Operating Systems

Jin-Soo Kim (*jinsookim@skku.edu*)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Standalone Applications

- Often no OS involved
- One large loop
- Microcontroller-based embedded systems

"Real-Time" OS

- Scheduling of threads (or tasks)
  - Creation and termination of threads
  - Timing of thread activations
- Synchronization
  - Semaphores and locks
- Input and output
  - Interrupt handling with predictable latency
- A small kernel size

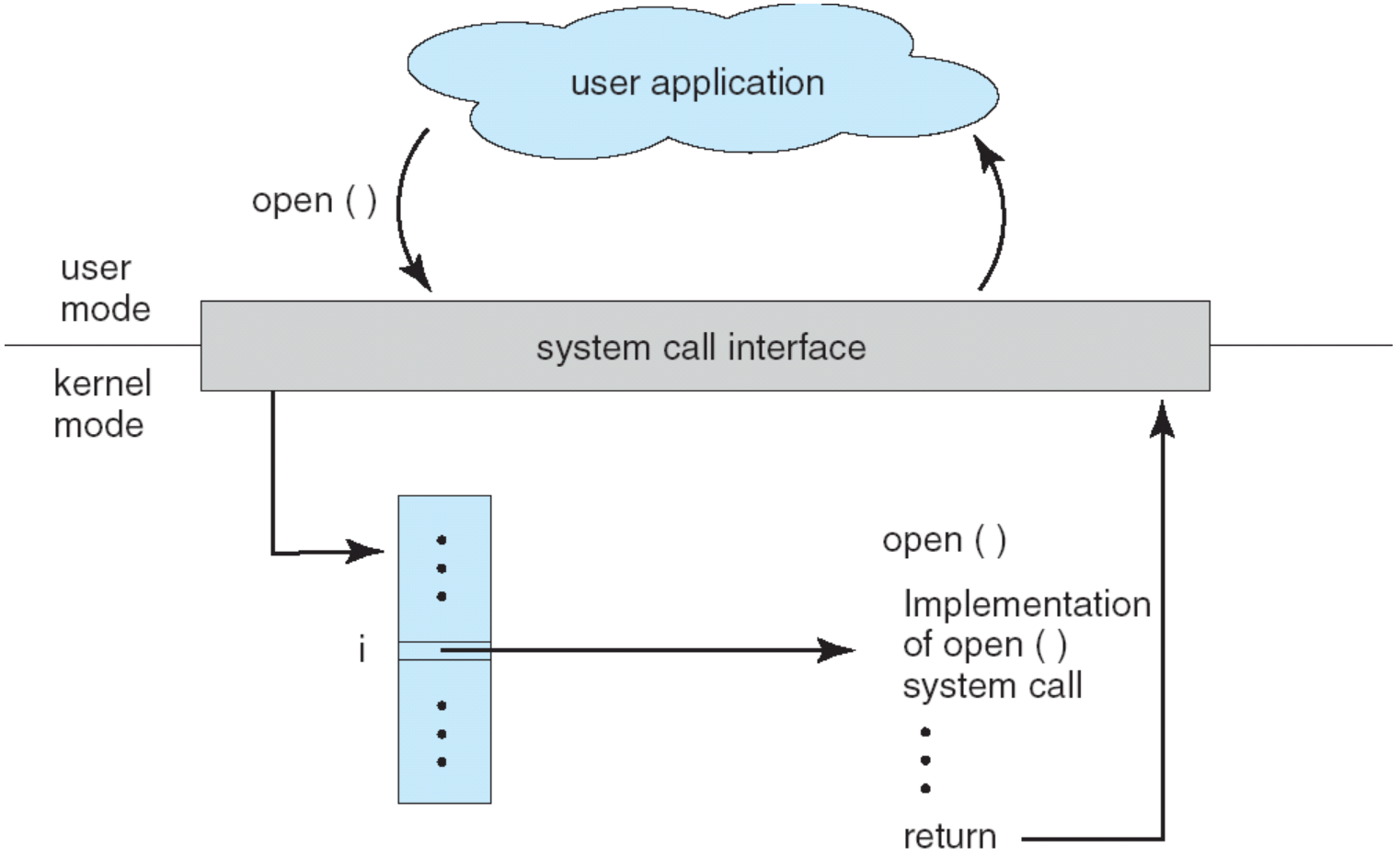
Full-Fledged OS

- Applications are huge and complex
- Multiple processes
- Virtual memory
  - Requires MMU-enabled CPU
- File systems with persistent storage
- Networking: TCP/IP stack
- Security: User vs. kernel space
- Windowing & GUI support

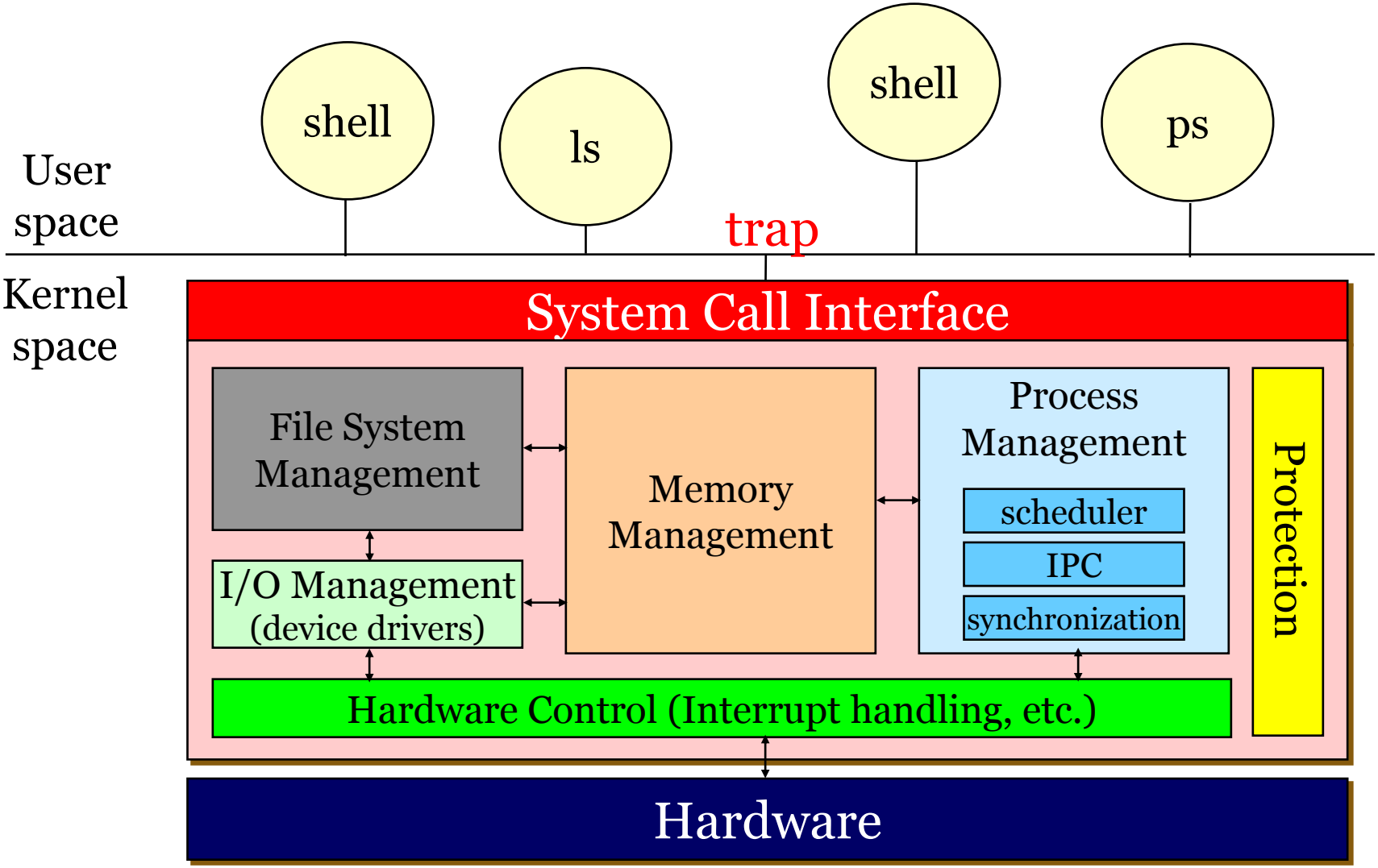
What is an OS?

- Provides an abstract view of the underlying computer system
  - Processors → Processes, Threads
  - Memory → Address space (virtual memory)
  - Storage, I/O → Files
- Manages various resources of a computer system
- Highly-concurrent, event-driven software

System Calls



Kernel Internals



Processes and Threads

Abstracting CPUs

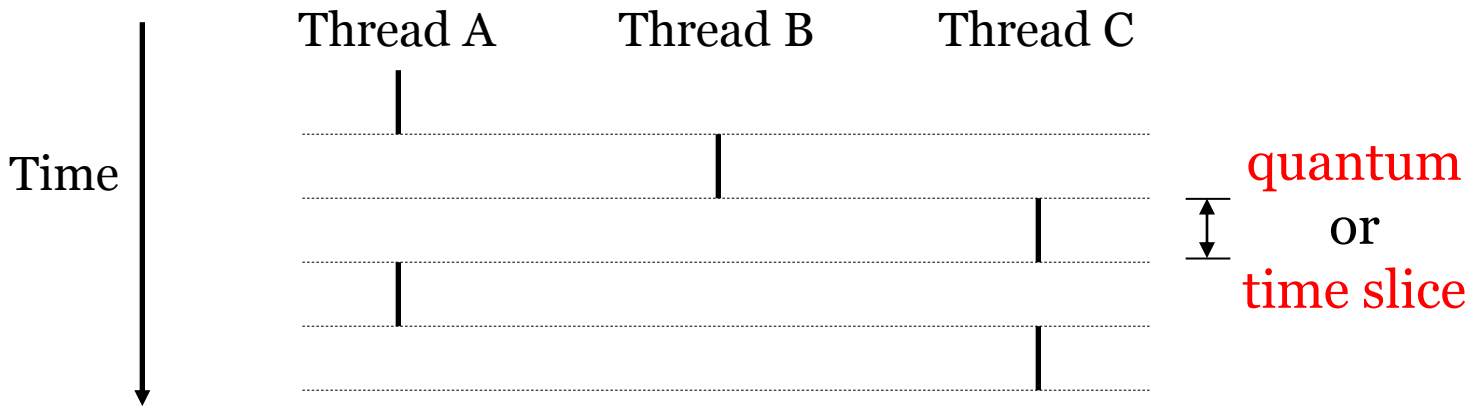
- Process
  - An instance of a program in execution
  - Memory protection
  - Resource allocation unit
- Thread
  - A sequential flow of control
  - Scheduling unit
- Task
  - Process or thread

Thread (1)

- A thread of control
- Usually consists of
  - A program counter (PC)
  - A stack to keep track of local variables and return addresses
  - Registers
- Threads share the instructions and most of its data

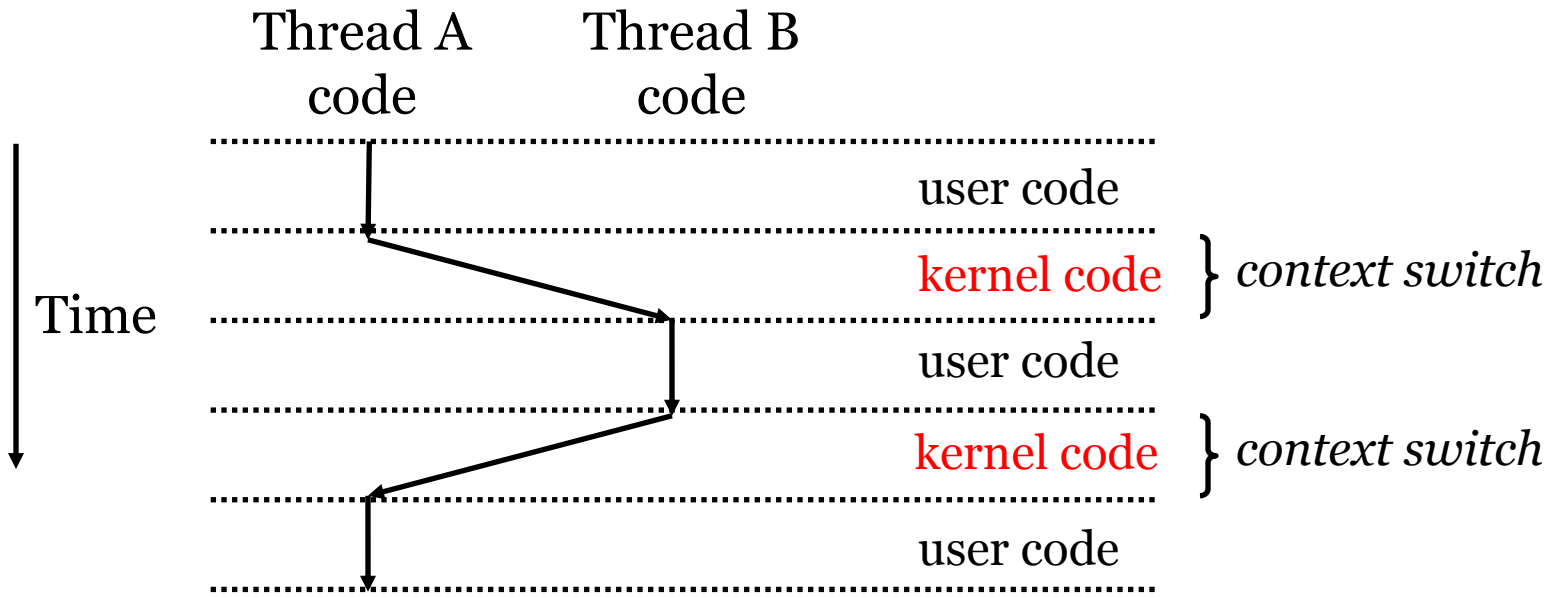
Thread (2)

- Key abstraction
  - Each thread has its own logical control flow.
  - Thread executions interleaved by the scheduler
  - What is running a thread?

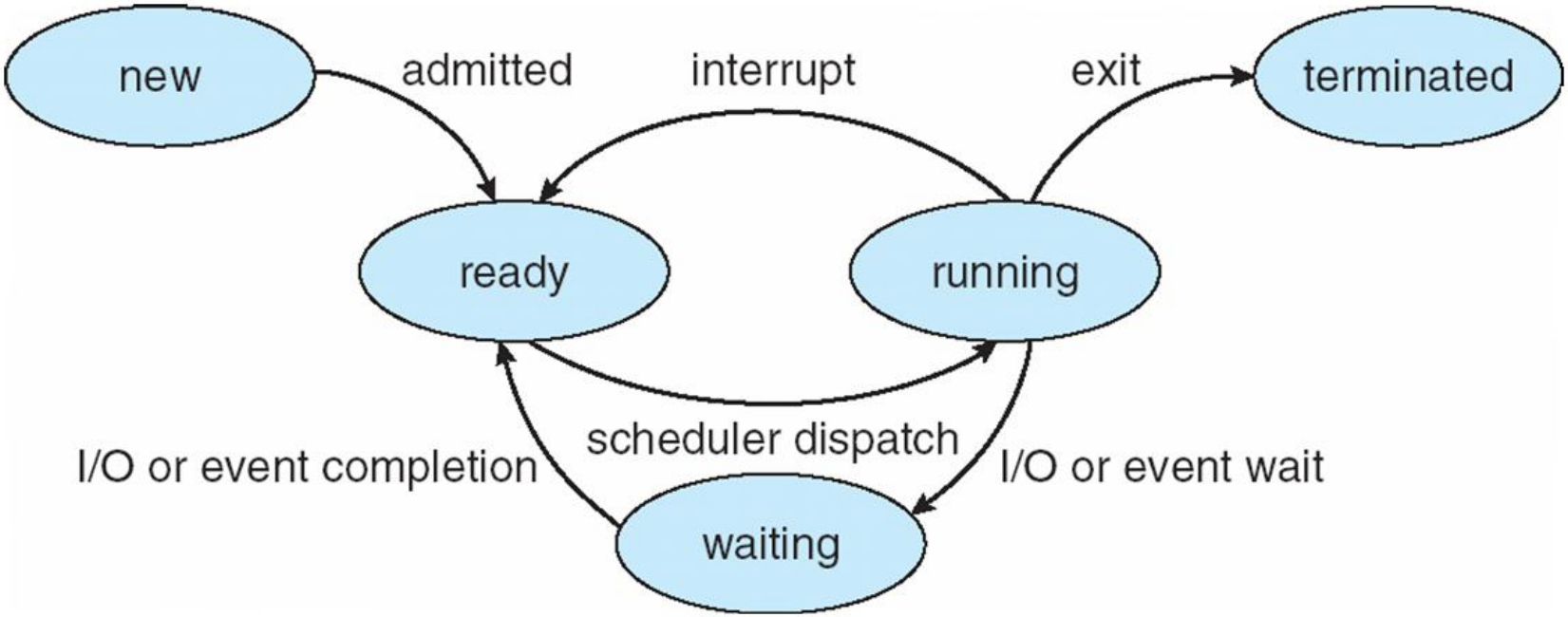


Thread (3)

- Context switching
  - Control flow passes from one thread to another via a context switch.



Thread State Transition



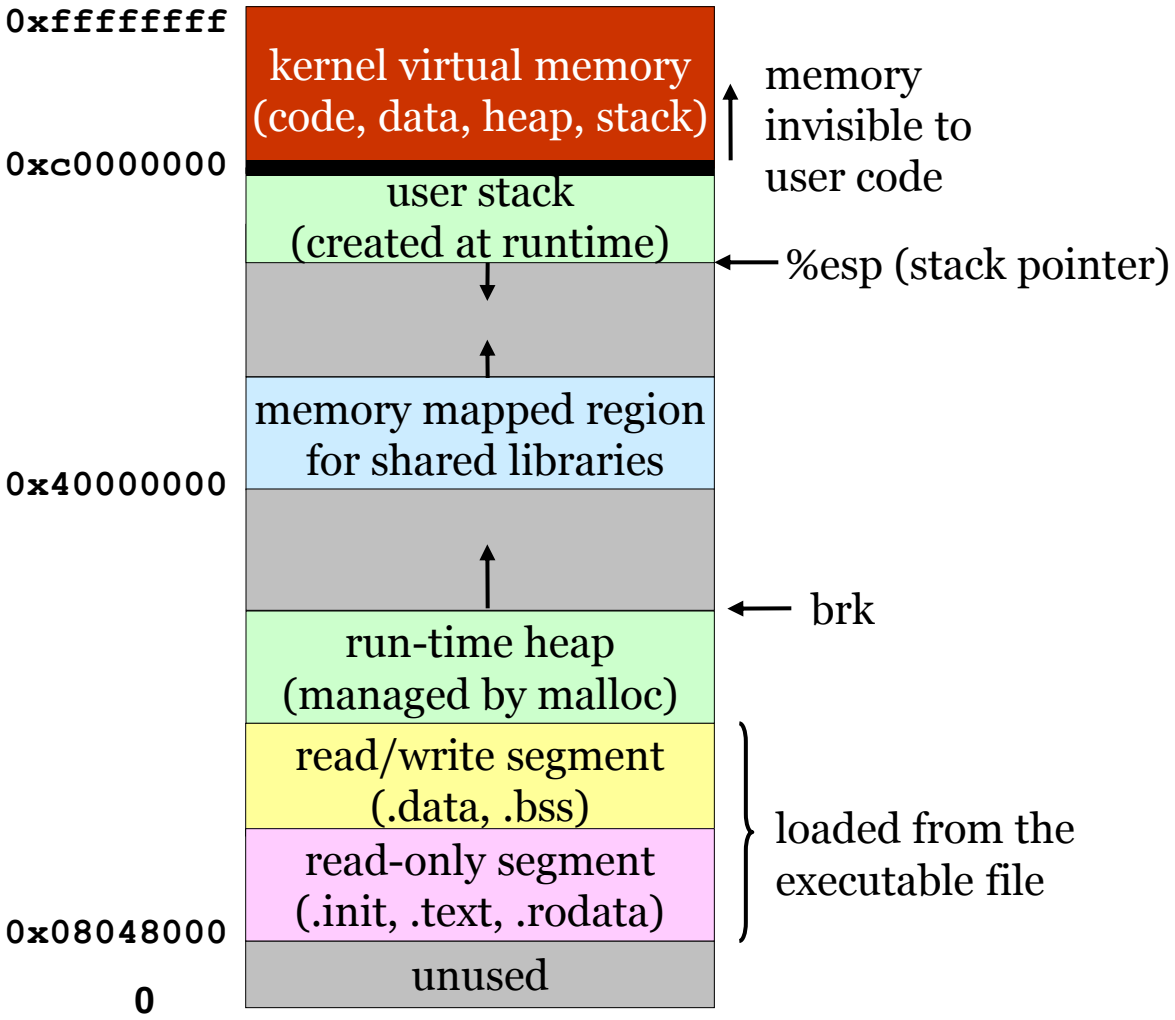
Process (1)

- An instance of a program in execution
- A process includes
  - Hardware execution state (PC, SP, registers, …)
  - OS resources (e.g., memory, open files)
  - Other information (PID, state, owner, …)
- Two key abstractions
  - Logical control flow (virtual CPU)
  - Private address space (virtual memory)

Process (2)

- Used to run a user program
- Protection domain
- Creating a new process is costly
- Inter-process communication is costly, since it must usually go through the OS

Private Address Spaces



Virtual Memory

- Allows to run programs much larger than the available physical memory size
- Simplifies memory management
- Provides memory protection

Threads vs. Processes

- A thread is bound to a single process (or a single address space)
- A process can have multiple threads
- Sharing data between threads is cheap: all see the same address space
- Threads are the unit of scheduling
- Processes are containers in which threads execute
- Processes become static, threads are the dynamic entities

Classification

# threads per addr space: / # of addr spaces:	One	Many
One	MS/DOS Early Macintosh	Traditional UNIX
Many	Many embedded OSes (VxWorks, uClinux, uC/OS-II, …)	Mach, OS/2, Linux, Windows, Mac OS X, Solaris, HP-UX

Example: µC/OS-II

What is µC/OS-II?

- A priority-based preemptive RT kernel
- Memory footprint: ~20KB
- Written mainly in C (open but not free)
- Ported to over 100 processors
- Manage up to 64 tasks
- Support 8-bit to 64-bit processors
- GUI, file system, etc. modules available
- http://www.micrium.com

Task

- A task is a simple program that thinks it has the CPU all to itself
- Each task has
  - Its own stack space
  - A priority based on its importance
- A task contains your application code
- A task is an infinite loop

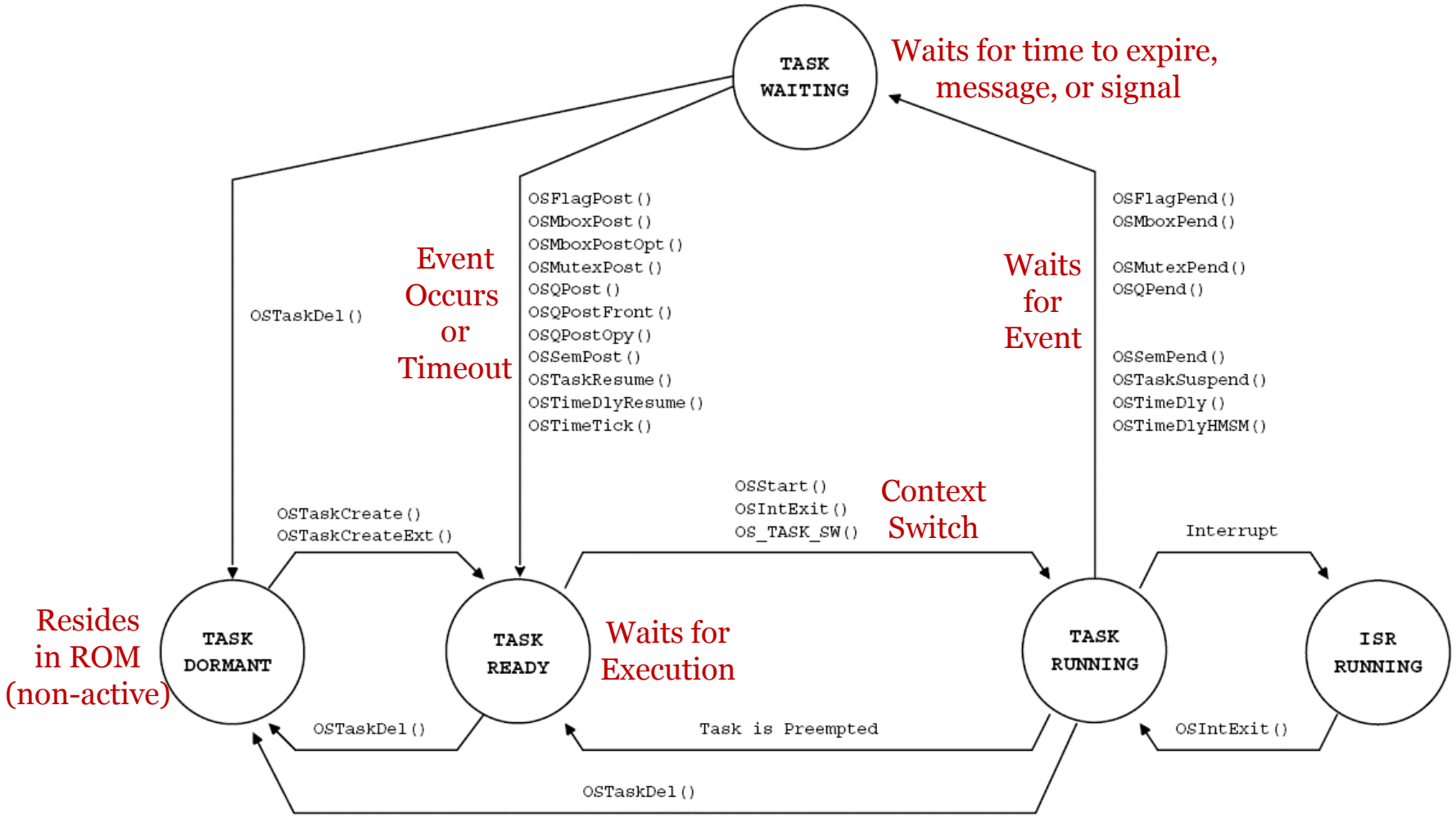
Task Example

```
void Task (void *p_arg)
{
    Do something with argument p_arg;
    Task initialization;
    for (;;) {
        Processing (your code)
        Wait for event;        /* Time to expire ..  */
                               /* Signal from ISR    */
                               /* Signal from task   */
        Processing (your code)
    }
}
```

