Thread and Multithreaded Programming

Thread

- Reading: OS essential Page 163~190
- abstraction of a processor
- a *thread of control*
- previous programs: single-threaded programs

Why Threads

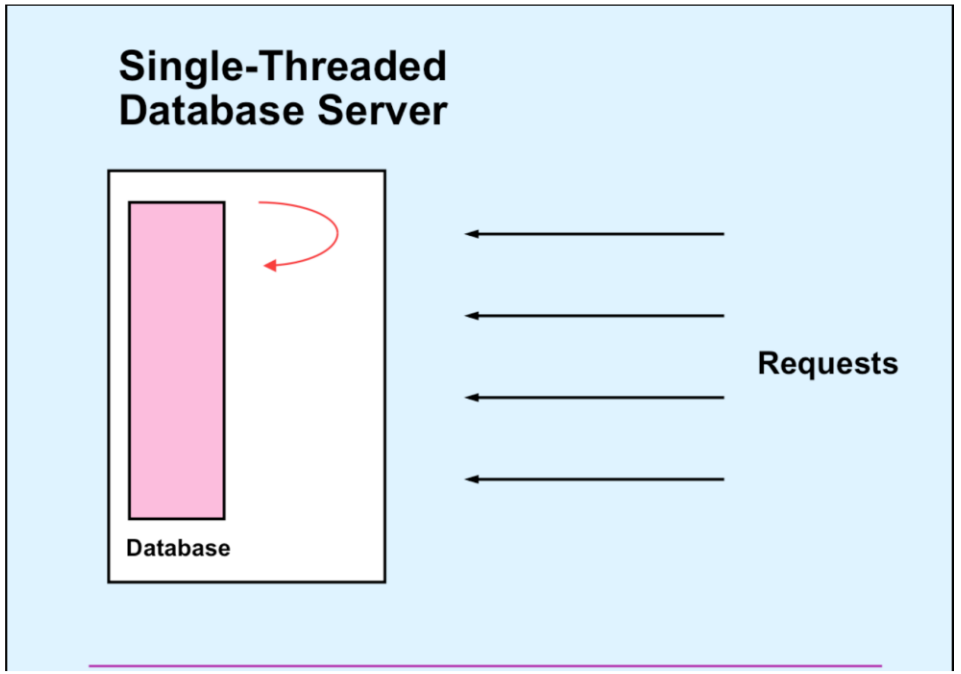
- a natural means for dealing with *concurrency*
  - *Concurrency: multiprocessor and equally useful on uniprocesors*
- Many things become easier to do with threads
- Many things run faster with threads

Processes vs. Threads

- Both provides concurrency
- What's the difference?
- Two single-threaded processes vs. one two-threaded process
  - First, for an existing process, it is much cheaper to add a thread than creating a new process
  - Context switching: it is also cheaper between context of two threads in the same process
  - Two threads in one process share everything: address space, open files.

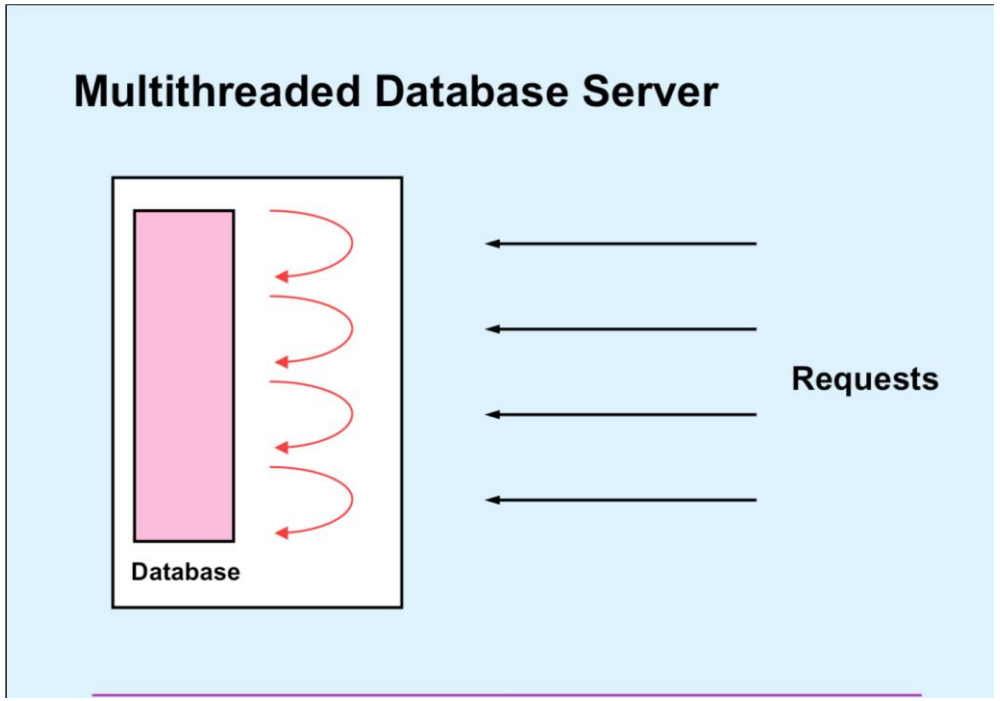
Example of a Database Server

- Single-Threaded Database Server
- Sequentially handle multiple clients
- unfair to quick requests behind lengthy requests



