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:

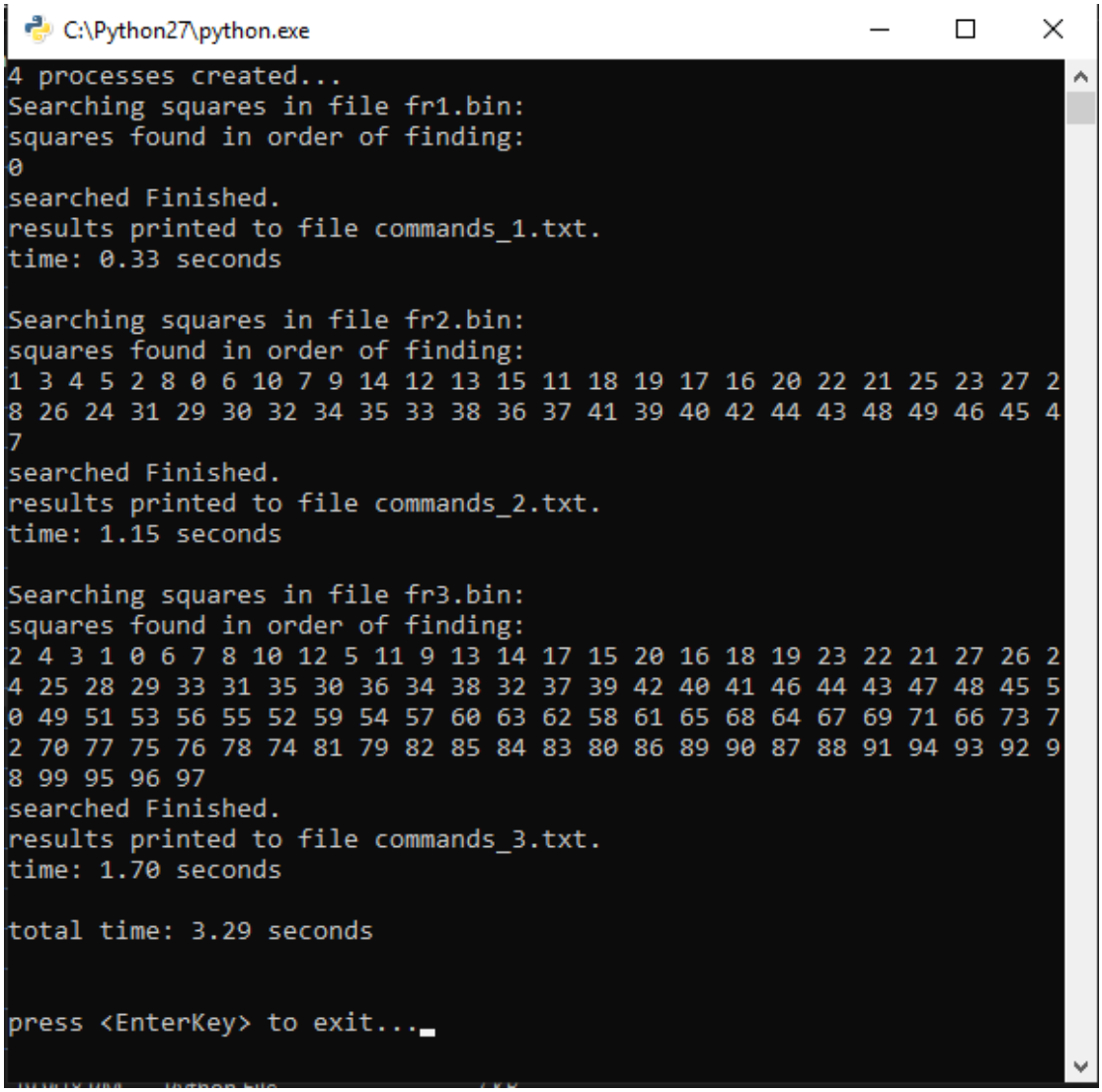
הקוד מתועד ומוסבר בצורה מלאה.

