# Thread & PThread

Chapter 4.1, 4.2, 4.3, 4.4, 4.6

# Agenda

1. What is thread?
2. User vs kernel threads
3. Thread models
4. Thread issues
5. Pthread library
6. An example

# 1. What is thread (1): Program, process, thread

In one process

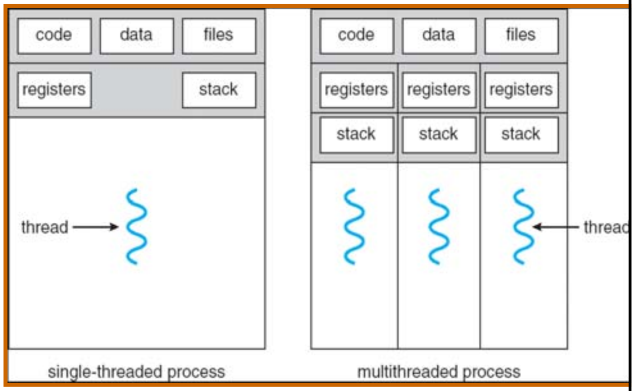
- easy to share

Btw processes

- multitasking

Best of both

- thread
  - one process
  - multitasking



Q: browsers to use multi-process?

# 1. What is thread (2): Threads

Thread

- a basic unit of CPU utilization
  - thread state, program counter, register set, stack
- share with other threads in the same process
  - code, data, opened files, signals, etc

Benefits

- responsiveness: multithreading
- resource sharing, efficiency, MP architectures

Q: potential problems?

## 1. What is thread (3): Single-threaded Web server

Web server with cache and disk

- wait for a request
- process the request
  - – check cache; if hit, break
  - – otherwise, retrieve from disk (relatively slow)
- respond the request

One request at a time

- or create a new process on each request
  - – expensive!

## 1. What is thread (4): Multi-threaded Web server

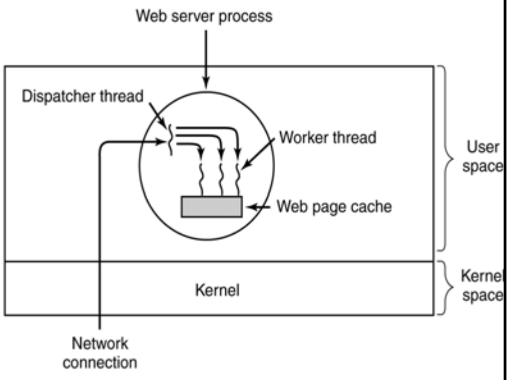
*Dispatcher* thread

- wait for a request
- handoff the request

*Worker* threads

- process the request
  - – disk I/O
- respond the request

“Many” requests at a time



# 2. User vs kernel threads

User threads: e.g., pthread library

- each process schedules its own threads
- no context switch between these threads
- a blocking call blocks the entire process

Kernel threads: in almost all modern OS

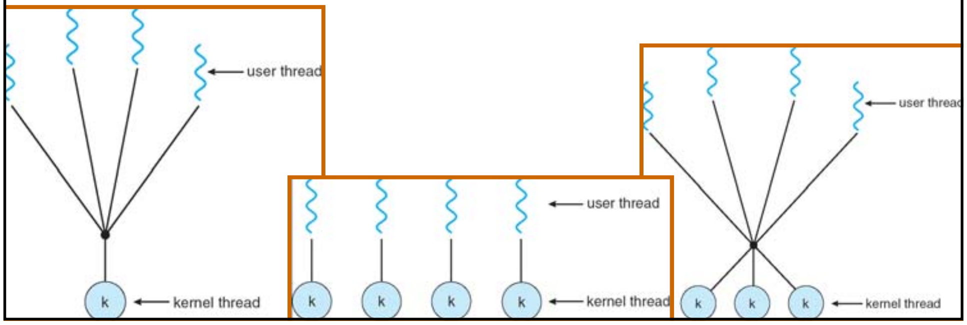
- kernel manages all threads
- can pickup another thread if one blocks

