GUI_support.py

```
#! /usr/bin/env python
#  -*- coding: utf-8 -*-
#
# Support module generated by PAGE version 4.12
# In conjunction with Tcl version 8.6
#    Mar 27, 2018 06:51:49 PM

#Author: Mike Ford
#Date: 4/5/18

import sys

try:
    from Tkinter import *
    import  tkFileDialog
    import math
    from collections import namedtuple
except ImportError:
    from Tkinter import *

try:
    import ttk
    py3 = False
except ImportError:
    import tkinter.ttk as ttk
    py3 = True

'''
Class: pageTable
Inputs: pages - how many pages will be in the pagetable

Summary: This object represents a page table with "pages" number of pages.
This creates a (pages x 2) 2d list. The page column values range from 0-pages.
The corresponding frame value b=can either be "get" or "set"
'''
class pageTable():
    def __init__(self,pages):
        self.pt = [[0 for x in range(2)] for y in range(pages)]

    def setPageTableItem(self,page,frameNum):
        self.pt[page][1] = frameNum

    def getPageTableFrameNum(self, page):
        return self.pt[page][1]
'''
Class: process
Inputs: pid - The processes ID number
codelength - length of code section in bytes
dataLength - length of data section in bytes
codePages - Number of pages required to hold code bytes (pages are 512 bytes)
dataPages - Number of pagers required to hold data bytes
codePageTable - a pageTable object to act as the processes' code page table
dataPageTable - a pageTable oject to act as the processes' data page table

Summary: This object holds all of the data for a loaded in process.
'''
class process():
    def __init__(self, pid, codeLength, dataLength, codePages, dataPages,codePageTable, dataPageTables):
        self.pid = pid
        self.codeLength = codeLength
        self.dataLength = dataLength
        self.codePages = codePages
        self.dataPages = dataPages
        self.codePageTable = codePageTable
        self.dataPageTable = dataPageTables
    def getCodeLength(self):
        return self.codeLength
    def getDataLength(self):
        return self.dataLength
    def getPID(self):
        return self.pid
    def getCodePages(self):
        return self.codePages
    def getDataPages(self):
        return self.dataPages
    def getCodePageTableFrame(self,page):
        return self.codePageTable.getPageTableFrameNum(page)
    def setCodePageTableFrame(self,page, frame):
        self.codePageTable.setPageTableItem(page,frame)
    def getDataPageTableFrame(self,page):
        return self.dataPageTable.getPageTableFrameNum(page)
    def setDataPageTableFrame(self,page,frame):
        self.dataPageTable.setPageTableItem(page, frame)
    def toString(self):
        return "Loading program %d into RAM: code=%d (%d pages), data=%d (%d pages)\n" % (self.pid,self.codeLength,self.codePages,self.dataLength,sel

'''
Class: frame
Inputs: frameNumber - how many pages will be in the pagetable

Summary: This object represents a page table with "pages" number of pages.
This creates a (pages x 2) 2d list. The page column values range from 0-pages.
The corresponding frame value can either be "get" or "set"
'''
```

```
95  class frame():
96      def __init__(self, frameNumber):
97          self.frameNumber = frameNumber
98          self.occupied = 0
99          self.presentProcess = 999
100     def getFrameNumber(self):
101         return self.frameNumber
102     def getOccupied(self):
103         return self.occupied
104     def getPresentProcess(self):
105         return self.presentProcess
106     def setOccupied(self, isOccupied):
107         self.occupied = isOccupied
108     def setPresentProcess(self,pid):
109         self.presentProcess = pid
110
111 #sets up globals for program
112 def set_Tk_var():
113
114     global FRAME_SIZE
115
116
117     #all string variables below are used to set GUI message box texts
118     global fileContent
119     fileContent = StringVar()
120     global pagesLoaded
121     pagesLoaded = StringVar()
122     global ramSize
123     ramSize = StringVar()
124     ramSize.set("512")
125     global pagesString
126     pagesString = ""
127     global messageBoxStrings
128     messageBoxStrings = []
129     for num in range(0,8):
130         messageBoxStrings.append(StringVar())
131         messageBoxStrings[num].set("Free")
132     #current line from file to read
133     global line
134     line = -1
135     #holds file contents
136     global operationList
137     operationList = []
138     #list for loaded processes and free frames
139     global processList
140     processList = []
141     global freeFrames
142     freeFrames = [frame(0),frame(1),frame(2),frame(3),frame(4),frame(5),frame(6),frame(7)]
143
144 #sorts free frames list by frame number (lowest to highest)
145 def sortFreeFrames(frames):
146     frames.sort(key = lambda frame: frame.frameNumber)
147
148 #if the file menu option is selected, pop up a file selection window
149 def fileSelected():
150     global FRAME_SIZE
151     global ramSize
152     #set frame size to whatever user entered
153     FRAME_SIZE = float(ramSize.get())
154     fileLines = ""
155     filename = tkFileDialog.askopenfilename(initialdir="", title="Select file", filetypes=(
156     ("text files", "*.txt"), ("all files", "*.*")))
157     #open file selected
158     inputFile = open(filename, "r")
159     #read in lines
160     for line in inputFile:
161         operationList.append(line.rstrip())
162     for element in operationList:
163         fileLines+=(element + '\n')
164     #set message window to file content
165     fileContent.set(fileLines)
166
167 #processes each file line
168 def handleLine(line):
169     global pagesString
170     #split the line into the individual numbers
171     numbersInLine = line.split(" ")
172     numbersInLine = map(int, numbersInLine)
173     #if line is 2 numbers, remove a process
174     #if line is 3 numbers, load process
175     if(len(numbersInLine) is 2):
176        removeProcess(numbersInLine[0])
177     if(len(numbersInLine) is 3):
178         #create new process, add it to process list
179         currentProcess = process(numbersInLine[0],numbersInLine[1],numbersInLine[2],(int)(math.ceil(numbersInLine[1]/FRAME_SIZE)),(int)(math.ceil(numl
180         processList.append(currentProcess)
181         pagesString += currentProcess.toString()
182         #print out process to message window and load process
183         pagesLoaded.set(pagesString)
184         addProcess(currentProcess)
185
186 def addProcess(process):
187     global pagesString
188     global freeFrames
189     global line
190     page = 0
191     codeString = ""
192     dataString = ""
```

```
193
194     #check if theres enough available frames to add process
195     if (process.getCodePages() + process.getDataPages()) > len(freeFrames):
196         pagesString += "NOT ENOUGH FRAMES TO ALLOCATE PAGES.\n"
197         pagesLoaded.set(pagesString)
198         line -=1
199     else:
200         #while theres still code pages to add
201         while (process.getCodePages() > page):
202             #add page to first available frame and updae process page table
203             process.setCodePageTableFrame(page, freeFrames[0].getFrameNumber())
204             #get messages to print out ready, and then set message frames
205             codeString = "Code-%d of P%d" % (page, process.getPID())
206             pagesString += "Load code page %d of process %d to frame %d\n" % (page,process.getPID(),freeFrames[0].getFrameNumber())
207             pagesLoaded.set(pagesString)
208             messageBoxStrings[freeFrames[0].getFrameNumber()].set(codeString)
209             #delete fram just used from free frames, resort free frames list
210             del freeFrames[0]
211             sortFreeFrames(freeFrames)
212             page += 1
213
214         page = 0
215         #same process as code pages, except applied to data pages
216         while (process.getDataPages() > page):
217
218             freeFrames[0].setOccupied(1)
219             process.setDataPageTableFrame(page, freeFrames[0].getFrameNumber())
220             dataString = "Data-%d of P%d" % (page, process.getPID())
221             pagesString += "Load data page %d of process %d to frame %d\n" % (
222             page, process.getPID(), freeFrames[0].getFrameNumber())
223             pagesLoaded.set(pagesString)
224             messageBoxStrings[freeFrames[0].getFrameNumber()].set(dataString)
225             del freeFrames[0]
226             sortFreeFrames(freeFrames)
227             page += 1
228
229         page = 0
230
231 #remove process passed in
232 def removeProcess(processID):
233     #return process object that corresponds to ID passed in
234     processToRemove = findProcess(processID)
235     global pagesString
236     global freeFrames
237     #loop through however many code pages to remove
238     for x in range(processToRemove.getCodePages()):
239         #add frame released back to free frames list, and resort
240         freeFrames.append(frame(processToRemove.getCodePageTableFrame(x)))
241         sortFreeFrames(freeFrames)
242         #set message box of that frame to free
243         messageBoxStrings[processToRemove.getCodePageTableFrame(x)].set("Free")
244     #same process as removing code pages, but to data pages
245     for x in range(processToRemove.getDataPages()):
246         freeFrames.append(frame(processToRemove.getDataPageTableFrame(x)))
247         sortFreeFrames(freeFrames)
248         messageBoxStrings[processToRemove.getDataPageTableFrame(x)].set("Free")
249     #setup and print end program message
250     pagesString += "End of Program %d\n" % (processID)
251     pagesLoaded.set(pagesString)
252
253 #return process object that matches ID passed in
254 def findProcess(processID):
255     for process in processList:
256         if processID is process.getPID():
257             return process
258
259 #callback for next button
260 def nextButtonClicked():
261     global line
262     global pagesString
263     line += 1
264     #if line is more than lines in file, print out no more processes
265     if line > len(operationList)-1:
266         pagesString += "No more processes.\n"
267         pagesLoaded.set(pagesString)
268     else:
269         handleLine(operationList[line])
270 #init GUI
271 def init(top, gui, *args, **kwargs):
272     global w, top_level, root
273     w = gui
274     top_level = top
275     root = top
276 #closes window
277 def destroy_window():
278     # Function which closes the window.
279     global top_level
280     top_level.destroy()
281     top_level = None
282 #start GUI
283 if __name__ == '__main__':
284     import GUI
285     GUI.vp_start_gui()
286
287
288
```