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

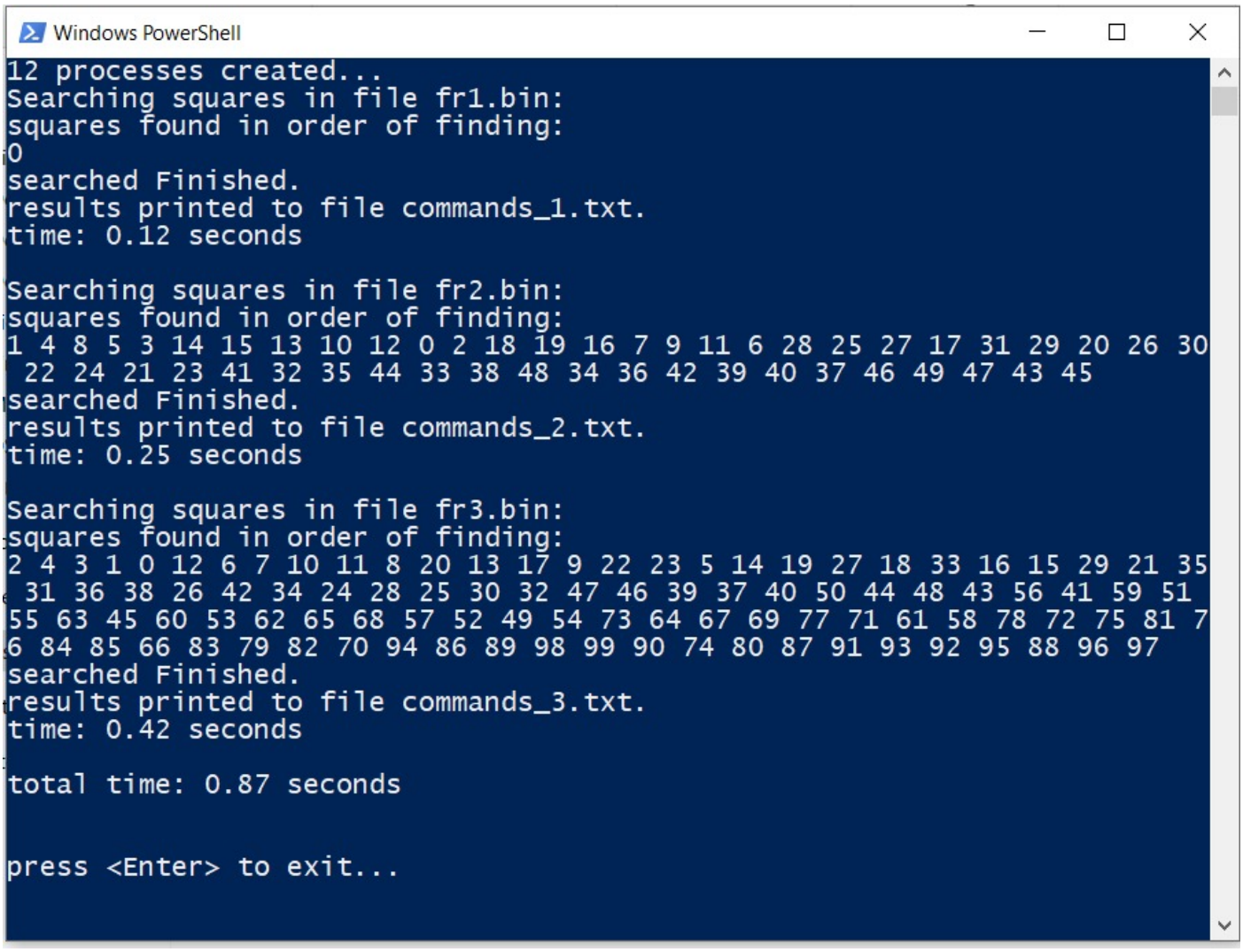
זמן ביצוע עבור מחשב עם ליבה בודדת יכול להגיע ל-6-8 שניות (לשלושת הקבצים).
מחשב מרובה ליבות יכול להגיע לזמן ביצוע של פחות משניה אחת (לשלושת הקבצים).

