Pemrograman Paralel dengan PThread

Kuliah#5 TSK205 Sistem Digital - TA 2011/2012

Eko Didik Widianto

Teknik Sistem Komputer - Universitas Diponegoro

Tentang Kuliah

- Sebelumnya dibahas tentang:
  - Desain model pemrograman di atas mesin memori bersama (misalnya SMP/symetric multiprocessor) menggunakan thread
    - Posix Thread/Pthread: menggunakan pustaka untuk paralelisme
    - OpenMP: menggunakan compiler directive untuk paralelisme
- Dalam kuliah ini, akan dibahas pemrograman menggunakan pustaka Posix Thread
  - Pustaka Pthread
  - API Pthread
  - Mengkompile program multithread
  - Mengelola Thread
  - Variabel Mutex
  - Variabel Kondisional

Kompetensi Dasar

- Setelah mempelajari bab ini, mahasiswa akan mampu:
  1. [C3] Mahasiswa akan mampu mengaplikasikan rutin-rutin pustaka pthread untuk memprogram paralel berbasis thread
  2. [C4] Mahasiswa akan mampu memprogram suatu aplikasi berbasis pthread sesuai dengan kebutuhan desain
- Acknowledment
  Materi dan gambar didapat dari:
  - POSIX Threads Programming di `https://computing.llnl.gov/tutorials/pthreads/`
- Link
  - Website: `http://didik.blog.undip.ac.id/2012/02/25/kuliah-tsk-617-pengolahan-paralel-2011/`
  - Email: didik@undip.ac.id

Bahasan

Bahasan

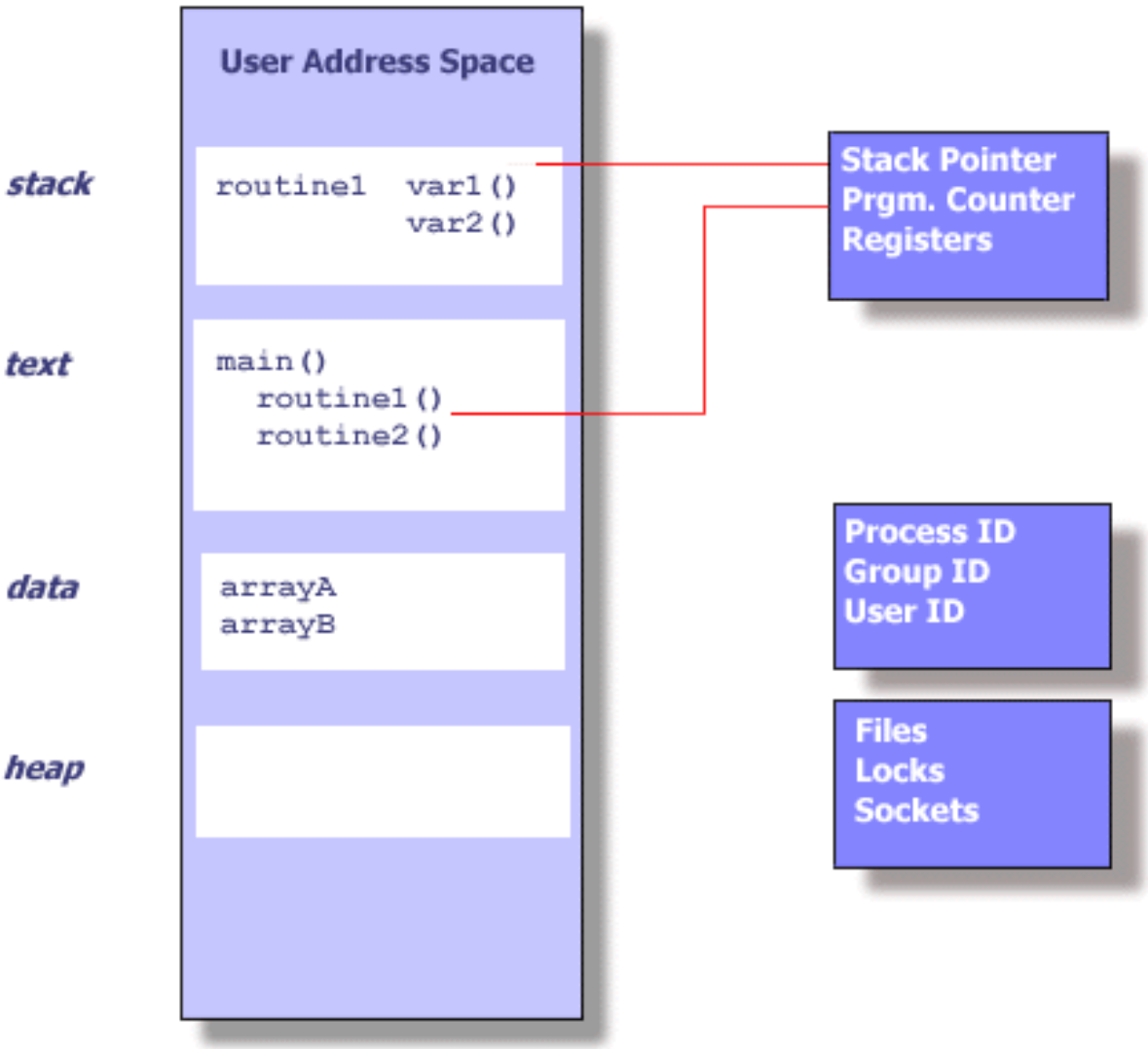
Apa itu Thread?

