

Processes and Threads

Marie Roch

Tanenbaum 2-2.2, 2.4

Policy vs. Mechanism

- Policy – Decisions about what should be done.
- Mechanism – Algorithms and data structures that *implement* policy

Processes

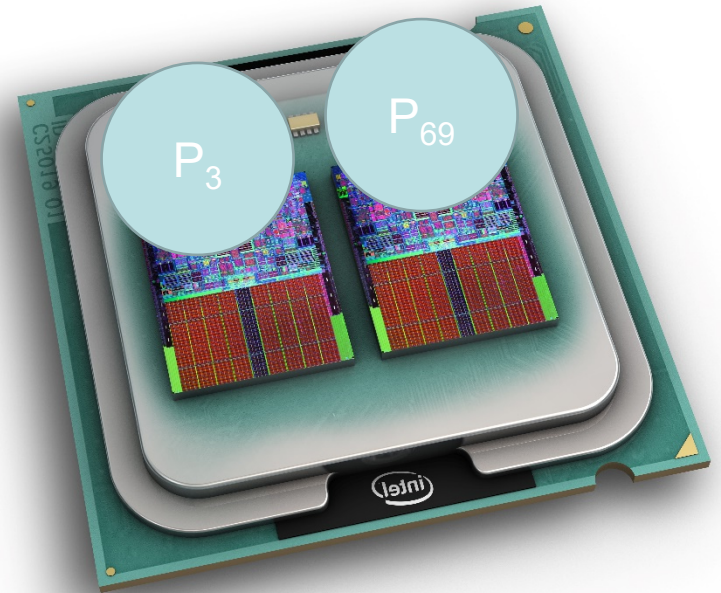
- Process → program in execution
- Processes
 - are assigned resources
 - run in user or kernel mode
- Things we do with resources
 - share
 - acquire
 - release

Multiprogramming

- Multiple processes on a single computer
- Scheduling algorithm selects which processes are allocated CPU