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
4.
5.
6.  def int2bytes(i):
7.      return hex2bytes(hex(i))
8.
9.
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')
16.
17.
18. row_res = 768
19. col_res = 1024
20. frame_size = row_res*col_res
21.
22. try:
23.     cpu_num = mul.cpu_count()
24. except NotImplementedError:
25.     cpu_num = raw_input('Enter how may CPU\'s this computer have: ')
26.
27. square_height = 5
28. square_width = 5
29.
30. upper_lim = int2bytes(110)
31. lower_lim = int2bytes(100)
32.
33. max_frames = 1000  # maximum frames that can be processed(in the same file)(can't resize shared me
    mory arrays)
34.
35.
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,
45.
46.
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)
52.
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(signi
    ficant improvement!)
58.     this is the best results we could get.
59.
60.
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 ran
    ge in all columns),
64.     lets define each row that checked 'checked row'.
```

```
    if found value in range check if this column has enough relevant sequentially values to be a left side of a square
    (from the rows from up to down),
    if there is a few possible squares check them all and if found full square return results(and stop checking)

    example:
    lets say our square start from row 3 to row 7(in some arbitrary column).
    lets say not found relevant value in row 3,then skipped to row 7 and found relevant value, then checks rows
    3-6,8-10 in the same column(notice: each checked row checking checked row above it but not the checked row below it)
    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)
    we start from checking the upper one (3,7) and we find that it has a full upper side, lower side, and right side
    and therefore its owr square and we quiting from the loop and returning results.

    """
    for i in xrange(square_height-1, row_res-(square_height-1)+1, square_height-1):
        for j in xrange(col_res-(square_height-1)):
            is_square = True
            row_offset = 0
            up_offset = 0
            down_offset = 0
            if lower_lim <= frame[i*col_res+j] <= upper_lim:
                # check if right border can possibly exist
                if not lower_lim <= frame[i*col_res+j+square_width-1] <= upper_lim:
                    continue

                # check if left border fully exist
                for uprow in xrange(1, square_height):
                    if lower_lim <= frame[(i-uprow)*col_res+j] <= upper_lim:
                        row_offset += 1
                        up_offset += 1
                    else:
                        break
                if not is_square:
                    continue
                for downrow in xrange(1, square_height-1):
                    if lower_lim <= frame[(i+downrow)*col_res+j] <= upper_lim:
                        row_offset += 1
                        down_offset += 1
                    else:
                        break
                if row_offset < square_height-1:
                    is_square = False
                if not is_square:
                    continue

                # there are diff possible squares in this column
                diff = row_offset-(square_height-1)

                # if there more then one option to a square in this col, start from checking from up to down
                for possible_square in xrange(diff+1):
                    # check if up border fully exist
                    is_square = True
                    for upcols in xrange(square_width-1):
                        if not lower_lim <= frame[(i-up_offset+possible_square)*col_res+j+upcols] <= upper_lim:
                            is_square = False
                            break
                    if not is_square:
                        continue
                    # check if down border fully exist
                    for downcols in xrange(square_width-1):
                        if not lower_lim<=frame[(i+down_offset-diff+possible_square)*col_res+j+downcols]<=upper_lim:
                            is_square = False
                            break
```

```
127.                     if not is_square:
128.                         continue
129.
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
135.
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
140.
141.
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.
147.
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 sho
    w progress
162.         frames_q.task_done()
163.
164.
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()
177.
178.
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 pos
    itions of rows
190.     col_results = mul.Array('i', max_frames)            # shared memory array representing the pos
    itions 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 f
    rames
194.     print_q = mul.JoinableQueue()                            # Q for printing progress in right order

195.
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_cn
    t, lock, print_q))
200.         p.start()
201.         p_list.append(p)
202.     print '%d processes created...'%cpu_num
203.
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:"
214.
215.             # create and start the process manager
216.             procs_manager = mul.Process(target=manage_procs, args=(fname, frames_q))
217.             procs_manager.start()
218.
219.             # start the process that responsible for printing progress
220.             print_proc = mul.Process(target=print_progress, args=(print_q,))
221.             print_proc.start()
222.
223.             procs_manager.join()
224.             frames_q.join()
225.             print_q.put(-1)             # tell printing process to finish himself
226.             print_proc.join()
227.
228.             print '\nsearched Finished.'
229.
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)
239.
240.             frame_cnt.value = 0   # zeroing frame_cnt because starting a new file
241.
242.     for proc in p_list:     # finish all search processes
243.         proc.terminate()
244.
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.
247.
248.
249. if __name__ == '__main__':
250.     main()
```