- Secara teknis, sebuah thread didefinisikan sebagai satu aliran instruksi yang independen yang dapat dijalankan secara terjadwal oleh OS
  - Dapat dibayangkan seperti konsep "procedure" yang dapat berjalan independen dari program utamanya
  - Seperti program utama (a.out) yang berisi sejumlah prosedur/fungsi. Kemudian semua prosedur dapat dijadwal untuk dijalankan secara simultan dan/atau independen oleh OS
    - Ini menggambarkan program “multi-thread”
    - Bagaimana menjalankannya secara simultan?

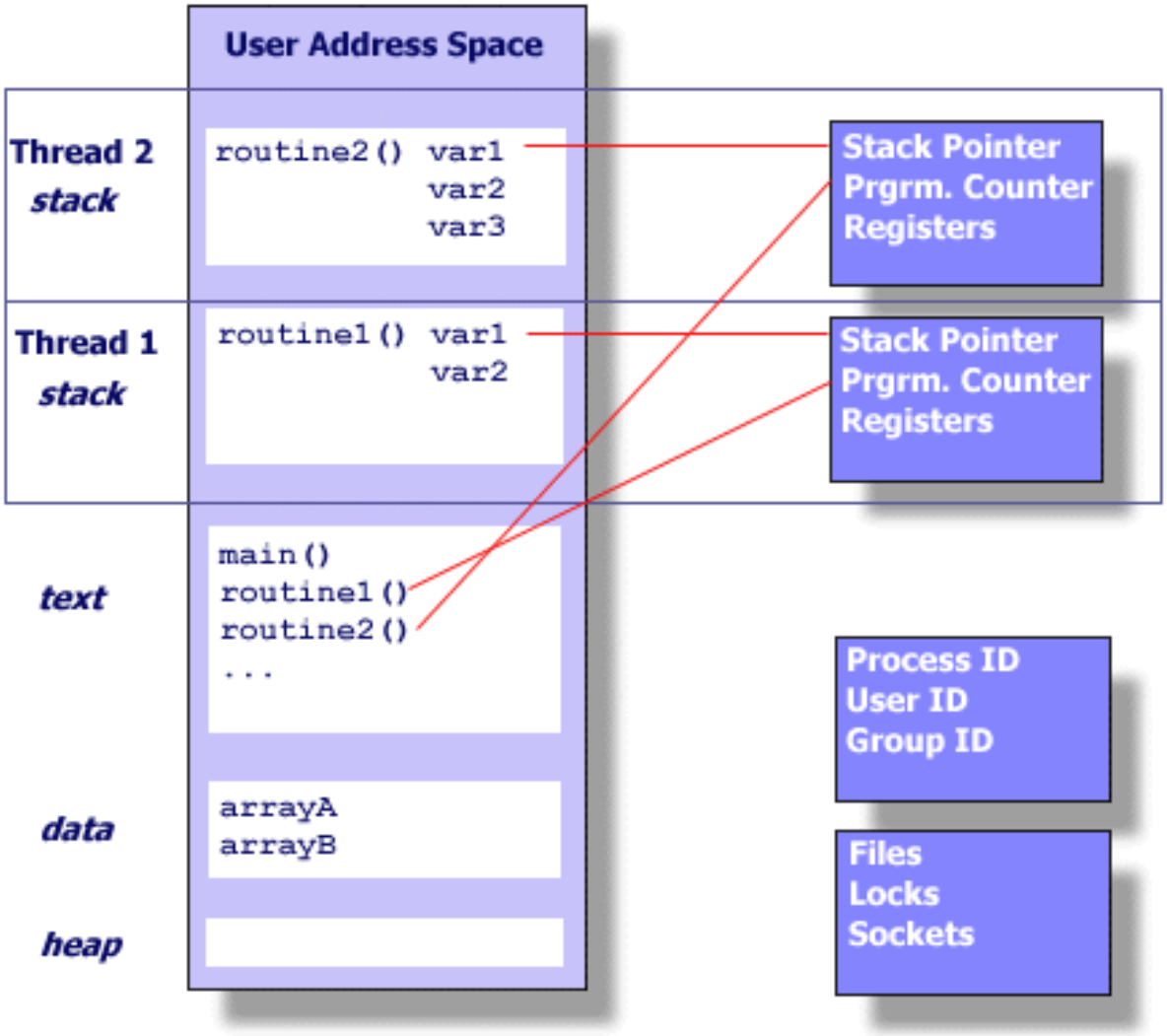
Process

- Sebuah proses dibuat oleh OS dan memerlukan **overhead** untuk menyimpan informasi tentang resource dan state eksekusi program
  - Process ID, process group ID, user ID, and group ID
  - Environment
  - Working directory
  - Program instructions
  - Registers
  - Stack
  - Heap
  - File descriptors
  - Signal actions
  - Shared libraries
  - Inter-process communication tools (such as message queues, pipes, semaphores, or shared memory)

UNIX Process



Thread dalam Sebuah Proses UNIX



Thread

- Exists **within a process** and **uses** the process resources
  - Changes made by one thread to shared system resources will be seen by all other threads
    - such as closing a file
  - Two pointers having the same value point to the same data
