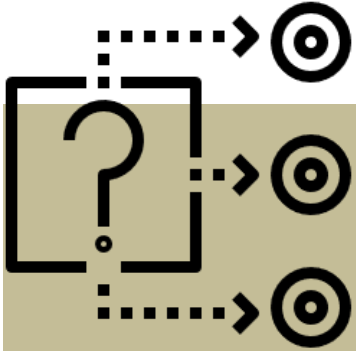
线程

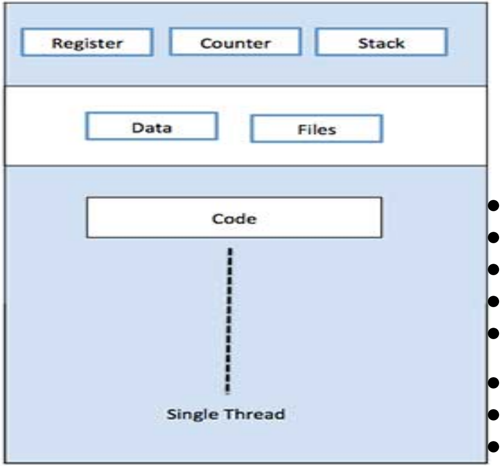
THREADS AND SYNCHRONIZATION 同步

DCS4103 Operating System



Threads

Single Thread



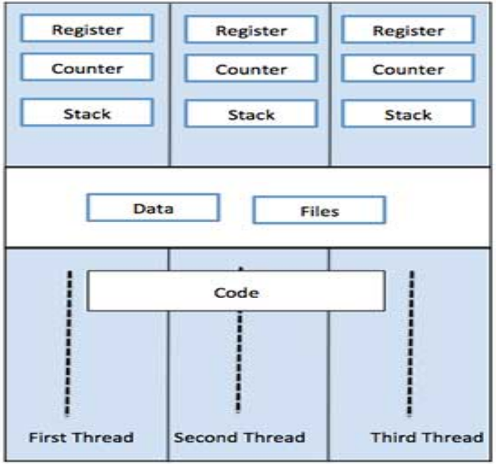
Single Process P with single thread

Thread

A path 路径 of execution 执行 through a program's code, plus a set of resources 资源 which assigned 分配 by the operating system

- Each thread belongs to exactly one process :
- Code
- Counter (next instruction)
- Registers (current variables)
- Stack (execution history)
- Application :
- Network servers
- Web server
- Shared memory multiprocessors

Multi-Threading



Single Process P with three threads

Multi-Threading

Dividing a process into multiple threads
Each thread has a private stack
All threads of a process share the code, the global data and heap
Application
- Browser (multiple tabs)
- MS Word (formatting text, process input)

Process and Thread

Process	Thread
Heavy weight or resource intensive.	Light weight, taking lesser resources
Switching needs interaction with operating system.	Switching does not need to interact with operating system.
In multiple processing environments, each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.

Threads

Advantages of Threads

- Responsiveness 响应能力 — allow program continue running if part of its blocked
- Faster context switch
- Effective utilization 利用 of multiprocessor system
- Resource sharing — share data, code, files
- Communication
- Enhanced throughout of the system

Types of Thread

User Level

- OS doesn't recognize user level threads.
- Thread library contains code
  - Create and destroy threads
  - Pass message and data between threads
  - Schedule thread execution
  - Save and restore thread contexts
- Examples
  - Java thread
  - POSIX threads.

Advantages

- Not require Kernel mode privileges
- Run on any operating system
- Designed as dependent threads
- Fast to create and manage

Disadvantages

- Most system calls are blocking
- Multithreaded application cannot take advantage of multiprocessing.
- If one user level thread perform blocking operation, then entire process will be blocked.

Types of Thread

Kernel Level

- Known as lightweight process
- Operating System managed threads acting on kernel
- Performs thread creation, scheduling and management in Kernel space
- Maintains context information for the process
- Example
  - Window
  - Solaris

Advantages

- Simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.

Disadvantages

- Slower to create and manage
- Hardware support is needed.
- Implementation is complicated. 复杂
- Specific to the operating system

执行

Types of Thread

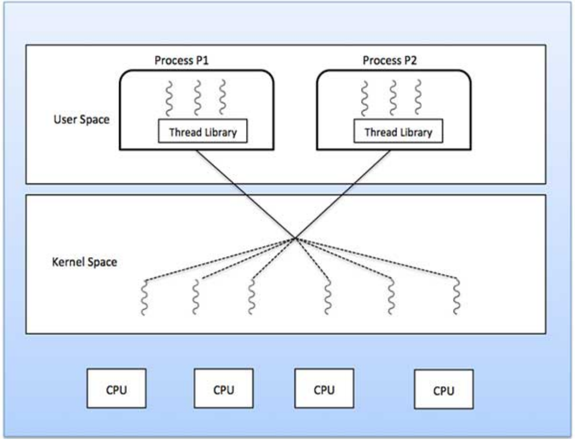
Combine

- Multiple threads within the same application can run in parallel on multiple processors
- Blocking system call need not block the entire process
- Example
  - Window
  - Solaris
- Categories into three models:
  - Many to many model
  - Many to one model
  - One to one model

