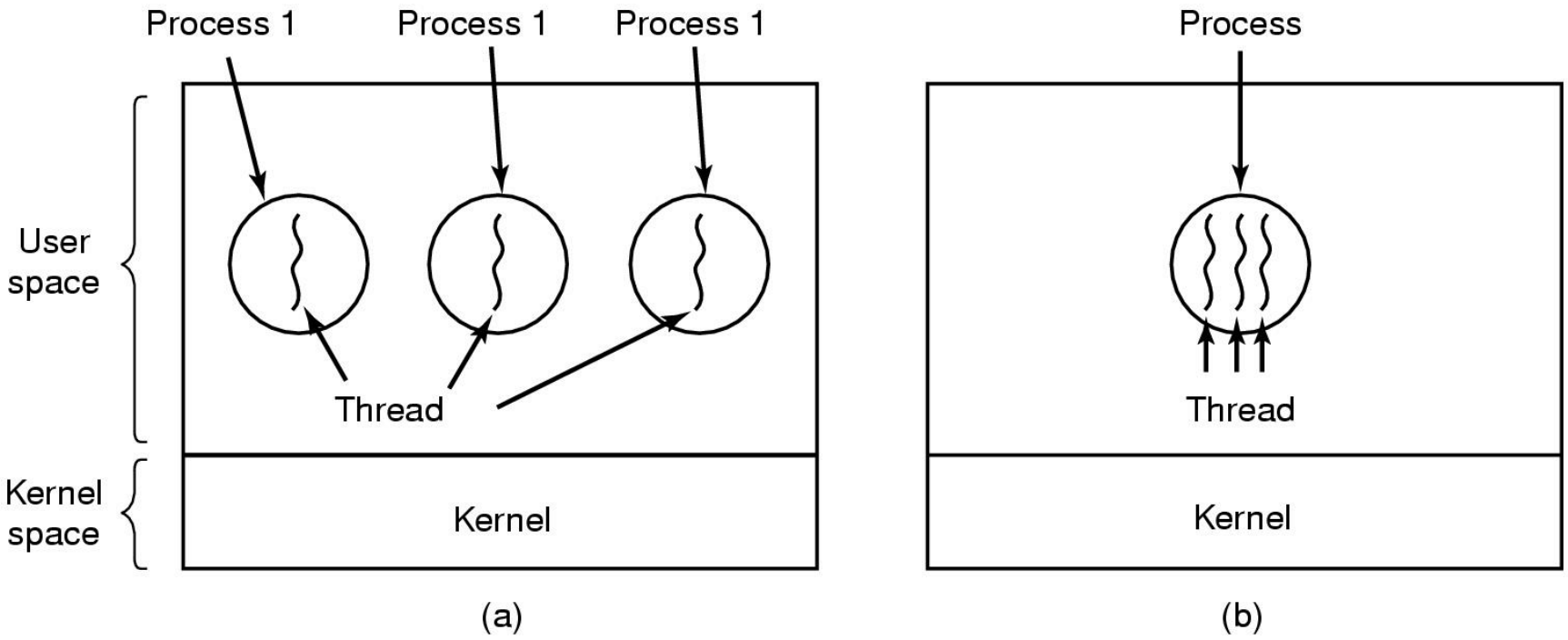
SISTEM OPERASI

Departemen Ilmu Komputer/ Informatika
Universitas Diponegoro
Semester Gasal 2018/ 2019

Process & Thread

- In traditional operating systems, each process has **an address space** and **a single thread of control**. In fact, that is almost the **definition of a process**.
- Nevertheless, there are frequently situations in which it is desirable to have multiple threads of control in the same address space running in quasi-parallel, as though they were (almost) separate processes (except for the shared address space).

Process & Thread



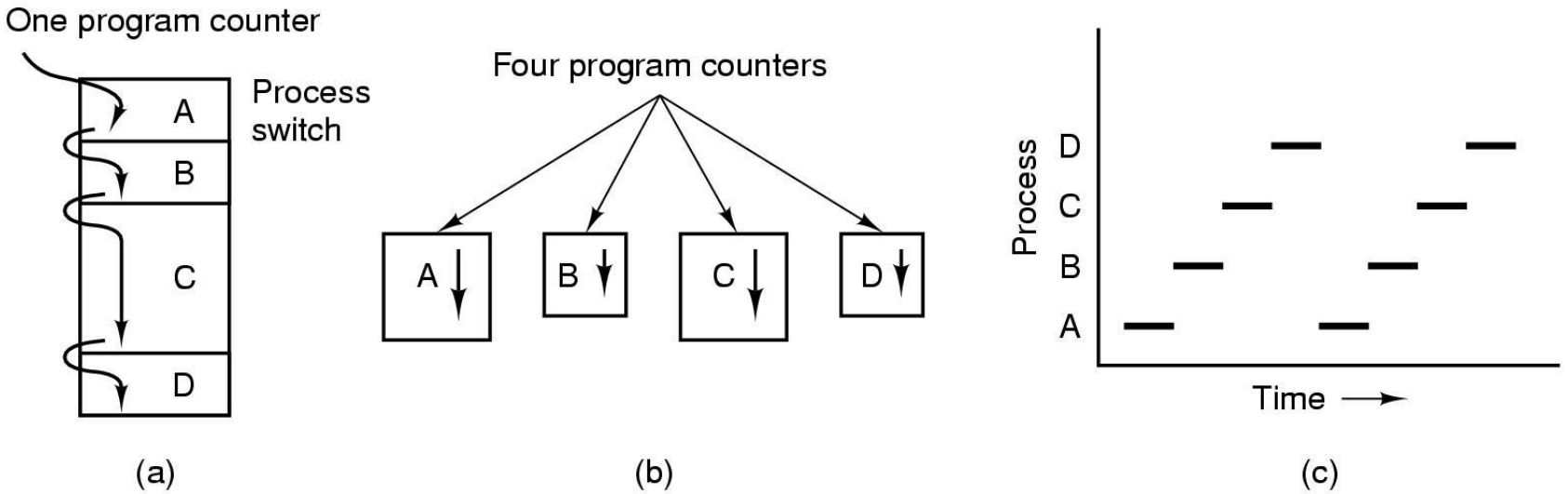
(a) Three processes each with one thread

(b) One process with three threads (threads share address space)

Agenda

1. Thread Model
2. Thread Usage
3. User Level Thread (ULT)
4. Kernel Level Thread (KLT)
5. Hybrid Implementation

Multiprogramming vs Multithreading ?