  - Reading and writing to the same memory locations is possible, and therefore requires **explicit synchronization** by the programmer
- Has its own independent flow of control as long as its parent process exists
  - Stack pointer, registers, scheduling properties (such as policy or priority), set of pending and blocked signals, thread specific data.
- Duplicates only the essential resources it needs to be independently schedulable
- May share the process resources with other threads that act equally independently (and dependently)
- **Dies if** the parent process dies - or something similar
- Is "lightweight" because most of the overhead has already been accomplished through the creation of its process

Bahasan

POSIX Thread

- POSIX: Portable Operating System Interface for UNIX
  - IEEE Std 1003.1c-1995 mendefinisikan API untuk membuat dan memanipulasi thread
  - System call untuk membuat dan mensinkronkan thread
  - Implementasi API: FreeBSD, NetBSD, GNU/Linux, Mac OS X dan Solaris
  - Windows? pthreads-w32 (third party)
- Support PThread
  - Membuat program paralel
  - Sinkronisasi
  - Memori shared implisit, pointer ke data shared diberikan ke suatu thread
- Implementasi: *pthread.h, libpthread*
- Some useful links:
  - standards.ieee.org/findstds/standard/1003.1-2008.html
  - www.opengroup.org/austin/papers/posix_faq.html
  - www.unix.org/version3/ieee_std.html

Mengapa PThread?

