()

{

OS os;

// memory representations

string blank1024;

for(int i = 0 ; i < MEM\_SIZE ; i++)

{

blank1024 += '.';

}

string character\_bit = "c";

for(int i = 1 ; i < MEM\_SIZE ; i++)

{

character\_bit += '.';

}

// test construction and printing

check(str(os.memory) == "(Memory " + blank1024 + ")");

// test copy construction

Memory mem\_block2 = os.memory;

check(str(mem\_block2) == "(Memory " + blank1024 + ")");

// test alloc

//// page name matches

int error;

// testing all of the threading stuff

pthread\_t process1 = 0;

error = pthread\_create(&process1, NULL, &test\_memory\_function,

(void\*) &(os));

// max threads

check(error != EAGAIN);

// invalid attributes

check(error != EINVAL);

// lacks permission with the pthread

// library's scheduler

check(error != EPERM);

pthread\_join(process1, NULL);

//// test multi block allocation

string freed32;

for(int i = 0 ; i < PAGE\_SIZE ; i++)

{

freed32 += '#';

}

string empty\_rest;

for(int i = 0 ; i < (PAGES - 1)\*PAGE\_SIZE ; i++)

{

empty\_rest += '.';

}

check(str(os.memory) == "(Memory " + freed32 + empty\_rest +

")");

// test free

//// page name becomes owner thread again

check(os.memory.coalesce() != -1);

for(int i = 0, j = PAGES; i < PAGES ; i++, j--)

{

check(os.memory.page.name[i] == pthread\_self());

}

for(int i = 0, j = PAGES - 1; i < PAGES ; i++, j--)

{

check(os.memory.page.free\_size[i] == j);

}

for(int i = 0, j = PAGES; i < PAGES ; i++, j--)

{

check(os.memory.page.end[i] == 0);

}

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: test\_file

|

| Purpose: Stress test for the file class. Test the class

| by creating I-nodes, doing filesystem, file and directory

| operations and print out appropriate information

|

| Precodition: create a file class and support functions

| Postcondition: have a fully tested and working file class

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void test\_file()

{

}

/\* done with unit tests \*/

// integrated testing

void test\_integration()

{

OS os;

// can task blocks allocate and deallocate well

//srand(time(0));

Queue<Scheduler::Task\_block> q;

//check(q == master.int\_q);

// basic tests

check(str(q) == "(Queue)");

check(q.empty() == true);

check(q.size() == 0);

Scheduler::Task\_block object;

//string obj\_str str(object);

q.push(object);

check(str(q) == ("(Queue " + str(object) +

")")); // not general

check(q.size() == 1);

//check(str(q.front()) == str(object));

check(q.empty() == false);

object = q.pop();

//check(str(object) == "1");

check(str(q) == "(Queue)");

check(q.empty() == true);

// test basic array of assertions

const string front = "(Queue";

const string end = ")";

string front\_mid;

string test\_str;

front\_mid = front;

auto input = object;

for(int i = 0 ; i < test\_size ; i++)

{

//input = rand();

front\_mid += ' ' + str(input);

test\_str = front\_mid + end;

q.push(input);

check(test\_str == str(q));

}

// test empty

while(q.empty() == false)

{

q.pop();

}

check(str(q) == "(Queue)");

os.scheduler.create\_task("test", &test\_sema\_sched\_fun1,

(void\*) &os);

os.scheduler.create\_task("test", &test\_sema\_sched\_fun2,

(void\*) &os);

os.scheduler.run();

// remove from scheduler instead of waiting on both of them to complete

auto temp = os.scheduler.task\_block\_q.pop();

pthread\_join(temp.thread, NULL);

temp = os.scheduler.task\_block\_q.pop();

pthread\_join(temp.thread, NULL);

// check for exit from structure

//check(str(scheduler) == "(Scheduler (Queue) 2 1)");

string one = "1";

string two = "2";

string blank2\_16;

for(int i = 1 ; i < MEM\_SIZE ; i++)

{

blank2\_16 += ".";

}

// check if data changed size 2

//check((str(test\_memory) == "(Safe\_memory (Semaphore memory 1) (Memory " + two + blank2\_16 + "))") or (str(test\_memory) == "(Safe\_memory (Semaphore memory 1) (Memory " + one + blank2\_16 + "))"));

#undef THREADS

//cout << \*test\_memory;

// needs clean up

/\*

#define THREADS 2

//create 2 tasks

for(int i = 0; i < THREADS; i++)

{

scheduler.create\_task("test", stub);

}

//while the task block isnt empty of tasks do stuff

while(scheduler.task\_block\_q.empty() == false)

{

Scheduler::Task\_block os = scheduler.task\_block\_q.pop();

scheduler.task\_block\_q.push(os);

//let one task send a message to another task

if(os.state == "ready" and os.semaphore == NULL)

{

os.state = "running";

pthread\_create(&os.thread, NULL, os.function, &os.data);

messenger.send("test message", scheduler.id = 1,

messagekey.down()); // 90% sure this line isn't right.

