**E1 Operating System**

Technical Manual

11/30/2014 \*Date of last edit

**Concord University, CS 361**

Ioan George Istrate

Table of contents

Technologies used, and useful links 3

DGML Diagram 4

Classes/Methods information 5

Core

Memory

PCB

PCBqueue

Scheduler

StringParser aka Python Whisperer

TextureManager

UserInput

Image

GameObject

Gui

Overview 11

Dependency Graph 12

**Technologies used**

The E1 operating system has been developed using the C++, and Python programming languages. C++ was used for its OOP capabilities, and speed while Python was used as a scripting language for simplifying the user -> machine interface.

Libraries used for development are:

* SDL2 for the user interface, and controls
* SDL2 image for use of jpegs and pngs
* SDL2 TTF for true type fonts
* Python 3.4 for Python

C++ fires up the program, SDL2 controls the image display and the user input, while Python – due to its strong string processing - is used mostly for processing the written commands. C++ and Python are glued together with the use of the Python C Api.

Useful links:

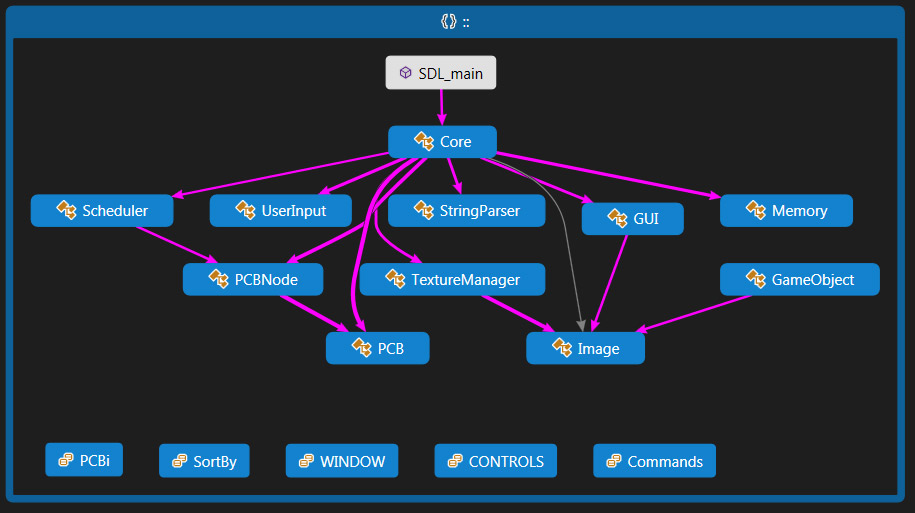
* <https://www.libsdl.org/download-2.0.php>
* <https://www.libsdl.org/projects/SDL_image/>
* <https://www.libsdl.org/projects/SDL_ttf/>
* <https://docs.python.org/3.4/c-api/>

Tutorial:

<http://lazyfoo.net/tutorials/SDL/index.php>

**DGML Diagram**

1) Class Interaction Overview



2) PCBi, SortBy, Window, Controls, Commands are all contained inside our Globals.h, and are used to store integer constants. Free floating constant integers are protected inside the Global namespace.

**Core**

Core contains our sentinel loop, and is where most action takes place.

Composition:

TextureManager Tmanager;

UserInput Ui;

GUI\* Panel;

StringParser\* Parser;

Memory\* m\_Memory;

//process queues

PCBQueue\* m\_Ready;

PCBQueue\* m\_Blocked;

PCBQueue\* m\_Completed;

//image sets vector < Image\* > m\_Images;

vector < Image\* > m\_Images\_CMD;

vector < Image\* > m\_Images\_TM;

Important methods:

Init() – Initialises SDL, and sentinel loop run condition

Run() – Initialises Memory Block, Ready/Blocked/Completed Queues, GUI panels

fpsCap() – Caps FPS for SDL

getTime() – gets current time

listDir() – gets directory explorer by communicating with Python

addSystem() – forces system commands by bypassing user commands

runPrograms() – runs our PCBs

**Image**

Composition: none

Important methods:

Load() – loads the properties (X, Y, Sprite X, Sprite Y, Width, Height) for the image as well as the filename

setFlipped() – set if image is flipped, needed for the Texture Manager

setCurrentFrame() – sets the current frame, needed for animation

getLayer() – zindex for Texture Manager, decides which images gets drawn on top

Setters and getters for the X, Y, width and height of an image.

