Chapter 3: Process concept (cont)

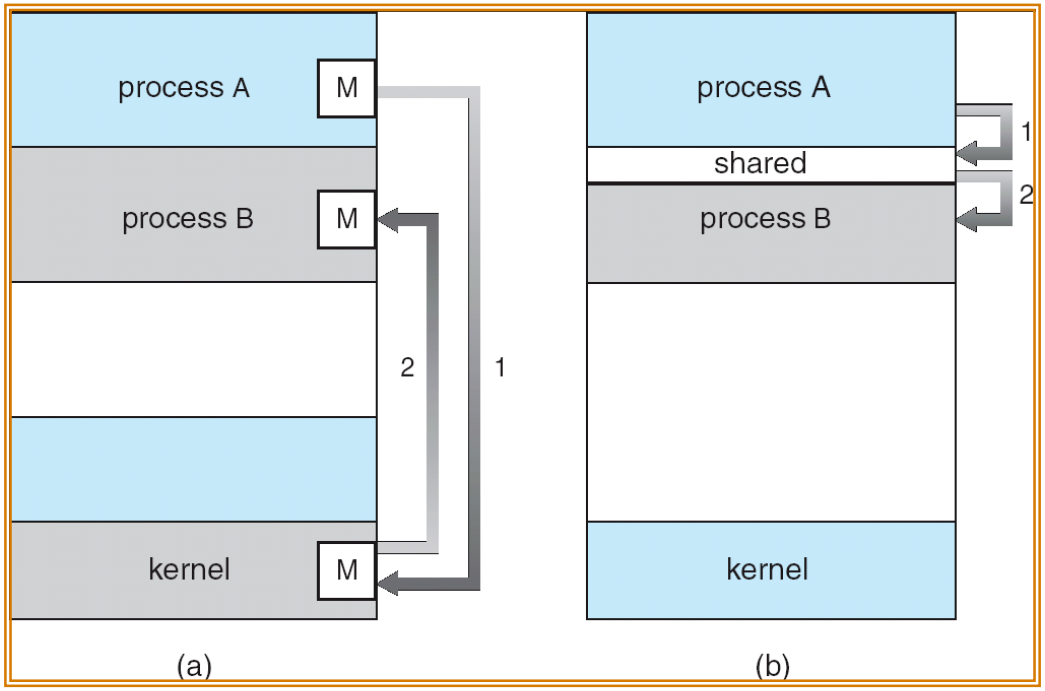
- So far, we discussed notion of processes, how they are represented (PCB), created (fork) and terminated (exit). Processes are isolated from each other. We'll see how they communicate with each other
- Chapter 4: Threads

Interprocess communications

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

IPC mechanisms

- Shared memory
  - Create shared memory region
  - When one process writes into this region, the other process can see it and vice versa
- Message passing
  - Explicitly send() and receive()



Producer/consumer using shared memory

- Shared data

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Insert/Remove methods

```
while (true) {
/* Produce an item */
 while (((in = (in + 1) % BUFFER SIZE count)  == out)
        ;   /* do nothing -- no free buffers */
   buffer[in] = item;
   in = (in + 1) % BUFFER SIZE;
}

while (true) {
   while (in == out)
       ; // do nothing -- nothing to consume

   // remove an item from the buffer
   item = buffer[out];
   out = (out + 1) % BUFFER SIZE;
   return item;
}
```