- The primary motivation is to realize potential program performance gains
  - When compared to the cost of creating and managing a process, a thread consumes much less operating system overhead
  - Managing threads requires fewer system resources than managing processes
- Function:
  - ***fork()*** subroutine: creating a new process
  - ***pthread_create()*** subroutine: creating a new thread

Fork vs Thread

- All threads within a process share the same address space
- Inter-thread communication is more efficient and easier to use than inter-process communication

Platform	fork()			pthread_create()		
	real	user	sys	real	user	sys
Intel 2.8 GHz Xeon 5660 (12cpus/node)	4.4	0.4	4.3	0.7	0.2	0.5
AMD 2.3 GHz Opteron (16cpus/node)	12.5	1.0	12.5	1.2	0.2	1.3
AMD 2.4 GHz Opteron (8cpus/node)	17.6	2.2	15.7	1.4	0.3	1.3
IBM 4.0 GHz POWER6 (8cpus/node)	9.5	0.6	8.8	1.6	0.1	0.4
IBM 1.9 GHz POWER5 p5-575 (8cpus/node)	64.2	30.7	27.6	1.7	0.6	1.1
IBM 1.5 GHz POWER4 (8cpus/node)	104.5	48.6	47.2	2.1	1.0	1.5
INTEL 2.4 GHz Xeon (2 cpus/node)	54.9	1.5	20.8	1.6	0.7	0.9
INTEL 1.4 GHz Itanium2 (4 cpus/node)	54.5	1.1	22.2	2.0	1.2	0.6

Timings reflect 50,000 process/thread creations

Aplikasi Single-Thread vs Multi-Thread

- Offers potential performance gains and practical advantages
  - Overlapping CPU work with I/O
    - For example, a program may have sections where it is performing a long I/O operation. While one thread is waiting for an I/O system call to complete, CPU intensive work can be performed by other threads
  - Priority/real-time scheduling: tasks which are more important can be scheduled to supersede or interrupt lower priority tasks
  - Asynchronous event handling: tasks which service events of indeterminate frequency and duration can be interleaved
    - For example, a web server can both transfer data from previous requests and manage the arrival of new requests

Pthread di Komputer SMP

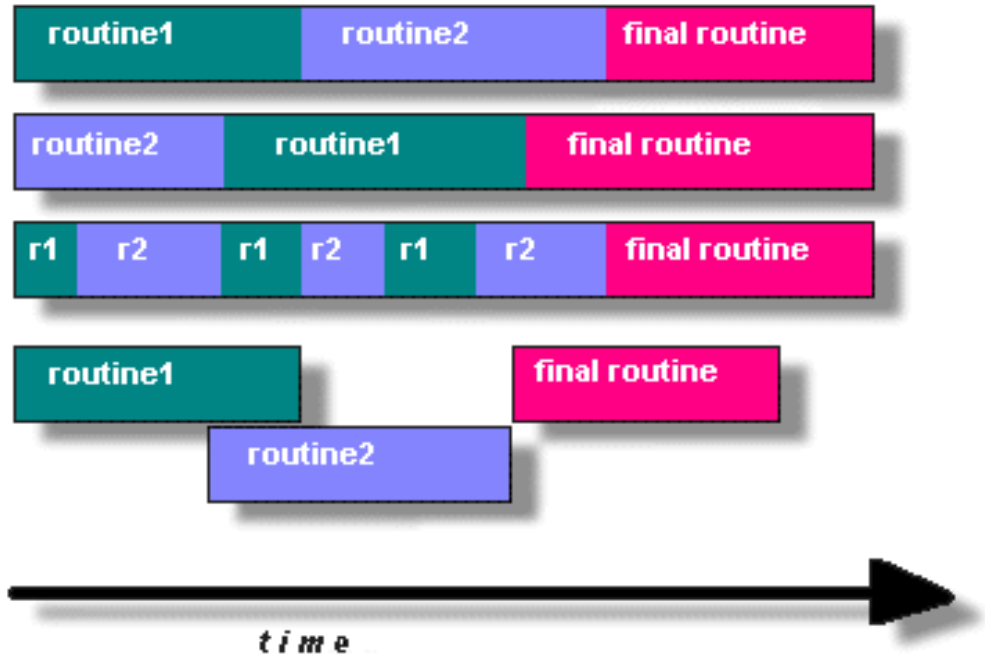
- To achieve optimum performance
  - Performance of an application can be greatly improved by using Pthread on-node data transfer/communication, instead of using MPI
    - MPI libraries usually implement on-node task communication via shared memory, which involves at least one memory copy operation (process to process)
    - For Pthreads there is no intermediate memory copy required because threads share the same address space within a single process
      It becomes more of a cache-to-CPU or memory-to-CPU bandwidth (worst case) situation. These speeds are much higher

