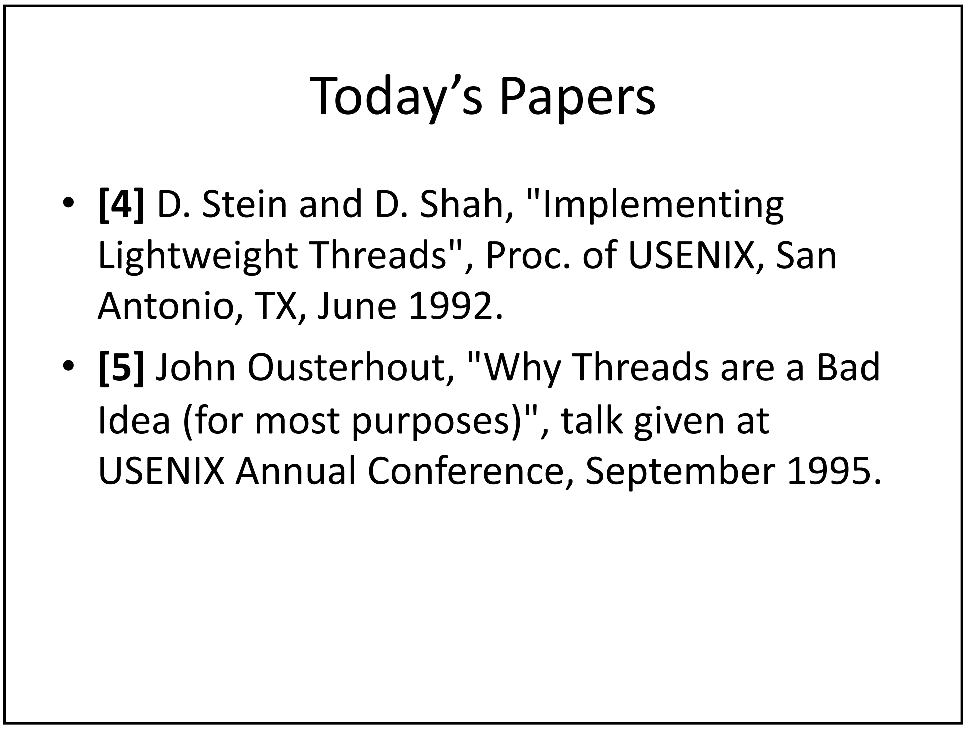
Graduate Operating Systems
(Threads & Events)

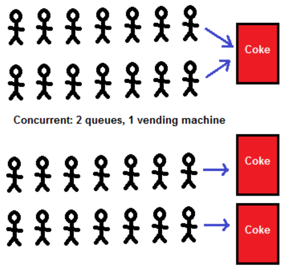
Fall 2020



Today's Papers

- **[4]** D. Stein and D. Shah, "Implementing Lightweight Threads", Proc. of USENIX, San Antonio, TX, June 1992.
- **[5]** John Ousterhout, "Why Threads are a Bad Idea (for most purposes)", talk given at USENIX Annual Conference, September 1995.

Concurrency vs. Parallelism



Concurrent: 2 queues, 1 vending machine

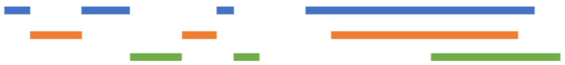
Parallel: 2 queues, 2 vending machines

Concurrency

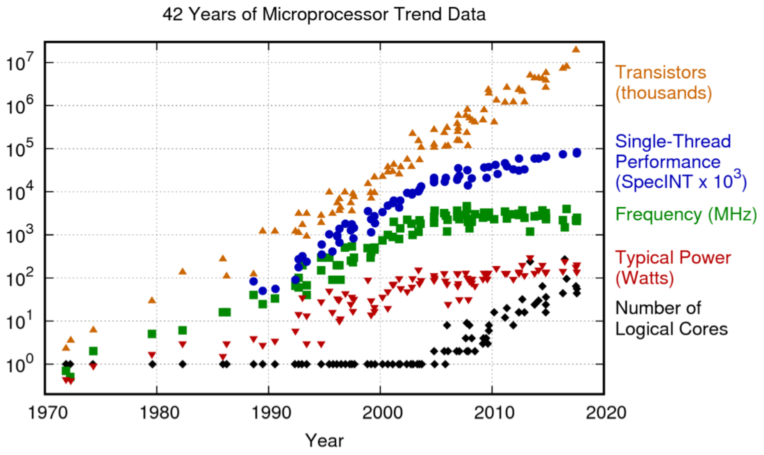
Tasks start, run and complete in an interleaved fashion

