**Threads**

**Definition 5.3 Thread**
An execution of a program that uses the resources of a process.

- A thread is an alternative model of program execution
- A process creates a thread through a system call
- Thread operates within process context
- Use of threads effectively splits the process state into two parts
  - Resource state remains with process
  - CPU state is associated with thread
- Switching between threads incurs less overhead than switching between processes

---

**Figure 5.11**
Threads in process $P_i$: (a) concept, (b) implementation.

TCB contains: 1) thread id, priority, and state; 2) CPU state; 3) pointer to PCB of parent process; 4) TCB pointer.

---

**Figure 5.12**

(a) Context of process $P_i$
(b) Context of process $P_j$, TCB contains: 1) thread id, priority, and state; 2) CPU state; 3) pointer to PCB of parent process; 4) TCB pointer.
Kernel-Level, User-Level, and Hybrid Threads

- Kernel-Level Threads
  - Threads are implemented by the kernel

- User-Level Threads
  - Threads are implemented by a thread library

- Hybrid Threads
  - Combination of kernel-level and user-level threads

---

**Table 5.8 Advantages of Threads over Processes**

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower overhead of creation</td>
<td>Thread state consists only of the state of a</td>
</tr>
<tr>
<td>and switching</td>
<td>computation. Resource allocation state and</td>
</tr>
<tr>
<td></td>
<td>communication state are not a part of the thread</td>
</tr>
<tr>
<td></td>
<td>state, an creation of threads and switching between</td>
</tr>
<tr>
<td></td>
<td>them incurs a lower overhead.</td>
</tr>
<tr>
<td>More efficient communication</td>
<td>Threads of a process can communicate with one another through</td>
</tr>
<tr>
<td></td>
<td>shared data, thus avoiding the overhead of system calls for</td>
</tr>
<tr>
<td></td>
<td>communication.</td>
</tr>
<tr>
<td>Simplification of design</td>
<td>Use of threads can simplify design and coding of</td>
</tr>
<tr>
<td></td>
<td>applications that service requests concurrently.</td>
</tr>
</tbody>
</table>

**Figure 5.12** Use of threads in structuring a server: (a) server using sequential code; (b) multithreaded server; (c) server using a thread pool.
Kernel-Level Threads

• A kernel-level thread is like a process except that it has a smaller amount of state information

• Advantages:
  – Provide parallelism in a multiprocessor system
  – If a thread performs a blocking system call, the kernel can schedule another thread in the application for execution

• Disadvantage: Switching between threads of same process incurs the overhead of event handling

User-Level Threads

• Thread library maintains thread state, performs scheduling and switching of threads

• Advantage: Fast thread switching because kernel is not involved

• Disadvantages:
  – Blocking of a thread blocks all threads of the process
  – No parallelism
Scheduling of User-Level Threads

- Many-to-many association permits a user-level thread to be mapped into different kernel-level threads at different times.
- Can provide a combination of parallelism and low overhead of switching.

Hybrid Thread Models

- Many-to-many association permits a user-level thread to be mapped into different kernel-level threads at different times.
- Can provide a combination of parallelism and low overhead of switching.
Pthreads

- The ANSI/IEEE Portable Operating System Interface (POSIX) standard defines threads API
  - For use by C language programs
  - Provides 60 routines that perform the following:
    - Thread management
    - Assistance for data sharing — mutual exclusion
    - Assistance for synchronization — condition variables
- On Linux machines
  - #include <pthread.h>
  - Link with the pthread library
    g++ threads.cc -lpthread

Pthreads: Creating

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr, 
void * (* start_routine) (void *), void *arg);
```
- On success
  - A new thread is created with attributes “attr” (NULL=default), id of new thread is placed in “thread”.
  - New thread starts executing function “start_routine” with parameter “arg”
  - New thread executes concurrently with the calling thread
  - New thread runs until it returns from “start_routine” or it calls “pthread_exit”
  - Return 0
- On failure
  - Return non-zero error code
pthread_create Example

```c
void* my_thread (void *arg)
{
    char *msg = (char *) arg;
    cout << “Thread says “ << msg <<”\n”;
}

int main()
{
    pthread_t t;
    char *msg = “Hello World”;
    pthread_create(&t, NULL, my_thread, msg);
    return 0;
}
```

Pthreads: Waiting

```c
int pthread_join(pthread_t th, void **thread_return) – wait for a thread to terminate
```

- th: the thread to wait for
- thread_return: NULL or address of buffer to store return value of th
- When a thread terminates, its memory resources (TCB and stack) are not deallocated until another thread performs pthread_join on it.
- At most one thread can wait for the termination of a given thread.
- Return 0 on success, non-zero on failure
#include <pthread.h>
#include <iostream>
#include <unistd.h>
using namespace std;

int N;

int main()
{
    N = 0;
    pthread_t t1, t2, t3;
    cout << "Parent creating threads\n";
    pthread_create(&t1, NULL, thread, new int(1));
    pthread_create(&t2, NULL, thread, new int(2));
    pthread_create(&t3, NULL, thread, new int(3));
    cout << "Threads created\n";
    N = 3;
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    pthread_join(t3, NULL);
    cout << "Threads are done\n";
    return 0;
}

void *thread(void *x)
{
    int *id = (int *)x;
    while (N != *id);
    cout << "Thread " << *id << endl;
    N--;
    delete id;
    pthread_exit(0);
}

Summary

• A process is a model of execution of a program
  – Can create other processes by making requests to the OS through system calls
    • Provides parallelism or concurrency
• OS allocates resources to a process and stores information about them in the context of the process
• To control operation of the process, OS uses notion of a process state
• OS keeps information concerning each process in a process control block (PCB)
  – Process state and CPU state associated with process
  – Scheduler selects a ready process and dispatcher switches CPU to selected process through information found in its process context and the PCB
Summary (continued)

• A thread is an alternative model of execution of a program
  – Overhead of switching between threads is much less than the overhead of switching between processes

• Three models of threads:
  – Kernel-level threads
  – User-level threads
  – Hybrid threads

• Thread models have different implications for switching overhead, concurrency, and parallelism