Process & Thread

Single-threading ← traditional process

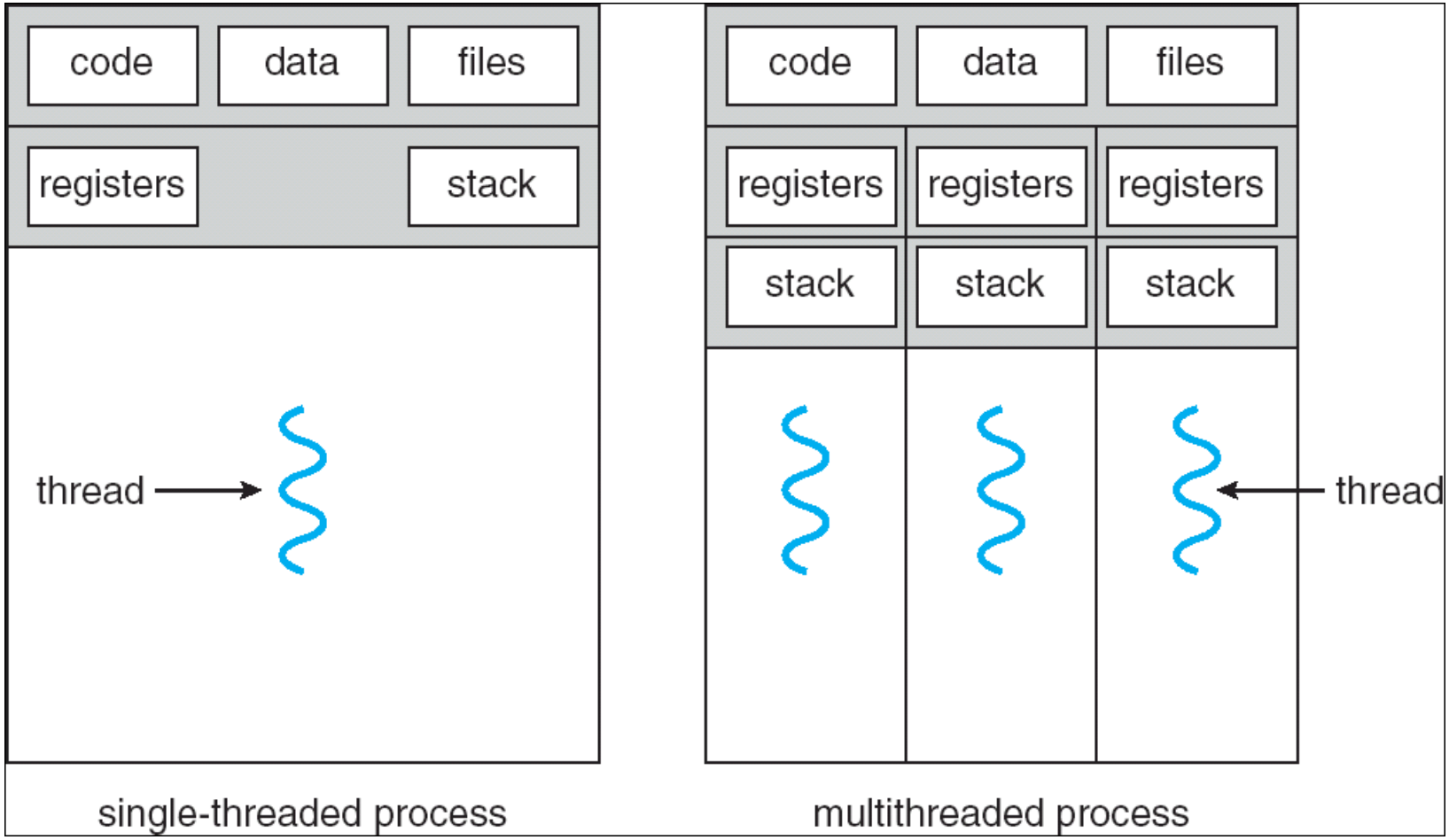
Process = **resource grouping** (code, data, open files, etc.) + **execution** (program counter, registers, stack)

Multi-threading:

- multiple **execution** takes place in the same process environment
- co-operation by **sharing resources** (address space, open files, etc.)