**Gui**

Contains information used to draw our panels. Can be extended to draw a GUI panel on click location, everything needed is in, but not made available for current version.

Composition:

vector <Image \*> m\_Images;

Important methods:

getImages() – communicates the Image sets for each GUI to the Core, which passes it on to the Texture Manager.

setVisible() – display panel or not

**PCB**

Contains info for a process.

Composition: none

Important Methods:

Setters and Getters for memory usage, cpu usage, priority, class, time of arrival, execution time, name, state.

**PCBNode**

Contains PCBs and are the building blocks for our PCBQueues.

Composition:

PCBNode\* m\_next;

PCBNode\* m\_prev;

PCB\* m\_Pcb;

Important Methods:

getPCB() – returns the PCB assigned to the node.

getNext() – get next node in the PCBQueue linked list.

getPrev() – get previous node in the PCBQueue linked list.

setPCB(), setNext(), setPrev() – self explanatory

**PCBQueue**

Is a doubly linked list of PCBNodes. Used for our Ready/Completed/Blocked queues.

Composition:

PCBNode\* m\_head;

PCBNode\* m\_tail;

Important methods:

allocatePCB() - Creates memory for a new PCB.

freePCB() - Frees all memory associated with a PCB

setupPCB() - Initializes a PCB’s content (name, priority, class).

findPCB() – Finds a PCB by name.

insetPCBatEnd() – inserts a PCB at the end of the Queue.

getPCBatIndex() – gets a PCB by index.

printPCBs() and printPCBsReverse() – used for testing linked list

**Scheduler**

Loads processes from a file indicated by the user, then sorts them using shell sort (by time to completion or time to arrival) and makes them available to the Core by means of the Blocked and Ready queues.

Composition:

PCBQueue& m\_Ready;

PCBQueue& m\_Blocked;

Important methods:

parseFile() – loads info from a file and returns a vector of PCBs.

addPCBS() – sorts and loads PCBS to their appropriate Queues.

**StringParser – aka the Python Whisperer**

Uses the Python C Api to connect C++ to Python.

Composition: Python.py python module

Important methods:

Init() – opens up the Python module, initializes Python and locks threads.

Close() – releases threads

parseString() – sends to Python a string and gets back 2 sets of integer tuples and a sanitized string. The string is displayed through SDL, 1 tuple holds the integer commands, 1 tuple holds error codes.

**TextureManager**

Used to draw sets of Images, and text.

Composition: none

Important methods:

drawImage() – draws one single Image on the screen.

draw() – draws a set of Images on the screen.

drawText() – draws text on screen. Used for all our written messages. Inside here we find the newline, and character enter code.

sortByLayer() – sorts images by layer, and decides which image gets drawn on top zindex style

initFont() – loads and inits a font

**UserInput**

UserInput listens for user events. Captures clicks (left, right and position of click), listens for specially linked buttons like ctrl left, ctrl right or escape, everything else is counted as a written word.

Composition: none

Important methods:

getCommand() – polls for click, button events. Fills is a string and returns an integer command.

getStringCommand() – returns a full string

getMouseX() and getMouseY() return the X and Y position of the mouse cursor.

**Memory:**

Allocates and frees memory for the PCBs. Is a singly linked list of memory Nodes (referred as memory fragments).