}

else if(os.state == "ready")

{

//since the semaphore is not NULL but the next message is in the ready state it has to wait before it can send a message.

if(os.semaphre->is\_down())

{

os.state = "blocked";

}

if(os.state != "blocked")

{

// semaphore has been released so now we can send a message.

os.semaphore->down();

os.state = "running";

//messenger.send

os.state = "ready";

os.semaphore->up();

}

}

scheduler.destroy\_task(

kill\_id); // destroy the tasks after they send messages

kill\_id++;

scheduler.garbage\_collect(); // call garbage collect to get rid of stuff

}

// run and kill of the threads slowly

int kill\_id = 0;

while(scheduler.task\_block\_q.empty() == false

and kill\_id < THREADS)

{

if(kill\_id == 0)

{

check();

}

if(os.state == "ready" and os.semaphore == NULL)

{

}

else if(os.state == "ready")

{

if(os.semaphore->is\_down())

{

os.state = "blocked";

}

if(os.state != "blocked")

{

os.semaphore->down();

os.state = "running";

pthread\_create(&os.thread, NULL, os.function, &os.data);

os.state = "ready";

os.semaphore->up();

}

}

scheduler.destroy\_task(kill\_id);

kill\_id++;

if(kill\_id == 1)

{

check();

}

scheduler.garbage\_collect();

}

#undef THREADS

\*/

}

/\*{\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| Name: run\_tests

|

| Purpose: Create a run function that can be used to call all

| of the unit and integration test cases. These will

| run/be called by phases of the project.

|

| Precodition: create functions to be called and tested

| Postcondition: run all the tests from one function call

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

\\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*}\*/

void run\_tests()

{

/\*\*\* data structures \*\*\*/

// unit tests

test\_q\_int();

test\_stack\_int();

test\_binary\_tree();

test\_priority\_q();

/\*\*\* phase 1 \*\*\*/

// unit tests

test\_scheduler();

test\_semaphore();

/\*\*\* phase 2 \*\*\*/

// unit tests

test\_mailbox();

/\*\*\* phase 3 \*\*\*/

// unit tests

test\_memory();

/\*\*\* phase 4 \*\*\*/

// unit tests

test\_file();

// integration tests

test\_integration();

// called to display the totals

end\_tests();

}

};

MCB mcb;

int main()

{

mcb.run\_tests();

return 0;

}

/\*

\*/

## Data Structures

### Class: Binary\_search\_tree

/\*================================================================

||

|| Class: binary search tree

||

|| Purpose: make message class faster

||

|| Attributes:

|| Nodes (elements of tree)

|| datum

|| left, right pointers

|| root (location of base)

||

|| Functions:

|| initializer

|| deconstructor

|| insert

|| delete\_branch

|| search

|| minimum

|| maximum

|| preorder

|| inorder

|| postorder

|| print (default inorder)

||

|| Friends with:

|| Test\_suite

||

\\*==============================================================\*/

template<class type> class Binary\_search\_tree

{

private:

struct Node

{

type datum;

Node\* left, \*right;

};

Node\* root;

public:

Binary\_search\_tree()

{

root = NULL;

}

~Binary\_search\_tree()

{

delete\_branch(root);

}

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

| Name: insert

|

| Precodition: nothing

| Postcondition: tree has new leaf

|

| Parameters:

|

| IN: datum of the declare binary tree type

| Out: nothing

|

| Returns: nothing

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

void insert(type datum)

{

// Ask for memory

Node\* new\_node = (Node\*) memory.alloc(sizeof(Node)/PAGE\_SIZE);

new\_node->datum = datum;

new\_node->right = NULL;

new\_node->left = NULL;

// start at the base, get into position

Node\* p = root;

Node\* trail\_p = NULL;

while(p != NULL)

{

trail\_p = p;

if(new\_node->datum < p->datum)

{

p = p->left;

}

else

{

p = p->right;

}

}

// Assign the data

if(trail\_p == NULL) // tree was empty

{

root = new\_node;

root->datum = new\_node->datum;

}

else if(new\_node->datum < trail\_p->datum)

{

trail\_p->left = new\_node;

}

else

{

trail\_p->right = new\_node;

}

}

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

| Name: delete\_branch

|

| Precodition: nothing

| Postcondition: section of binary search tree is removed

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

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

void delete\_branch(Node\* node)

{

if(node == NULL)

{

return;

}

delete\_branch(node->right);

delete\_branch(node->left);

delete node;

node = NULL;

}

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

| Name: search

|

| Precodition: nothing

| Postcondition: location of data, if it is on the tree

|

| Parameters:

|

| IN: node of binary search tree, datum of search

| Out: nothing

|

| Returns: location of datum or NULL

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

Node\* search(Node\* node, type key)