Hybrid approaches

# 3. Thread models (1)

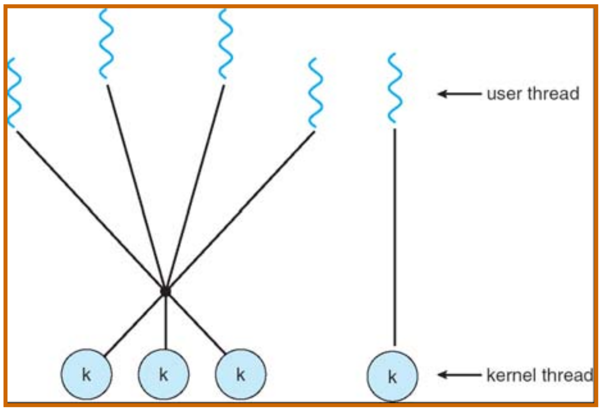
User-kernel mapping

- many-to-one: low cost, (lower) parallelism
- one-to-one: high parallelism, (higher) cost
- many-to-many: limited kernel threads



# 3. Thread models (2)

Two-level model



# 4. Threading issues

When a new process is created

- fork(), and usually then exec()
  - – duplicate all threads or just the calling thread?

When a signal to be delivered

- signal: event notification to be handled
  - – to all, some, or a specific thread?

Thread pool

- keep a pool of threads to be used
  - – and reuse

# 5. Pthread library (1)

Create a thread

- – int **pthread_create** (thread, attributes, start_routine, arguments);
- • PC: start_routine(arguments);
- • default attributes: joinable and non-realtime

Exit from a (created) thread

- – void **pthread_exit** (return_value);
- • cleanup handlers by **pthread_cleanup_push** ();
  - –stack-like “reverse” execution order

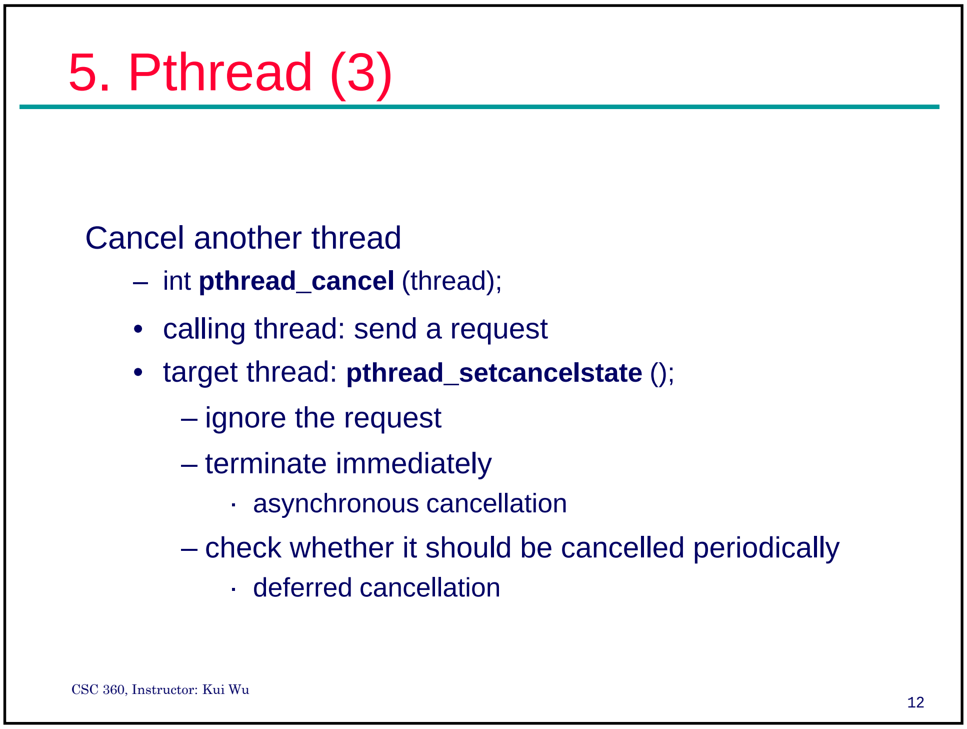
# 5. Pthread library (2)

