SURVEY METHODOLOGY

SURVEY METHODOLOGY
This is the Subtitle

Robert M. Groves
Universitat de les Illes Balears

Floyd J. Fowler, Jr.
University of New Mexico

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Survey Methodology / Robert M. Groves . . . [et al.].
p. cm.—(Wiley series in survey methodology)
"Wiley-Interscience."
Includes bibliographical references and index.
ISBN 0-471-48348-6 (pbk.)
1. Surveys—Methodology. 2. Social sciences—Research—Statistical methods. I. Groves, Robert M. II. Series.

HA31.2.S873 2007
001.4'33—dc22 2004044064
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To my parents

CONTRIBUTORS

MASAYKI ABE, Fujitsu Laboratories Ltd., Fujitsu Limited, Atsugi, Japan

L. A. AKERS, Center for Solid State Electronics Research, Arizona State University, Tempe, Arizona

G. H. BERNSTEIN, Department of Electrical and Computer Engineering, University of Notre Dame, Notre Dame, South Bend, Indiana; formerly of Center for Solid State Electronics Research, Arizona State University, Tempe, Arizona

CONTENTS IN BRIEF

CONTENTS

LIST OF FIGURES

LIST OF TABLES

FOREWORD

This is the foreword to the book.

PREFACE

This is an example preface. This is an example preface. This is an example preface. This is an example preface.

R. K. WATTS

Durham, North Carolina
September, 2007

ACKNOWLEDGMENTS

From Dr. Jay Young, consultant from Silver Spring, Maryland, I received the initial push to even consider writing this book. Jay was a constant “peer reader” and very welcome advisor durying this year-long process.

To all these wonderful people I owe a deep sense of gratitude especially now that this project has been completed.

G. T. S.

ACRONYMS

ACGIH	American Conference of Governmental Industrial Hygienists
AEC	Atomic Energy Commission
OSHA	Occupational Health and Safety Commission
SAMA	Scientific Apparatus Makers Association

GLOSSARY

NormGibbs	Draw a sample from a posterior distribution of data with an unknown mean and variance using Gibbs sampling.
pNull	Test a one sided hypothesis from a numberically specified posterior CDF or from a sample from the posterior
sintegral	A numerical integration using Simpson's rule

SYMBOLS

A Amplitude
& Propositional logic symbol
a Filter Coefficient

$\mathcal{B}$ Number of Beats

INTRODUCTION

CATHERINE CLARK, PHD.

Harvard School of Public Health
Boston, MA, USA

The era of modern began in 1958 with the invention of the integrated circuit by J. S. Kilby of Texas Instruments [1]. His first chip is shown in Fig. I. For comparison, Fig. I.2 shows a modern microprocessor chip, [4].

This is the introduction. This is the introduction. This is the introduction. This is the introduction. This is the introduction. This is the introduction.

$$ABCDEF\alpha\beta\Gamma\Delta\sum_{def}^{abc} \tag{I.1}$$

REFERENCES

1. J. S. Kilby, "Invention of the Integrated Circuit," *IEEE Trans. Electron Devices,* **ED-23,** 648 (1976).
2. R. W. Hamming, *Numerical Methods for Scientists and Engineers*, Chapter N-1, McGraw-Hill, New York, 1962.
3. J. Lee, K. Mayaram, and C. Hu, "A Theoretical Study of Gate/Drain Offset in LDD MOSFETs" *IEEE Electron Device Lett.,* **EDL-7**(3). 152 (1986).

PART I

SUBMICRON SEMICONDUCTOR MANUFACTURE

CHAPTER 1

HOME

CHAPTER 2

OVERVIEW

CHAPTER 3

ENVIRONTMENT SETUP

CHAPTER 4

BASIC SYNTAX

CHAPTER 5

VARIABEL TYPE

CHAPTER 6

BASIC OPERATOR

CHAPTER 7

DESICION MAKING

CHAPTER 8

LOOP

CHAPTER 9

NUMBERS

CHAPTER 10

STRINGS

CHAPTER 11

LISTS

CHAPTER 12

TUPLES

CHAPTER 13

DICTIONARY

CHAPTER 14

FUNCTIONS

CHAPTER 15

MODULES

CHAPTER 16

FILES I/O

CHAPTER 17

EXCEPTIONS

CHAPTER 18

CLASESS/OBJECT

CHAPTER 19

REG EXPRESSION

CHAPTER 20

CGI PROGRAMMING

CHAPTER 21

DATABASES ACCESS

CHAPTER 22

SENDING EMAIL

CHAPTER 23

MULTITHREADING

BAB I

PYTHON MULTITHREADED PROGRAMMING

Menjalankan beberapa *thread* mirip dengan menjalankan beberapa program yang berbeda secara bersamaan, namun dengan manfaat berikut :