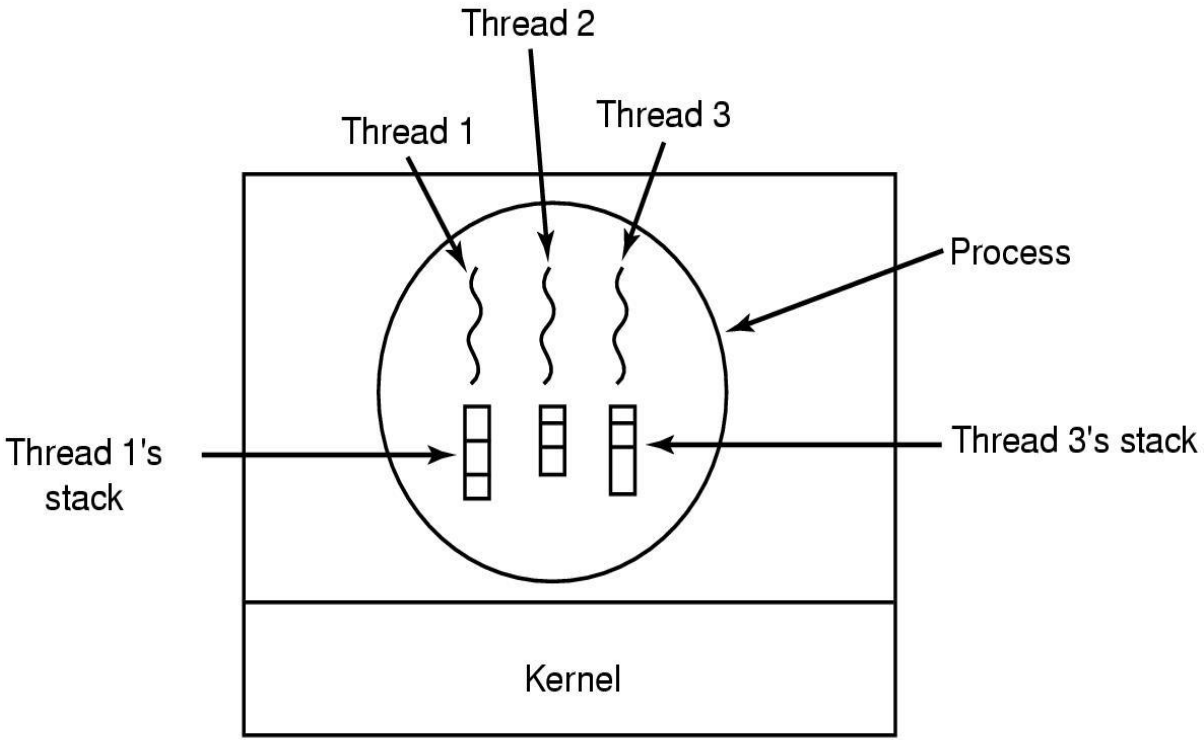
The Thread Model

by Silberschatz



The Thread Model

by Tanenbaum



Each thread has **its own stack** to keep track execution history (called procedures)

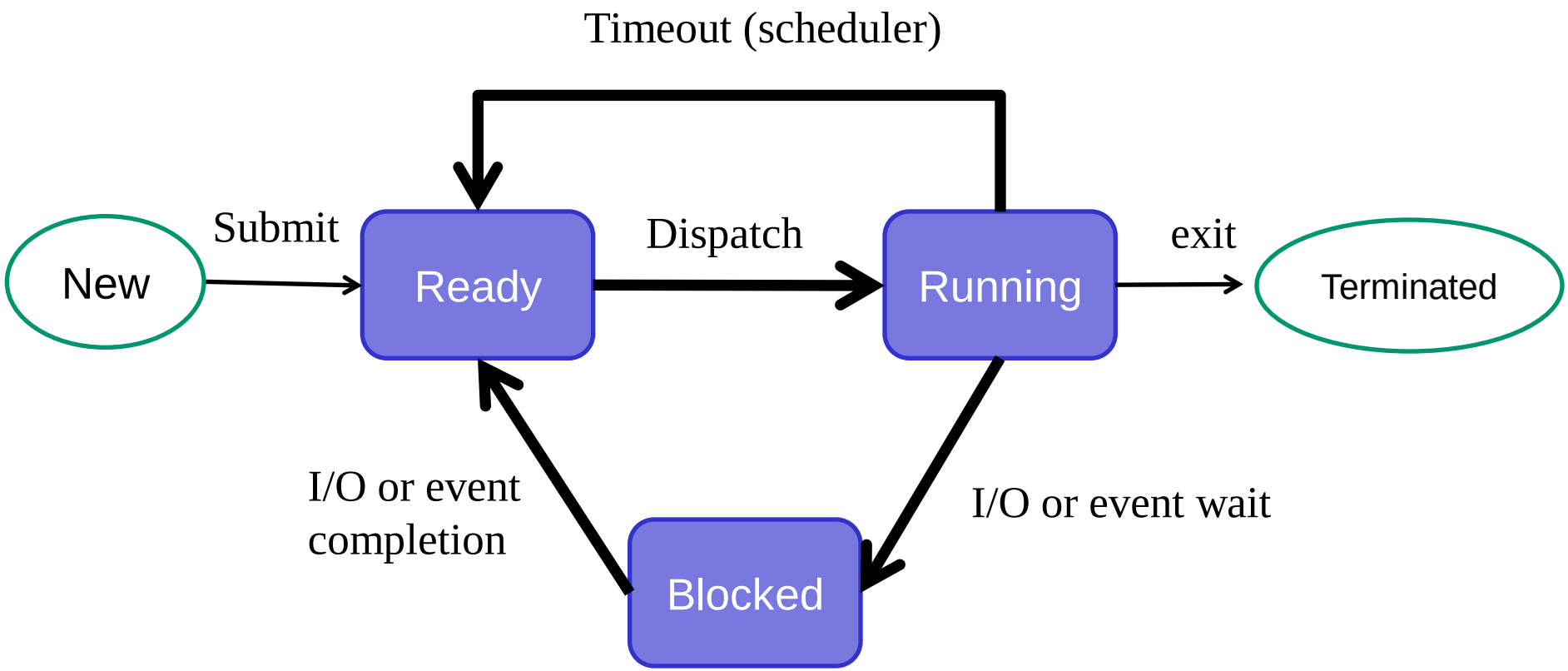
The Thread Model

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

What we are trying to achieve with the thread concept is the ability for multiple threads of execution to share a set of resources so that they can work together closely to per-form some task

The Thread Model

Threats lifecycle is Similar to The Process



How do threads work?