Composition: Private class with methods Node

Important methods:

getMaxSize() – returns the maximum memory

getFreeMemory() – returns the free memory

getFragmentCount() – counts unused memory fragments

allocate() – allocates memory for a process

free() – frees memory used by a process

Unused classes:

**GameObject**

GameObject contains the interface for items such as clock, or other apps available on the future releases of the E1 Op Sys.

Composition:

Image\* m\_Image;

Methods:

Comprised of mostly getters and setters for the X, Y coordinates; Width, Height; and Image.

**Overview**

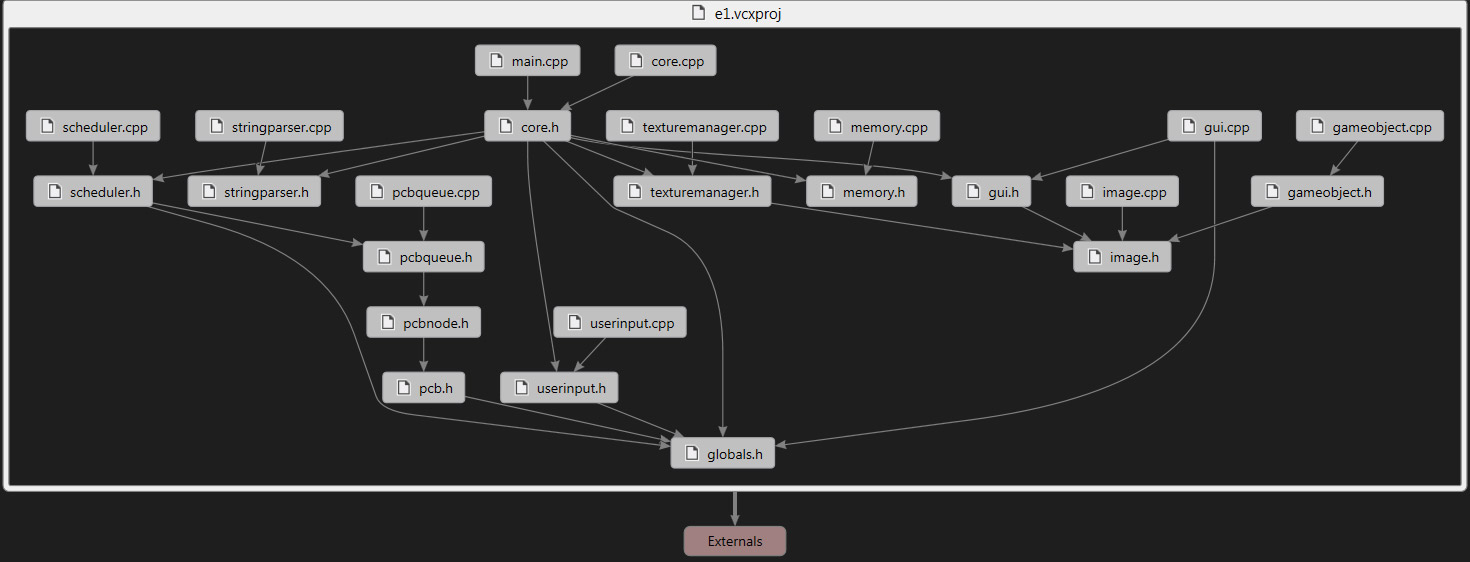
Once Core has been initialized and set to running it will start listening to user events. If the user events system catches mouse clicks that correspond with certain linked areas of the screen/or a user writes a certain valid command the Core will issue a system command.

The command pipeline consists mostly of integer commands sent from C++ to Python to SDL and back to C++. The Python module only sanitizes and looks for valid commands and returns integers. In case of an invalid command the command won’t run and the user won’t see a change in the program. However Python DOES send a set of **Error Codes** which currently are not implemented but are very easy to do.   
**Test cases**

Throughout the Code you will see test cases where blocks of code are commented out and there’s a short description for each test.

Whenever you want you can enter debug mode and you will have a series of tests that are outlined in the User Manual.

**Dependency Graph**

****

The end