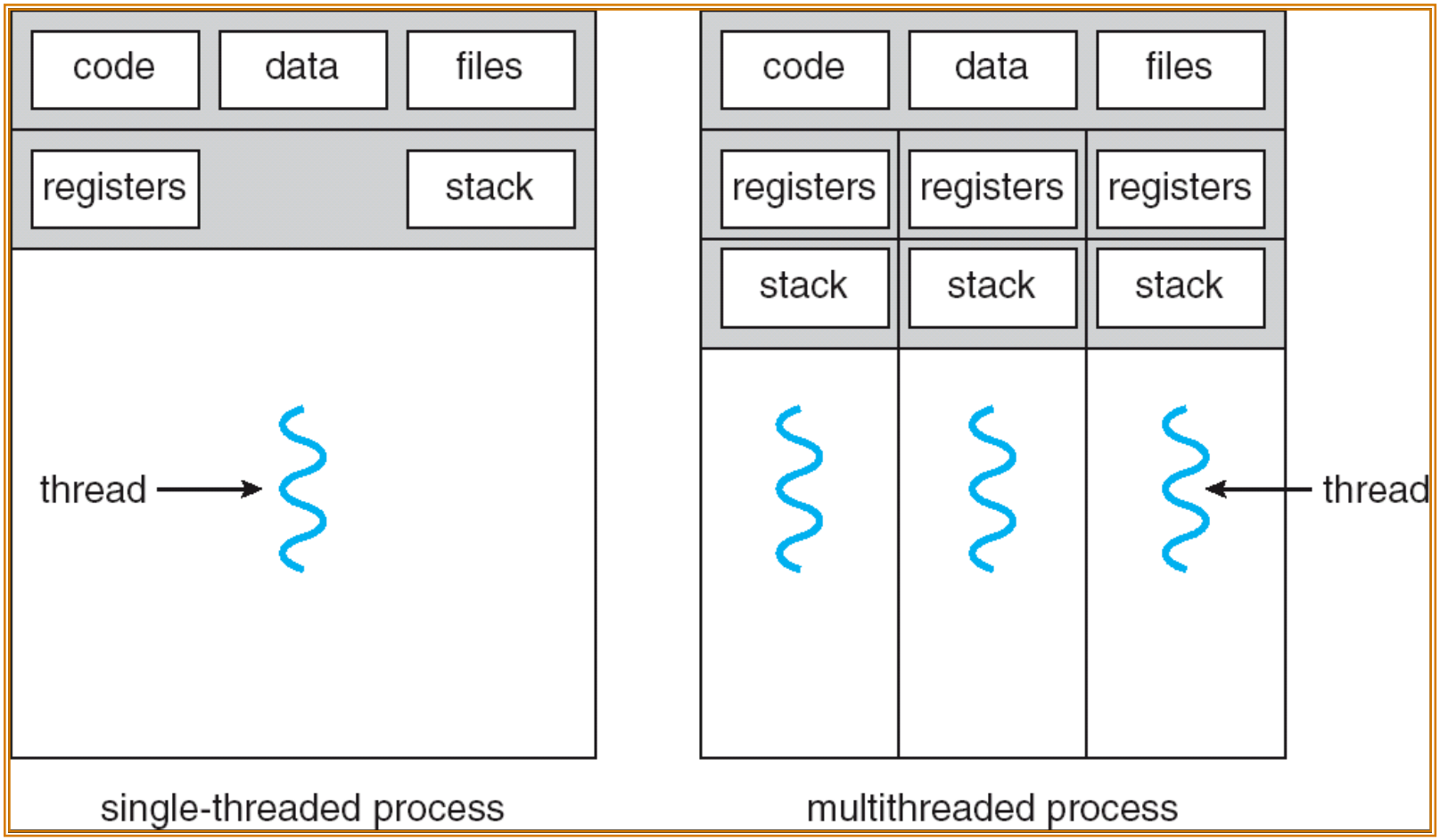
Message passing

- Requires ways to name objects (same machine or different machine).
- Communications can be synchronous or asynchronous.
- May need to buffer messages that are not ready to be read

Chapter 4: Threads

- Thread is the basic unit of CPU utilization. So far, our implicit assumption was that each process has a single thread of execution. However, each process can have multiple threads of execution, potentially working on more than one thing at the same time
- Threads in the same process share text, data, open files, signals and other resources. Each thread has its own execution context and stack.

Single and Multithreaded Processes



Benefits

- Responsiveness - Interactive applications can be performing two tasks at the same time (rendering, spell checking)
- Resource Sharing - Sharing resources between threads is easy (too easy?)
- Economy - Resource allocation between threads is fast (no protection issues)
- Utilization of MP Architectures - seamlessly assign multiple threads to multiple processors (if available). Future appears to be multi-core anyway.

Thread types

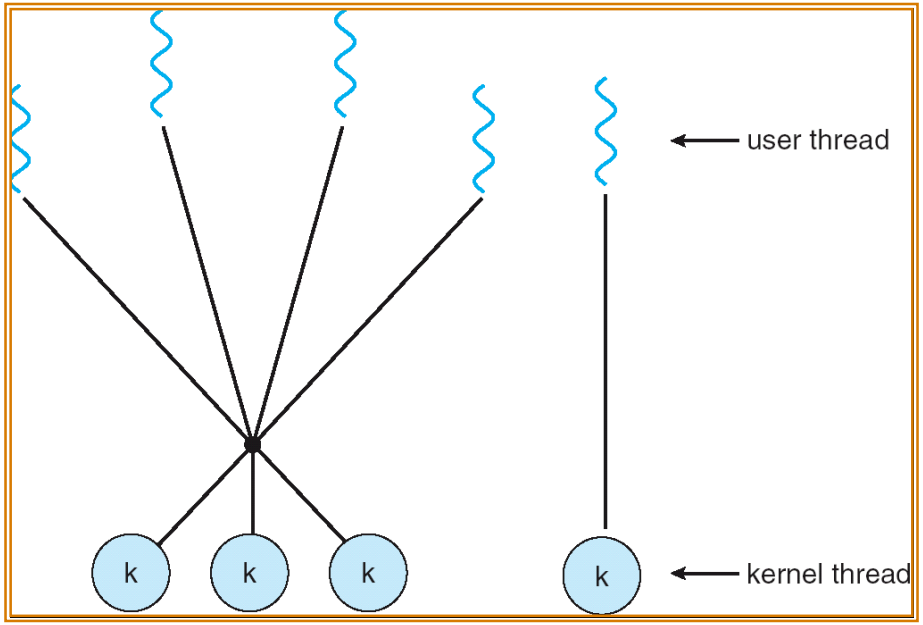
- User threads: thread management done by user-level threads library. Kernel does not know about these threads
  - Three primary thread libraries:
    - POSIX Pthreads
    - Win32 threads
    - Java threads
- Kernel threads: Supported by the Kernel and so more overhead than user threads
  - Examples: Windows XP/2000, Solaris, Linux, Mac OS X
- User threads map into kernel threads

Multithreading Models

- Many-to-One: Many user-level threads mapped to single kernel thread
  - If a thread blocks inside kernel, all the other threads cannot run
  - Examples: Solaris Green Threads, GNU Pthreads
- One-to-One: Each user-level thread maps to kernel thread
- Many-to-Many: Allows many user level threads to be mapped to many kernel threads
  - Allows the operating system to create a sufficient number of kernel threads

Two-level Model

- Similar to M:M, except that it allows a user thread to be bound to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



Pthreads library

- Discuss the sample pthread program