Chapter 2
Processes and Threads

2.1 Processes
2.2 Threads
2.3 Interprocess communication
2.4 Classical IPC problems
2.5 Scheduling

2.1 Process

1. Sequential Execution (单道程序系统中，一个具有独立功能的程序独占处理器直至最终结束的过程称为程序的顺序执行)

- Sequential
  - the order of program structure (maybe switch or loop)
- Close
  - Occupy all resource alone
- Reappear
  - if the initial condition be same, the result be same.

2. Concurrency Execution: Lose all the character

- Concurrency → Process
  - Multiprogramming
  - Problem:

    compete the resource;

    communication;

    collaboration ….

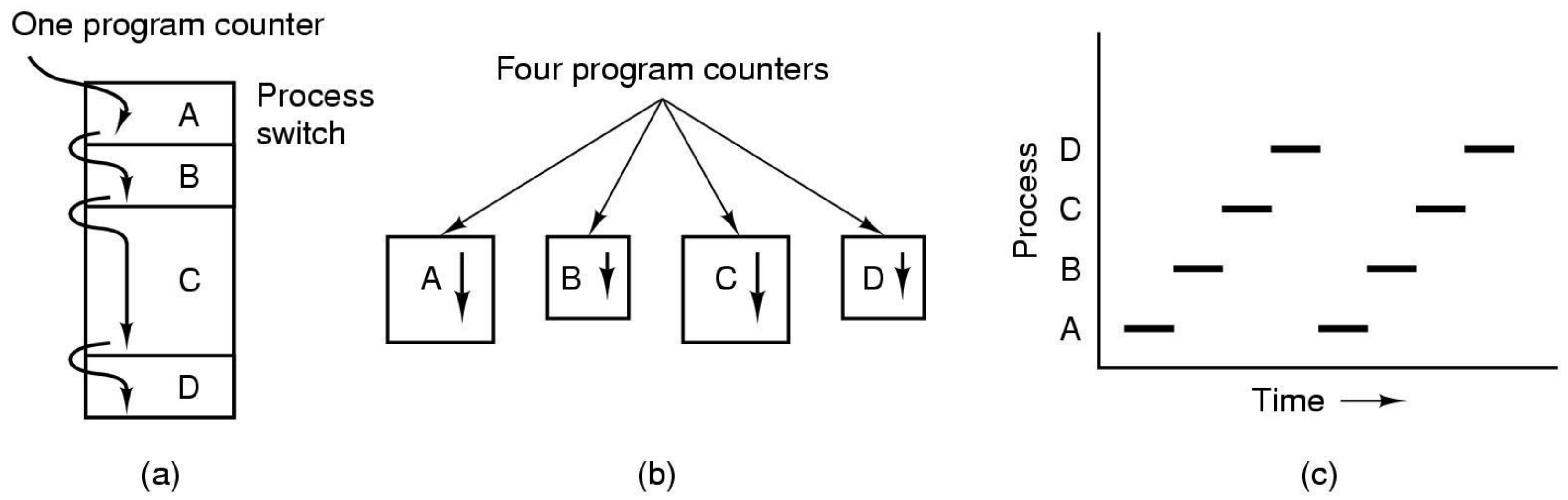
  - Solution: process concept

The Process Model

- In the process model, all runnable software is organized as a collection of processes.
- A process is an instance of an executing program, including the current values of the program counter, registers and variables.
- Conceptually,each process has its own virtual CPU. In reality, the real CPU switches back and forth from process to process.

The Process Model

- Example



a) Multiprogramming of four programs
b) Conceptual model of 4 independent, sequential processes
   - Each has its own flow of control (i.e. its own logical PC)
c) Only one program active at any instant

Process Concept

- from MULTICS and CTSS/360 system (task)
- An abstraction of a running program
- a program in execution

- 进程，即是一个具有一定独立功能的程序关于某个数据集合的一次活动。

- Analogy: birthday cake recipe;
  activity (reading, fetching, baking)

Difference between process and program

1. Program: the collection of instruction, static concept; Process: describe concurrency, dynamic concept
2. Process include program, data, and PCB
3. Process: temporary; Program: permanent
4. A program can be the execution program of multiple process; also, a process can call multiple programs.
5. Process can create other process in it.

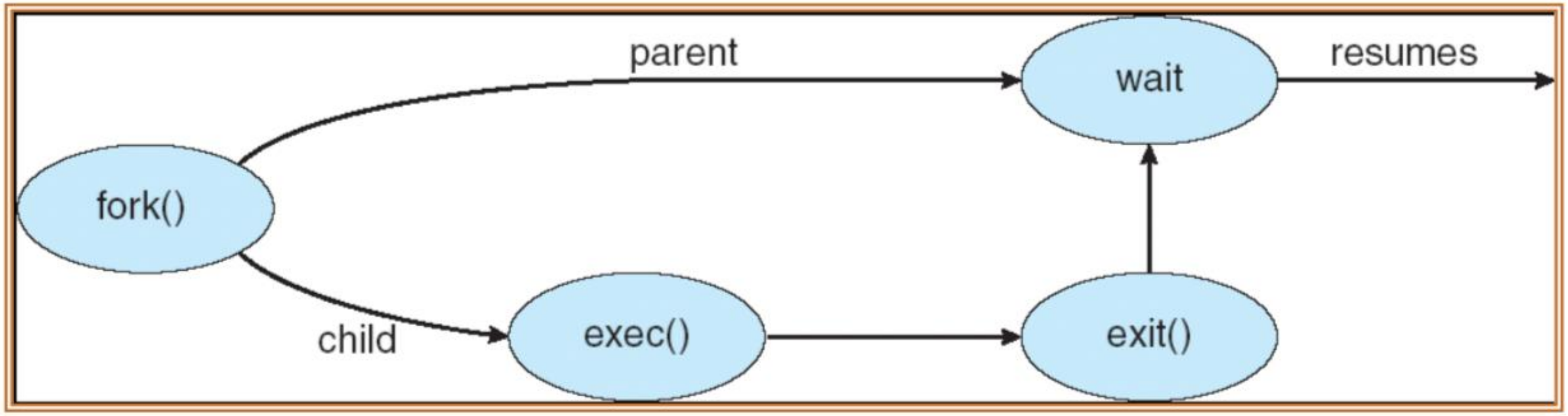
Process Creation (when?)