Task Creation/Deletion

- A task is created by OSTaskCreate()
- Make the new task ready for multitasking
- The followings passed to the kernel
  - Starting address
  - Stack
  - Priority
  - Arguments passed to the task, etc.
- A task is deleted by OSTaskDel()

Task States


TASK WAITING
Waits for time to expire, message, or signal
OSTaskDel()
Event Occurs or Timeout
OSFlagPost()
OSMboxPost()
OSMboxPostOpt()
OSMutexPost()
OSQPost()
OSQPostFront()
OSQPostOpy()
OSSemPost()
OSTaskResume()
OSTimeDlyResume()
OSTimeTick()
Waits for Event
OSFlagPend()
OSMboxPend()
OSMutexPend()
OSQPend()
OSSemPend()
OSTaskSuspend()
OSTimeDly()
OSTimeDlyHMSM()
OSStart()
OSIntExit()
OS_TASK_SW()
Context Switch
OSTaskCreate()
OSTaskCreateExt()
Interrupt
Resides in ROM (non-active)
TASK DORMANT
TASK READY
Waits for Execution
TASK RUNNING
ISR RUNNING
OSTaskDel()
Task is Preempted
OSIntExit()
OSTaskDel()

Task Priority

- Max 64 priority levels
- 8 levels reserved for OS, 56 levels for user
- Low number means high priority
- Priority changed by OSTaskChangePrio()

- Suspending a task: OSTaskSuspend()
- Resuming a task: OSTaskResume()

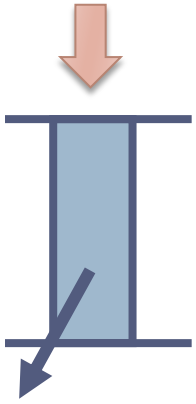
Task Synchronization

- Semaphore Mailbox Message Queue

OSSemPost()

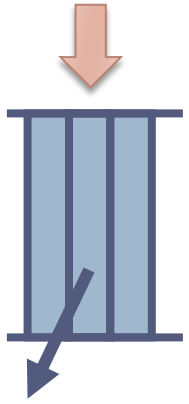
OSSemPend()

OSMboxPost()



OSMboxPend()

OSQPost()
OSQPostFront()



OSQPend()

Semaphores

- OSSemPend()
  - Counter--
  - If (counter < 0), the task is blocked and moved to the wait list
  - A time-out value can be specified
- OSSemPost()
  - Counter++
  - If (counter >= 0), the highest priority task in the wait list is removed from the wait list

Mailbox

- OSMboxPen()
  - The message in the mailbox is retrieved
  - If empty, the task is immediately blocked
  - A time-out value can be specified
- OSMboxPost()
  - A message is posted in the mailbox
  - If there is already a message, returns an error
  - The task with the highest priority is removed from the wait list and scheduled to run

Message Queues

- OSQPend()
  - A message is removed from queue
  - If empty, the task is moved to the wait list and becomes blocked
- OSQPost()/OSQPostFront()
  - A message is appended to the queue
  - OSQPost(): FIFO, OSQPostFront(): LIFO
  - The highest-priority pending task receives the message and scheduled to run, if any

Memory Management

- To reduce memory fragmentation
  - Use of memory blocks with fixed size
  - All blocks same size
  - The size of blocks is configurable
- Creation of a memory partition: OSMemCreate()
- Requesting a memory block: OSMemGet()
- Releasing a memory block; OSMemPut()