***עבודת סוף במערכות הפעלה***

קורס: מערכות הפעלה חורף **–** 2019

עבודה מס' 2

תאריך ההגשה בפועל: 16/2/19

שם הסטודנט, תעודת זהות, מייל: לוסקי, אליאב, 316219526, Louski.a@gmail.com

שם הסטודנט, תעודת זהות, מייל: אלקבס, דוד,206007494 , David.elkabass@gmail.com

מגישים:

אליאב לוסקי (ת"ז 316219526)

דוד אלקבס (ת"ז 206007494)

תאריך הגשה:

18.2.19

הסבר קצר:

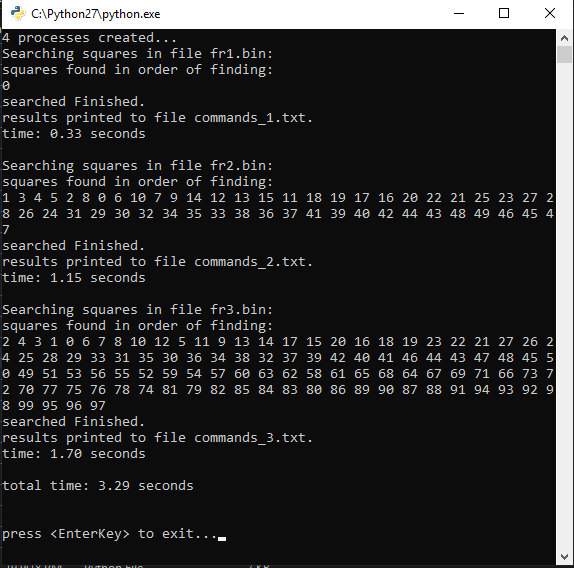
**הקוד מתועד ומוסבר בצורה מלאה.**

**הקוד מחפש את שלושת הקבצים ומריץ אותם אחד אחרי השני, במידה וקובץ לא נמצא הוא מדולג ומודפסת הודעת שגיאה.**

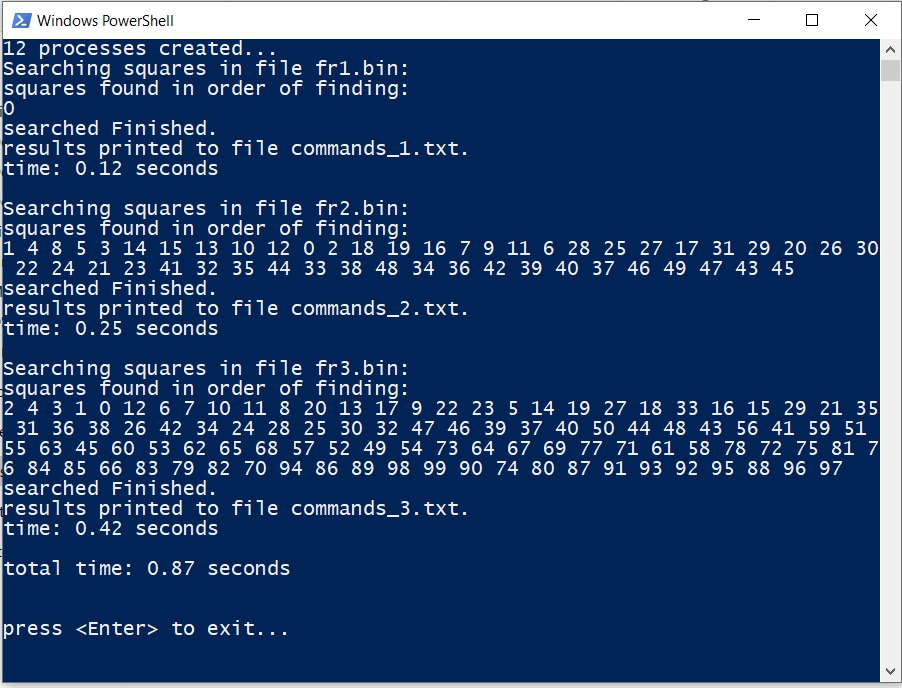
**זמן ביצוע עבור מחשב עם ליבה בודדת יכול להגיע ל-6-8 שניות (לשלושת הקבצים).   
מחשב מרובה ליבות יכול להגיע לזמן ביצוע של פחות משניה אחת (לשלושת הקבצים).**

**הקוד עבר שינויים רבים וגרסאות רבות ובהתחלה זמן הריצה(לכל הקבצים) היה כ-80 שניות, לאחר שיפורים שונים ירד ל-50, לאחר שינוי אלגוריתם חיפוש ירד ל-15, ולאחר שינויים והתאמות נוספות(אי שימוש בפונקציות בתוך לולאה, שינוי הטיפוס של התור מרשימה של רשימות לרשימה בלבד(מסתבר שמאוד משמעותי), סנכרון טוב יותר ושימוש במערכים בזיכרון משותף כך שכל תהליך יוכל לגשת לתא המתאים לו ללא צורך בנעילה) ירד ל-**4-5 **שניות למחשב ממוצע עם 2 ליבות(4 ליבות לוגיות).  
מספר הליבות נמצא אוטומטית(במידה וכשל, מבקש מהמשתמש להקליד כמה ליבות יש לו).**