- Principal events that cause process creation

1. System initialization (reboot)
   - Foreground processes are those that interact with users and perform work for them.
   - Background processes that handle some incoming request are called **daemons**.
2. Execution of a process creation system call (fork())
3. User request to create a new process
   - Command line or click an icon
4. Initiation of a batch job

Process Creation

- In UNIX, a fork() system call is used to create a new process.
  - Initially, the parent and the child have the same memory image, the same environment strings, and the same open files.
  - The execve() system call can be used to load a new program.
  - But the parent and child have their own distinct address space.



Process Creation

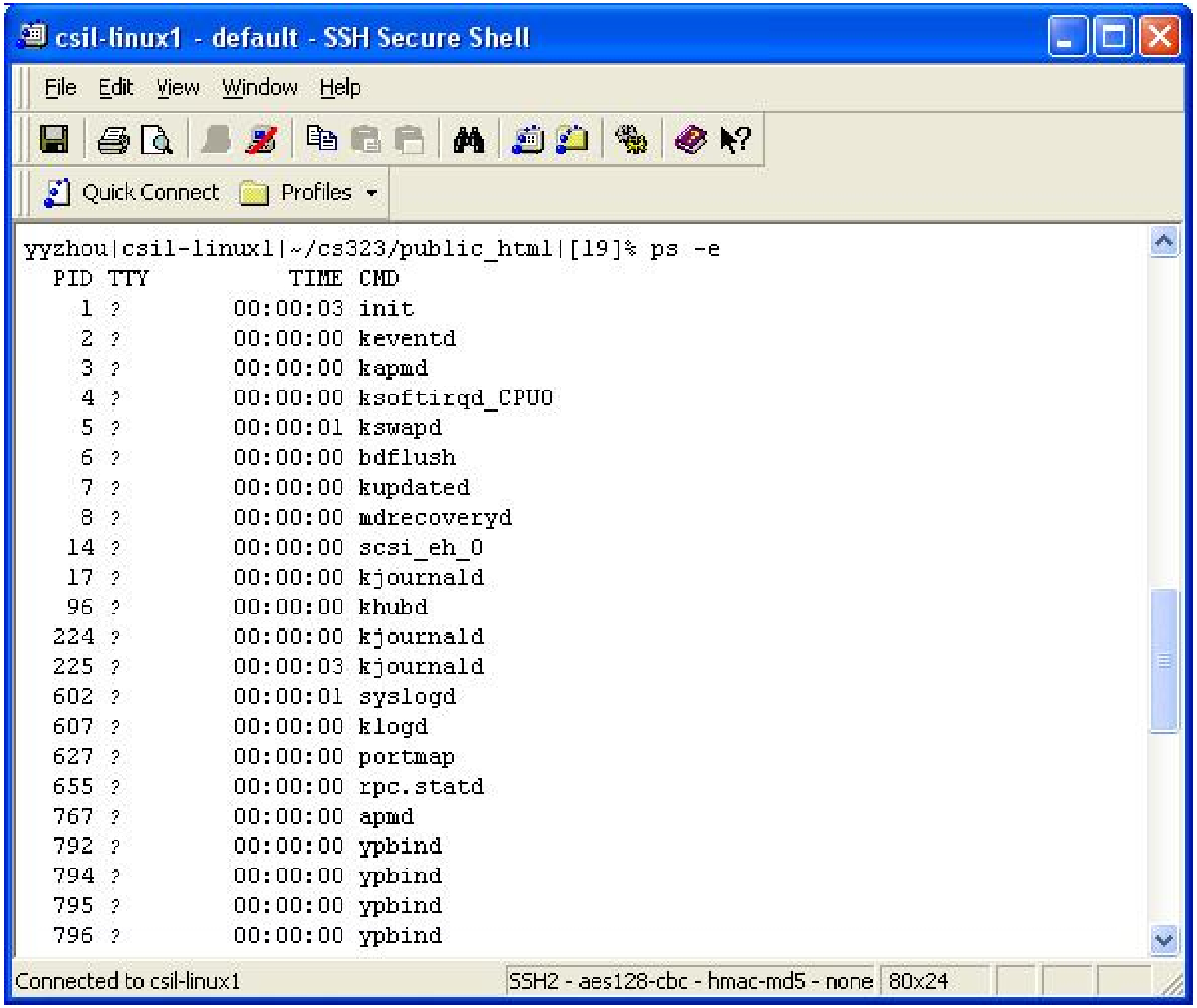
- In Windows, CreateProcess handles both creation and loading the correct program into the new process.

BOOL CreateProcess(
LPCTSTR *lpApplicationName*, // pointer to name of executable module
LPTSTR *lpCommandLine*, // pointer to command line string
LPSECURITY_ATTRIBUTES *lpProcessAttributes*, // process security attr.
LPSECURITY_ATTRIBUTES *lpThreadAttributes*, // thread security attr.
BOOL *bInheritHandles*, // handle inheritance flag
DWORD *dwCreationFlags*, // creation flags
LPVOID *lpEnvironment*, // pointer to new environment block
LPCTSTR *lpCurrentDirectory*, // pointer to current directory name
LPSTARTUPINFO *lpStartupInfo*, // pointer to STARTUPINFO
LPPROCESS_INFORMATION *lpProcessInformation* // pointer to PROCESS_INFORMATION)

Process Creation

- How to list the running processes?
  - In UNIX, use the ps command.
  - In Windows, use the task manager.