1. Start with one thread in a process
2. Thread contains (id, registers, attributes)
3. Use library call to create new threads and to use threads, eg:
   - **Thread_create** includes parameter indicating what procedure to run
   - **Thread_exit** causes thread to exit and disappear (can't schedule it)
   - **Thread_join** Thread blocks until another thread finishes its work
   - **Thread_yield** to let another thread run

POSIX Threads (Pthreads)

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

Portable Operating System Interface for UNIX (POSIX)

Pthreads are IEEE Unix standard library calls

A Pthreads example-"Hello,world"

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
        /* This function prints the thread's identifier and then exits. */
        printf("Hello World. Greetings from thread %d0, tid);
        pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
        /* The main program creates 10 threads and then exits. */
        pthread_t threads[NUMBER_OF_THREADS];
        int status, i;

        for(i=0; i < NUMBER_OF_THREADS; i++) {
                printf("Main here. Creating thread %d0, i);
                status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

. . .           if (status != 0) {
                        printf("Oops. pthread_create returned error code %d0, status);
                        exit(-1);
                }
        }
        exit(NULL);
}
```

Threads Complications

- If the parent process has multiple threads, should the child also have them?
- What happens if one thread closes a file while another one is still reading from it?

Agenda

1. Thread Model
2. **Thread Usage**
3. User Level Thread (ULT)
4. Kernel Level Thread (KLT)
5. Hybrid Implementation

Threads Advantages

1. Pseudo-parallelism with shared address space and data

- In many applications, multiple activities are going on at once.
- Some of these may block from time to time.
- Ability for the parallel entities to share an address space and all of its data among themselves.
- By decomposing such an application into multiple sequential threads that run in quasi-parallel, the programming model becomes simpler.

Pseudo-paralelism (2)

- We have seen this argument before. It is precisely the argument for having processes. Instead of thinking about interrupts, timers, and context switches, we can think about parallel processes.
- Only now with threads we add a new element: **the ability for the parallel entities to share an address space and all of its data among themselves.**
- This ability is essential for certain applications, which is why having multiple processes (with their separate address spaces) will not work.

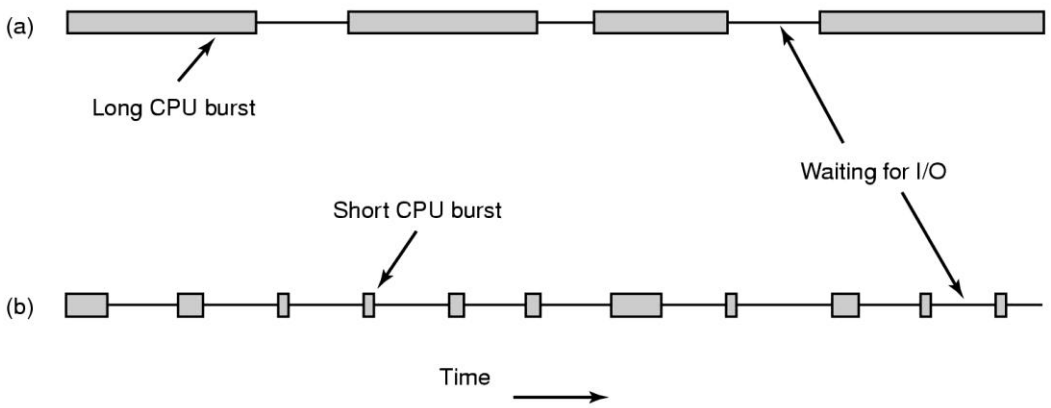
Threads Advantages

2. Easier to create and destroy than processes
   - since they do not have any resources attached to them
   - creating a thread goes 100 times faster than creating a process

Threads Advantages

3. Better performance for I/O bound applications

- Threads yield no performance gain when all of them are CPU bound,
- but when there is substantial computing and also substantial I/O, having threads allows these activities to overlap, thus speeding up the application.



Threads Advantages

4. Threads are useful on systems with multiple CPUs
   - Real paralellism is possible

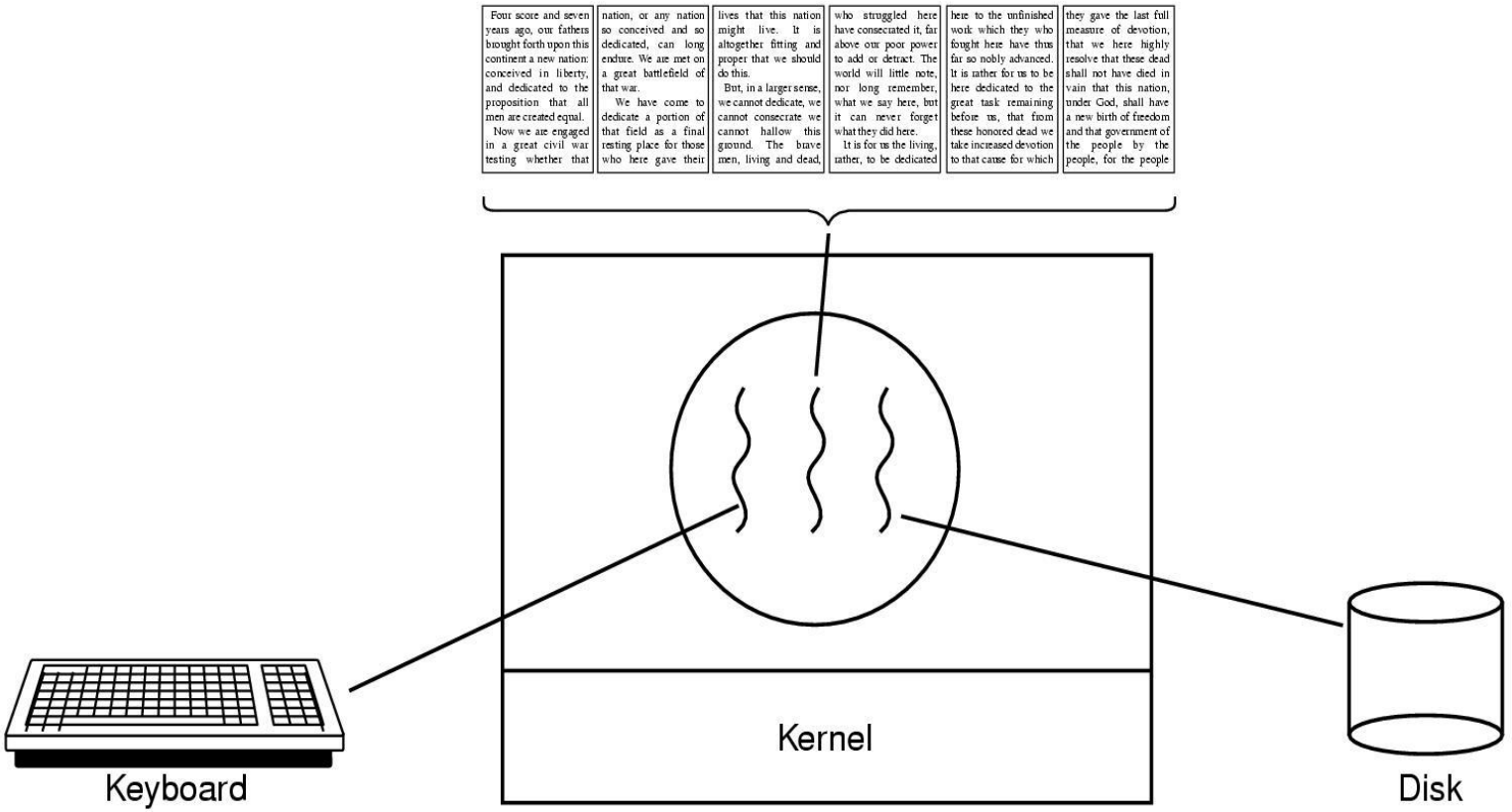
   → Discussed in ch 8 (Tanenbaum)

Threads Usage

- It is easiest to see why threads are useful by looking at some concrete examples
  1. Word processor
  2. Spread sheet
  3. Web Server
  4. Applications that must process very large amounts of data

Thread Usage (1)

Word processor



Writing a book: interactive and background threads sharing the same file

Problem with one thread

- Consider what happens when the user suddenly deletes one sentence from page 1 of an 800-page document.
  - After checking the changed page for correctness, he now wants to make another change on page 600 and types in a command telling the word processor to go to that page (possibly by searching for a phrase occurring only there).
