OPERATING SYSTEMS

by
Marwa Yusuf

Lecture 4
Sunday 1-11-2020

Chapter 2 (2.2.1 to 2.2.3)

Processes and Threads

Threads

- Traditional: single process with single thread.
- Now: quasi-parallel threads sharing address space (miniprocess).
- Reasons:

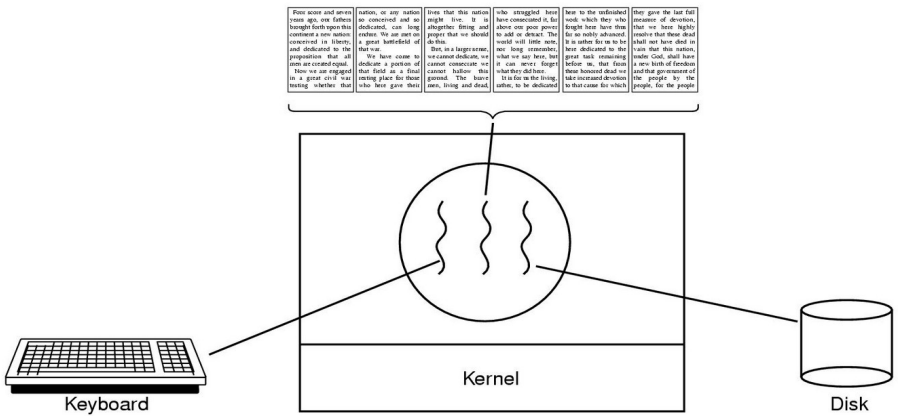
1) Parallelizable activities within an application (some block).
   - Easier programming model, separate processes not suitable.
2) Lighter, easier, faster (10-100 times).
   - Dynamic and rapid change in threads number.
3) Performance: cpu and I/O mix → faster application.
4) With multiple CPUs → real parallelism.

Threads Examples

- Word processor:
  - Editing large book 800 pages in 1 file, delete in page 1, go to page 600 --> delay.
  - Threads for : user interaction + reformatting + saving
  - 1 process & 3 processes won here. Why?

1 process: blocking interaction while saving or interrupt-driven complex model.

3 processes: no shared address space (document).



Threads Examples (cont.)

- Spreedsheet:
  - Threads: computations, interaction, saving.
- Large data applications:
  - Threads: reading (to in buffer), processing (to out buffer), writing.
  - Useful only if blocking a thread not entire process.

Threads Examples (cont.)

- Web server:
  - Cache of common pages (ex: home)
  - Threads: dispatcher, (awaken) worker
- Single thread server on a dedicated machine → idle CPU.
- Non-blocking read with a table for request and signals or interrupts for replies, mimicking threads the hard way, losing the sequential model. (finite state machines).

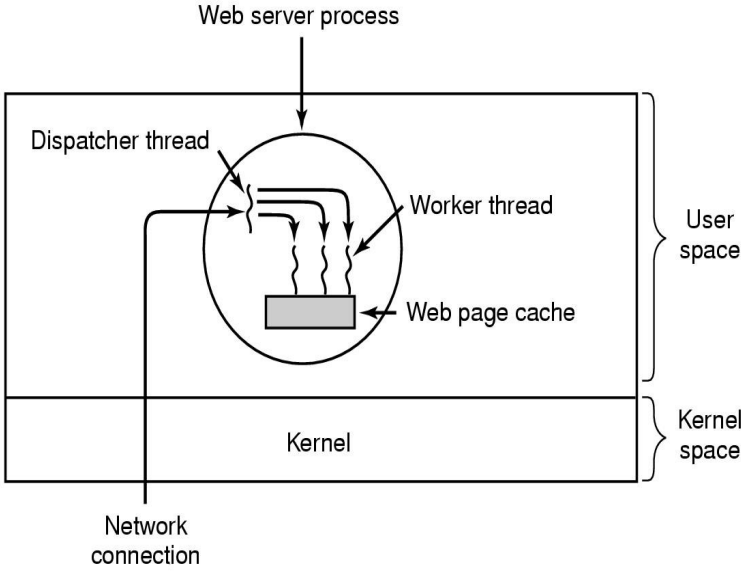
```
while (TRUE) {
  get_next_request(&buf);
  handoff_work(&buf);
}
```