תמונת הריצה במחשב שלי(i5-4200m):



**תמונת הריצה עם i7-8700k(0.42 שניות לקובץ של המאה!!):**



**קוד התוכנית:**

1. **from** os **import** path
2. **import** time
3. **import** multiprocessing as mul

6. **def** int2bytes(i):
7. **return** hex2bytes(hex(i))

10. **def** hex2bytes(h):
11. **if** len(h) > 1 **and** h[0:2] == '0x':
12. h = h[2:]
13. **if** len(h) % 2:
14. h = "0" + h
15. **return** h.decode('hex')

18. row\_res = 768
19. col\_res = 1024
20. frame\_size = row\_res\*col\_res
22. **try**:
23. cpu\_num = mul.cpu\_count()
24. **except** NotImplementedError:
25. cpu\_num = raw\_input('Enter how may CPU\'s this computer have: ')
27. square\_height = 5
28. square\_width = 5
30. upper\_lim = int2bytes(110)
31. lower\_lim = int2bytes(100)
33. max\_frames = 1000  # maximum frames that can be processed(in the same file)(can't resize shared memory arrays)

36. **def** print\_progress(print\_q):
37. """
38. responsible for printing the frame number that just been found(just to show progress).
39. """
40. **while** True:
41. num = print\_q.get()
42. **if** num == -1:
43. **break**
44. **print** num,

47. **def** find\_square(frame):
48. """
49. very efficient algorithm to fo find and return a location of a square in a frame.
50. jump every time square\_height-1(in our case 4) rows.
51. a single process can find 100 squares in 100 frames in 3-3.5 seconds (on single average CPU core)
53. execute time improved by:
54. not calling to any help functions(turns out that using functions inside the loop costing with a lot of time)
55. using for's and not while's(turns out for's faster)
56. numpy arrays did not improved results
57. decoding int to bytes (upper\_lim and lower\_lim) instead decoding byte to int every check(significant improvement!)
58. this is the best results we could get.

61. detailed explanation:
62. in case of square 5X5:
63. starting checking in row 3(count start from 0),then 7,11,15...(checking if value in wanted range in all columns),
64. lets define each row that checked 'checked row'.
65. if found value in range check if this column has enough relevant sequentially values to be a left side of a square
66. (from the rows from up to down),
67. if there is a few possible squares check them all and if found full square return results(and stop checking)
69. example:
70. lets say our square start from row 3 to row 7(in some arbitrary column).
71. lets say not found relevant value in row 3,then skipped to row 7 and found relevant value, then checks rows
72. 3-6,8-10 in the same column(notice: each checked row checking checked row above it but not the checked row below it)
73. rows 3-9 had values in range then the lower and upper side of the squares can possibly be (3,7) or (4,8) or (5,9)
74. we start from checking the upper one (3,7) and we find that it has a full upper side, lower side, and right side
75. and therefore its owr square and we quiting from the loop and returning results.
77. """
78. **for** i **in** xrange(square\_height-1, row\_res-(square\_height-1)+1, square\_height-1):
79. **for** j **in** xrange(col\_res-(square\_height-1)):
80. is\_square = True
81. row\_offset = 0
82. up\_offset = 0
83. down\_offset = 0
84. **if** lower\_lim <= frame[i\*col\_res+j] <= upper\_lim:
85. # check if right border can possibly exist
86. **if** **not** lower\_lim <= frame[i\*col\_res+j+square\_width-1] <= upper\_lim:
87. **continue**
89. # check if left border fully exist
90. **for** uprow **in** xrange(1, square\_height):
91. **if** lower\_lim <= frame[(i-uprow)\*col\_res+j] <= upper\_lim:
92. row\_offset += 1
93. up\_offset += 1
94. **else**:
95. **break**
96. **if** **not** is\_square:
97. **continue**
98. **for** downrow **in** xrange(1, square\_height-1):
99. **if** lower\_lim <= frame[(i+downrow)\*col\_res+j] <= upper\_lim:
100. row\_offset += 1
101. down\_offset += 1
102. **else**:
103. **break**
104. **if** row\_offset < square\_height-1:
105. is\_square = False
106. **if** **not** is\_square:
107. **continue**
109. # there are diff possible squares in this column
110. diff = row\_offset-(square\_height-1)
112. # if there more then one option to a square in this col, start from checking from up to down
113. **for** possible\_square **in** xrange(diff+1):
114. # check if up border fully exist
115. is\_square = True
116. **for** upcols **in** xrange(square\_width-1):
117. **if** **not** lower\_lim <= frame[(i-up\_offset+possible\_square)\*col\_res+j+upcols] <= upper\_lim:
118. is\_square = False
119. **break**
120. **if** **not** is\_square:
121. **continue**
122. # check if down border fully exist
123. **for** downcols **in** xrange(square\_width-1):
124. **if** **not** lower\_lim<=frame[(i+down\_offset-diff+possible\_square)\*col\_res+j+downcols]<=upper\_lim:
125. is\_square = False
126. **break**
127. **if** **not** is\_square:
128. **continue**
130. # check if right border fully exist
131. **for** row **in** xrange(square\_height-1):
132. **if** **not** lower\_lim<=frame[(i-up\_offset+row+possible\_square)\*col\_res+j+square\_width-1]<=upper\_lim:
133. is\_square = False
134. **break**
136. # if you got to this point this is a square
137. row\_result = i-up\_offset+int(square\_height/2) + possible\_square
138. col\_result = j+int(square\_width/2)
139. **return** row\_result, col\_result