Perbandingan Pthread vs MPI di Komputer SMP

Platform	MPI Shared Memory Bandwidth (GB/sec)	Pthreads Worst Case Memory-to-CPU Bandwidth (GB/sec)
Intel 2.8 GHz Xeon 5660	5.6	32
AMD 2.3 GHz Opteron	1.8	5.3
AMD 2.4 GHz Opteron	1.2	5.3
IBM 1.9 GHz POWER5 p5-575	4.1	16
IBM 1.5 GHz POWER4	2.1	4
Intel 2.4 GHz Xeon	0.3	4.3
Intel 1.4 GHz Itanium 2	1.8	6.4

Desain Program Multithread

- On modern, multi-cpu machines, pthreads are ideally suited for parallel programming
  - whatever applies to parallel programming in general, applies to parallel pthreads programs
- In order for a program to take advantage of Pthreads, it must be able to be organized into discrete, independent tasks which can execute **concurrently**
  - Routines can be interchanged, interleaved and/or overlapped in real time



Program yang Akan Memerlukan Pthread

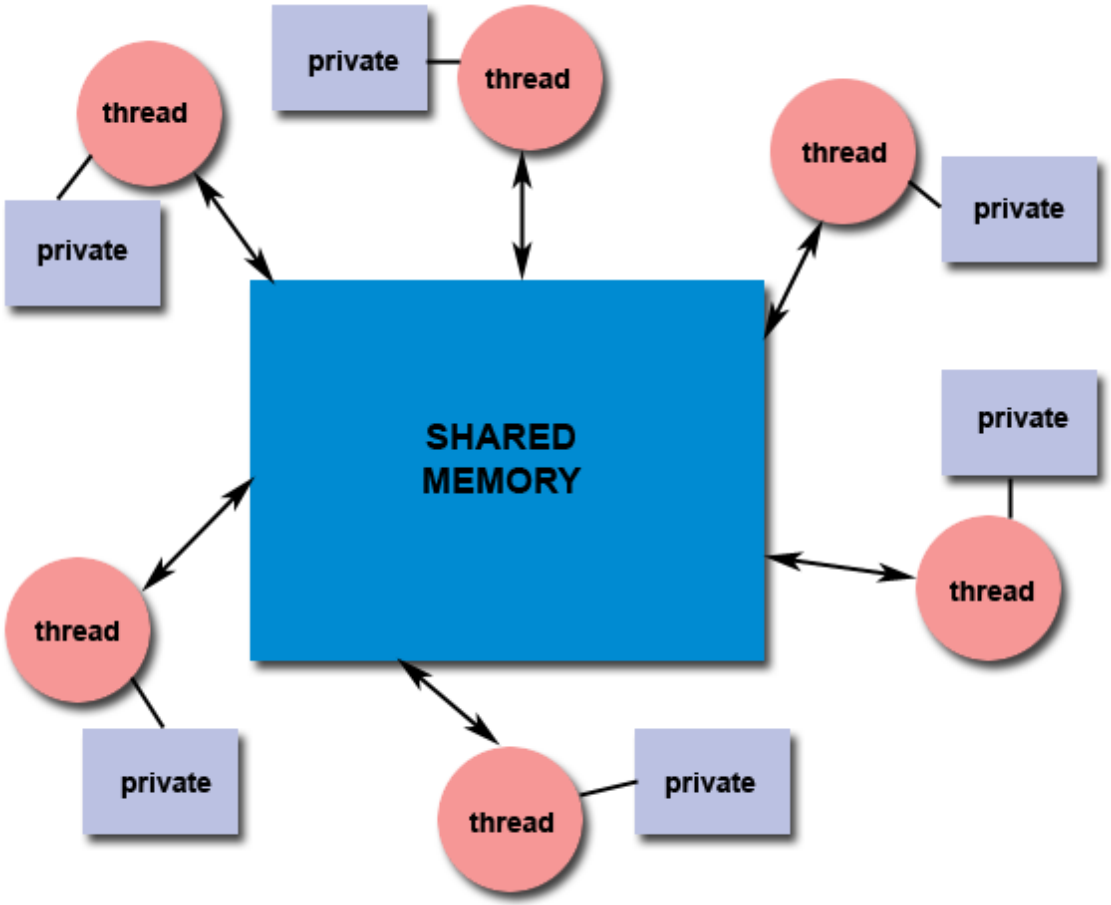
- Programs having the following characteristics may be well suited for pthreads:
  - Work that can be executed, or data that can be operated on, by multiple tasks simultaneously
  - Block for potentially long I/O waits
  - Use many CPU cycles in some places but not others
  - Must respond to asynchronous events
  - Some work is more important than other work (priority interrupts)