  - The word processor is now forced to reformat the entire book up to page 600 on the spot because it does not know what the first line of page 600 will be until it has processed all the previous pages.
  - There may be a substantial **delay** before page 600 can be displayed, leading to an **unhappy** user.

Add 2nd thread: "Reformat thread"

- Threads can help here. Suppose that the word processor is written as a two-threaded program.
  - One thread interacts with the user and the other handles reformatting in the background. As soon as the sentence is deleted from page 1, the interactive thread tells the reformatting thread to reformat the whole book.
  - Mean-while, the interactive thread continues to listen to the keyboard and mouse and responds to simple commands like scrolling page 1 while the other thread is com-puting madly in the background.
  - With a little luck, the reformatting will be completed before the user asks to see page 600, so it can be displayed **instantly**.

Add the 3rd thread "Backup thread"

If the program were single-threaded:

- whenever a disk backup started, commands from the keyboard and mouse would be ignored until the backup was finished.
- Alternatively, keyboard and mouse events could interrupt the disk backup, allowing good performance but leading to a complex interrupt-driven programming model.

Compare it with three thread ...

- The programming model is much simpler. The **first** thread just interacts with the user. The **second** thread reformats the document when told to. The **third** thread writes the contents of RAM to disk periodically.

Word Processor Case

- Having **three separate processes** would not work here because all three threads need to operate on the document.
- By having **three threads** instead of three processes, they share a common memory and thus all have access to the document being edited.

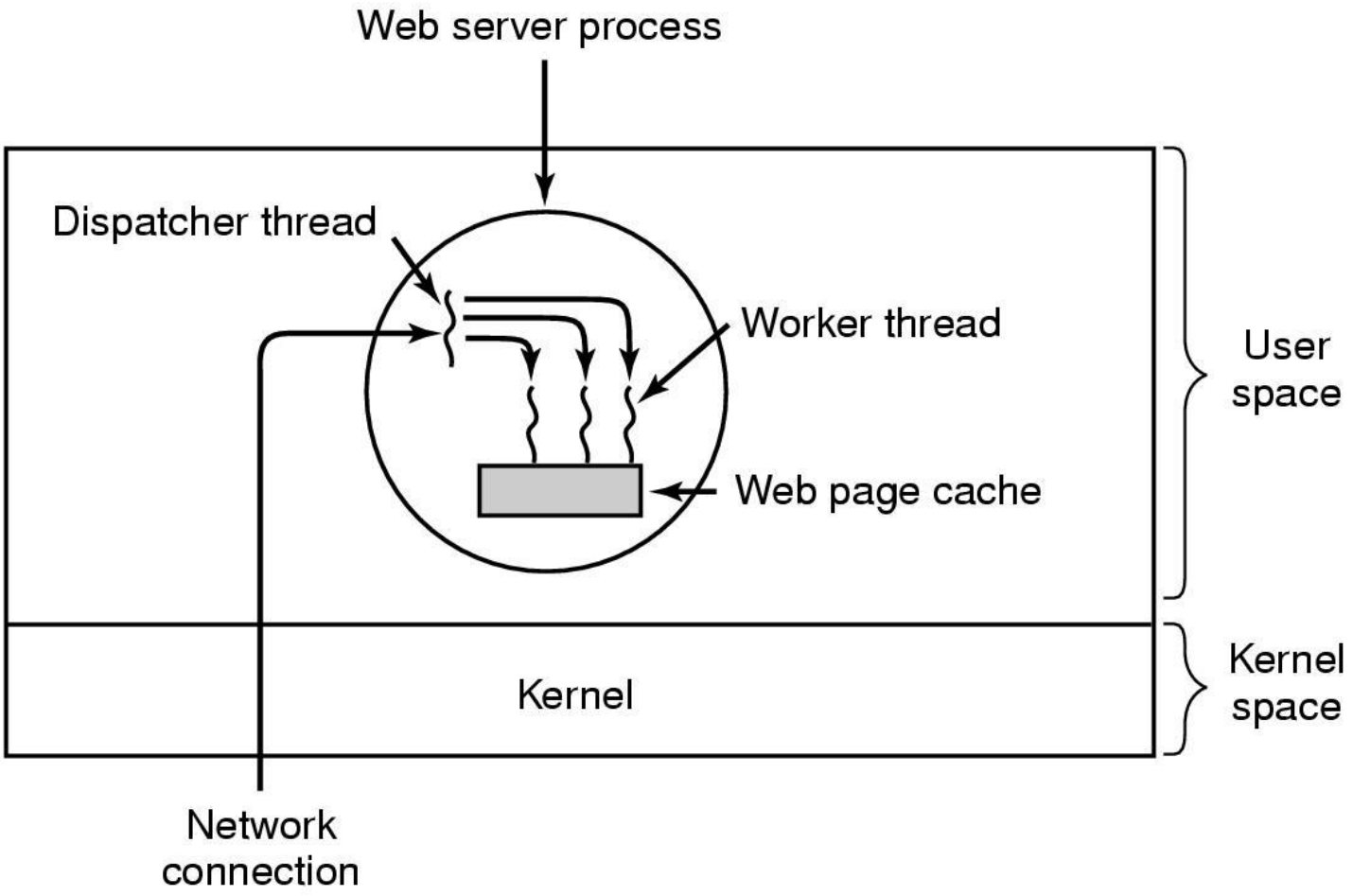
Thread Usage (2)

Spreadsheet

- An electronic spreadsheet is a program that allows a user to maintain a matrix, some of whose elements are **data provided by the user**.
- Other elements are **computed based on the input data** using potentially complex formulas. When a user changes one element, many other elements may have to be recomputed.
- By having a background thread do the recompute on, the interactive thread can allow the user to make additional changes while the computation is going on.
- Similarly, a third thread can handle periodic **backups** to disk on its own.

Thread Usage (3)

Web server



Web Server

- Here one thread, the **dispatcher,** reads incoming requests for work from the network.
- After examining the request, it chooses an idle (i.e., blocked) **worker thread** and hands it the request, possibly by writing a pointer to the message into a special word associated with each thread.
- The dispatcher then wakes up the sleeping worker, moving it from blocked state to ready state.

Web Server

- When the worker wakes up, it checks to see if the request can he satisfied from the Web page cache, to which all threads have access.
- If not, it starts a read operation to get the page from the disk and blocks until the disk operation completes.
- When the thread blocks on the disk operation, another thread is chosen to run, possibly the dispatcher, in order to acquire more work, or possibly another worker that is now ready to run

Web Server

- This model allows the server to be written as a collection of sequential threads.
- The dispatcher's program consists of an infinite loop for getting a work request and handing it off to a worker.
- Each worker's code consists of an infinite loop consisting of accepting a request from the dispatcher and checking the Web cache to see if the page is present.
- If so, it is returned to the client and the worker blocks waiting for a new request. If not, it gets the page from the disk, returns it to the client, and blocks waiting for a new request.

Web Server