{

if(node == NULL or key == node->datum)

{

return node;

}

if(key < node->datum)

{

return search(node->left, key);

}

else

{

return search(node->right, key);

}

}

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

| Name: minimum

|

| Precodition: nothing

| Postcondition: location of minimum datum data has been found

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: location of min datum or NULL

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

Node\* minimum(Node\* node)

{

while(node->left != NULL)

{

node = node->left;

}

return node;

}

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

| Name: maximum

|

| Precodition: nothing

| Postcondition: location of maximum datum data has been found

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: location of max datum or NULL

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

Node\* maximum(Node\* node)

{

while(node->right != NULL)

{

node = node->right;

}

return node;

}

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

| Name: preorder

|

| Precodition: nothing

| Postcondition: preorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

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

void preorder(Node\* node)

{

if(node != NULL)

{

//cout << node->datum << ' ';

preorder(node->left);

preorder(node->right);

}

}

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

| Name: inorder

|

| Precodition: nothing

| Postcondition: inorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

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

void inorder(Node\* node)

{

if(node != NULL)

{

inorder(node->left);

//cout << node->datum << endl;//' ';

inorder(node->right);

}

}

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

| Name: postorder

|

| Precodition: nothing

| Postcondition: postorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: node of binary search tree

| Out: nothing

|

| Returns: nothing

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

void postorder(Node\* node)

{

if(node != NULL)

{

postorder(node->left);

postorder(node->right);

//cout << node->datum << ' ';

}

}

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

| Name: print

|

| Precodition: nothing

| Postcondition: inorder walk of a binary search tree was

| printed to standard out

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

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

void print()

{

inorder(root);

}

friend class Test\_suite;

};

### Class: Stack

/\*============================================================

||

|| Class: Stack<datum\_type>

||

|| Purpose: A handcrafted textbook Stack, easy data

|| structure to work with.

||

|| Attributes:

|| Node

|| TYPE Element

|| Node\* next

|| Top

|| stack\_size

||

|| Functions:

|| Stack

|| ~Stack

|| reset

|| empty

|| push

|| pop

|| peek

|| print

||

\\*==========================================================\*/

template <class TYPE> class Stack

{

struct Node // each node holds an element of type TYPE

{

TYPE datum;

Node\* next;

};

Node\* top;

int \_size;

public:

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

| Name: Stack

|

| Summary: This is a simple constructor

|

| Precodition: none

| Postcondition: stack is created

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

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

Stack()

{

top = NULL;

\_size = 0;

}

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

| Name: ~Stack

|

| Summary: This is a simple deconstructor

|

| Precodition: there is a Stack

| Postcondition: that Stack memory is freed

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

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

~Stack()

{

reset();

}

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

| Name: push

|

| Summary: take a value and push/add it to the stack

|

| Precodition: there is a stack

| Postcondition: the value has been added to the stack

|

| Parameters:

|

| IN: type and value to add to the stack

| Out: stack size increases

|

| Returns: nothing

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

void push(TYPE& value)

{

Node\* Temp\_node = new Node;

if(Temp\_node == NULL)

{

//cout << "Push: memory allocation error" << endl;

exit(1);

}

Temp\_node->datum = value;

Temp\_node->next = top;

top = Temp\_node;

\_size++;

}

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

| Name: pop

|

| Summary: if there is an element pop it from the stack

|

| Precodition: there is a stack

| Postcondition: the element is popped from the stack

|

| Parameters:

|

| IN: nothing

| Out: popped element

|

| Returns: element was popped from the stack and returned

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

TYPE pop()

{

TYPE Return\_datum;

if(top != NULL)

{

Node\* Temp\_node = top;

Return\_datum = top->datum;

top = top->next;

delete Temp\_node;

\_size--;

}

return(Return\_datum);

}

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

| Name: size

|

| Summary: return the size of the stack

|

|

| Precodition: there is a stack

| Postcondition: the size of the stack is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: size of stack

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

int size()

{

return \_size;

}

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

| Name: reset

|

| Summary: pops all the elements of the stack to empty it

|

| Precodition: there is a stack

| Postcondition: the stack is now empty

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

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

void reset()

{

while(empty() != TRUE)

{

pop();

}

}

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

| Name: empty

|

| Summary: checks to see if any elements are in the stack

|

| Precodition: there is a stack

| Postcondition: if the stack is empty

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: true or false depending if anything is in the stack

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

int empty()

{

if(\_size == 0)

{

return(true);

}

else

{

return(false);

}

}

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

| Name: peek

|

| Summary: if there is an element return what the top elements

| value is

|

| Precodition: there is a stack

| Postcondition: the top elements value is returned

|

| Parameters:

|

| IN: nothing

| Out: top elements value

|

| Returns: element at the tops of the stacks value

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

TYPE peek()