Type of Thread

Combine (Many to Many Model)

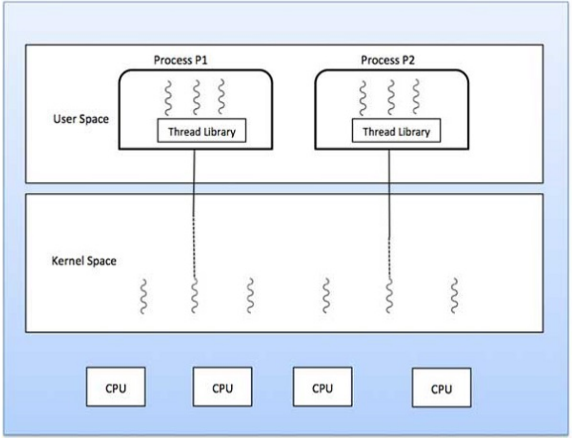
- Multiplexes any number of user threads onto an equal or smaller number of kernel threads
- Can create as many user threads.
- Can run in parallel on a multiprocessor machine
- When a thread performs a blocking system call, the kernel can schedule another thread for execution.
- Example :
- Unix
- Solaris 9



Type of Thread

Combine (Many to One Model)

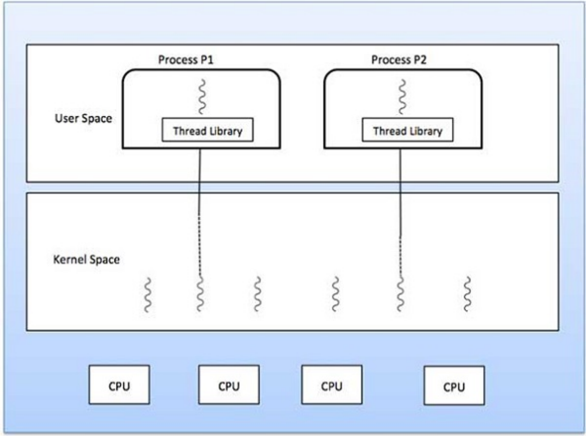
- Maps many user level threads to one Kernel-level thread
- Thread management is done in user space by the thread library
- When thread makes a blocking system call, the entire process will be blocked
- Only one thread can access the Kernel at a time
- Example :
- Solaris



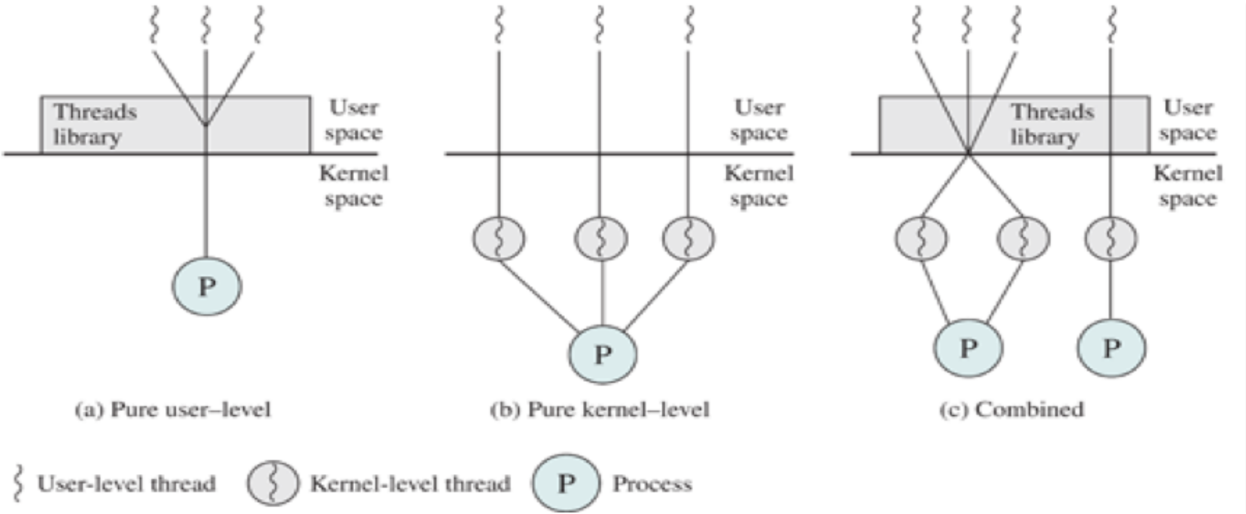
Type of Thread

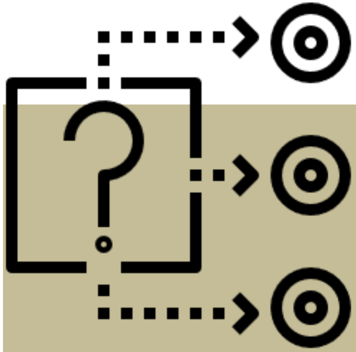
Combine (One to One Model)