- Beberapa *thread* dalam proses berbagi ruang data yang sama dengan benang induk dan karena dapat saling berbagi informasi atau berkomunikasi satu sama lain dengan lebih muda daripada jika prosesnya terpisah
- *thread* terkadang disebut proses ringan dan tidak membutuhkan banyak memori atas, mereka lebih murah daripada proses.

Sebuah *thread* memiliki permulaan, urutan eksekusi dan sebuah kesimpulan. Ini memiliki pointer perintah yang melacak dari mana dalam konteksnya saat ini berjalan.

- Hal ini dapat dilakukan sebelum pre-*empted* (*inturrepted*)

- Untuk sementara dapat ditunda sementara *thread* lainnya yang sedang berjalan ini disebut unggul.

1.1 Memulai *Thread* Baru

Untuk melakukan *thread* lain, perlu memanggil metode berikut yang tersedia dimodul *thread* :

Thread.start _new _thread (function, args [, kwargs])

Pemanggilan metode ini memungkinkan cara cepat dan tepat untuk membuat *thread* baru di linux dan window.

Pemanggilan metode segera kembali dan anak *thread* dimulai dan fungsi pemanggilan dengan daftar *args* telah berlalu. Saat fungsi kembali ujung *thread* akan berakhir.

Disini, *args* adalah tupel argumen. Gunakan tupel kosong untuk memanggil fungsi tanpa melewati argumen. *Kwargs* adalah kamus opsional argumen kata kunci.

Contoh :

```
#!/usr/bin/python

Import thread
Import time

# Define a function for the thread
Def print _time (threadNamw, delay):
        Count = 0
        While count ¡5:
        Time.sleep(delay)
        Count +=1
        Print " %s : %s " % (threadName, time.ctime(time.time()))

# Create two thread as follows
try:
thread.start _new _thread(print _time, ( "Thread-1 ", 2, ))
thread.start _new _thread(print _time, ( "Thread-2 ", 4,))
except:
  print "Error: unable to start thread "

while 1:
pass
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut :

```
Thread-1 : Thu Jan 22 15:42:17 2009

Thread-1 : Thu Jan 22 15:42:19 2009

Thread-2 : Thu Jan 22 15:42:19 2009

Thread-1 : Thu Jan 22 15:42:21 2009

Thread-2 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:25 2009

Thread-2 : Thu Jan 22 15:42:27 2009

Thread-2 : Thu Jan 22 15:42:31 2009

Thread-2 : Thu Jan 22 15:42:35 2009
```

Meskipun sangat efektif untuk benang tingkat rendah, namun modul *thread* sangat terbatas dibandingkan dengan modul yang baru.

1.2 Modul Threading

Modul threading yang lebih baru disertakan dengan Python 2.4 memberikan jauh lebih kuat, dukungan tingkat tinggi untuk *thread* dari modul *thread* dibahas pada bagian sebelumnya.

The *threading* modul mengekpos semua metode dari *thread* dan menyediakan beberapa metode tambahan :

- **threading.activeCount()**

  Mengembalikan jumlah objek *thread* yang aktif

- **threading.currentThread()**

  Mengembalikan jumlah objek *thread* dalam kontrol benang pemanggil

- **threading.enumerate()**

Mengembalikan daftar semua benda *thread* yang sedang aktif

Selain metode, modul *threading* memiliki *thread* kelas yang mengimplementasikan *threading*. Metode yang disediakan oleh *thread* kelas adalah sebagai berikut :

- **run()**

  Metode adalah titik masuk untuk *thread*

- **start()**

  Metode dimulai *thread* dengan memanggil metode run

- **join([time])**

  Menunggu benang untuk mengakhiri

- **isAlive()**

  Metode memeriksa apakah *thread* masih mengeksekusi

- **getName()**

  Metode mengambalikan nama *thread*

- **setName()**

  Metode menetapkan nama *thread*

1.3 Membuat *Thread* Menggunakan *Threading* Modul

Untuk melaksanakan *thread* baru menggunakan *threading* harus melakukan hal berikut :

- Mendefinisikan subclass dari *thread* kelas
- Menimpa _init _ (self [args]) metode untuk menambahkan argumen tambahan
- Menimpa run(self[args]) metode untuk menerapkan apa *thread* harus dilakukan ketika mulai

Setelah membuat baru *thread* subclass, dapat membuah seuah instance dari itu dan kemudian memulai *thread* baru dengan menerapkan *start(),* yang ada gilirinnya panggilan *run()* metode.

Contoh :

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
        def _init _(self, threadID, name, counter) :
    threading.Thread. _init _(self)
        self.threadID = threadID
        self.name = name
self.counter = counter
def run (self) :
        print "Starting " + self.name
        print _time(self.name, self.counter, 5)
        print "Exiting "+ self.name
def print _time(threadName, delay, counter):
while counter:
        if exitFlag:
                threadName.exit()
        time.sleep(delay)
        print " %s: %s " % (threadName, time.ctime(time.time()))
counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1 ", 1)
thread2 = myThread(2, "Thread-2 ", 2)

# Start new threads
thread1.start()
thread2.start()
print "Exiting Main Thread "
```

Ketika kode diatas dijalankan, menghasilkan hasil sebagai berikut:

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1 : Thu Mar 21 09:10:03 2013
Thread-1 : Thu Mar 21 09:10:04 2013
Thread-2 : Thu Mar 21 09:10:04 2013
```

```
Thread-1 : Thu Mar 21 09:10:05 2013
Thread-2 : Thu Mar 21 09:10:06 2013
Thread-1 : Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2 : Thu Mar 21 09:10:08 2013
Thread-2 : Thu Mar 21 09:10:10 2013
Thread-2 : Thu Mar 21 09:10:12 2013
Exiting Thread=2
```

1.4 Sinkronisasi *Thread*

Threading modul disediakan dengan Python termasuk sederhana untuk menerapkan mekanisme bahwa memungkinkan untuk menyinkronkan *thread* penguncian. Sebuah kunci baru dibuat dengan memanggil *lock()* metode yang mengembalikan kunci baru.

The *acquire (blocking)* metode objek kunci baru digunakan untuk memaksa *thread* untuk menjalankan serempak. Opsional *blocking* parameter memungkikan untuk mengontrol apakah *thread* menunggu untuk mendapatkan kunci.

Jika *blocking* diatur ke 0, *thread* segera kembali dengan nilai 0 jika kunci tidak dapat diperoleh dan dengan 1 jika kunci dikuisisi. Jika pemblokiran diatur ke 1, blok dan menunggu kunci yang akan dirilis.

The *release()* metode objek kunci baru digunakan untuk melepaskan kunci ketika tidak lagi diperlukan.

Contoh:

```
#!/usr/bin/python

import threading
import time

class myThread (threading.Thread):
  def _init _(self, threadID, name, counter):
    threading.Thread. _init _(self)
    self.threadID = threadID
    self.name = name
    self.counter = counter
  def run(self)
    print "Starting "+ self.name
    # Get lock to synchronize threads
    ThreadLock.acquire()
    print _time(self.name, self.counter, 3)
    # Free lock to realease next thread
    ThreadLock.release()

  Def print _time(threadName, delay, counter):
   while counter:
    time.sleep(delay)
```

```
   print " %s: %s " % (threadName, time.ctime(time.time()))
   counter -= 1
 threadLock = threading.Lock()
 threads = []

# Create new threads
thread1 = myThread(1, "Thread-1,1 )
thread2 = myThread(2, "Thread-2,2 )

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
thread2.append(thread2)

# Wait for all threads to complete
Fort t in threads:
   t.join()
print "Exiting Main thread "
```

Bila kode diatas dieksekusi, maka menghasilkan sebagai berikut :