142. **def** search\_in\_frame(frames\_q, row\_results, col\_results, frame\_cnt, lock, print\_q):
143. """
144. search\_in\_frame is a process responsible for finding squares and putting the result
145. in the right index in the shared memory arrays row\_results and col\_results,
146. frame\_cnt updated each time process 'toke' a frame to process.
148. example:
149. if process p1 processed frame 4 and by the time took p1 to find the square in it,
150. frames 5,6 was given to p2,p3, then the next frame p1 will get is frame 7.
151. each process 'taking' a frame updating frame\_cnt so the next process will know what frame his
152. working on.
153. """
154. **while** True:
155. frame = frames\_q.get()
156. frame\_num = frame\_cnt.value
157. with lock:
158. frame\_cnt.value += 1
159. result = find\_square(frame)
160. row\_results[frame\_num], col\_results[frame\_num] = result
161. print\_q.put(frame\_num)  # on screen will be printed in which frame square was found to show progress
162. frames\_q.task\_done()

165. **def** manage\_procs(fname, frames\_q):
166. """
167. process responsible for filling the queue frame\_q with frames.
168. frames\_q has a limited size to contain frames as the number of CPU's only.
169. """
170. f = open(fname, 'rb')
171. **while** True:
172. frame = f.read(frame\_size)
173. **if** frame == '':
174. **break**
175. frames\_q.put(frame)
176. f.close()

179. **def** main():
180. """
181. the main is responsible for creating all processes:
182. creating one manage\_procs process,
183. creating search\_in\_frame processes as the number of CPU's,
184. the main also responsible for the user interface and showing progress of the search in the all the files,
185. ,showing time taken to each file and total time, and printing a error messages when needed(can't open etc).
186. when all processes finished, printing the results into files.
187. """
188. t = time.time()
189. row\_results = mul.Array('i', max\_frames)            # shared memory array representing the positions of rows
190. col\_results = mul.Array('i', max\_frames)            # shared memory array representing the positions of columns
191. frame\_cnt = mul.Value('i', 0)                       # shared memory value holding next number of next frame
192. lock = mul.Lock()                                   # Lock to give only one process permission to change frame\_cnt
193. frames\_q = mul.JoinableQueue(frame\_size\*cpu\_num)    # Q for holding and sharing data of next frames
194. print\_q = mul.JoinableQueue()                       # Q for printing progress in right order
196. # start search\_square processes as the number of CPU's
197. p\_list = list()
198. **for** j **in** xrange(cpu\_num):
199. p = mul.Process(target=search\_in\_frame, args=(frames\_q, row\_results, col\_results, frame\_cnt, lock, print\_q))
200. p.start()
201. p\_list.append(p)
202. **print** '%d processes created...'%cpu\_num
204. # start searching squares in all files
205. **for** i **in** xrange(1, 4):
206. fname = 'fr'+str(i)+'.bin'
207. **if** **not** path.exists(fname):
208. **print** 'Error! Cant open', fname
209. **continue**
210. **else**:
211. start\_time = time.time()
212. **print** "Searching squares in file %s:"%fname
213. **print** "squares found in order of finding:"
215. # create and start the process manager
216. procs\_manager = mul.Process(target=manage\_procs, args=(fname, frames\_q))
217. procs\_manager.start()
219. # start the process that responsible for printing progress
220. print\_proc = mul.Process(target=print\_progress, args=(print\_q,))
221. print\_proc.start()
223. procs\_manager.join()
224. frames\_q.join()
225. print\_q.put(-1)             # tell printing process to finish himself
226. print\_proc.join()
228. **print** '\nsearched Finished.'
230. # print result to files.
231. commName = 'commands\_'+str(i)+'.txt'
232. results\_f = open(commName, 'w')
233. **for** j **in** xrange(frame\_cnt.value):
234. results\_f.write("%-2d: %-3d %-3d\n"%(j, row\_results[j], col\_results[j]))
235. # print "%-2d: %-3d %-3d"%(j, row\_results[j], col\_results[j]) # remove comment('#') to see prints on cmd
236. results\_f.close()
237. **print** "results printed to file %s."%commName
238. **print** "time: %.2f seconds\n"%(time.time()-start\_time)
240. frame\_cnt.value = 0   # zeroing frame\_cnt because starting a new file
242. **for** proc **in** p\_list:     # finish all search processes
243. proc.terminate()
245. **print** 'total time: %.2f seconds\n'%(time.time()-t)
246. a = raw\_input('\npress <Enter> to exit...')    # wait for enter key press and then exit.

249. **if** \_\_name\_\_ == '\_\_main\_\_':
250. main()