```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

- Rough outline of code for multithreaded web server
  (a) Dispatcher thread
  (b) Worker thread

Consider Web Server in the absence of threads

- One possibility is to have it operate as a single thread.
  - The main loop of the Web server gets a request, examines it, and carries it **out to completion before getting the next one**.
  - While waiting for the disk, the server is idle and does not process any other incoming requests.
  - If the Web server is running on a dedicated machine, as is commonly the case, the CPU is simply idle while the Web server is waiting for the disk.

Consider Web Server in the absence of threads

- The net result is that many fewer requests/sec can be processed.
- Thus threads gain considerable performance, but each thread is programmed sequentially, in the usual way.

Three ways to construct a server

Model	Characteristics
Multi-threaded process	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

- With Multi-threaded process, blocking system calls make programming easier and parallelism improves performance.
- The single-threaded server retains the ease of blocking system calls but gives up performance.
- The third approach achieves high performance through parallelism but uses non-blocking calls and interrupts and is thus is hard to program. → simulating the threads and their stacks in hard way.

Thread Usage (4)

Applications that must process very large amounts of data

- The normal approach is to read in a block of data, process it, and then write it out again.
- *The problem here is that if only blocking system calls are available, the process blocks while data are coming in and data are going out.*
- Having the CPU go idle when there is lots of computing to do is clearly wasteful and should be avoided if possible.

