Національный технічний університет України
"Київський політехнічний інститут"

Лабораторна робота

Тема: Дослідження дисциплін обслуговування заявок при обмежених ресурсах

Виконав:
Радер Роман
залікова книжка: 215
група ІО-02

Київ 2013

Варіант: $15\%16 + 1 = 16$ 16. Змішаний алгоритм. FIFO + АП

Вхідні дані: тривалість кванту часу, кількість черг, кількість квантів часу для кожної черги, інтенсивність задач та інтенсивність задач з абсолютним пріоритетом.

Середній час очікування в черзі від інтенсивності заявок:

l3.py

```
import random
import logging
import math
from pylab import arange

logging.basicConfig(level=logging.INFO)


def log(name, text):
    logging.getLogger(name).debug(text)


TASK_NEW, TASK_PROGRESS, TASK_SUSPENDED, TASK_FINISHED, TASK_KILLED = range(5)


class Task(object):
    ALL_TASKS = []
    ALL_TASKS_AP = []

    def __init__(self, env, duration, ap):
        self.env = env
        self.duration = duration
        self.creation_time = env.time
        self.state = TASK_NEW
        self.ap = ap
        self.last_work_time = self.creation_time
        self.waiting_time = 0
        if ap:
            log('tsk', "[%d] task produced %ds ABSOLUTE PRIORITY" % (env.time, self.
                duration))
        else:
            log('tsk', "[%d] task produced %ds" % (env.time, self.duration))
        if ap:
            Task.ALL_TASKS_AP.append(self)
        else:
            Task.ALL_TASKS.append(self)

    def suspend(self):
        if self.state != TASK_SUSPENDED:
            self.last_work_time = self.env.time
            self.state = TASK_SUSPENDED
            log('tsk', "[%d] task #%d suspended" % (self.env.time, self.id))

    def resume(self):
        if self.state != TASK_PROGRESS:
            self.waiting_time += self.env.time - self.last_work_time
            self.state = TASK_PROGRESS
            log('tsk', "[%d] task #%d resumed" % (self.env.time, self.id))

    def finished(self):
        self.state = TASK_FINISHED
        self.finish_time = self.env.time
        self.turnaround = self.finish_time - self.creation_time
```

```
53          log('tsk', "[%d] task #%d finished. turnaround time: %d" %
54              (self.env.time, self.id, self.turnaround))
55
56      def kill(self):
57          self.state = TASK_KILLED
58          log('tsk', "[%d] task #%d killed" % (self.env.time, self.id))
59
60
61  def task_generator(env, task_f, task_d, ap):
62      time = task_f()  # once per period
63      prev_time = env.time
64      diff = env.time + time
65      while True:
66          diff = env.time - prev_time
67          while diff >= time:
68              diff -= time
69              task = Task(env, task_d(), ap)
70              yield task
71              prev_time = env.time
72              time = task_f()
73          yield None
74
75
76  def poisson(l):
77      return -(1/float(l)) * math.log(random.random())
78
79
80  def def_task_generator(env):
81      task_f = lambda: poisson(env.LAMBDA_DEF)  # random.random()*3*1000+2000  #
            2000..5000
82      task_d = lambda: random.random()*3*1000+1000  # 1000..4000
83      return task_generator(env, task_f, task_d, False)
84
85
86  def ap_task_generator(env):
87      task_f = lambda: poisson(env.LAMBDA_AP)  # random.random()*1.5*1000+4000  #
            4000..5500
88      task_d = lambda: random.random()*1*1000+500  # 500..1500
89      return task_generator(env, task_f, task_d, True)
90
91
92  class Queue(list):
93      pass
94
95
96  class BaseProcessor(object):
97      def __init__(self, environment):
98          self.new_tasks = []
99          self.tid_count = 1
100         self.environment = environment
101
102     def add_task(self, task):
103         task.id = self.tid_count
104         self.tid_count += 1
105         task.worktime = 0
106         self.new_tasks.append(task)
107
108     def tick(self):
109         pass
110
111 TIME_QUANT = 400
```

```
112 THRESHOLDS = [3, 6, 20]
113 QUEUE_NUM = 3
114
115
116 class Processor(BaseProcessor):
117     def __init__(self, env, quant=TIME_QUANT, q_num=QUEUE_NUM, thresholds=THRESHOLDS):
118         super().__init__(env)
119         self.qs = [Queue() for q in range(q_num)]
120         self.q_ap = []
121         self.current = None
122         self.thresholds = thresholds
123         self.quant = quant
124         self.prev_time = env.time
125
126     def get_new(self):
127         self.current = None
128
129         if len(self.q_ap):
130             self.current = self.q_ap[0]
131             self.current.queue = -1
132             self.current.resume()
133             self.q_ap.remove(self.current)
134             return
135
136         for q_id, queue in enumerate(self.qs):
137             if len(queue) > 0:
138                 self.current = queue[0]
139                 self.current.queue = q_id
140                 self.current.resume()
141                 self.qs[self.current.queue].remove(self.current)
142                 break
143
144     def tick(self):
145         for task in self.new_tasks:
146             task.loops = 0
147             task.queue = 0
148             if task.ap:
149                 self.q_ap.append(task)
150             else:
151                 self.qs[0].append(task)
152         self.new_tasks = []
153
154         elapsed = self.environment.time - self.prev_time
155         # log('cpu', "elapsed %d" % elapsed)
156         self.prev_time = self.environment.time
157
158         if not self.current:
159             self.get_new()
160             if self.current:
161                 self.current.session = 0
162
163         if self.current:
164             # elapsed = min(self.quant, self.current.duration -
165                  # self.current.worktime)
166             self.current.worktime += elapsed
167             self.current.session += elapsed
168             # log('cpu', "[%d] task #%d worktime %d AFTER" %
169             #     (env.time, self.current.id, self.current.worktime))
170             if self.current.worktime >= self.current.duration or \
171                self.current.session >= self.quant or \
172                (len(self.q_ap) and not self.current.ap):
```