- One-to-one relationship of user-level thread to the kernel-level thread
- Allows another thread to run when a thread makes a blocking system call
- Supports multiple threads to execute in parallel on microprocessors.
- Creating user thread requires the corresponding Kernel thread
- Example :
- Linux
- Windows 95 to XP



Type of Thread





Mutual Exclusion and Synchronization

同步 Process Synchronization

The task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources

Independent Process

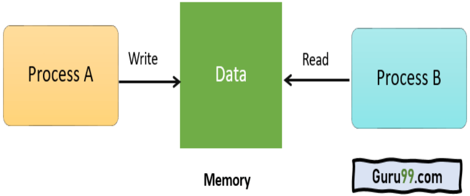
- Execution of one process does not affects the execution of other processes.

Cooperative Process

- Execution of one process affects the execution of other processes.

Race condition :

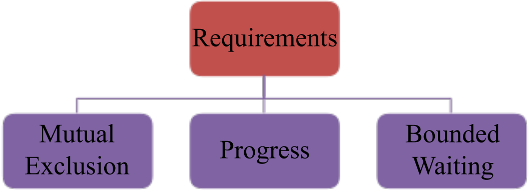
- Two or more threads try to read, write
- Make the decisions based on the memory access concurrently



Critical Section Problem

Critical Section

- The regions of a program that try to access shared resources
  - Memory location
  - Data structure
  - CPU
  - IO device
- Only one process at a time can execute within the critical section.
- Ensure that the Race condition among the processes will never arise.

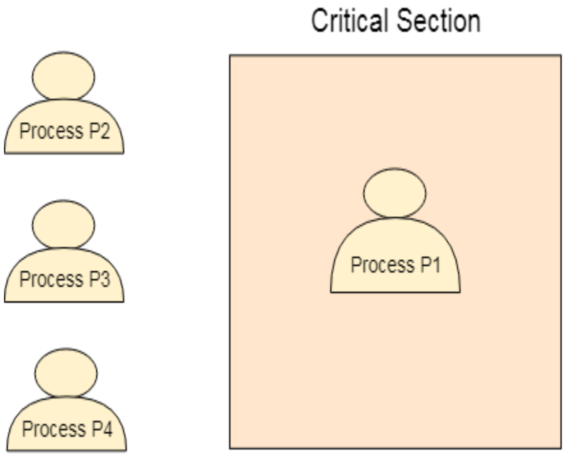


Synchronization Requirements

互斥

Mutual Exclusion

- If one process is executing inside critical section, then the other process must not enter in the critical section.
- Used for controlling access to the shared resource
- Include a priority mechanism



Synchronization Requirements

Progress

- If one process doesn't need to execute into critical section, then it should not stop other processes to get into the critical section.

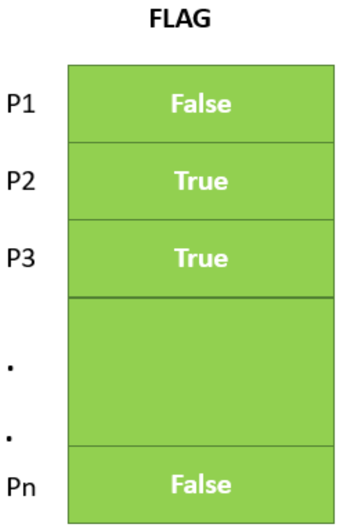
Bound Waiting

- When a process makes a request for getting into critical section
- There is a specific limit about number of processes can get into their critical section.
- When the limit is reached, the system must allow request to the process to get into its critical section

Solution to Critical Section Problem

Peterson's Algorithm

- Widely used
- When a process is executing in a critical state, then the other process only executes the rest of the code.
- Make sure that only a single process runs in the critical section at a specific time.
- Initialized to FALSE
- No one is interested in entering the critical section
- Satisfied all requirements



Solution to Critical Section Problem

Dekker's Algorithm

- If two processes attempt to enter a critical section at the same time
- The algorithm will allow only one process in, based on whose turn it is.
- If one process is already in the critical section, the other process will busy wait for the first process to exit
- Satisfied Mutual Exclusive

PROCESS A

```
While (TRUE)
{
   pAtrying = True;
   turn= B;
   While (pBtrying && turn ==B)
         do nothing;

   <critical section A>

   pAtrying = False;
}
```

PROCESS B

```
While (TRUE)
{
   pBtrying = True;
   turn= A;
   While (pAtrying && turn ==A)
         do nothing;

   <critical section A>

   pBtrying = False;
}
```

Solution to Critical Section Problem

Synchronization Hardware	• Lock functionality where a process acquires a lock when entering the Critical section and releases the lock after leaving it.
Mutex Locks	• In the entry section of code, a LOCK is obtained over the critical resources used inside the critical section. In the exit section that lock is released.
Semaphore	• A signaling mechanism and a thread that is waiting on a semaphore can be signaled by another thread • Wait and signal

Revision Questions

1. Distinguish between the single thread and multithreading
2. Define the process synchronization and its types
3. Evaluate the process synchronization requirements