Wait a target thread to exit: *synchronize*

- – int **pthread_join** (thread, return_value);
- • release resource allocated to the target thread

Put a target thread in detached state

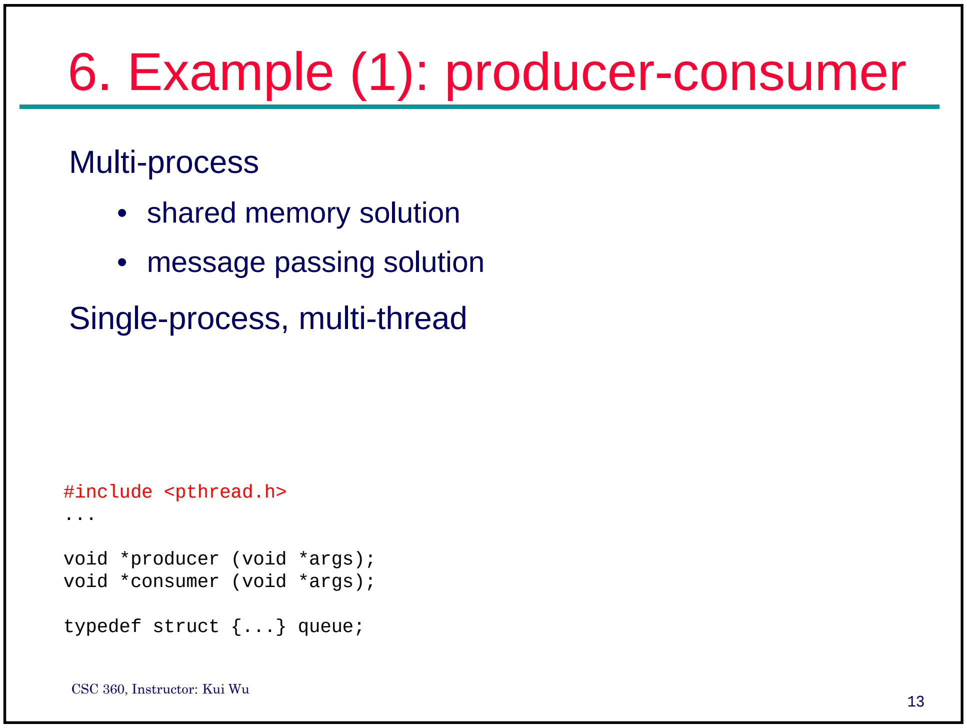
- – int **pthread_detach** (thread);
- • no other threads can “join” this one
  - – no “pthread_attach”
- • resource released once the thread exits
  - – thread can be created in detached state



# 5. Pthread (3)

Cancel another thread

- int **pthread_cancel** (thread);
  - calling thread: send a request
  - target thread: **pthread_setcancelstate** ();
    - ignore the request
    - terminate immediately
      - asynchronous cancellation
    - check whether it should be cancelled periodically
      - deferred cancellation