Unix Example: ps


csil-linux1 - default - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
yyzhou|csil-linux1|~/cs323/public_html|[19]% ps -e
  PID TTY          TIME CMD
    1 ?        00:00:03 init
    2 ?        00:00:00 keventd
    3 ?        00:00:00 kapmd
    4 ?        00:00:00 ksoftirqd_CPU0
    5 ?        00:00:01 kswapd
    6 ?        00:00:00 bdflush
    7 ?        00:00:00 kupdated
    8 ?        00:00:00 mdrecoveryd
   14 ?        00:00:00 scsi_eh_0
   17 ?        00:00:00 kjournald
   96 ?        00:00:00 khubd
  224 ?        00:00:00 kjournald
  225 ?        00:00:03 kjournald
  602 ?        00:00:01 syslogd
  607 ?        00:00:00 klogd
  627 ?        00:00:00 portmap
  655 ?        00:00:00 rpc.statd
  767 ?        00:00:00 apmd
  792 ?        00:00:00 ypbind
  794 ?        00:00:00 ypbind
  795 ?        00:00:00 ypbind
  796 ?        00:00:00 ypbind
```

Connected to csil-linux1 | SSH2 - aes128-cbc - hmac-md5 - none | 80x24

Windows Task Manager

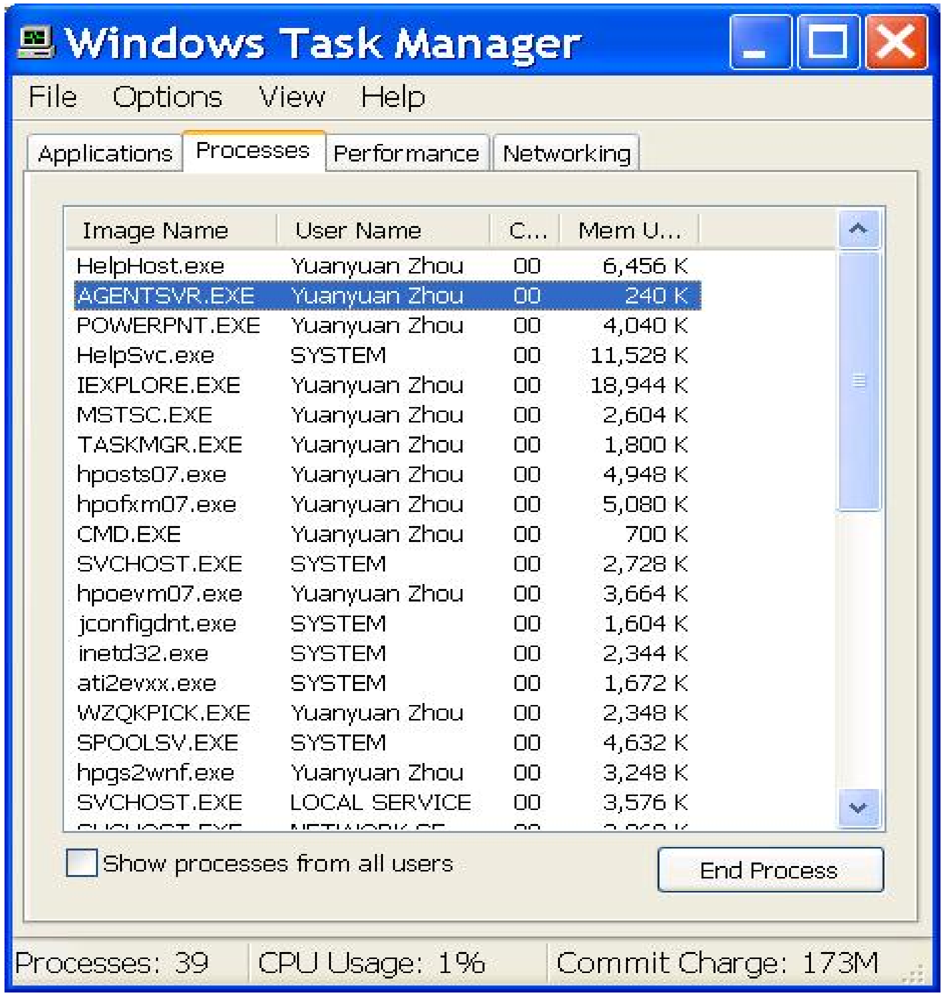


Image Name	User Name	C...	Mem U...
HelpHost.exe	Yuanyuan Zhou	00	6,456 K
AGENTSVR.EXE	Yuanyuan Zhou	00	240 K
POWERPNT.EXE	Yuanyuan Zhou	00	4,040 K