{

TYPE Return\_datum;

if(top != NULL)

{

Return\_datum = top->datum;

}

return (Return\_datum);

}

// needs to be made to conform to rest of code

friend ostream& operator<<(ostream& os, Stack<TYPE>& q)

{

Stack::Node\* temp = q.top;

if(q.\_size == 0)

{

os << "(Stack)";

}

else

{

os << "(Stack ";

while(temp->next != NULL)

{

os << (temp->datum) << ' ';

temp = temp->next;

}

os << temp->datum;

os << ")";

}

return os;

}

};

### Class: Queue

/\*============================================================

||

|| Class: Queue<datum\_type>

||

|| Purpose: A handcrafted textbook Queue, easy data

|| structure to work with.

||

|| Attributes:

|| Node

|| Q datum

|| Node\* next

|| \_size

|| head

|| tail

||

|| Functions:

|| ~Queue

|| push

|| pop

|| front

|| size

|| empty

|| << (is a friend)

|| >> (is a friend)

|| (friend) MCB

||

\\*==========================================================\*/

template<class Q> class Queue

{

private:

// block of Queue

struct Node

{

Q datum;

Node\* next = NULL;

};

// Queue knows the number of Nodes it has

int \_size = 0;

// For speed of access Queue's always know of its head

// and tail

Node\* head = NULL;

Node\* tail = NULL;

public:

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

| Name: ~Queue

|

| Summary: This is a simple deconstructor

|

| Precodition: there is a Queue

| Postcondition: that Queue memory is freed

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: nothing

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

~Queue()

{

Node\* temp = head;

while(head != NULL)

{

head = head->next;

#ifdef SELF\_MANAGED

memory.free(temp);

#else

delete temp;

#endif

temp = head;

}

// for safety

tail = NULL;

head = NULL;

//while(empty() == false)

// pop();

}

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

| Name: push

|

| Purpose: Push values into the queue

|

| Precodition: queue was declared

| Postcondition: queue as another element in the back

|

| Parameters:

|

| IN: value of queues datatype

| Out: nothing

|

| Returns: nothing

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

void push(Q value)

{

Node\* temp;

if(tail != NULL)

{

#ifdef SELF\_MANAGED

temp = (Node\*) memory.alloc(sizeof(Node));

#else

temp = new Node;

#endif

temp->datum = value;

temp->next = NULL;

tail->next = temp;

tail = temp;

}

else

{

#ifdef SELF\_MANAGED

tail = (Node\*) memory.alloc(sizeof(Node));

#else

tail = new Node;

#endif

tail->datum = value;

tail->next = NULL;

head = tail;

}

\_size++;

}

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

| Name: pop

|

| Purpose: remove element at the front of the q, return value

|

| Precodition: queue exists, has element

| Postcondition: element removed from front

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: item at front of queue

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

Q pop()

{

Q pop\_value;

if(head == NULL)

{

//cout << "Error: push on empty Queue\n";

exit(1);

}

else if(\_size != 1)

{

Node\* temp = head;;

pop\_value = head->datum;

head = head->next;

delete(temp);

temp = NULL;

\_size--;

}

else

{

pop\_value = head->datum;

delete(head);

head = NULL;

tail = NULL;

\_size = 0;

}

return pop\_value;

}

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

| Name: front

|

| Purpose: see the element at the front without removal

|

| Precodition: queue exists, has element

| Postcondition: head datum is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: copy of element at the front

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

Q front()

{

return head->datum;

}

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

| Name: size (accessor)

|

| Purpose: get the size of the queue

|

| Precodition: queue exists

| Postcondition: size of queue is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: size of queue

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

int size()

{

return \_size;

}

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

| Name: empty

|

| Purpose: tells if a queue is empty, useful for going

| through all the elements in a queue via while loops

|

| Precodition: queue exists

| Postcondition: boolean of empty or not empty is returned

|

| Parameters:

|

| IN: nothing

| Out: nothing

|

| Returns: whether the queue is empty or not

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

bool empty()

{

return (\_size == 0) ? true : false;

}

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

| Name: <<

|

| Purpose: allow for use in any standard stream mechanism

|

| Precodition: nothing

| Postcondition: object is sent along a stream as a string

|

| Parameters:

|

| IN: stream mechanism, queue

| Out: stream mechanism, queue

|

| Returns: stream mechanism

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

friend ostream& operator<<(ostream& os, Queue<Q>& q)

{

Queue::Node\* temp = q.head;

if(q.\_size == 0)

{

os << "(Queue)";

}

else

{

os << "(Queue ";

while(temp->next != NULL)

{

os << (temp->datum) << ' ';

temp = temp->next;

}

os << temp->datum;

os << ")";

}

return os;