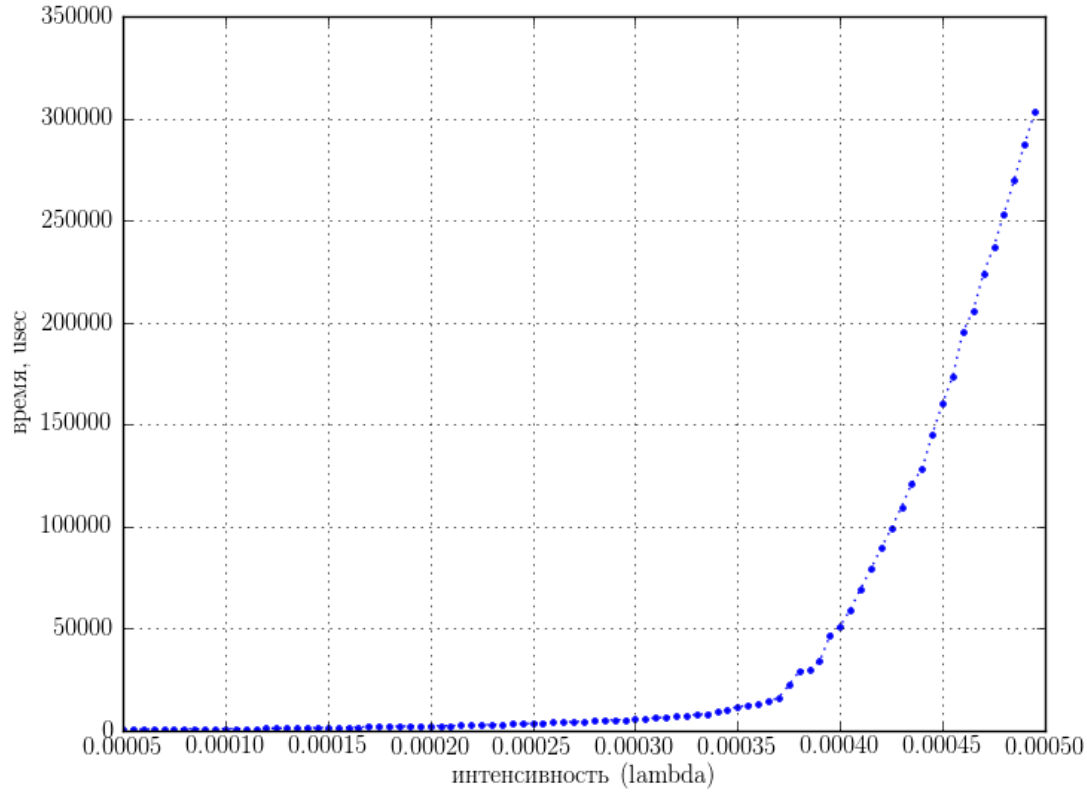
```
173                 self.current.session = 0
174                 self.current.suspend()
175                 self.current.loops += 1
176                 if self.current.loops >= self.thresholds[self.current.queue] and \
177                    not self.current.ap:
178                     self.current.loops = 0
179                     self.current.queue += 1
180                     if self.current.queue >= len(self.qs):
181                         self.current.kill()
182                     else:
183                         log('cpu', "task #%d moved to queue %d" %
184                             (self.current.id, self.current.queue))
185                 if self.current.worktime < self.current.duration and \
186                    self.current.state != TASK_KILLED:
187                     if self.current.queue == -1:
188                         self.q_ap.append(self.current)
189                     else:
190                         self.qs[self.current.queue].append(self.current)
191                 else:
192                     self.current.finished()
193 
194                 self.get_new()
195                 if self.current:
196                     self.current.session = 0
197         # return elapsed  # 0.6t period
198 
199     def is_empty(self):
200         return all(len(q) == 0 for q in self.qs) and not self.current
201 
202 
203 class Environment(object):
204     def __init__(self):
205         self.time = 0
206         self.time = 0
207         self.tasks = def_task_generator(self)
208         self.ap_tasks = ap_task_generator(self)
209         self.processor = Processor(self)
210         self.producing = True
211         self.utilization_time = 0
212         self.QUANT = 10
213         self.LAMBDA_DEF = 0.0003
214         self.LAMBDA_AP = 0.00005
215 
216     def is_empty(self):
217         return self.processor.is_empty()
218 
219     def tick(self):
220         if self.producing:
221             task = next(self.tasks)
222             while task is not None:
223                 self.processor.add_task(task)
224                 log('env', "[%d] added task #%d" % (self.time, task.id))
225                 task = next(self.tasks)
226 
227             task = next(self.ap_tasks)
228             while task is not None:
229                 self.processor.add_task(task)
230                 log('env', "[%d] added task #%d" % (self.time, task.id))
231                 task = next(self.tasks)