HelpSvc.exe	SYSTEM	00	11,528 K
IEXPLORE.EXE	Yuanyuan Zhou	00	18,944 K
MSTSC.EXE	Yuanyuan Zhou	00	2,604 K
TASKMGR.EXE	Yuanyuan Zhou	00	1,800 K
hposts07.exe	Yuanyuan Zhou	00	4,948 K
hpofxm07.exe	Yuanyuan Zhou	00	5,080 K
CMD.EXE	Yuanyuan Zhou	00	700 K
SVCHOST.EXE	SYSTEM	00	2,728 K
hpoevm07.exe	Yuanyuan Zhou	00	3,664 K
jconfigdnt.exe	SYSTEM	00	1,604 K
inetd32.exe	SYSTEM	00	2,344 K
ati2evxx.exe	SYSTEM	00	1,672 K
WZQKPICK.EXE	Yuanyuan Zhou	00	2,348 K
SPOOLSV.EXE	SYSTEM	00	4,632 K
hpgs2wnf.exe	Yuanyuan Zhou	00	3,248 K
SVCHOST.EXE	LOCAL SERVICE	00	3,576 K

☐ Show processes from all users

Processes: 39 CPU Usage: 1% Commit Charge: 173M

Process Termination (when?)

- Conditions which terminate processes
  1. Normal exit (voluntary)
     - Exit in UNIX and ExitProcess in Windows.
  2. Error exit (voluntary)
     - Example: Compiling errors.
  3. Fatal error (involuntary)
     - Example: Divide by 0, executing an illegal instruction, referencing nonexistent memory
  4. Killed by another process (involuntary)
     - Kill in UNIX and TerminateProcess in Windows.

Process Hierarchies

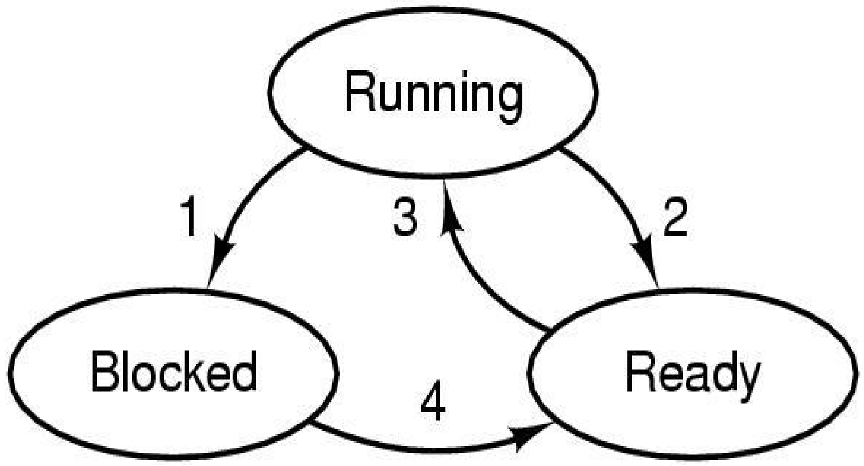
- Parent creates a child process, child processes can create its own processes