# 6. Example (1): producer-consumer

Multi-process

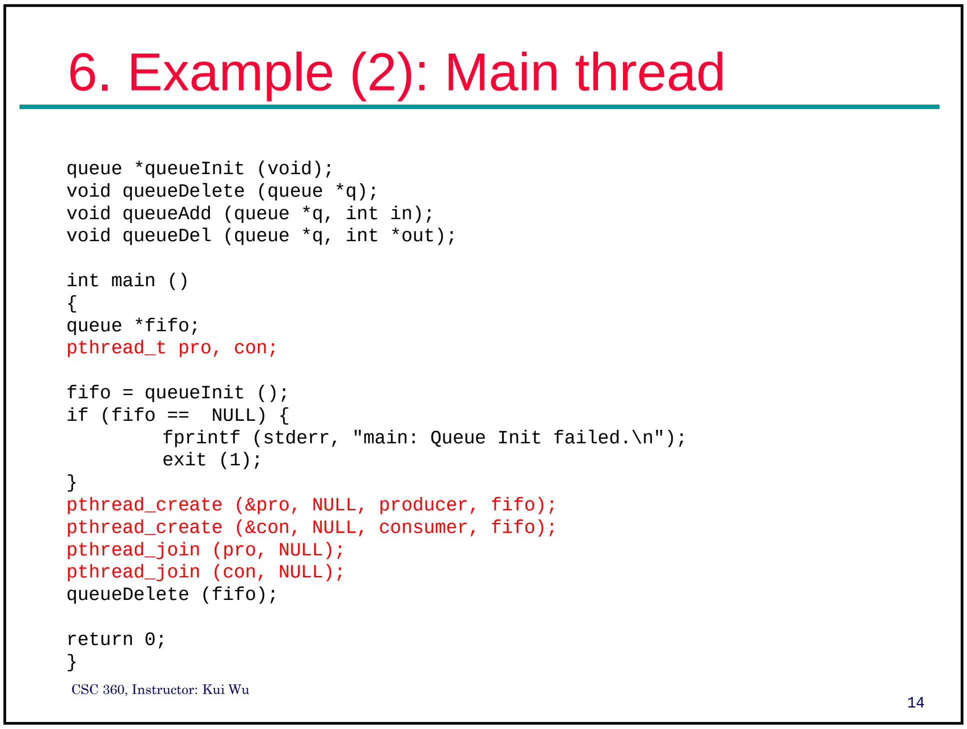
- shared memory solution
- message passing solution

Single-process, multi-thread

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

void *producer (void *args);
void *consumer (void *args);

typedef struct {...} queue;
```



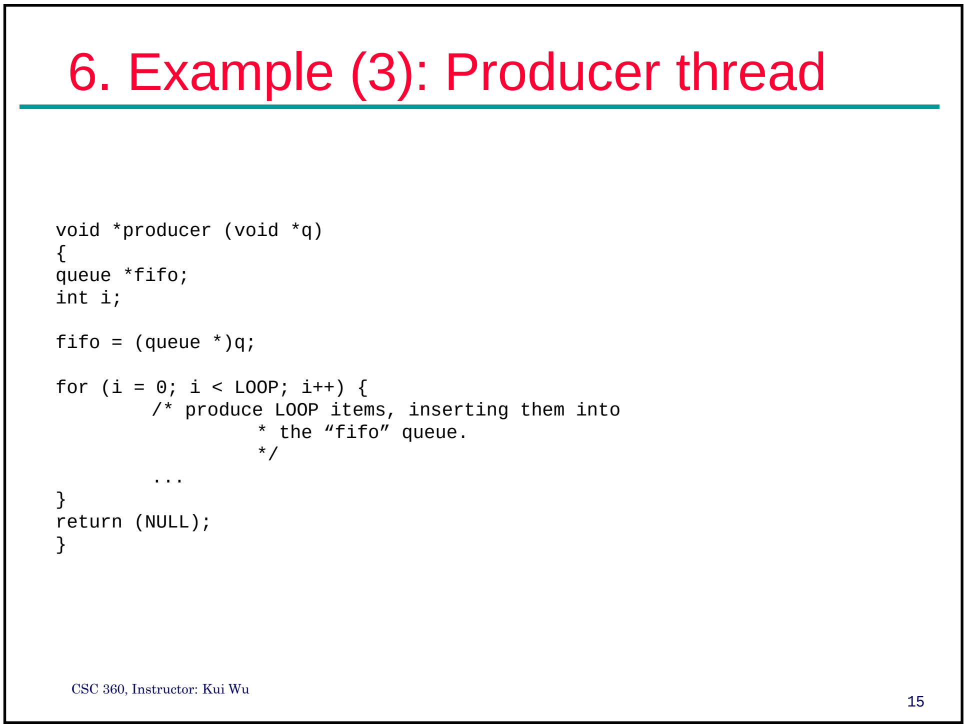
# 6. Example (2): Main thread