Model Program Multithread

- **Manager/worker**: a single thread, the manager assigns work to other threads, the workers
  - Typically, the manager handles all input and parcels out work to the other tasks.
  - At least two forms of the manager/worker model are common: static worker pool and dynamic worker pool
- **Pipeline**: a task is broken into a series of suboperations, each of which is handled in series, but concurrently, by a different thread. An automobile assembly line best describes this model.
- **Peer**: similar to the manager/worker model, but after the main thread creates other threads, it participates in the work

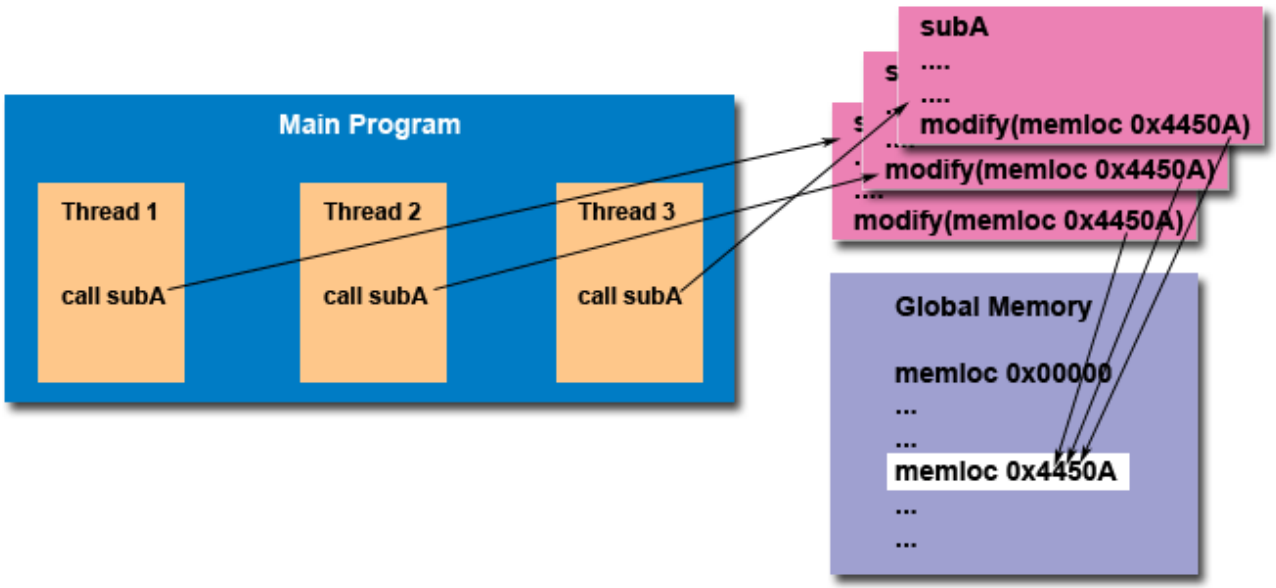
Model Memori Bersama

- All threads have access to the same global, shared memory
- Threads also have their own private data
- Programmers are responsible for **synchronizing access (protecting)** globally shared data