- Don't know when scheduled next...
avoid loops for timing

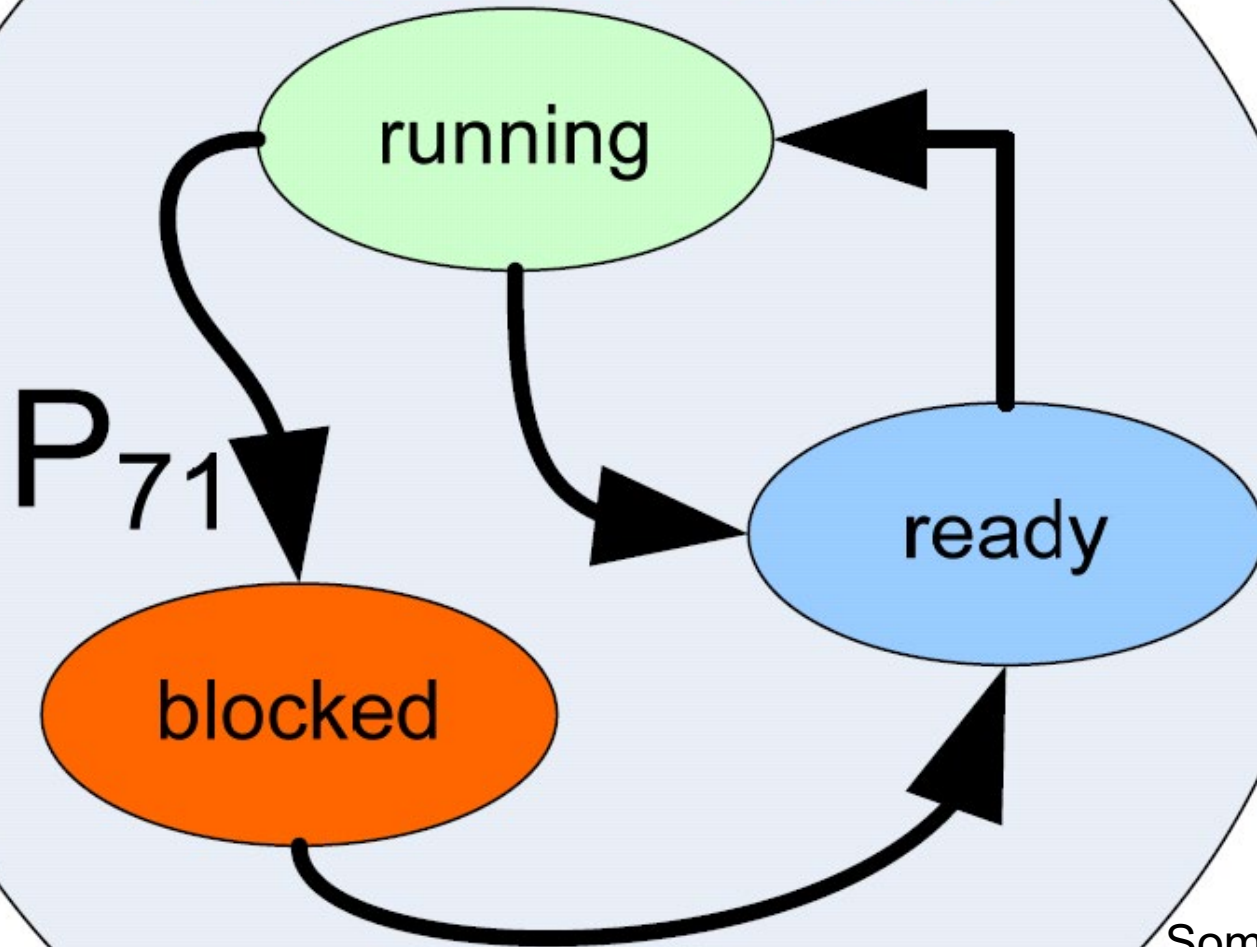


Multiprogramming

- How do we transition from one process to another?
 - cooperative multitasking
processes explicitly yield
 - preemptive multitasking
forced release of processor resource based
on external conditions



Process State



Some models use:
initializing
terminating

Process life cycle

- initialization – created by another process
- running / blocked / ready – Process is active