```
Starting Thread-1
Starting Thread-2
Thread-1: Thu Mar 21 09:11:28 2013
Thread-1: Thu Mar 21 09:11:29 2013
Thread-1: Thu Mar 21 09:11:30 2013
Thread-2: Thu Mar 21 09:11:32 2013
Thread-2: Thu Mar 21 09:11:34 2013
Thread-2: Thu Mar 21 09:11:36 2013
Exiting Main Thread
```

1.5 Multithreaded Antrian Prioritas

The queue modul memungkinkan untuk membuat objek antrian baru yang dapat menampung jumlah tertentu item. Ada metode berikut untuk mengontrol antrian :

- **get()**

  Menghapus dan mengembalikan item dari antrian

- **put()**

  Menambahkan item ke antrian

- **qsize()**

  Mengembalikan jumlah item yang saat ini dalam antrian

- **empty()**

  Mengembalikan benar jika antrian kosong jika tidak, salah

- **full()**

Mengembalikan benar jika antrian penuh jika tidak, salah

Contoh:

```
#!/usr/bin/python

import Queue
import threading
import time

exitFlag = 0

class myThread (threading.Thread):
  def _init_(self, threadID, name, q):
     threading.Thread. _init_(self)
    self.name = name
    self.q = q
def run(self):
      print "Starting "+ self.name
      process_data(self.name, self.q)
      print "Exiting "+ self.name

def process_data(threadName, q):
    while not exitFlag:
    queueLock.acquire()
    if not workQueu.empty():
        data = q.get()
        queueLock.release()
        print " %s processing %s " % (threadName, data)
     else:
        queueLock.release()
        time.sleep(1)

threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
queueLock = threading.Lock()
workLock = Queue.Queue(10)
threads = []
threadID = 1
```

```
# Create new threads
For tName in threadList:
   thread = myThread(threadID, tName, workQueue)
   thread.start()
   thread.append(thread)
   threadID +=1

# Fill the queue
queueLock.acquire()
for word in nameList:
   workQueue.put(word)
queueLock.release()

# Wait for queue to empty
while not workQueue.empty():
pass

# Notify threads it?s time to exit
exitFlag = 1

# Wait for all threads to complete
For t in threads:
   t.join()
print ”Exiting Main Thread ”
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut:

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
Exiting Thread-3
Exiting Thread-1
Exiting Thread-2
Exiting Main Thread
```

CHAPTER 24

XML PROCESSING

CHAPTER 25

GUI PROGRAMMING

CHAPTER 26

FUTHER EXPRESSION

REFERENCES

1. J. S. Kilby, "Invention of the Integrated Circuit," *IEEE Trans. Electron Devices,* **ED-23,** 648 (1976).
2. R. W. Hamming, *Numerical Methods for Scientists and Engineers*, Chapter N-1, McGraw-Hill, New York, 1962.
3. J. Lee, K. Mayaram, and C. Hu, "A Theoretical Study of Gate/Drain Offset in LDD MOSFETs" *IEEE Electron Device Lett.,* **EDL-7**(3). 152 (1986).
4. A. Berenbaum, B. W. Colbry, D.R. Ditzel, R. D Freeman, and K.J. O'Connor, "A Pipelined 32b Microprocessor with 13 kb of Cache Memory," it Int. Solid State Circuit Conf., Dig. Tech. Pap., p. 34 (1987).

REFERENCES

[Kil76] J. S. Kilby, "Invention of the Integrated Circuit," *IEEE Trans. Electron Devices,* **ED-23,** 648 (1976).

[Ham62] R. W. Hamming, *Numerical Methods for Scientists and Engineers*, Chapter N-1, McGraw-Hill, New York, 1962.

[Hu86] J. Lee, K. Mayaram, and C. Hu, "A Theoretical Study of Gate/Drain Offset in LDD MOSFETs" *IEEE Electron Device Lett.,* **EDL-7**(3). 152 (1986).

[Ber87] A. Berenbaum, B. W. Colbry, D.R. Ditzel, R. D Freeman, and K.J. O'Connor, "A Pipelined 32b Microprocessor with 13 kb of Cache Memory," it Int. Solid State Circuit Conf., Dig. Tech. Pap., p. 34 (1987).