- Forms a hierarchy (*init*)
  - UNIX calls this a "process group“
- Windows has no concept of process hierarchy
  - all processes are created equal

Process States (1)

- basic process states
  - Running - using the CPU.
  - Ready - runnable (in the ready queue).
  - Blocked - unable to run until an external event occurs; e.g., waiting for a key to be pressed.

Process States (2)

Transitions between process states:

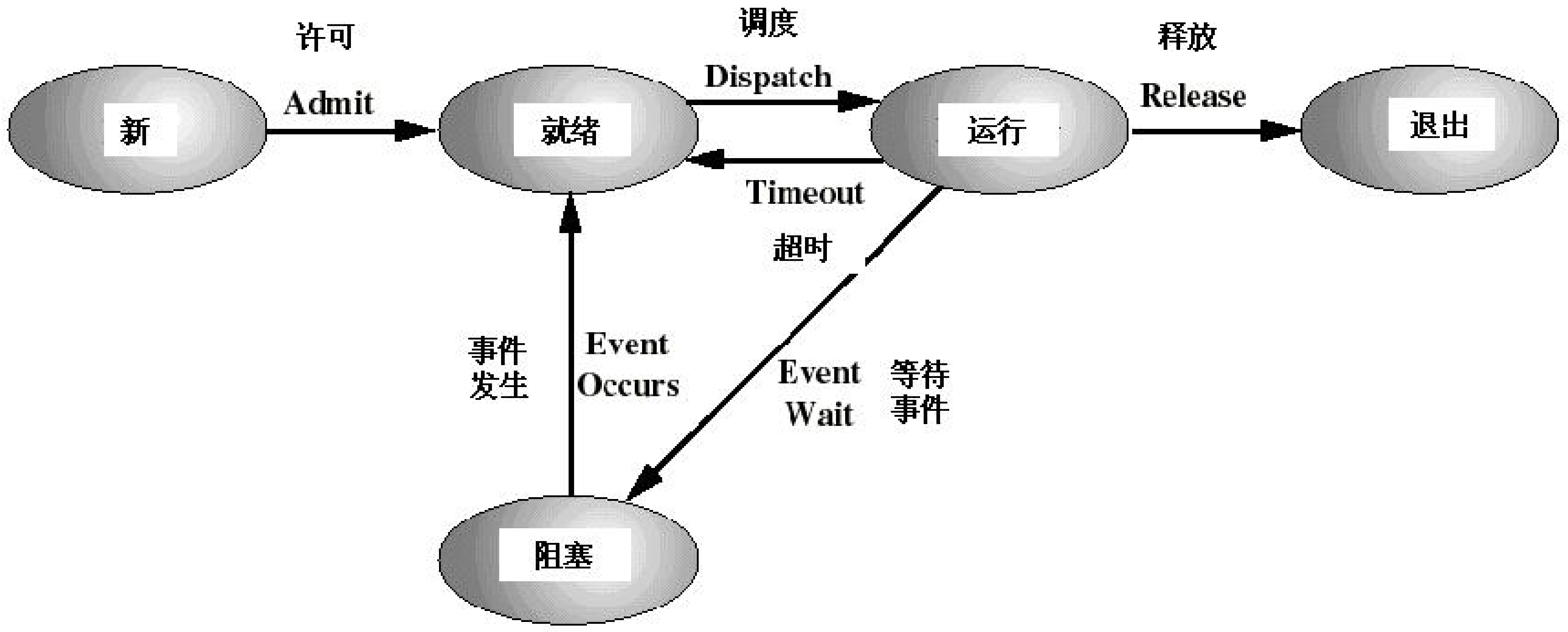


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Process States (3)

- Other process states

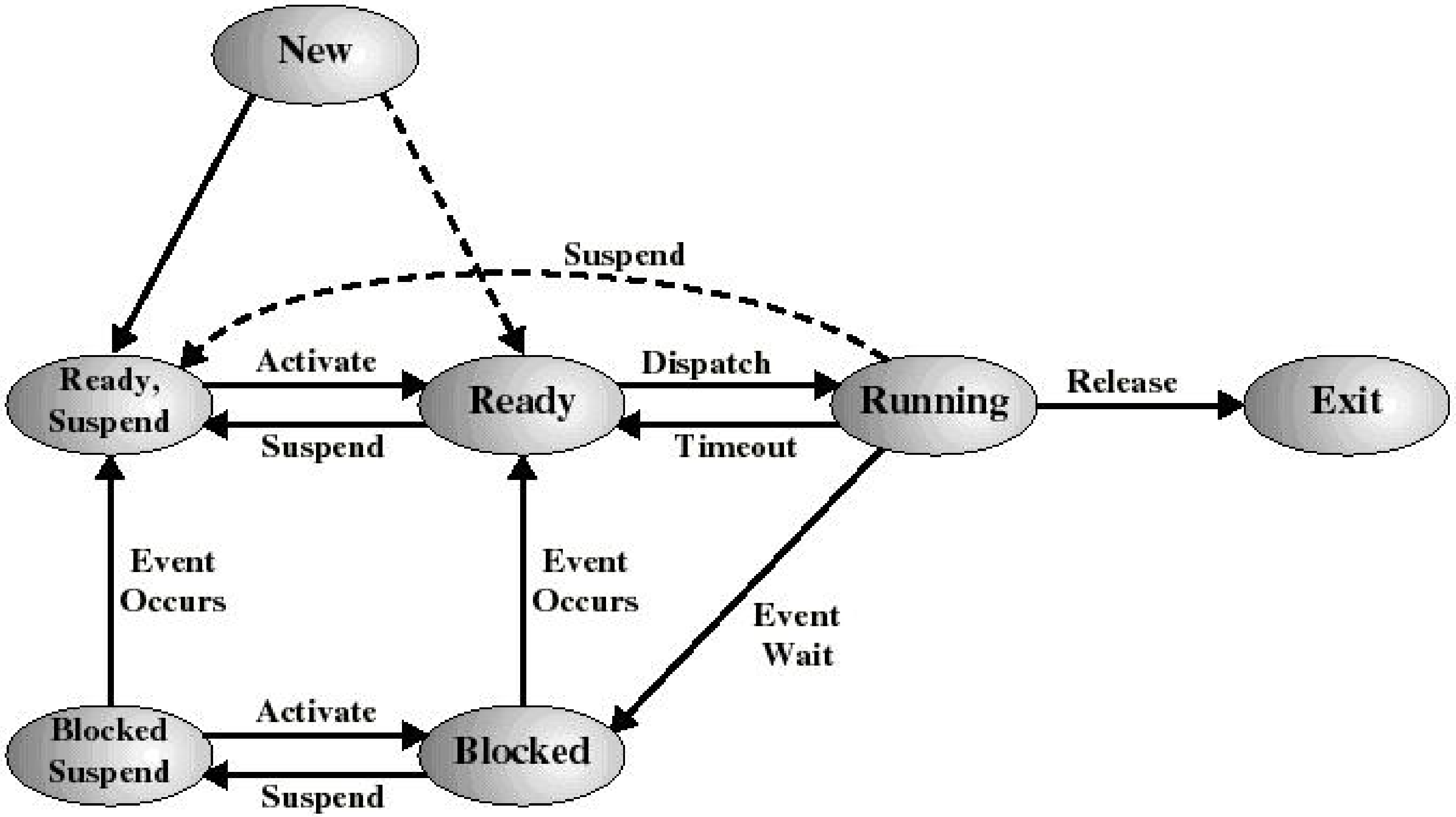
  New state, Exit state:



Process States (4)

Suspend :

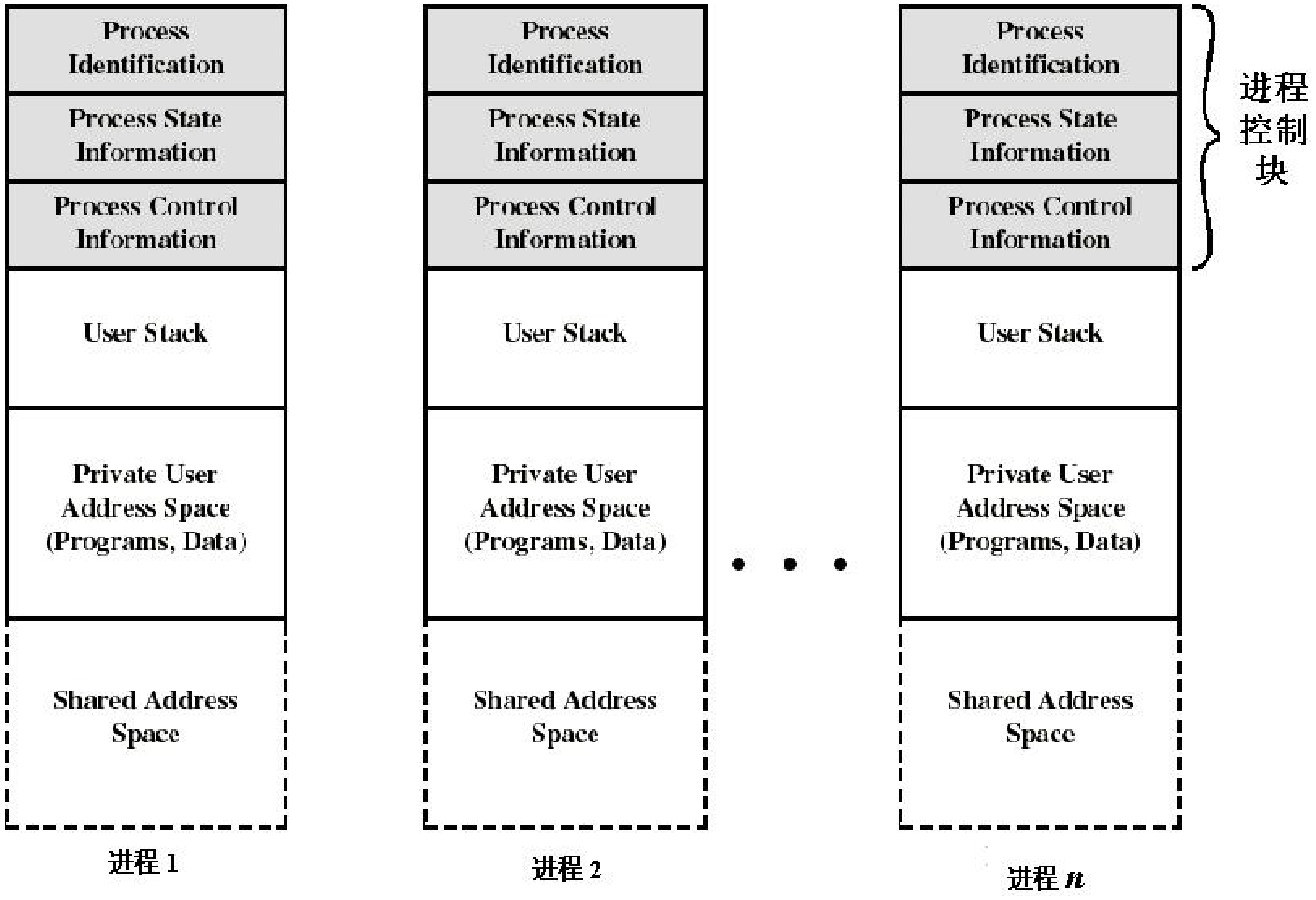
- Blocked, suspend state
- Ready, suspend



Implementation of Processes(1)

- What is in a process? - Process image
  - User program
  - User data
  - Stack (procedure call and parameter passing)
  - PCB - Process Control Block (process attribute)

在虚存中的进程映象



Process Identification
Process State Information
Process Control Information
User Stack
Private User Address Space (Programs, Data)
Shared Address Space
进程 1
进程 2
进程 *n*