Parallelism

Tasks run simultaneously

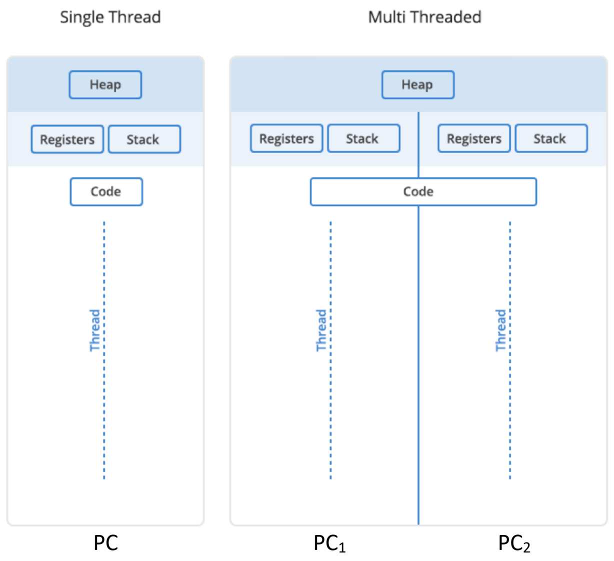


Microprocessor Trends



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Processes vs. Threads



Single Thread

Heap

Registers Stack

Code

Thread

PC

Multi Threaded

Heap

Registers Stack

Registers Stack

Code

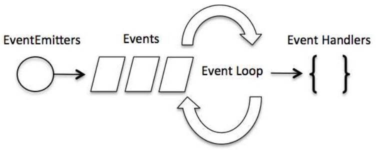
Thread

Thread

PC_1

PC_2

Events



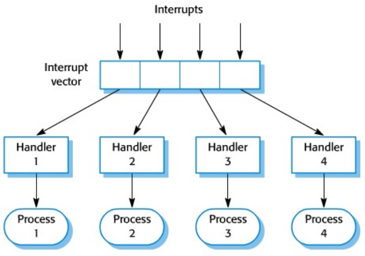
EventEmitters

Events

Event Loop

Event Handlers

{ }



Interrupts

Interrupt vector

Handler 1

Handler 2

Handler 3

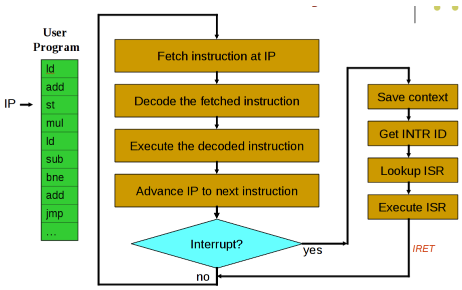
Handler 4

Process 1

Process 2

Process 3

Process 4



User Program

IP →

ld
add
st
mul
ld
sub
bne
add
jmp
…

Fetch instruction at IP

Decode the fetched instruction

Execute the decoded instruction

Advance IP to next instruction

Interrupt?

yes

no

Save context

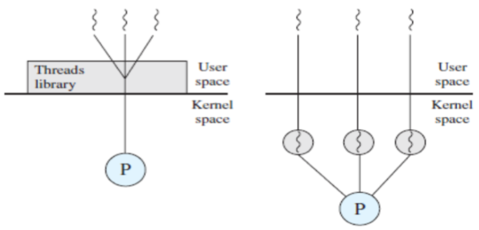
Get INTR ID

Lookup ISR

Execute ISR

IRET

Types of Threads



(a) Pure user-level

(b) Pure kernel-level

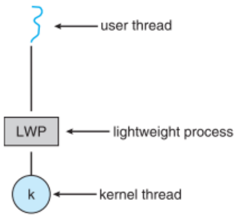
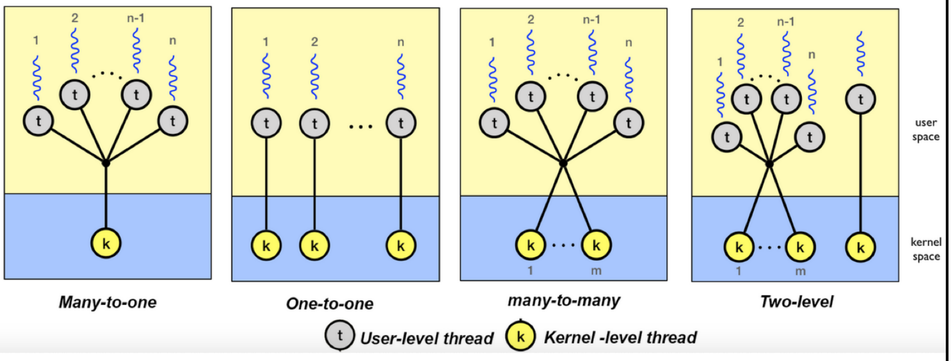


Figure 4.13 Lightweight process (LWP).