- termination

- Voluntary

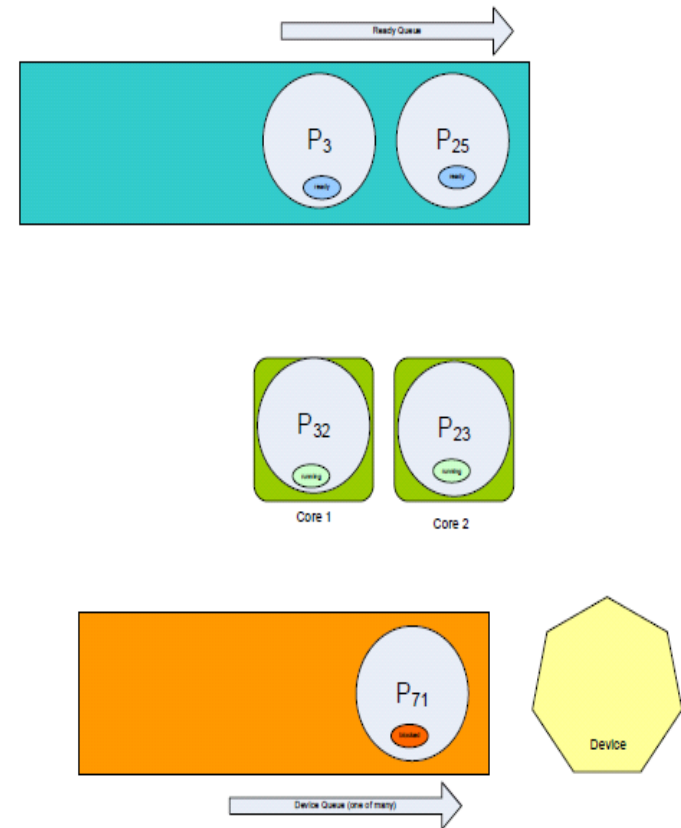
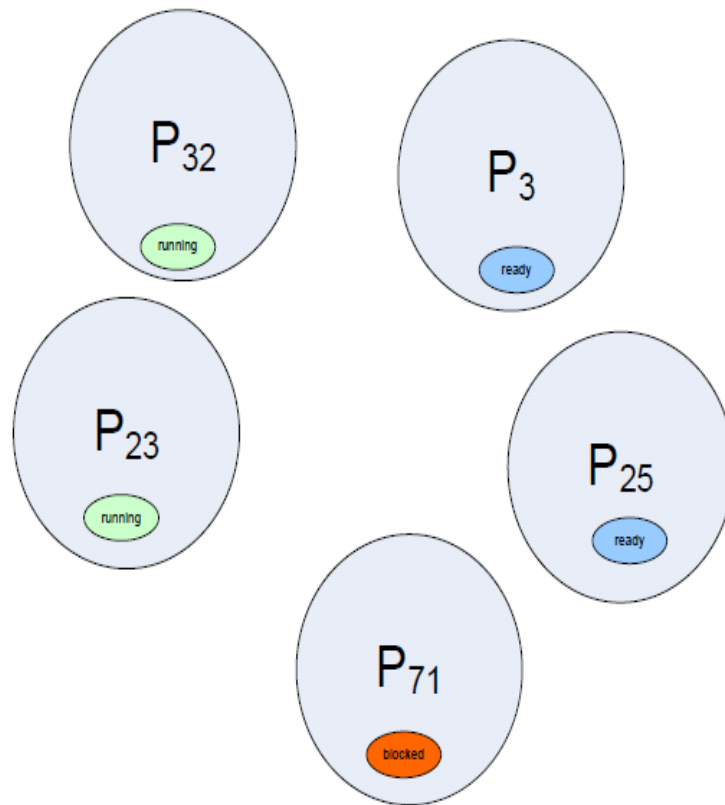
- error
- complete

- Involuntary

- unhandled exception
- killed by another process



Process State



Queuing diagram

Process Implementation

- Process control block (PCB)
 - kernel data structure representing processes
 - frequently implemented as a fixed size array
- Process control block contains
 - state
 - what else?

Typical PCB

Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment info Pointer to data segment info Pointer to stack segment info	Root directory Working directory File descriptors User ID Group ID

Process types

- Background – no user supervision
- Interactive – User I/O
- Daemon – Special background processes that provide services



Program thread

- Thread of execution
 - Stream of instructions being executed
 - CPU registers
 - Stack
 - current procedure calls
 - local variables
- Can we have more than one thread?

Multithreading

- Multiple threads (aka lightweight processes)
- Threads within process share resources:
 - files, heap, and any other allocated resources
 - and are allocated the CPU, much like processes
- Thread control block
 - Keeps registers, PC, PSW and state

Multithreading dangers

- Threads can
 - overwrite each others' stacks
 - access data structures in transient states
 - change heap values
 - access resources in unexpected interleaving
 - ...

