|  |  |  |  |
| --- | --- | --- | --- |
| 1. Thread (multithreading) | | 圖示: | benefits |
| 又稱為lightweight process，是OS配置CPU time的對象 | | **Responsiveness**--process內，執行中的thread is blocked則CPU可以切給其他可用的thread，只要還有thread可以執行process就不會被blocked。   * 用於user interactive app可增加對user的回應度 |
| 同一process內  不同的threads  彼此共享 | 1. code section 2. data section 3. other OS resource |
| **resource sharing**--同一process內不同的threads彼此共享code section可有多個task同時執行 |
| 同一process內  threads個別私有的 | 1. PC 2. Stack 3. Cpu register 4. Thread ID 5. Thread state | **Economy**--同一process內不同的threads彼此共享memory及other OS resource私有的context不多 |
| **utilization of multiprocessor**—同一個process內不同thread可以平行在不同CPU上執行。(平行度增加) |
| 同一時間點有很多工作要做multithreading  一個時間點只有一個工作要做process |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Thread | lightweight process | OS配置CPU time的對象 | 管理成本低 | 同一process內不同的threads彼此共享memory及other OS resource | Context switching時間短 | Process內的single thread 若blocked整個process則blocked | Multiprocessor架構效益發揮差 |
| process | heavtweight process | OS配置resource的對象 | 管理成本高 | 不同process之間不共享memory及other OS resource(除了share memory) | Context switching時間長 | process內，只要還有thread可以執行就不會被blocked。 | Multiprocessor架構效益發揮較優 |

|  |  |  |  |
| --- | --- | --- | --- |
| Thread pool | 緣由 : 在client server model中當server收到client的要求server才建立thread去執行，雖然thread建立很快，但還是需耗費時間，對client的回應不夠快 | 解決法 : process事先建立一些thread放在Thread pool，當有client要求，可直接指配一條thread去執行，可快速回應 | 缺點 : 建立過多thread在Thread pool則system resource消耗嚴重  通常會限制 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名稱 | 定義 | 優點 | 缺點 | 範例 |
| user thread | Thread的create、Context switch等管理是由user process負責  (不需kernel介入，kernel不知道thread存在) | Thread的create、Context switch快且成本低 | 1. 如user thread發出blocking且kernel是single thread，則整個process block 2. multiprocessor架構效益發揮差 | 1. POSIX Pthread library 2. MACH Cthread 3. Sun Solaris2以上—green thread 4. UI thread library |
| Kernel thread | Thread的create、Context switch等管理是由Kernel process負責(kernel知道thread存在) | 1. 如user thread發出blocking且kernel是single thread，則整個process不會block 2. multiprocessor架構效益發揮優 | Thread的create、Context switch慢且成本高 | 1. Windows NT 2. UNIX 3. Sun Solaris 2以上 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名稱 | 圖示 | 定義 | 優點 | 缺點 |
| Many to one |  | Many user thread to one kernel thread | fast thread create及Context switch  (因為所有thread的管理是在user space內做，很有效率) | 1. process內的執行thread若發出block，會導致整個process block 2. multiprocessor架構效益發揮差 |
| one to one |  | one user thread to one kernel thread | 1. 如user thread發出blocking且kernel是single thread，則整個process不會block 2. multiprocessor架構效益發揮優 | 1. Thread的create、Context switch慢且成本高 2. 有一條user thread就要建立一條kernel thread對應，user thread 生成數多，造成system overhead重 |
| Many to Many |  | Many user thread to a small or equal number kernel thread | 1. 如user thread發出blocking且kernel是single thread，則整個process不會block 2. multiprocessor架構效益發揮優 | 1. Thread的create、Context switch慢且成本高 2. 製作成本高較複雜 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Signal**—在UNIX一個signal被使用去通知process有特別的事件發生 | Signal種類 | Synchronous signal—process自己引起的 | Signal傳送在multithreading | 1. | 2. | 3. | 4. |
| Asynchronous signal—其他process引起，造成終止 |