```
queue *queueInit (void);
void queueDelete (queue *q);
void queueAdd (queue *q, int in);
void queueDel (queue *q, int *out);

int main ()
{
queue *fifo;
pthread_t pro, con;

fifo = queueInit ();
if (fifo ==  NULL) {
        fprintf (stderr, "main: Queue Init failed.\n");
        exit (1);
}
pthread_create (&pro, NULL, producer, fifo);
pthread_create (&con, NULL, consumer, fifo);
pthread_join (pro, NULL);
pthread_join (con, NULL);
queueDelete (fifo);

return 0;
}
```

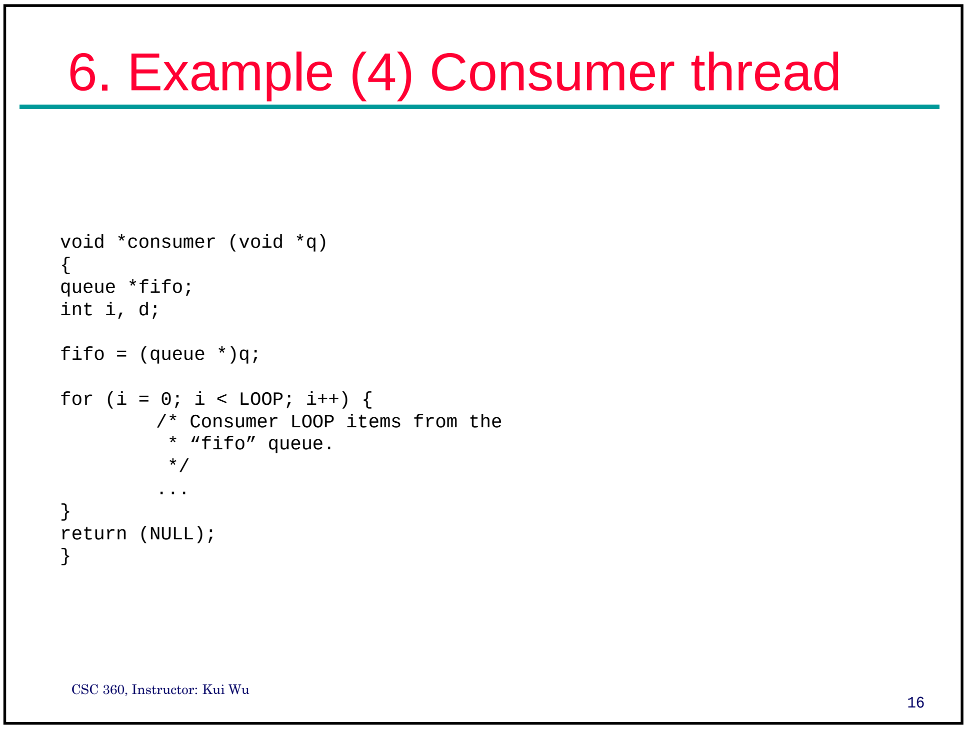


# 6. Example (3): Producer thread

```
void *producer (void *q)
{
queue *fifo;
int i;

fifo = (queue *)q;

for (i = 0; i < LOOP; i++) {
        /* produce LOOP items, inserting them into
                 * the “fifo” queue.
                 */
        ...
}
return (NULL);
}
```



# 6. Example (4) Consumer thread

```
void *consumer (void *q)
{
queue *fifo;
int i, d;

fifo = (queue *)q;

for (i = 0; i < LOOP; i++) {
         /* Consumer LOOP items from the
          * “fifo” queue.
          */
         ...
}
return (NULL);
}
```