Thread-safeness

- Thread-safeness: refers an application's ability to execute multiple threads simultaneously without "clobbering" shared data or creating "race" conditions.
  - If the routine does not employ synchronization constructs to prevent data corruption, then it is not thread-safe



Bahasan

API PThread

- Pthreads API was defined in the ANSI/IEEE POSIX 1003.1 - 1995 standard
- The subroutines which comprise the Pthreads API can be informally grouped into:
  1. **Thread management**: Routines that work directly on threads - creating, detaching, joining, etc
     - Also include functions to set/query thread attributes (joinable, scheduling etc.)
  2. **Mutexes**: Routines that deal with synchronization, called a "mutex", which is an abbreviation for "mutual exclusion"
     - Provide for creating, destroying, locking and unlocking mutexes
     - Supplemented by mutex attribute functions that set or modify attributes associated with mutexes
  3. **Condition variables**: Routines that address communications between threads that share a mutex.
     - Based upon programmer specified conditions
     - Includes functions to create, destroy, wait and signal based upon specified variable values. Functions to set/query condition variable attributes are also included
  4. **Synchronization**: Routines that manage read/write locks and barriers

API PThread

Routine Prefix	Functional Group
pthread_	Threads themselves and miscellaneous subroutines
pthread_attr_	Thread attributes objects
pthread_mutex_	Mutexes
pthread_mutexattr_	Mutex attributes objects.
pthread_cond_	Condition variables
pthread_condattr_	Condition attributes objects
pthread_key_	Thread-specific data keys
pthread_rwlock_	Read/write locks
pthread_barrier_	Synchronization barriers

Kompilasi dan Linking

Compiler / Platform	Compiler Command	Description
INTEL Linux	`icc -pthread`	C
	`icpc -pthread`	C++
PathScale Linux	`pathcc -pthread`	C
	`pathCC -pthread`	C++
PGI Linux	`pgcc -lpthread`	C
	`pgCC -lpthread`	C++
GNU Linux, BG/L, BG/P	`gcc -pthread`	GNU C
	`g++ -pthread`	GNU C++
IBM BG/L and BG/P	`bgxlc_r / bgcc_r`	C (ANSI / non-ANSI)
	`bgxlC_r, bgxlc++_r`	C++

Bahasan

Rutin

- *pthread_create (thread,attr,start_routine,arg)*
- *pthread_exit (status)*
- *pthread_cancel (thread)*
- *pthread_attr_init (attr)*
- *pthread_attr_destroy (attr)*