}

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

| Name: >>

|

| Purpose: convert appropriate strings to queues

|

| Precodition: there is a queue declared

| Postcondition: queue is built from strong representation

|

| Parameters:

|

| IN: stream mechanism, queue

| Out: stream mechanism, queue

|

| Returns: stream mechanism

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

friend istream& operator>> (istream& in, Queue<Q>& queue)

{

char ch;

string keyword = "Queue";

// capture the chracters and see if it matches

// the string

in >> ch;

if(ch != '(')

{

//cout << "mismatched parens\n";

exit(-1);

}

for(unsigned int i = 0 ; i < keyword.length() ; i++)

{

in >> ch;

if(ch != keyword[i])

{

//cout << "wrong object\n";

exit(-1);

}

}

in >> ch; // for space, or end

if(ch == ')')

{

return in;

}

// to be continued

//Q element;

stringstream ss;

//for (int i = 0 ; i < 6 ; i++)

// in >> char\_temp;

//Q type\_temp;

//cout << "Enter Real Part ";

//in >> c.real;

//cout << "Enter Imaginary Part ";

//in >> c.imag;

return in;

}

// MCB is used in testing software

friend class MCB;

};

## Makefile

# Jordan Winkler, Kyle Lucas, Joshua Carter

# Thu Mar 5 12:56:12 EST 2020

# makefile to construct things

# Note: if you don't have google test and valgrind installed then, testing and debugging makes may not work for you

# description of key arugments

# -c compile but don't link

# -Wall check for all warnings

# -Werror counts warnings as errors

# -g is to leave tags for gdb

# common arguments

CC = gcc

CCC = g++

DEBUG\_FLAGS = -g -Wall -Werror -fconcepts

LIB\_PATH =

LIB\_BIN = -lm -lpthread -lncurses

LIBS = $(LIB\_PATH) $(LIB\_BIN)

objects = main.o

default : run

# meaning build and run

run :

$(CCC) $(DEBUG\_FLAGS) main.cpp $(LIBS) -o main

./main

# catch test run

test :

$(CCC) $(DEBUG\_FLAGS) test.cpp $(LIBS) -o test\_bin

valgrind -q ./test\_bin

#rm test\_bin

val : main

valgrind -q ./main

rm -f main #\*.o # keep test runs clean

debug : main

valgrind --vgdb=yes --vgdb-error=0 ./main

#terminal -e gdb ./main -ex target remote | vgdb

main :

$(CCC) $(DEBUG\_FLAGS) main.cpp $(LIBS) -o main

clean :

rm -f main test\_bin #\*.o

# PATH\_TO\_VALGRIND = /usr/include/valgrind/memcheck.h

# PATH\_TO\_GTEST = /usr/include/gtest

# ifneq ("$(wildcard $(PATH\_TO\_VALGRIND))","") # and

# ifneq ("$(wildcard $(PATH\_TO\_GTEST))","")

# endif

#else # no installed libraries

#################### Main Compilation ######################

#main.o : main.cpp

# g++ $(DEBUG\_FLAGS) -c main.cpp -o main.o

#queue.o : queue.cpp

# g++ $(DEBUG\_FLAGS) -c queue.cpp -o queue.o

# 

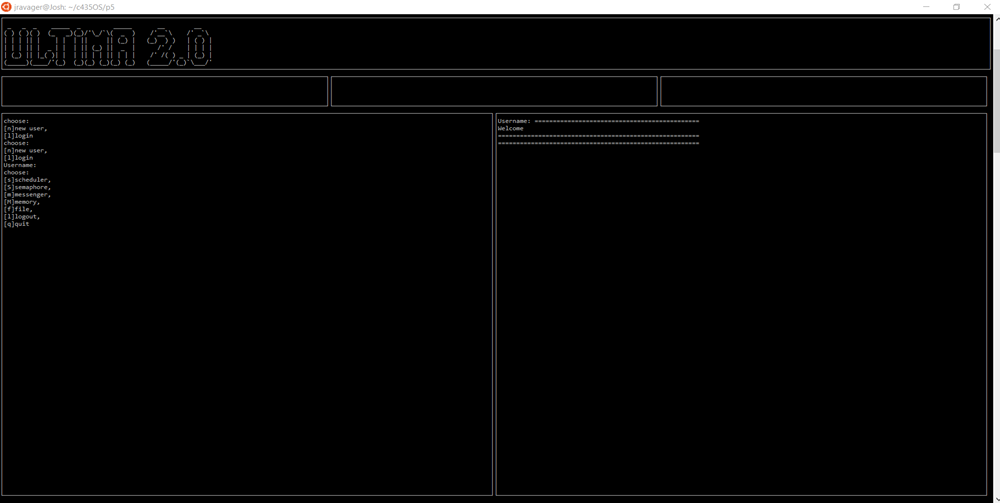
# **Output**

## **Screen Dumps**

Screenshots from the ULTIMA 2.0 operating system will illustrate the use of the system. The system is portable across a few different architectures and was built using the Valgrind debugging tool. Memory leaks are trapped and easier to debug using this suite.

