이 강좌는 연세대학교 이용석 교수 연구실에서 제작되었으며 copyright가 없으므로 비영리적인 목적에 한하여 누구든지 복사, 배포가 가능합니다. 연구실 홈페이지는 고성능 마이크로프로세서에 관련된 많은 강의가 있으며 누구나 무료로 다운로드 받을 수 있습니다.

멀티스레드 프로그래밍
(Multithreaded Programming)

연세대학교 이용석 교수 연구실
박사과정 김재억
E-mail: yonglee@yonsei.ac.kr
Homepage: http://mpu.yonsei.ac.kr

전화: 02-2123-2872

Reference

1. JAVA Concurrency in Practice, Brian Goetz, Addison-Wesley, 2006
3. Python 개정판 Ver.2 page 681 ~ 720, 이강성, 프리액, 2005
4. Tutorial on Threads Programming with Python, Norman Matloff and Francis Hsu, University of California, 2007

Content

- Why Multi-core?
- Process and Thread
- Three Hazards of Multi-threaded Programming
- Comparison between C and Python
- Simple Python Tutorial
- Multi-threaded Programming

Reference

5. Operating System Concepts, Silberschatz, John Wiley & Sons, INC, 2005
8. 운영체제 --- 그림으로 배우는 원리와 구조, 구현회, 한빛미디어, 2007
Why Multi-core?

Multi-core

- Clock rates are saturated.
  - Power consumption
  - Heat dissipation
- Multi-core CPUs
  - CPU contains two or more cores.
    - Intel
      - Core Duo
      - Core 2 Duo
      - Core 2 Quad
    - AMD
      - Athlon64
      - Opteron
      - Phenom

Amdahl’s law

\[
\text{Speed Improvement} \leq \frac{1}{F \cdot (1 - F) \cdot N}
\]

N : Number of processors
F : the proportion that cannot be parallelized
F limits throughput of a Multi-core CPU.

Definitions of the term process

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources

Process and Thread
Two essential elements of a process

- Program code (which may be shared with other processes that are executing the same program)
- A set of data associated with that code

Process in Memory

- Stack
  - Temporary data
  - Function parameters
  - Return addresses
  - Local variables
- Heap
  - Dynamically allocated
- Data
  - Global variables
- Text
  - Program code
- Current activity
  - The value of the PC
  - The contents of registers
Two characteristics of a process

- Resource ownership
  - a virtual address space to hold the process image
  - program
  - stack
  - attributes defined in the process control block
  - resources
    - main memory
    - I/O channels
    - I/O devices
    - files
- Scheduling/execution
  - the execution of a process follows an execution path (trace) through one or more programs
  - a process has an execution state and a dispatching priority

Two-State process model

Five-State Process Model
Thread

- A thread is a flow of control within a process.
- A thread is a basic unit of CPU utilization.
- A multithreaded process contains several different flows of control within the same address space.
  - Each thread has a thread ID, a program counter, a register set, and a stack.
  - Each thread shares code section, data section, operating-system resources.
- Light-weighted process.

Thread VS Process

- It takes far less time to create a new thread in an existing process.
- It takes less time to terminate a thread than a process.
- It takes less time to switch between threads within the same process.
- Threads enhance efficiency in communication between different executing programs.

Multithreading

- Merits
  - Parallel computation.
  - Parallel I/O.
  - Asynchronous I/O events.
- Demerits
  - Safety hazards.
  - Liveness hazards.
  - Performance hazards.

Three Hazards of Multi-threaded Programming

- Safety hazards.
- Liveness hazards.
- Performance hazards.
Safety hazards

- 9 + 1 → 10
- 9 - 1 → 8
- Store Count(=10)

Mutual exclusion using critical regions

Thread A
- A enters critical region
- B attempts to enter critical region
- B blocked
- B leaves critical region
- A leaves critical region

Thread B
- Y enters critical region
- Y leaves critical region

- Race condition
  - A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution

- Critical section
  - A section of code within a process that requires access to shared resources and that may not be executed while another process is in a corresponding section of code.

- Mutual exclusion
  - The requirement that when one process is in a critical section that accessed shared resources, no other process may be in a critical section that accesses any of those shared resources.

- Deadlock
  - A situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something
Comparison between C and Python

- The first or second most popular language
- But...
  - Data structure
  - Class
  - Pointer
  - Incompatible problems between C Compilers

Python
- Easy to learn
- Executable code examples
- Code compatibility
- Cross platform
- Easy to install
  - Linux: included in the standard package
  - Windows: Downloads one install file
- Support POSIX Threads
  - "thread", "threading" module
- Support List, tuple, dictionary data type

Installing Python
- Python Official Website http://www.python.org/
- Download Standard Python Software
- Recommend
  - Python 2.5.4

C Language

Simple Python Tutorial
Develop environment

Run source code (1/3)

Run source code (2/3)

Run source code (3/3)
Block Delimiter

- No begin/end statements
- No { / } statements
- Indentation

Import Modules

- Modules are libraries that can be called from other scripts.

Variable

- No explicit variable declaration
- Variables are created by assigning them a name and a value.