232         self.time += self.QUANT
233         if self.processor.current:
```

```
234             self.utilization_time += self.QUANT
235         self.processor.tick()
236         # log('env', "[%d] ticked" % (env.time))
237
238
239 def test(l_def, l_ap, time):
240     env = Environment()
241     env.LAMBDA_DEF = l_def
242     env.LAMBDA_AP = l_ap
243     while env.time < time:
244         env.tick()
245
246     prod = env.time
247     env.producing = False
248     while not env.is_empty():
249         env.tick()
250     print("SIMULATION FINISHED")
251
252     results = {}
253
254     results['overtime'] = env.time - prod
255     print("%d overtime" % results['overtime'])
256
257     results['utilization'] = (env.utilization_time / env.time)*100
258     results['tasks processed'] = env.processor.tid_count
259     results['AP tasks'] = len(Task.ALL_TASKS_AP)
260     results['tasks/s'] = (env.processor.tid_count / (env.time / 1000))
261     print("%4.2f%% utilization" % results['utilization'])
262     print("%d tasks processed" % results['tasks processed'])
263     print("%d AP tasks" % results['AP tasks'])
264     print("%4.4f tasks/s" % results['tasks/s'])
265
266     mean_turnaround_def = sum(t.turnaround for t in Task.ALL_TASKS) / len(Task.
        ALL_TASKS)
267     results['mean turnaround def'] = mean_turnaround_def
268     mean_turnaround_ap = sum(t.turnaround for t in Task.ALL_TASKS_AP) / len(Task.
        ALL_TASKS_AP)
269     results['mean turnaround ap'] = mean_turnaround_ap
270     print("%4.4f mean turnaround" % (mean_turnaround_def))
271     print("%4.4f mean turnaround AP" % (mean_turnaround_ap))
272
273     waiting_time = sum(t.waiting_time for t in Task.ALL_TASKS) / len(Task.ALL_TASKS)
274     results['waiting time'] = waiting_time
275     print("%4.4f waiting time" % waiting_time)
276     waiting_time_AP = sum(t.waiting_time for t in Task.ALL_TASKS_AP) / len(Task.
        ALL_TASKS_AP)
277     results['waiting time AP'] = waiting_time_AP
278     print("%4.4f waiting time AP" % waiting_time_AP)
279
280     sorted_tasks = sorted(Task.ALL_TASKS, key=lambda x: x.duration)
281
282     waiting_times = []
283     durations = []
284     last_dur = sorted_tasks[0].duration
285     vals = []
286
287     waiting_times_qs = [[] for x in range(QUEUE_NUM)]
288     durations_qs = [[] for x in range(QUEUE_NUM)]
289     start_q_id = 0
290
291     for i, dur in enumerate([t.duration for t in sorted_tasks]):
```