进程控制块

Implementation of Processes(2)

- The architecture of a PCB is completely dependent on operating system and may contain different information in different operating system. (Linux PCB Structure: *task_struct*, 106 fields)

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
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		

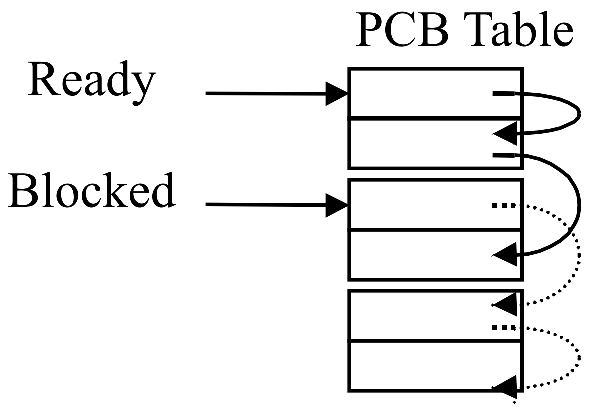
Some of the fields of a typical process table entry

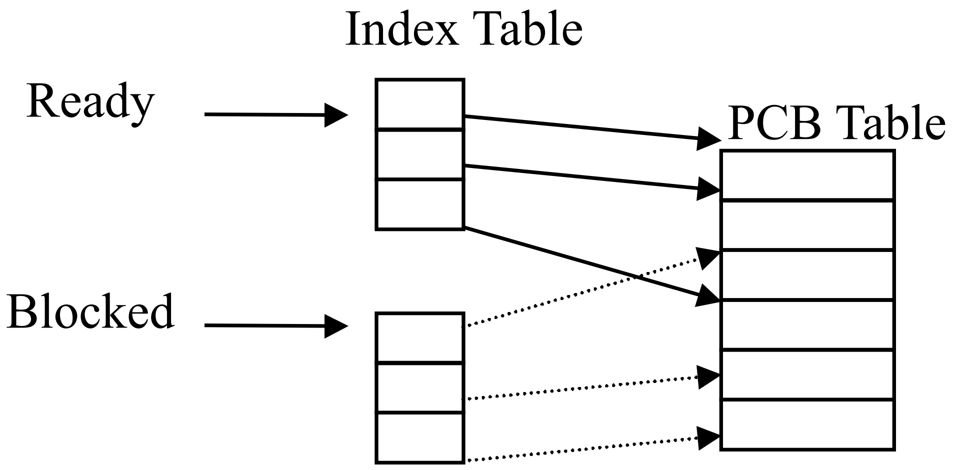
Implementation of Processes(3)

- **Process Context**
  - The static description of the whole execution process
  - User Context, Register Context, System Context
- **Context Switch**
  - Switch CPU from one process to another
  - Performed by scheduler (chapter 2.4)
  - It includes:
    - save PCB state of the old process;
    - load PCB state of the new process;
    - Flush memory cache;
    - Change memory mapping;
  - Context switch is expensive(1-1000 microseconds)
    - No useful work is done (pure overhead)
    - Can become a bottleneck

Implementation of Processes(4)

- Process table or PCB table
  - Os maintain a process table, each entry is PCB.