Object

- Everything in Python is an object.
  - Attributes
  - Methods

Useful functions - dir()

- dir() used to find out which names a module defines
**Useful function - help()**

- `help()`
  - used to invoke the built-in help system
  ```python
  def help():
      """Help on built-in function help.
      """
  ```

- Constant
  - A constant, in Python, is a variable whose value is initialized but not changed.

- Function
  - Doesn’t define a return datatype
  - All function start with `def`
  - The argument doesn’t specify a datatype.

- Typing: Strong VS Weak, Static VS Dynamic
  - Statically typed language
    - A language in which types are fixed at compile time. Most statically typed languages enforce this by requiring you to declare all variables with their types before using them. Java and C are statically typed languages.
  - Dynamically typed language
    - A language in which types are discovered at execution time, the opposite of statically typed. VBScript and Python are dynamically typed, because they figure out the type of a variable only when you first assign it a value.
  - Strongly typed language
    - A language in which types are always enforced. Java and Python are strongly typed. If you have an integer, you can't treat it like a string without explicitly converting it.
  - Weakly typed language
    - A language in which types may be ignored; the opposite of strongly typed. VBScript is weakly typed. In VBScript, you can concatenate the string '12' and the integer 3 to get the string '123'. You then treat that as the integer 123, all without any explicit conversion.
  - Python is a dynamically typed language and also a strongly typed language.

- List
  - List is An ordered group of items.
  - List items don’t have to be of the same type.
Defining a List

- range([start], stop, [step])
  -=[start, start + step, start + step *2, ...]
  (all items < stop)

List : Index, Slice

List : Add, Delete

List : Queue, Stack

Conditional Statements

- if
- else
- elif
  - it stands for "else if."

Python doesn't officially support "switch statement."

<table>
<thead>
<tr>
<th>operator</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>equal</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
</tr>
</tbody>
</table>
For Loops

```
for (int i = 0; i < 10; i++) {
    printf("Hello, World!
");
}
```

Python

```
for i in range(10):
    print("Hello, World!
");
```

- 65 -

While Loops

```
while (i < 10) {
    printf("Hello, World!
");
    i++;
}
```

Python

```
while i < 10:
    print("Hello, World!
");
    i += 1
```

- 67 -

Class

Data and Function - together

```
class MyClass:
    def __init__(self):
        self.data = "init data"

    def function(self):
        print("Hello, World!
");
```

Class Inheritance

```
class Subclass(MyClass):
    def __init__(self):
        super().__init__()
```

- 69 -

Do ~ While Loops

Python doesn’t support do ~ while statement.

```
while i < 10:
    print("Hello, World!
");
    i += 1
```

- 68 -
Recommended Python Online Books

- Jump to Python (Korean)
  - http://wikidocs.net/mybook/read/page?pageid=1
- Dive into Python (English)
  - http://diveintopython.org/
- Dive into Python (Korean translation)
  - http://coreapython.hosting.paran.com/dive

Example #1

"thread"

- start_new_thread(FunctionName, Args)
- Global variable
- Local variable

Multi-threaded Programming

forces to wait using the time module sleep
Example #2

Expectation: \( \text{GlobalCount} = 0 + 2 \times 10 = 20 \)

-- 77 --

Example #2

Safety hazards

-- 78 --

Load Count(=9)

\[ 9 + 1 \rightarrow 10 \]

Thread A

Store Count(=10)

Load Count(=10)

\[ 10 + 1 \rightarrow 11 \]

Thread B

Store Count(=11)

Lock Object

- A synchronization primitive
- Not owned by a particular thread when locked
- `thread allocate_lock()` - Create a new lock object
- Usage of lock object
  - `acquire()` - Acquire a lock
  - `Critical Section` - codes between `acquire()` and `release()`
  - `release()` - Release a lock

-- 79 --

Example #3

-- 80 --

Example #3

-- 81 --

Example #3

-- 82 --
Example #3-2

```python
thread.start_new_thread()
```

Example #3-2

```
Main Thread is finished before other threads haven't finished executing.
```

Example #4

```
"threading"

- Higher-level threading interface
- class Thread - A class that represents a thread of control.
  - start() - Start the thread's activity.
  - run() - Method representing the thread's activity.
    - override this method in a subclass
  - join() - Wait until the thread terminates
    - A thread cannot join itself because this would cause a deadlock.
```
### Condition Object

- **Methods**
  - `acquire()` - Acquire the underlying lock.
  - `release()` - Release the underlying lock.
  - `wait()` - Wait until notified or until a timeout occurs.
  - `notify()` - Wakes up one of the threads waiting for the condition variable
  - `notifyAll()` - Wake up all threads waiting on this condition
Semaphore Object

- One of the oldest synchronization primitives
- Methods
  - `acquire()`
    - Acquire a semaphore
    - `internal counter > 0`
      - Decrementing it by one
    - `wait`
  - `release()`
    - Release a semaphore
    - Incrementing the internal counter by one

Event Object

- One of the simplest mechanisms for communication between threads
- One thread signals an event and other threads wait for it