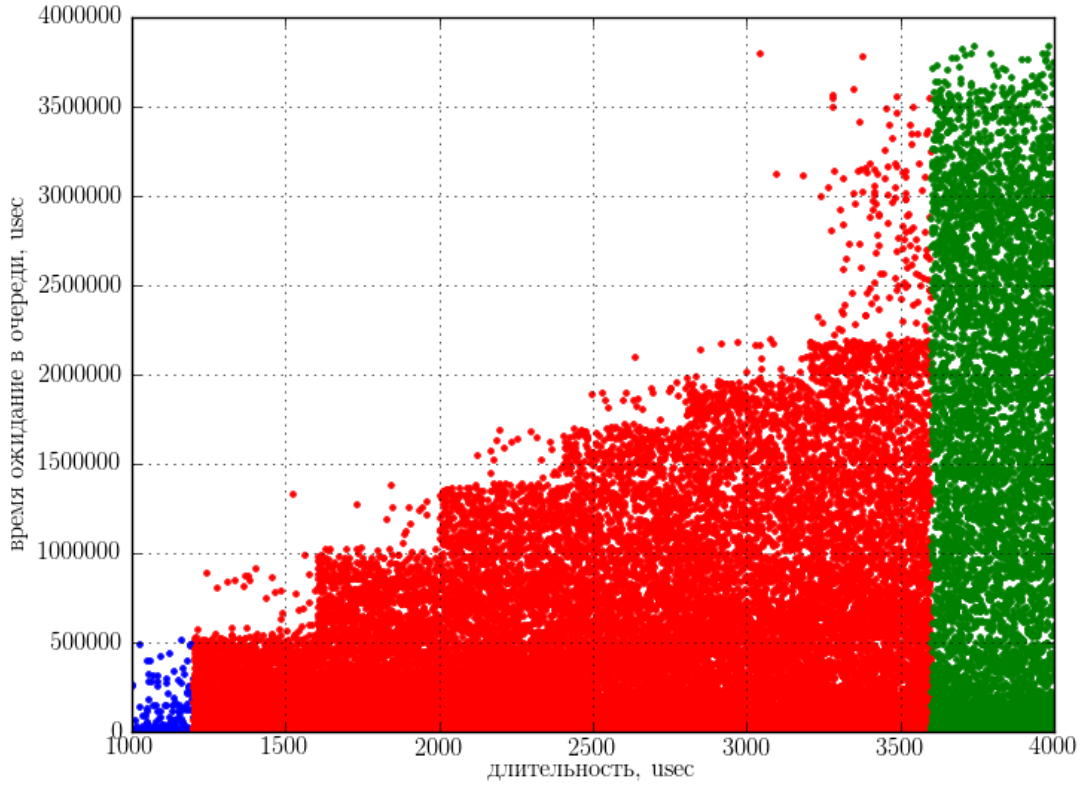
```
292         if last_dur != dur:
293             durations.append(last_dur)
294             waiting_times.append(sum(vals)/len(vals))
295             while last_dur > TIME_QUANT*(sum(THRESHOLDS[q_id] for q_id in range(0,
                    start_q_id+1))):
296                 start_q_id += 1
297             waiting_times_qs[start_q_id].append(sum(vals)/len(vals))
298             durations_qs[start_q_id].append(last_dur)
299             vals = []
300         vals.append(sorted_tasks[i].waiting_time)
301         last_dur = dur
302     else:
303         durations.append(last_dur)
304         waiting_times.append(sum(vals)/len(vals))
305
306     results['waiting times'] = waiting_times  # [t.waiting_time for t in sorted_tasks]
307     results['durations'] = durations  # [t.duration for t in sorted_tasks]
308
309     results['waiting times qs'] = waiting_times_qs
310     results['durations qs'] = durations_qs
311     return results
312
313
314 def make_plot(x, y, title_t, xlabel_t=u'интенсивность (lambda)', ylabel_t=u'время, usec
        ', dot='b.:', clear=True):
315     from pylab import plot, axis, xlabel, ylabel, grid, show, savefig, cla, rc
316     rc('font', **{'family': 'serif'})
317     rc('text', usetex=True)
318     rc('text.latex', unicode=True)
319     rc('text.latex', preamble='\\usepackage[utf8]{inputenc}')
320     rc('text.latex', preamble='\\usepackage[russian]{babel}')
321
322     plot(x, y, dot)
323     # axis([0, 0.01, 0, 10000])
324     ylabel(ylabel_t)
325     xlabel(xlabel_t)
326     grid()
327
328     savefig('%s.png' % title_t.replace(" ", "_"))
329     if clear:
330         cla()
331
332
333 if __name__ == '__main__':
334     time = 5000*600
335     l_def_r = arange(0.00005, 0.0005, 0.000005)
336     results = []
337     for l_def in l_def_r:
338         print ("Test: %f" % l_def)
339         res = test(l_def, 0.00005, time)
340         results.append(res)
341
342     make_plot(l_def_r, [x['waiting time'] for x in results], 'waiting time')
343     make_plot(l_def_r, [100-x['utilization'] for x in results], 'idle time')
344     make_plot(l_def_r, [x['waiting time AP'] for x in results], 'waiting time AP')
345
346     res = test(0.0002, 0.00005, time)
347     make_plot(res['durations'], res['waiting times'], 'waiting by duration',
348               'длительность, usec', 'время ожидание в очереди, usec', dot='b.')
349
```

```
350     make_plot(res['durations qs'][0], res['waiting times qs'][0], 'waiting by duration
            with queues',
351               'длительность, usec', 'время ожидание в очереди, usec', dot='b.', clear=
                  False)
352     make_plot(res['durations qs'][1], res['waiting times qs'][1], 'waiting by duration
            with queues',
353               'длительность, usec', 'время ожидание в очереди, usec', dot='r.', clear=
                  False)
354     make_plot(res['durations qs'][2], res['waiting times qs'][2], 'waiting by duration
            with queues',
355               'длительность, usec', 'время ожидание в очереди, usec', dot='g.', clear=
                  True)
356
357     make_plot([x['waiting time'] for x in results],
358               [x['tasks processed'] for x in results], 'tasks count by waiting time',
359               'время ожидание в очереди, usec', 'задач обработано', dot='b.:')
```