  - The size of PCB table determine the concurrency degree of system.
- Two organization form:
  - Link
  - Index





Summary(1)

- What is process
- Difference between process and program
- Process states and model
- What information in a process
- Process Context switch

2. Thread

- Basic attribute of Process:
  - Resource allocate unit
  - Scheduler unit
- OS must do:
  - Create process
  - Terminate process
  - Switch Process

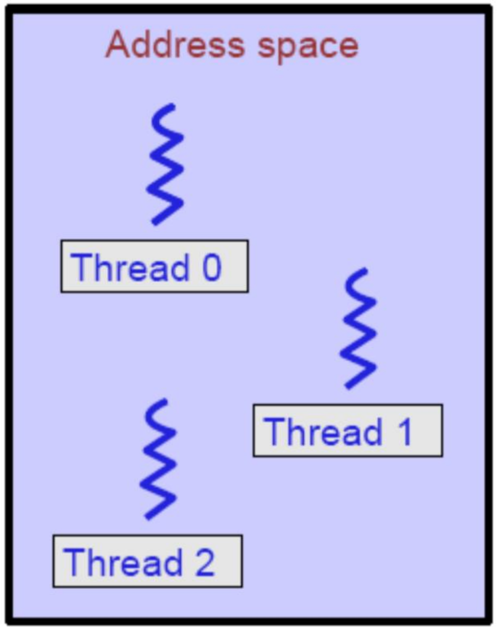
Spend much time and space, restrict the concurrency.

Thread concept

- 如何能使多个程序更好地并发执行，同时又尽量减少系统的开销，成为设计操作系统的重要目标。
- 将进程的两个基本属性分开，由操作系统分开处理：
    - 对于作为调度和分派的基本单位，不同时作为拥有资源的单位，以做到“轻装上阵”；
    - 对于拥有资源的基本单位，又不对之进行频繁的切换。
- A thread – **lightweight process (LWP)**
    - Thread: a basic unit of CPU utilization.
    - Process: a basic unit of resource allocation.

Thread concept

- What is a thread?
  - A sequential execution stream within a process.
- Threads separate the notion of execution from the process abstraction.



Thread concept

- 原进程PCB的内容分成两部分：
  - 描述进程资源和空间的部分；
  - 描述执行现场、状态及调度的部分。

  将第二部分内容作为线程控制块TCB的内容，且一个进程内允许多个线程存在。

- 新进程描述为：
  - 一个独立的进程空间，可装入进程映像；
  - 一个独立的进程相关联的执行文件；
  - 进程所用的系统资源；