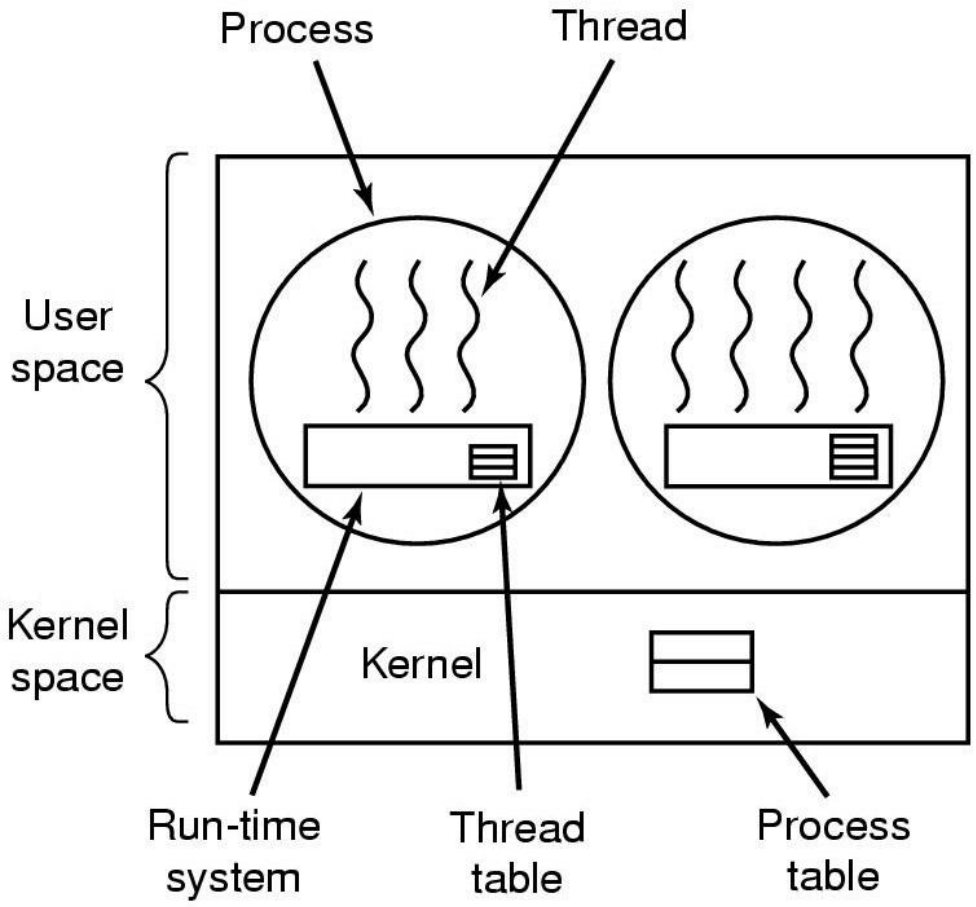
...Applications that must process very large amounts of data

- Threads offer a solution
- The process could be structured with an input thread, a processing thread, and an output thread.
  - The input thread reads data into an input buffer.
  - The processing thread takes data out of the input buffer, processes them, and puts the results in an output buffer.
  - The output buffer writes these results back to disk.
- In this way, input, output, and processing can all be going on at the same time.
- Of course, this model only works if a system-call blocks only the calling thread, not the entire process
  - Thread which waiting for I/O is blocked
  - Another thread activated

Agenda

1. Thread Model
2. Thread Usage
3. **User Level Thread (ULT)**
4. Kernel Level Thread (KLT)
5. Hybrid Implementation

Implementing Threads in **User Space**



A user-level threads package

Thread Table

- When threads are managed in user space, each process needs its own **private thread table** to keep track of the threads in that process.
- This table is analogous to the kernel's process table, except that it keeps track only of the per-thread properties, such as each thread's ***program counter, stack pointer, registers, state***, and so forth

Run-time System

- When a thread does something that may cause it to become blocked locally, for example, waiting for another thread in its process to complete some work, it calls a **run-time system** procedure.
- This procedure checks to see if the thread must be put into blocked state. If so, it stores the thread's registers (i.e., its own) in the thread table, looks in the table for a ready thread to run, and reloads the machine registers with the new thread's saved values

User Space Advantages

+: No specific OS support needed

Faster than kernel instructions

Process-specific scheduling algorithms

User Space Dis-advantages

- : Blocking system calls (select)

Suppose that a thread reads from the keyboard before any keys have been hit. Letting the thread actually make the system call is unacceptable, since this will **stop all the threads.**

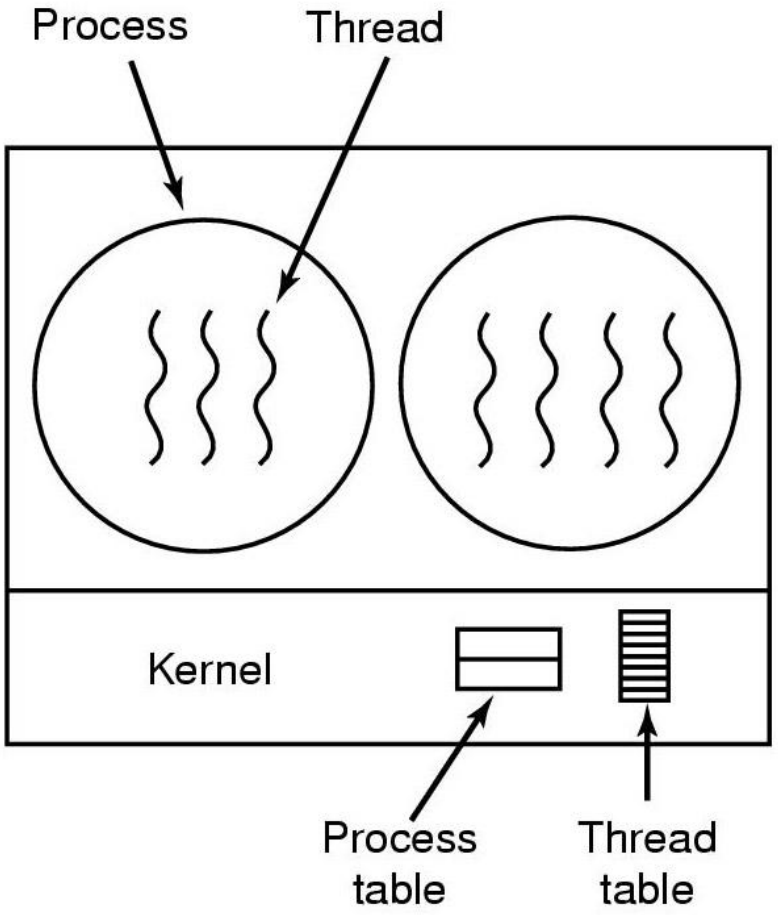
Page faults

If a thread causes a page fault, the kernel, not even knowing about the existence of threads, naturally **blocks the entire process** until the disk I/O is complete, even though other threads might be runnable.