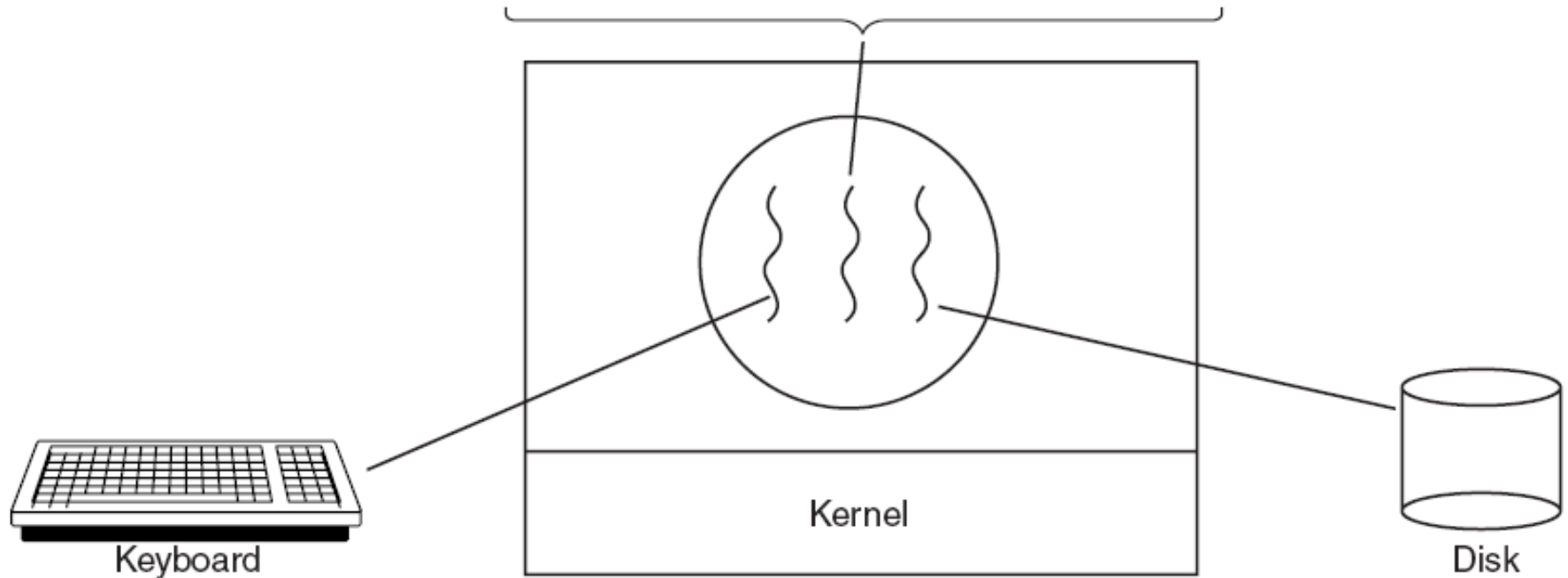


Lost in Space, Irwin Allen Productions

Okay, so why bother?