(a)

```
while (TRUE) {
  wait_for_work(&buf)
  look_for_page_in_cache(&buf, &page);
  if (page_not_in_cache(&page)
      read_page_from_disk(&buf, &page);
  return_page(&page);
}
```

(b)

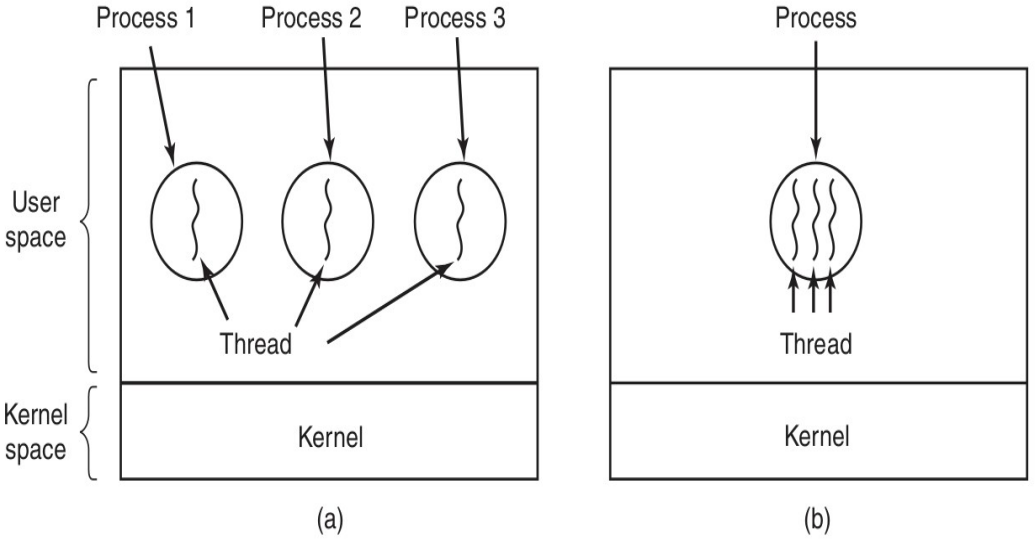
Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts



The Classical Thread Model

- Resource grouping vs. execution.
- A **process** groups related resources to manage easily:
  - Code, data, files, children, pending alarms, signal handlers, accounting info, … etc.
- A **thread** is an entity of execution:
  - Program counter, registers, stack.
- **Multithreading**:
  - multiple threads (lightweight processes)
  - Analogy: multiple processes in a computer sharing CPU, memory, ...etc. multiple threads in an application sharing address space, … etc.
  - Some CPUs support threads directly → faster switching.

The Classical Thread Model (cont.)



- CPU switches rapidly between threads → illusion of parallelism.
- Threads share global variables, no protection:
  - Not possible, not necessary. They are from the same application, cooperating not fighting.
  - Use processes for unrelated jobs, threads for parts of the same job.
- Thread states: running, blocked (waiting for external event or another thread), ready, terminated.
- Each thread calls different procedures, hence has its own stack.

The Classical Thread Model (cont.)

- Usually, process starts with 1 thread, which creates others.
- `thread_create` with the starting procedure name, returning the new thread's thread identifier.
- Sometimes, thread hierarchy, but not usually.
- `thread_exit`, `thread_join`, `thread_yield`, waiting, announcing.
- Complications: ex: does the child process has the same number of threads as the parent process? What if a thread closes a file being read by another thread? What if 2 threads start allocating memory?
  - Be careful when programming multithreaded apps.

POSIX Threads

- **Pthreads**: IEEE standard:
  - Over 60 function calls.
  - Each thread has: identifier, set of registers (including PC), a structure of attributes (stack size, scheduling parameters, others).

pthread_create	Takes procedure name, return tid
pthread_exit	Stop thread, release stack
pthread_join	Wait for another thread to finish, tid as a parameter
pthread_yield	Voluntarily release CPU
pthread_attr_init	Create attributes structure, initialize it by defaults
pthread_attr_destroy	Remove, free memory, does not affect the thread