### $ make

#### User creation & logon



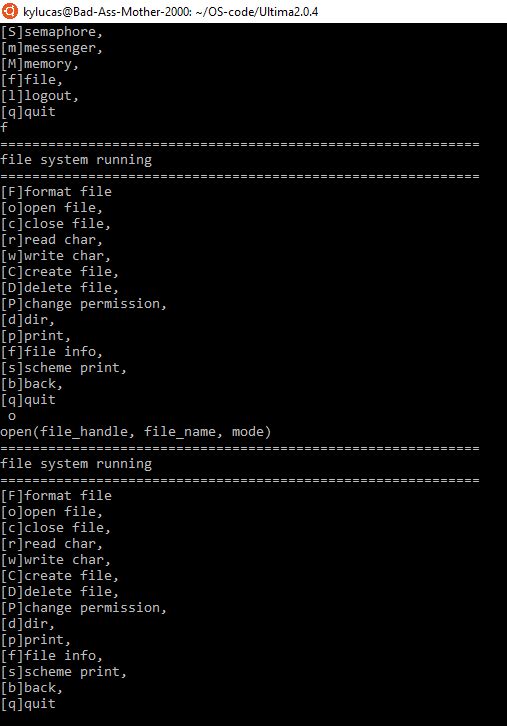
#### 

#### Testing the file system

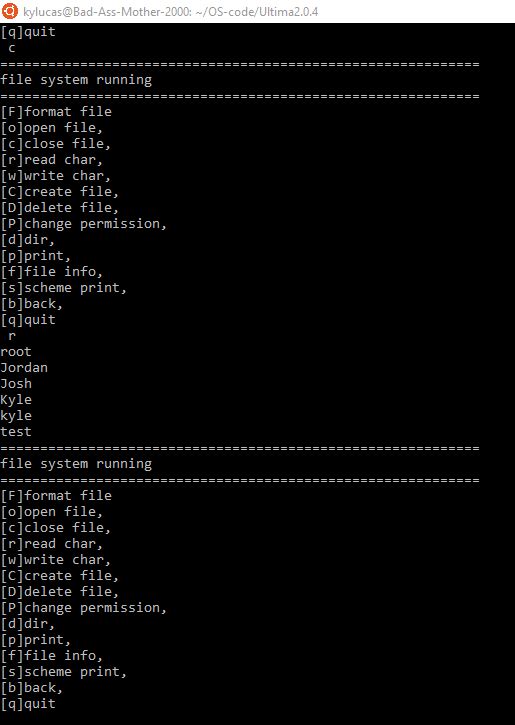
Format, creates a file called inode.txt:



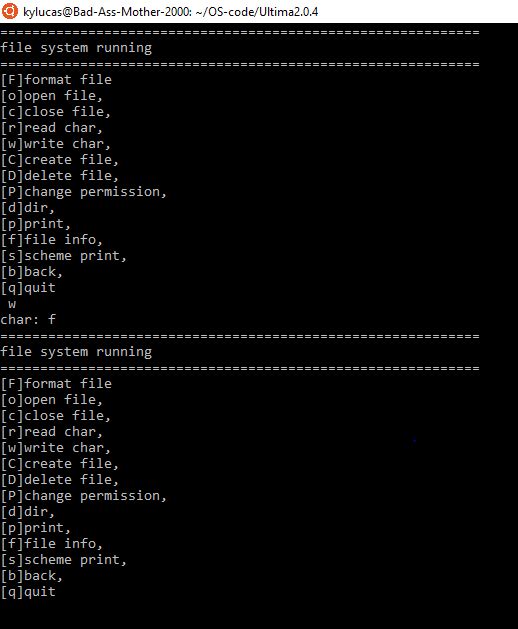
Open file:



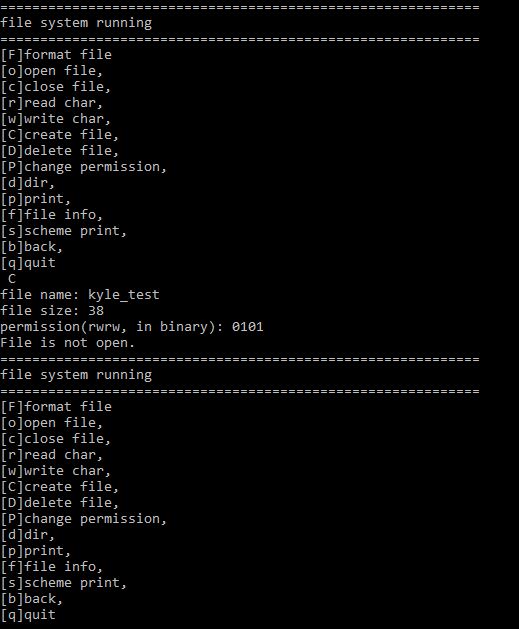
Read char:



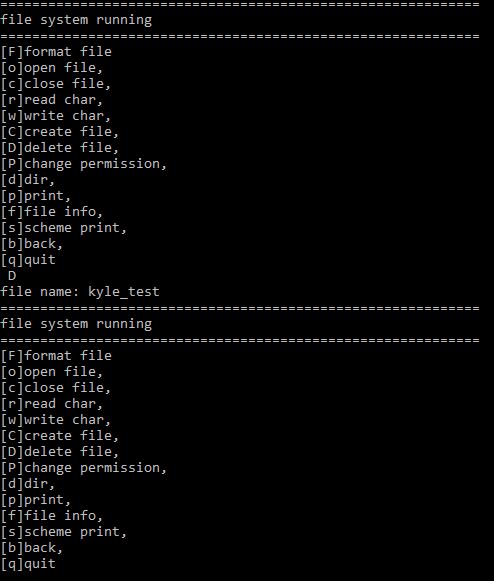
Write char:



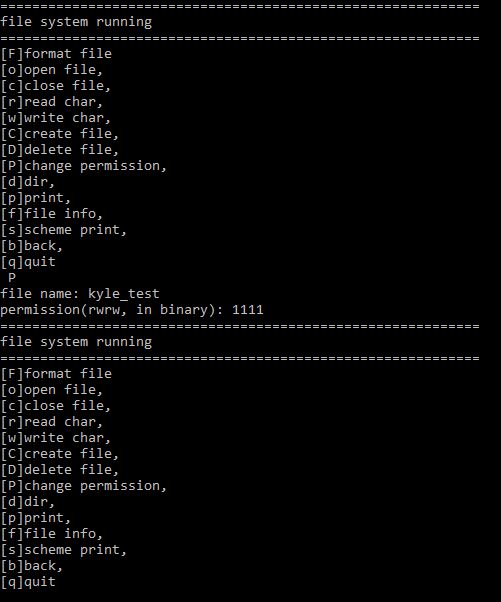
Create file:



Delete file:



Change permission:

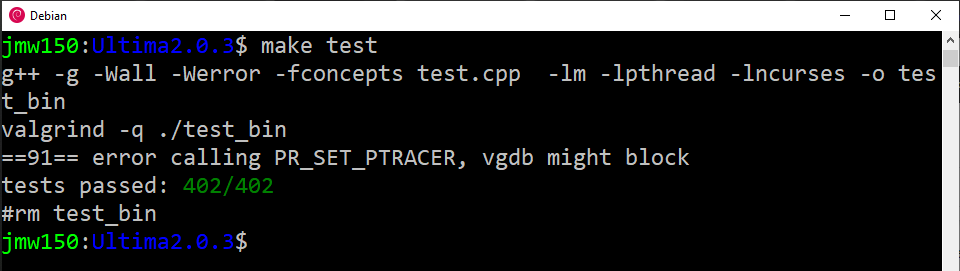


### 

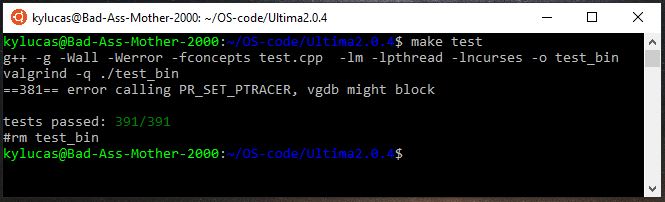
### 

### $ make test

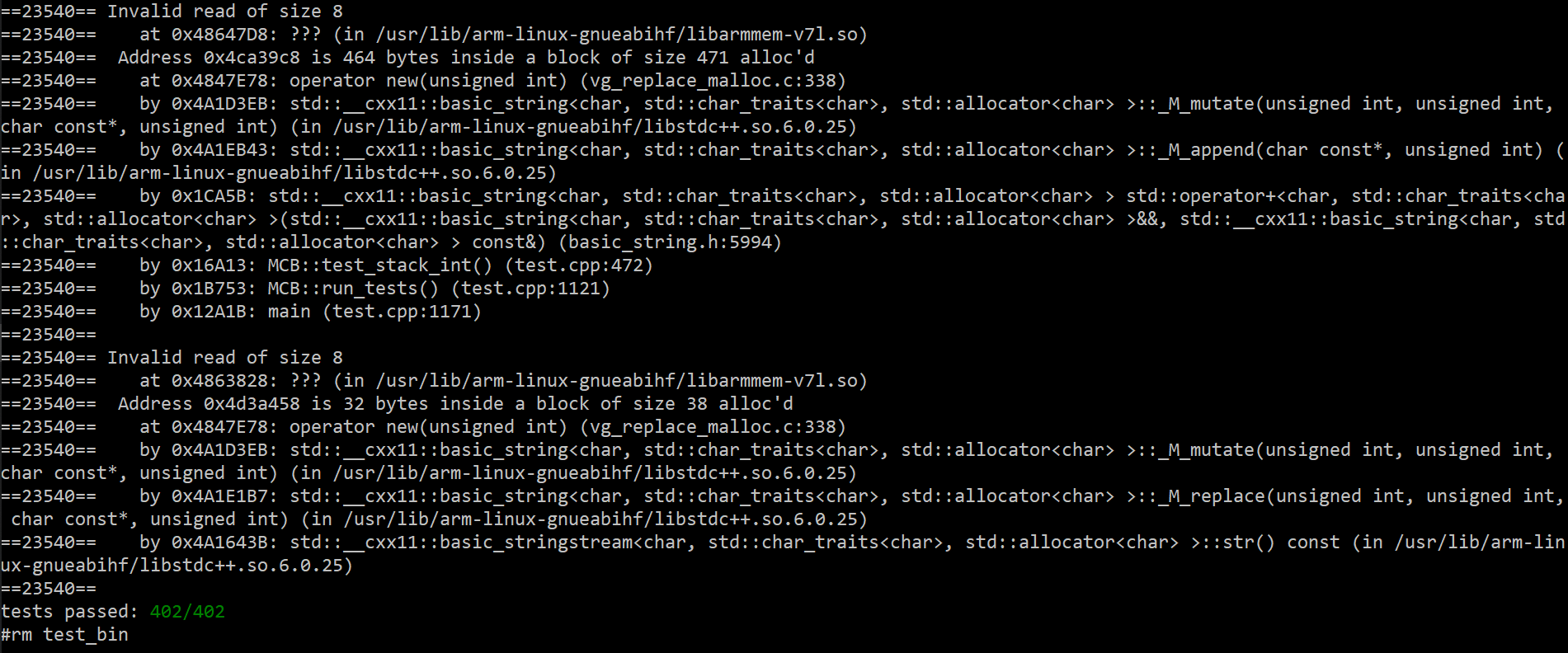
On Debian WSL



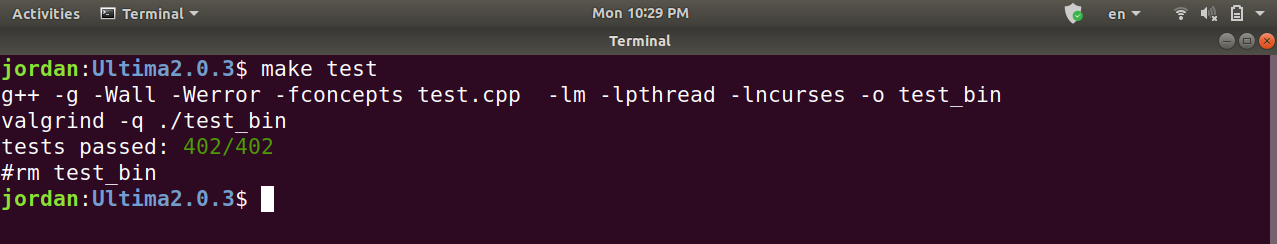
On Ubuntu WSL

****

On Raspberry pi 4 Raspian with the new ARMv8 chip (new stack leaks memory)



On Ubuntu on x86-64 chip



# 

# 

# 

# Needed Fixes/Future Features:

## Test File

* Develop an organizing scheme for integration tests
* Make combinatorial outline of test coverage

## Scheduler Class

* Would like to implement a priority queue template for the task blocks so they are orderable by priority.

## Messenger Class

* Would like to implement a binary tree for the message storage system.

## Memory Class

* Would like to implement pointer arithmetic functions such as ‘->’ and ‘.’.

## File System Class

* The file system needs more testing to ensure the robustness of the class.

# 

# 

# 

# **References:**

Grenning, James W. Test Driven Development for Embedded C. 1 edition, Pragmatic Bookshelf, 2011.

Hakimzadeh, Hossein. *PHASE 3 – Memory Management*. PDF, Indiana University South Bend, Spring 2020, p. 1-2.

Tanenbaum, Andrew, and Herbert Bos. *Modern Operating Systems*. 4th ed., Pearson, 2015.

Nicolai M. Josuttis. *The C++ Standard Library*. 2nd ed., Addison Wesley, 2012.

Stanley B. Lippman, et al. *C++ Primer*. 5th ed., 2013.

“http://www.Cplusplus.Com/.” *Cplusplus.Com*, 6 Apr. 2020, http://www.cplusplus.com/.

1. From Hakimzadeh, Hossein. *PHASE 5 - Final Submission & Presentation/Demo* [↑](#footnote-ref-0)