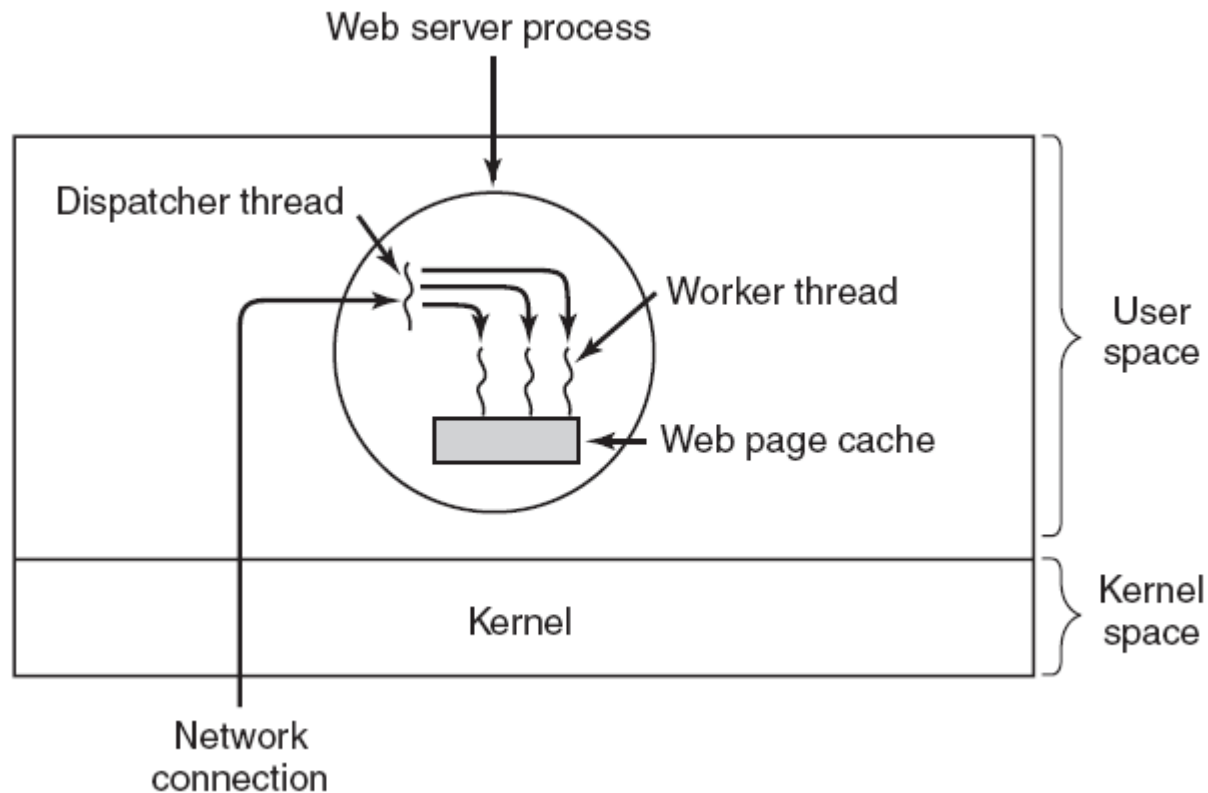
Threads are convenient

Four score and seven years ago, our fathers brought forth upon this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war testing whether that	nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field as a final resting place for those who here gave their	lives that this nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we cannot dedicate, we cannot consecrate we cannot hallow this ground. The brave men, living and dead,	who struggled here have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember, what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated	here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us, that from these honored dead we take increased devotion to that cause for which	they gave the last full measure of devotion, that we here highly resolve that those dead shall not have died in vain that this nation, under God, shall have a new birth of freedom and that government of the people, for the people
---	--	---	---	---	---



Threads are convenient

A multithreaded Web server.



Creating a thread

- Implementation dependent
- POSIX implementation (man pages/FAQ for details)
 - headers: `<pthread.h>` `<sched.h>`
 - `pthread_create(...)` – Create a thread
 - `sched_yield(...)` – Next thread runs
 - `pthread_exit(...)` – Terminate thread
 - `pthread_join(...)` – Wait for specific thread to exit
 - `pthread_attr_init(...)` – Initialize options structure to be passed to `pthread_create`

POSIX thread example

```
int main(int argc, char *argv[])
{
    /* The main program creates 10 threads and then exits. */
    pthread_t threads[NUMBER_OF_THREADS];
    int status, i;

    for(i=0; i < NUMBER_OF_THREADS; i++) {
        printf("Main here. Creating thread %d0, i);
        status = pthread_create(&threads[i], NULL, print_hello_world, (void *)i);

        if (status != 0) {
            printf("Oops. pthread_create returned error code %d0, status);
            exit(-1);
        }
    }
    exit(NULL);
}
```

See the course FAQ for a concrete example.

POSIX thread example

- Previous example needs header:

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

- Compilation on a linux box

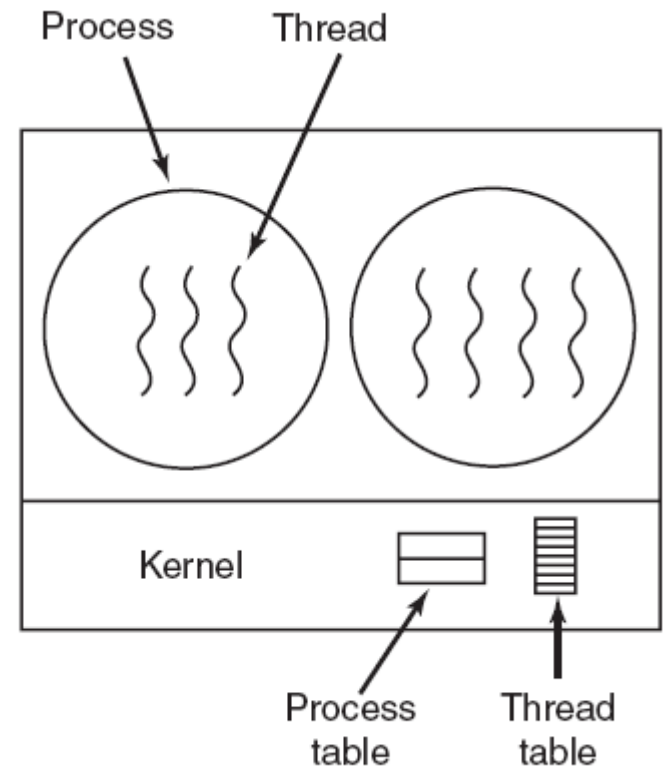
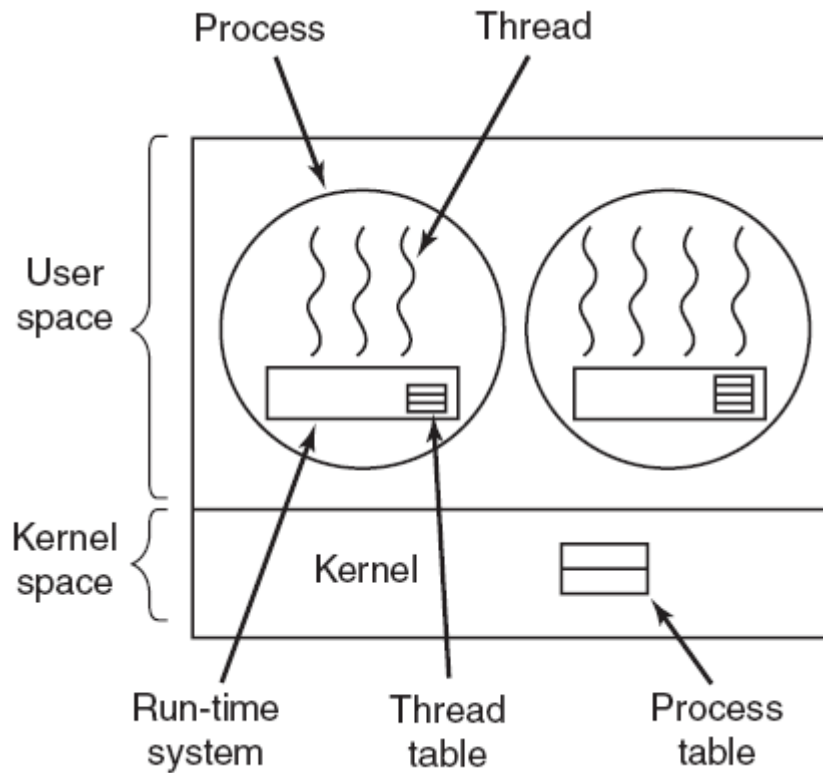
```
gcc -o threadeg threadeg.c -l pthread -l rt
```

Note: gcc puts library flags **after** list of files (rarely done) and the order of libraries is important.

Thread mechanisms

- user-level threads
 - implemented via a user library
 - scheduling occurs in user code
- kernel-level threads
 - part of OS implementation
 - data structures are maintained in kernel code

User- vs. Kernel- mode threads

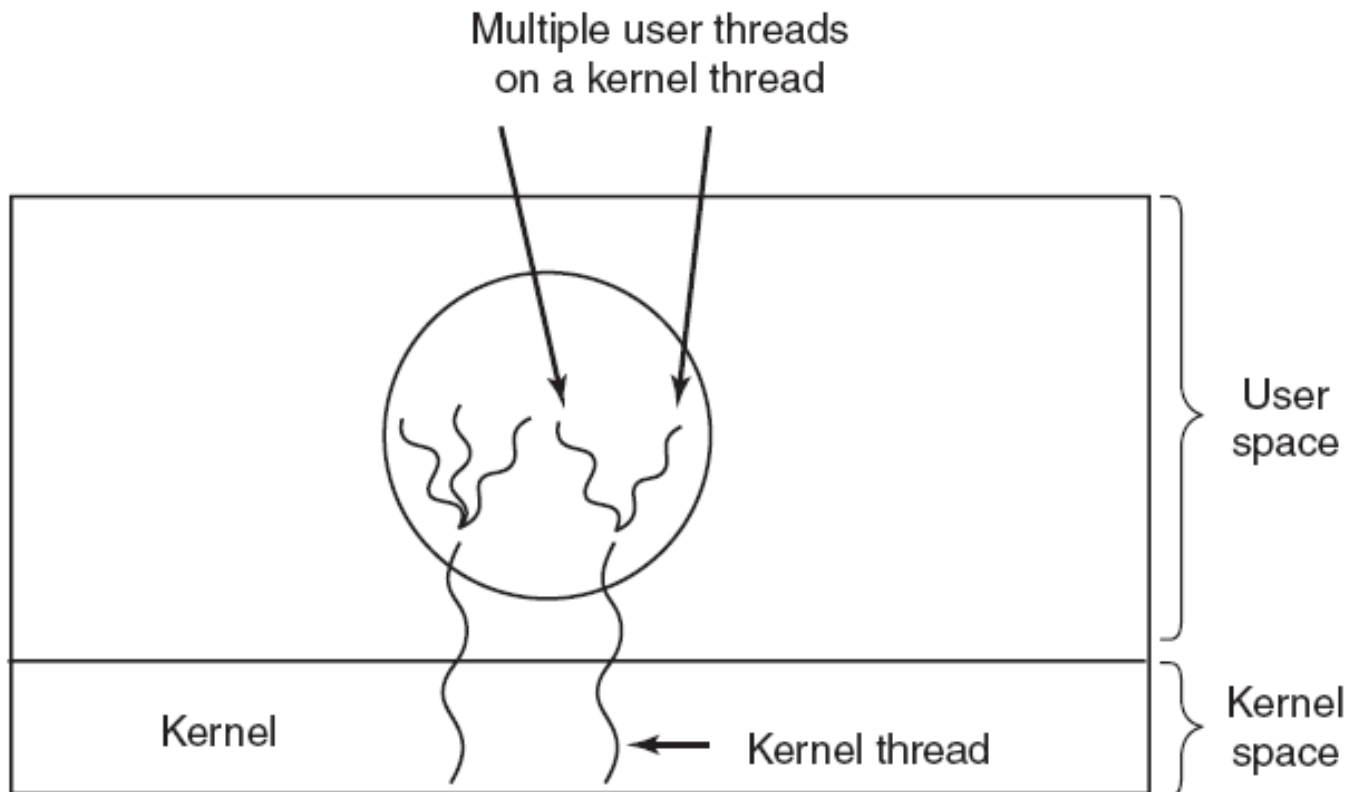


User- vs. Kernel- mode threads

- Kernel-mode
 - Kernel schedules threads, not processes
 - Thread operations and switches require shift to kernel mode (\$\$\$)
- User-mode
 - Kernel unaware of user threads
What happens when one thread blocks?
 - Very fast thread operations

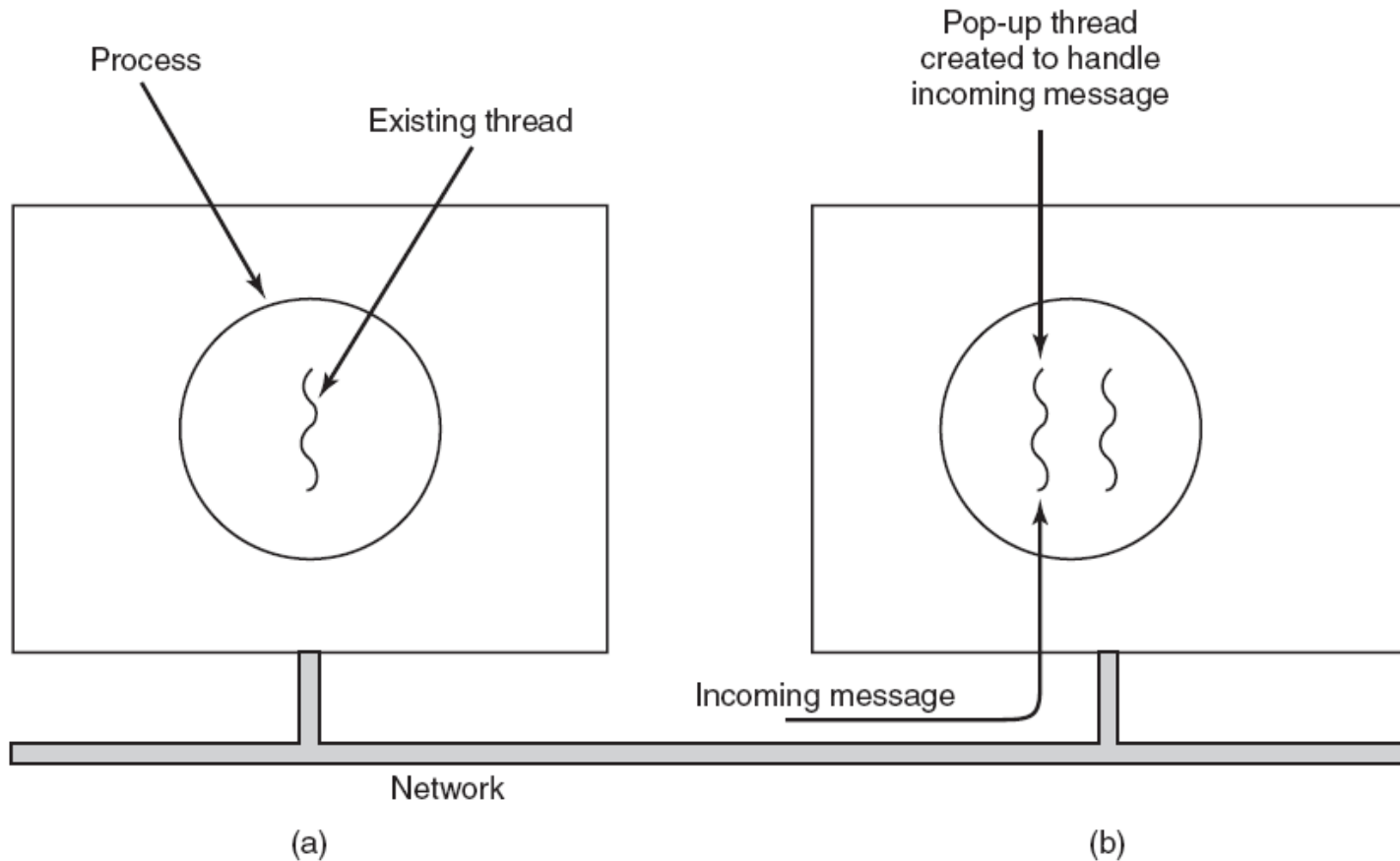
Hybrid thread design

- User-threads mapped onto kernel-threads
- Best of both worlds



Pop-up threads

- Dynamic creation of threads to handle events



Threads

- Suppose a thread calls a function that sets a global return code (e.g. UNIX errno)
- Can we run into problems?