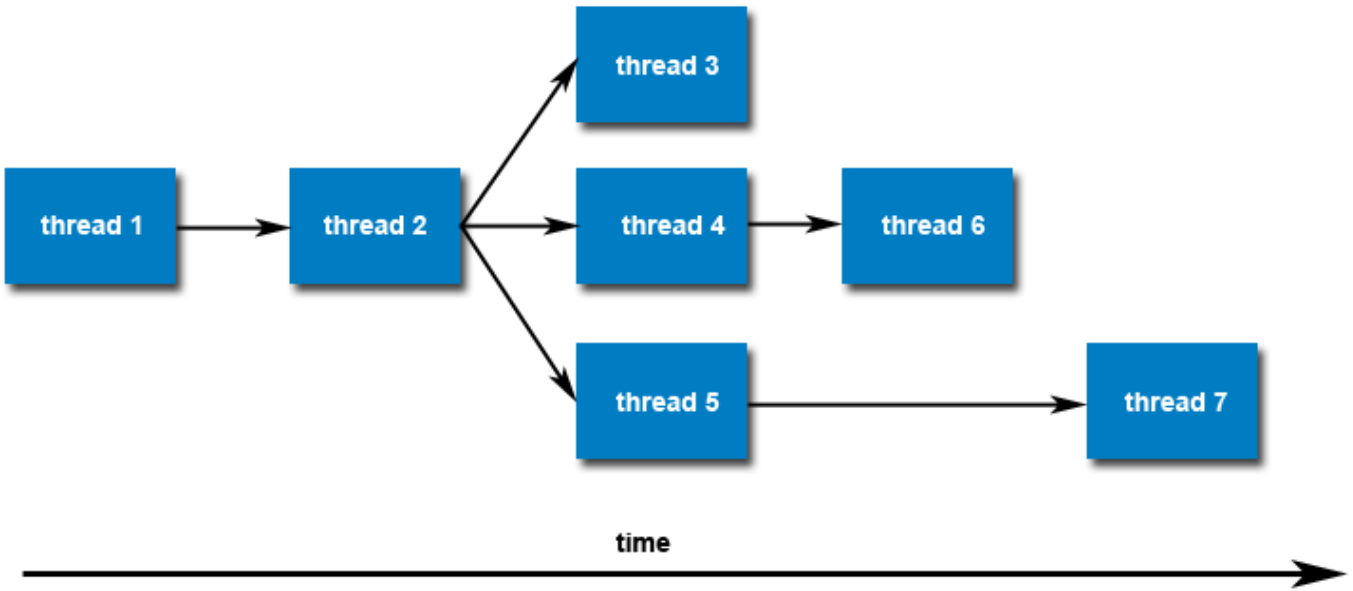
Membuat Thread

Rutin: pthread_create (thread,attr,start_routine,arg)

- Argument:
  - *thread*: An opaque, unique identifier for the new thread returned by the subroutine
  - *attr*: An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values
  - *start_routine*: the C routine that the thread will execute once it is created
  - *arg*: A single argument that may be passed to start_routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed
- Return: *thread_id*

Membuat Thread

- Once created, threads are peers, and may create other threads



Atribut Thread

- Can be set when creating a thread
- Or initialize thread attribute in main thread: *pthread_attr_init* and *pthread_attr_destroy*
- Attributes:
  - Detached or joinable state
  - Scheduling inheritance
  - Scheduling policy
  - Scheduling parameters
  - Scheduling contention scope
  - Stack size
  - Stack address
  - Stack guard (overflow) size

Terminasi Thread

Routine: pthread_exit (status)

- A thread may be terminated:
  - The thread returns normally from its starting routine. It's work is done
  - The thread makes a call to the pthread_exit subroutine - whether its work is done or not
  - The thread is canceled by another thread via the pthread_cancel routine
  - The entire process is terminated due to making a call to either the exec() or exit()
  - If main() finishes first, without calling pthread_exit explicitly itself
- Optional termination status can be specified
  - Typically returned to threads joining the terminated thread
- Cleanup: the pthread_exit() doesnot close files
  - Any files opened inside the thread will remain open after the thread is terminated

Contoh Kode: Pembuatan dan Terminasi Thread

See code

Bahasan

Passing Argumen ke Thread

See code

Bahasan

Joining dan Detaching Thread

See code

Umpan Balik

- Yang telah kita pelajari hari ini:
  - tbd
- Yang akan kita pelajari di pertemuan berikutnya adalah <tbd>
  - Pelajari: tbd

Lisensi

Creative Common Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)

- Anda bebas:
  - untuk **Membagikan** — untuk menyalin, mendistribusikan, dan menyebarkan karya, dan
  - untuk **Remix** — untuk mengadaptasikan karya
- Di bawah persyaratan berikut:
  - **Atribusi** — Anda harus memberikan atribusi karya sesuai dengan cara-cara yang diminta oleh pembuat karya tersebut atau pihak yang mengeluarkan lisensi.
  - **Pembagian Serupa** — Jika Anda mengubah, menambah, atau membuat karya lain menggunakan karya ini, Anda hanya boleh menyebarkan karya tersebut hanya dengan lisensi yang sama, serupa, atau kompatibel.
- Lihat: Creative Commons Attribution-ShareAlike 3.0 Unported License