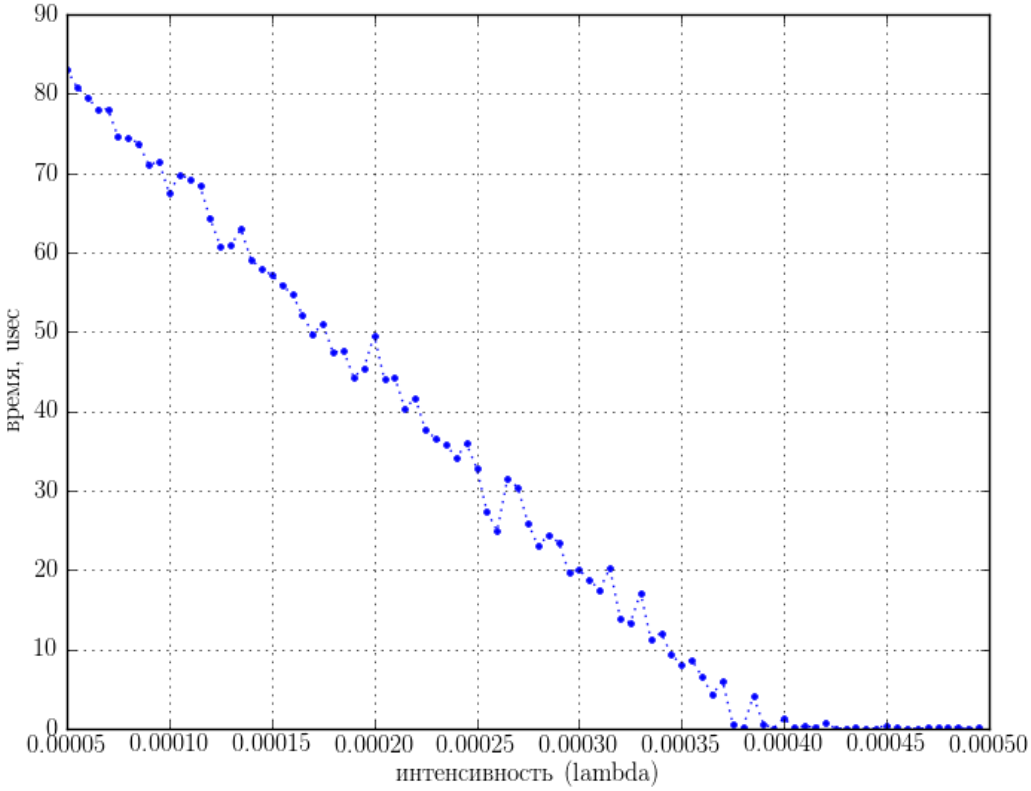
Час чекання в черзі від інтенсивності:



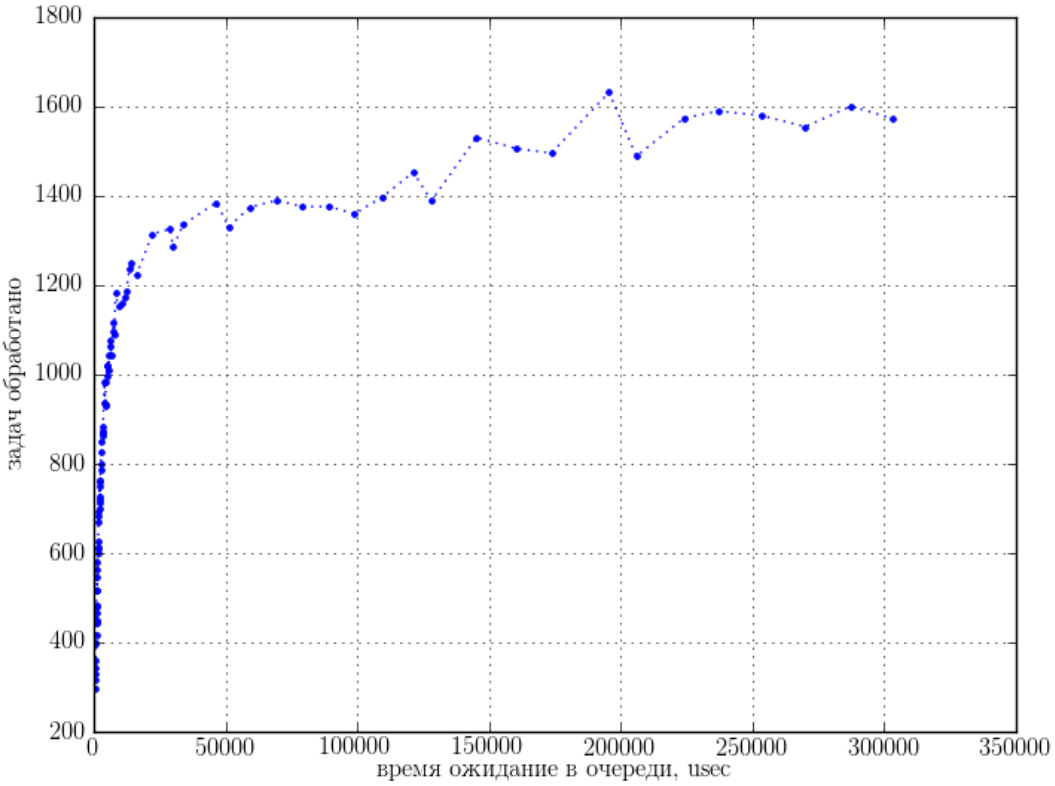
Час чекання в черзі від довжини задачі:



Час простою від інтенсивності заявок:



Кількість опрацьованих задач від часу чекання:



Середній час очікування задач абсолютного приорітету в черзі від інтенсивності заявок: