Threads

- Silberschatz Chapter 4

Parallelism in operating systems

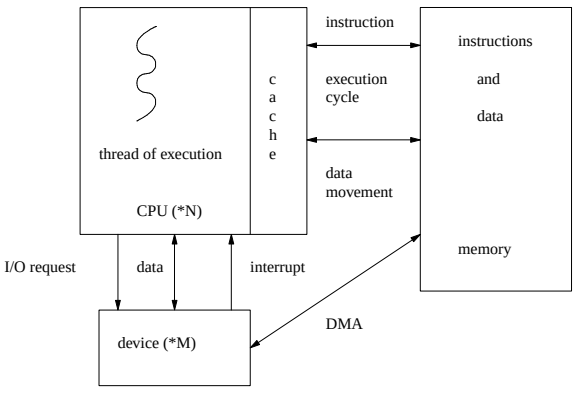
- typical activity on modern operating systems is achieved by multiple processes (Unix ps, top commands + Windows Task Manager)
- techniques originally designed for single CPU made extensible to multiple CPUs
- additionally the nature of applications run by OS has changed - more complexity

Thread concept

- modern complex applications require a degree of parallelism
- thread concept is derived from process concepts sometimes defined as a ‘‘light weight process’’
- both exist in modern operating systems

Modern computer system operation

-

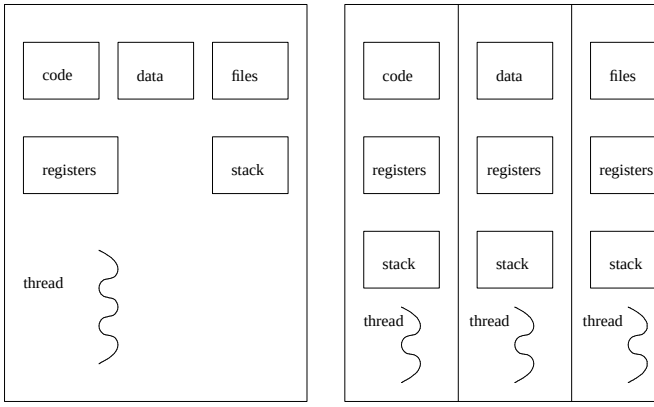


Silberschatz figure 1.5

Thread vs Process

- traditional processes often have a single thread of control per address space - parallelism is achieved by creating a new process via fork and exec
- threads were originally a programming technique allowing multiple ‘‘threads of control’’ in a single address space

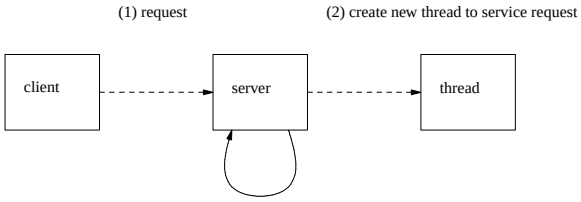
Traditional and multithreaded processes



single-threaded process

multithreaded process

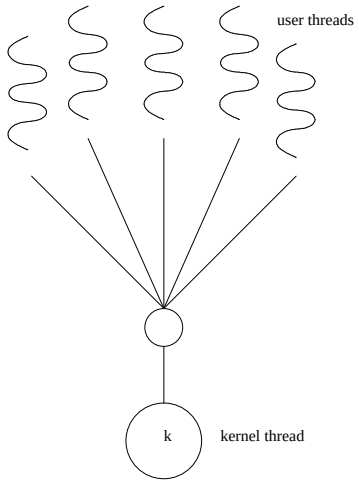
Multithreaded server architecture



Multithreaded models

- user level threads
  - supported above the kernel and managed without kernel support
- kernel threads
  - supported and managed directly by the operating system kernel

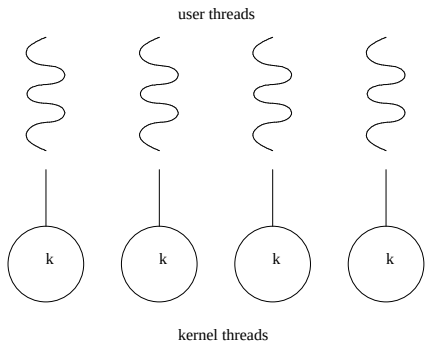
Relationship between kernel and user threads: many to one



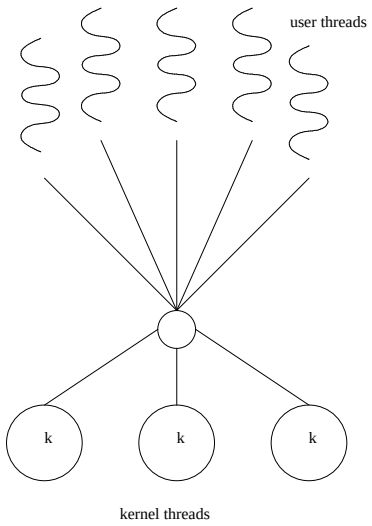
Relationship between kernel and user threads: many to one

- example is GNU/Pth (not to be confused with Pthreads)
  - GNU/Pth allows you to write your own thread scheduler
  - Penguin tower uses this approach

Relationship between kernel and user threads: one to one



Relationship between kernel and user threads: many to many



PThreads

- a POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- common in UNIX operating systems (GNU/Linux, Mac OS X)

An example of threads in a pedagogical program

- $x = \sum_{i=1}^{3} i$
- $x = 1 + 2 + 3 = 6$
- the programs calculate, x, using a thread

Multithreaded C program using the Pthread API: Adapted from Silberschatz figure 4.6

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

int sum;
void *runner(void *param);

int main()
{
  pthread_t tid;
  char *param = "3";

  printf ("%s\n", param);
  pthread_create (&tid, NULL, runner, param);
  pthread_join (tid, NULL);
  printf ("sum = %d\n", sum);
}
```


Multithreaded C program using the Pthread API: Adapted from Silberschatz figure 4.6

```
void *runner (void *param)
{
  int i;
  int upper = atoi (param);
  sum = 0;

  for (i=1; i<= upper; i++) {
    sum = sum + i;
    printf ("sum = %d (added %d)\n", sum, i);
  }
  pthread_exit (0);
}
```

Multithreaded Python program using Threads

```
#!/usr/bin/python

# adapted from Programming Python - O'Reilly
# mutex included to make output clean

import thread, time

sum = 0
mutex = thread.allocate_lock ()

def counter (myId, count):
    global sum
    for i in range (1, count+1):
        time.sleep (1)
        mutex.acquire ()
        print "thread_id[%s] => %s" % (myId, i)
        sum = sum + i
        print "sum =", sum, "(added", i, ")"
        mutex.release ()

thread.start_new (counter, (1, 3))
time.sleep (4)
print "Main thread exiting, sum =", sum
```