Thread Models



Many-to-one

One-to-one

many-to-many

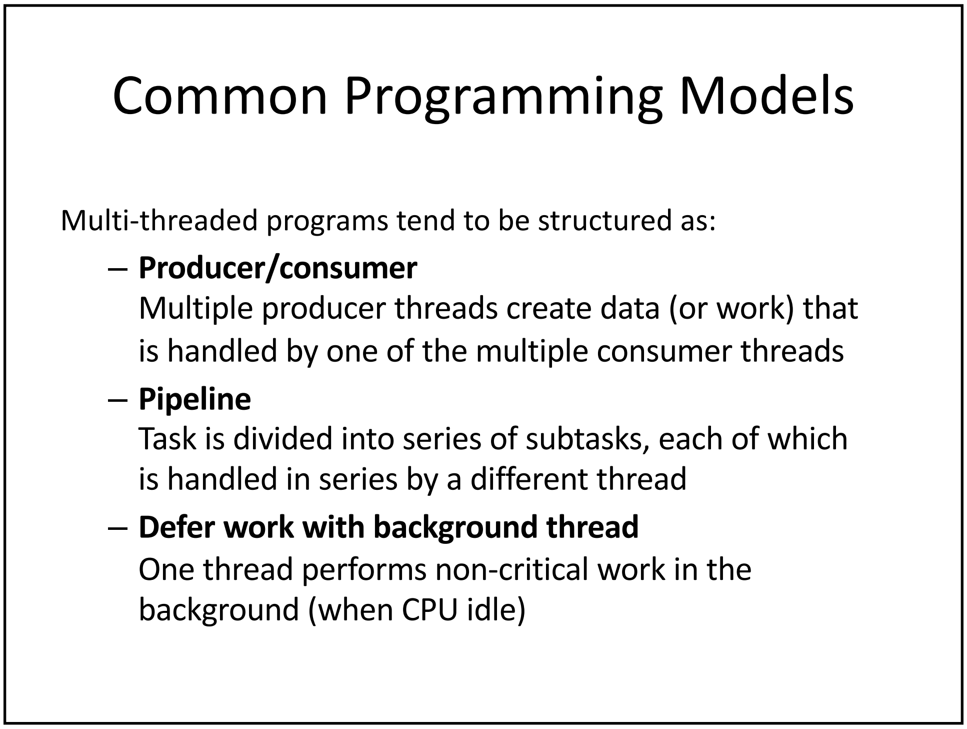
Two-level

Paper Discussion

- Why are threads cheaper than processes?
- How is IPC performed using threads?
- Why is synchronization between threads needed?
- Two creation approaches: create ALL threads or create only CALLING thread; difference?
- What is “thread-local storage”?
- What are bound threads and why are they useful?
- Why is signaling challenging?

Pthreads (POSIX 1003.1c)

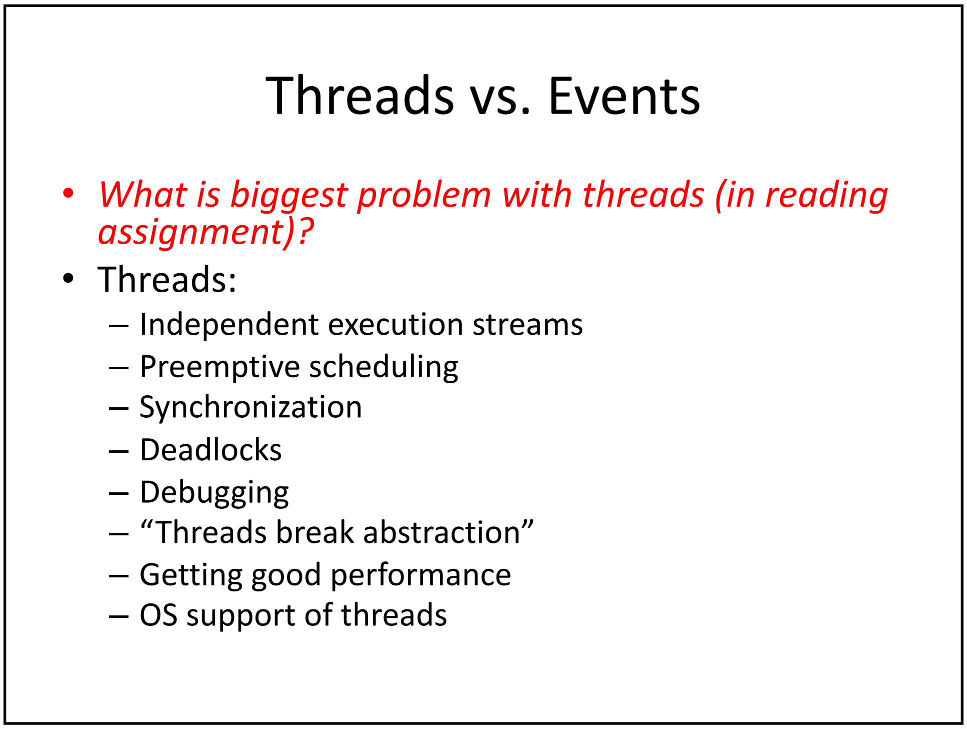
```
#include <stdio.h>
#include <pthread.h>
void printMsg(char* msg) {
    int status = 0;
    printf(“%s\n”, msg);
    pthread_exit(&status);
}
int main(int argc, char** argv) {
    pthread_t thrdID;
    int* status = (int*)malloc(sizeof(int));

    printf(“creating a new thread\n”);
    pthread_create(&thrdID, NULL, (void*)printMsg, argv[1]);
    printf(“created thread %d\n”. thrdID);
    pthread_join(thrdID, &status);
    printf(“Thread %d exited with status %d\n”, thrdID, *status);

    return 0;
}
```