Agenda

1. Thread Model
2. Thread Usage
3. User Level Thread (ULT)
4. **Kernel Level Thread (KLT)**
5. Hybrid Implementation

Implementing Threads in the Kernel



A threads package managed by the kernel

(Dis)advantages

+: Handling blocking and page faults

if one thread in a process causes a page fault, the kernel can easily check to see if the process has any other runnable threads, and if so, run one of them while waiting for the required page to be brought in from the disk

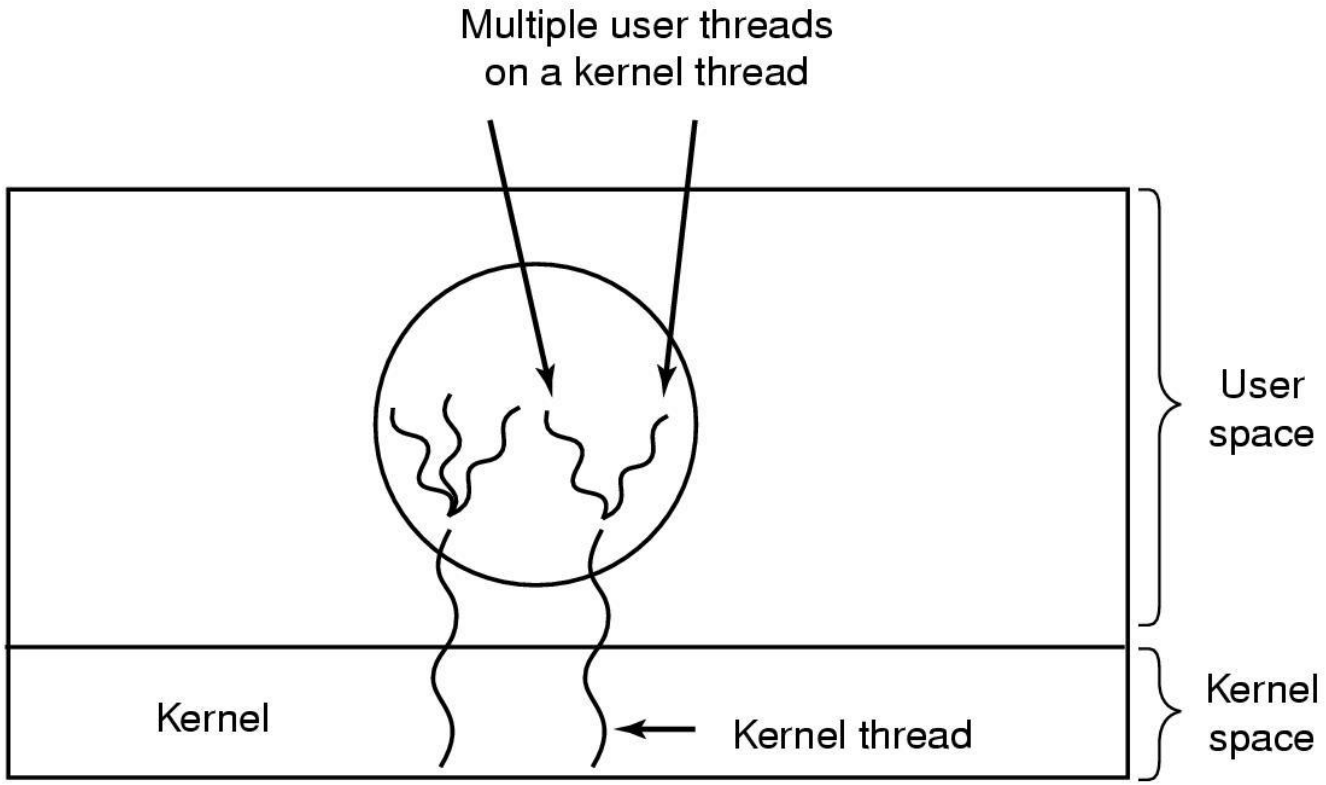
-: More costly (but recycling threads)

if thread operations (creation, termination, etc.) are common, much **more overhead will be incurred**

Agenda

1. Thread Model
2. Thread Usage
3. User Level Thread (ULT)
4. Kernel Level Thread (KLT)
5. Hybrid Implementation

Hybrid Implementations



Multiplexing user-level threads onto kernel- level threads

Hybrid

- Kernel is aware of kernel threads only
- User level threads are scheduled, created destroyed independently of kernel thread
- Programmer determines how many user level and how many kernel level threads to use

REVIEW

1. Sebutkan alasan mengapa pemindahan (switching) antar threads lebih “murah” dibandingkan antar proses?
2. Atribut apa saja yang dimiliki secara khusus oleh tiap thread dalam suatu proses? Jelaskan alasannya.
3. Jelaskan dengan contoh bagaimana threads dapat memberi solusi bagi *word processor* dan *web server*.
4. Bandingkan kelebihan dan kekurangan ULT (*User Level Thread*) dibandingkan KLT (*Kernel level thread*).
5. Apa yang dimaksud dengan *blocking system call*? Bagaimana pengaruhnya terhadap sistem yang menggunakan thread?