Example of a Database Server

- Multithreaded Database Server
- A separate thread to each request
- Synchronization of access to the database



Standards

- POSIX 1003.4a, 1003.1c(1995), 1003.1j(2000)
- Microsoft: Win32

Matrix Multiplication

- C= A *B
- A: m x n
- B: n x p
- C: ?
  - m x p

Example

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

#define M 3
#define N 4
#define P 5
```

```
long A [M][N]
long B [N][P]
long C [M][P]

void *matmul(void *)

main(){
        long i
        pthread_t thr[m];
        int error;
…..
```

Creating Threads

```
long A[m][n], B[n][p], C[m][p];
……
for (i = 0; i< m; i++)          // create worker threads
    pthread_create(&thr[i], 0, matmult, i);


 void *matmult(void *arg){
        //computer row I of C
}
```

Example

```
void *matmult(void *arg){
    long row = (long) arg;
    long col;
    long i;
    long t;
    for (col = 0; col < P; col++){
        t = 0;
        for (I = 0; i<N; i++)
                t += A[row][i]*B[i][col];
        C[row][col] = t;
    }
}
```

When Finish?

```
long A[m][n], B[n][p], C[m][p];
……
for (i = 0; i< m; i++)          // create worker threads
    pthread_create(&thr[i], 0, matmult, i);

for (i = 0; i< m; i++)          //wait for termination
    pthread_join(thr[i], 0);

printResult(C);
```

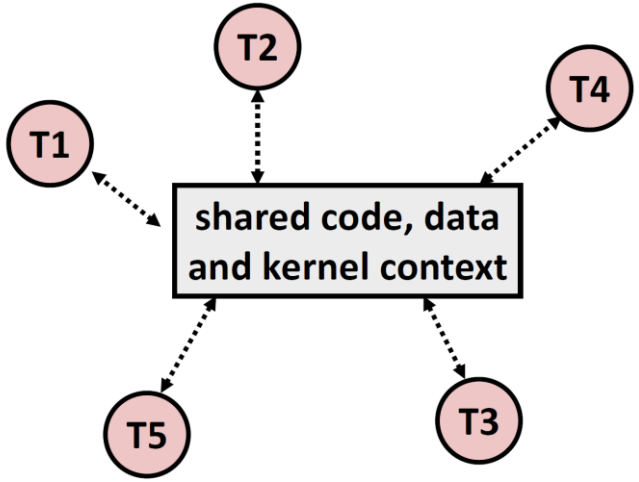
A Process With Multiple Threads

- **Multiple threads can be associated with a process**
  - Each thread has its own logical control flow
  - Each thread shares the same code, data, and kernel context
  - Each thread has its own stack for local variables
  - Each thread has its own thread id (TID)
    - pthread_t pthread_self(void);

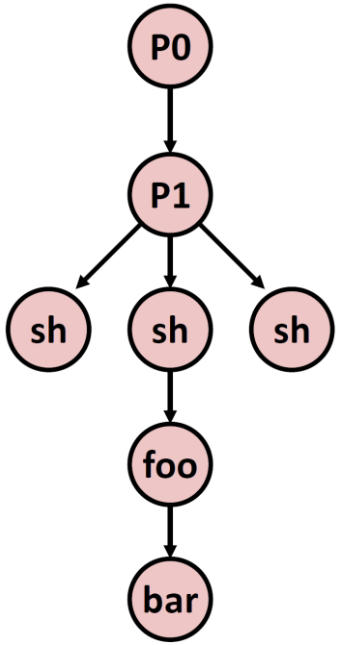
Logical View of Thread

- **Threads associated with a process form a pool of peers**
  - Unlike processes which form a tree hierarchy

Threads in a process



Process hierarchy



Threads vs. Processes

- Threads share all code and data (except local stacks)
  - Processes (typically) do not
- Threads are somewhat less expensive than processes
  - Process control (creating and reaping) twice as expensive as thread control
  - Linux numbers:
    - ~20K cycles to create and reap a process
    - ~10K cycles (or less) to create and reap a thread

Termination

- pthread_exit
- return
- pthread_join

Termination

- The thread terminates
  - phread_exit
    - When main thread calls pthread_exit, it waits for all other peer threads to terminate and then terminates the main thread.
  - Top-level thread routine returns
  - Some peer calls exit, which terminates the process