Common Programming Models

Multi-threaded programs tend to be structured as:

- **Producer/consumer**
  Multiple producer threads create data (or work) that is handled by one of the multiple consumer threads
- **Pipeline**
  Task is divided into series of subtasks, each of which is handled in series by a different thread
- **Defer work with background thread**
  One thread performs non-critical work in the background (when CPU idle)



Threads vs. Events

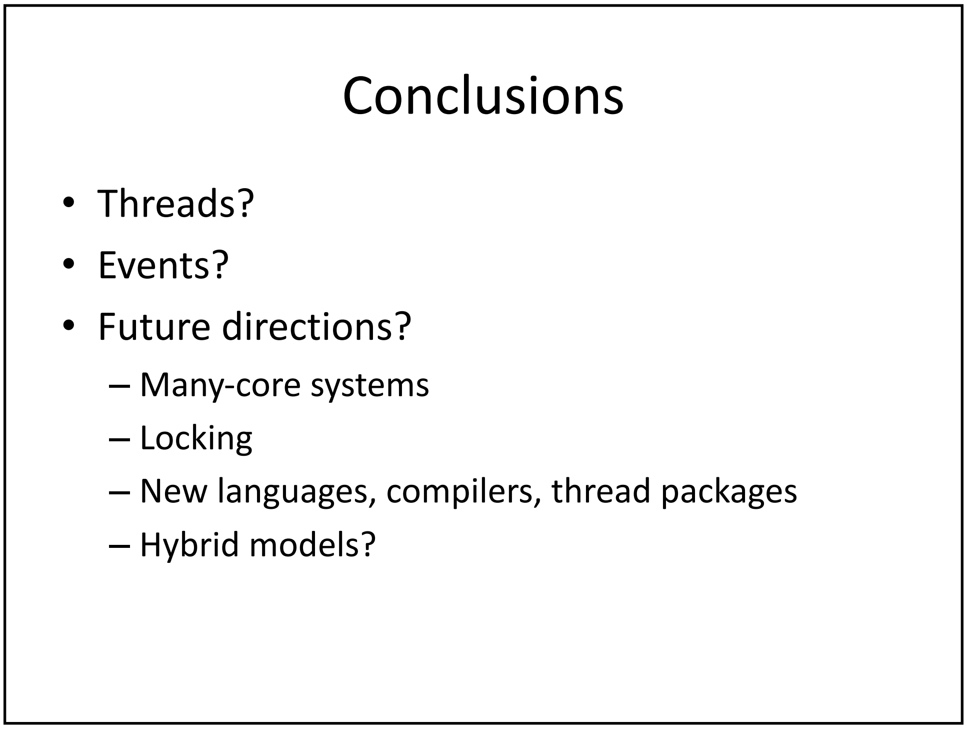
- *What is biggest problem with threads (in reading assignment)?*
- Threads:
  - Independent execution streams
  - Preemptive scheduling
  - Synchronization
  - Deadlocks
  - Debugging
  - “Threads break abstraction”
  - Getting good performance
  - OS support of threads

Threads vs. Events

- Events:
  - No CPU concurrency
  - Callbacks; event handlers
  - No preemption
  - Long-running handlers
  - State across handler invocations
  - Debugging
  - Overheads
  - Portability

Problems with Threads (Paper)

- Performance
  - Poor design; not intrinsic properties
- Control flow
  - Complicated control flow patterns are rare (call/return most common)
- Synchronization
  - Cooperative multitasking (no preemption)
- State management
  - Minimize live stack (dynamic stack growth and live state management)
- Scheduling
  - Event scheduling tricks can be applied to threads too



Conclusions

- Threads?
- Events?
- Future directions?
  - – Many-core systems
  - – Locking
  - – New languages, compilers, thread packages
  - – Hybrid models?