  - 一个或多个线程。（进程在创建时一般同时创建好第一个线程，其他线程按需要由用户程序请求创建）

Thread concept

- 线程不拥有系统资源，只有其运行所必需的一些数据结构：TCB, a program counter, a register set, and a stack. 它与该进程内其它线程共享该进程所拥有的全部资源。
  - 堆栈和寄存器用来存储线程内的局部变量；
  - 线程控制块TCB用来说明线程存在的标识、记录线程属性和调度信息

    (Linux TCB: *thread_struct*, 24 fields)
- A traditional (**heavyweight**) process has a single thread of control.
- If the process has multiple threads of control, it can do more than one task at a time. This situation is called **multithreading**.

Thread vs Process

① 进程是资源分配的基本单位，所有与该进程有关的资源分配情况，如打印机、I/O缓冲队列等，均记录在进程控制块PCB中，进程也是分配主存的基本单位，它拥有一个完整的虚拟地址空间。而线程与资源分配无关，它属于某一个进程，并与该进程内的其它线程一起共享进程的资源。

② 不同的进程拥有不同的虚拟地址空间，而同一进程中的多个线程共享同一地址空间。

③ 进程调度的切换将涉及到有关资源指针的保存及进程地址空间的转换等问题。而线程的切换将不涉及资源指针的保存和地址空间的变化。所以，线程切换的开销要比进程切换的开销小得多。

Thread vs Process

④ 进程的调度与切换都是由操作系统内核完成，而线程则既可由操作系统内核完成，也可由用户程序进行。

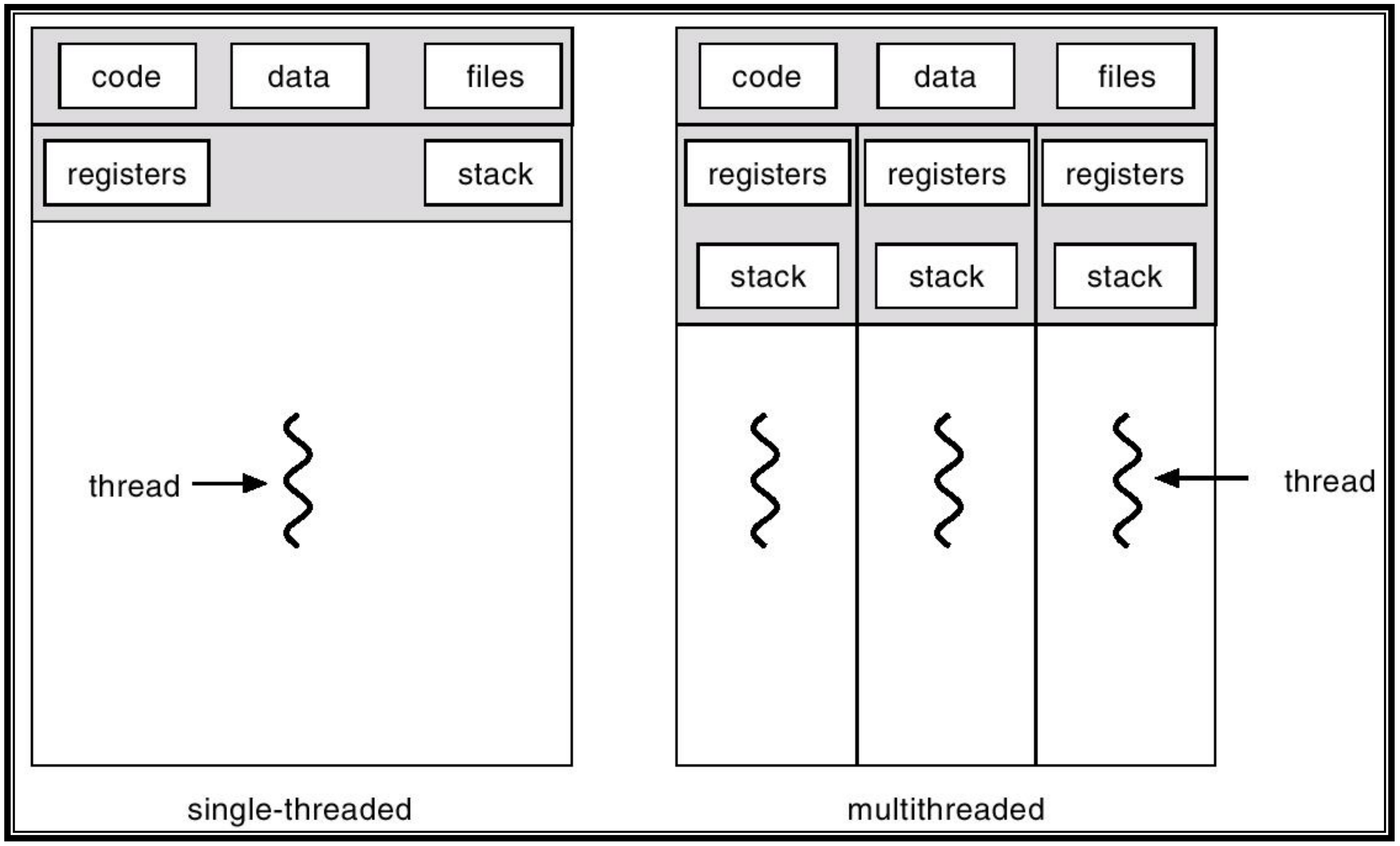
⑤ 进程可以动态创建进程。被进程创建的线程也可以创建其它线程。

⑥ 进程有创建、执行、消亡的生命周期。线程也有类似的生命周期。

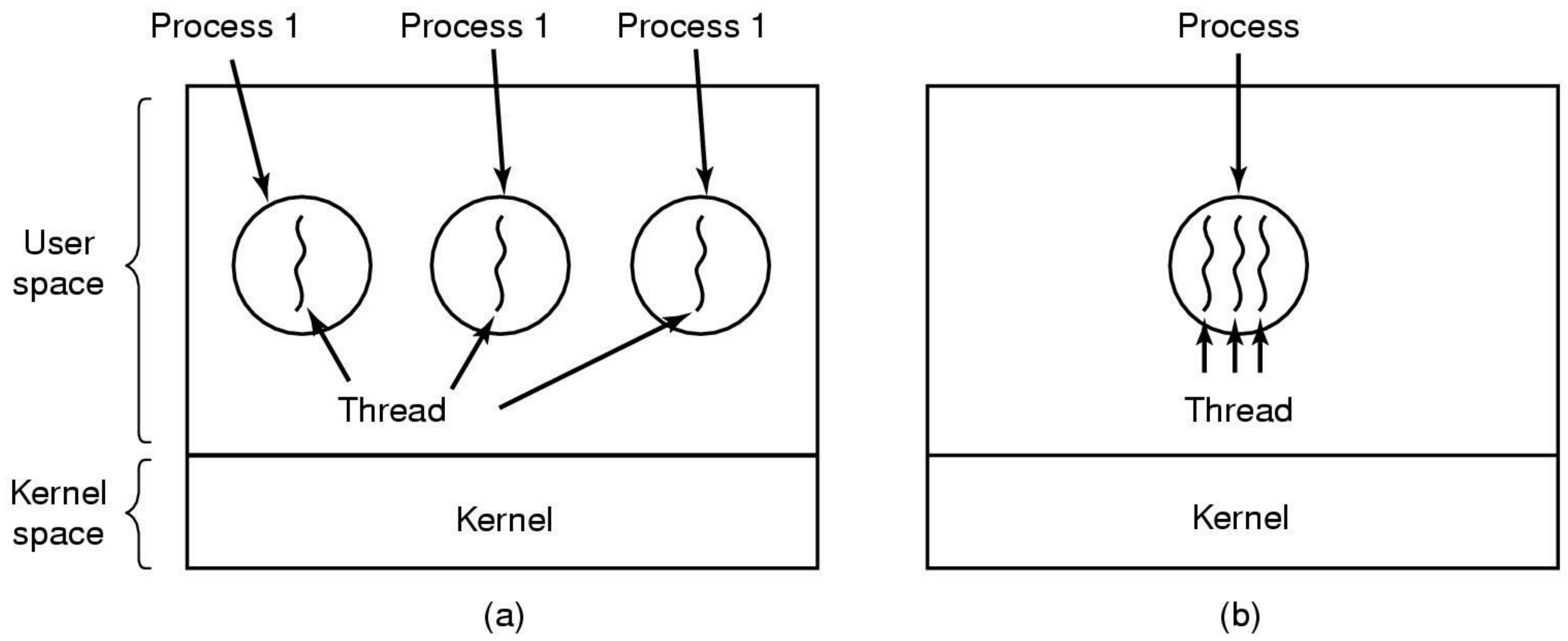
Thread Advantage

- Thread creation, short
- Thread exit, short
- Thread switch, short
- Communication of threads, simple
  (share the memory and file resource of process)

Single and Multithreaded Processes

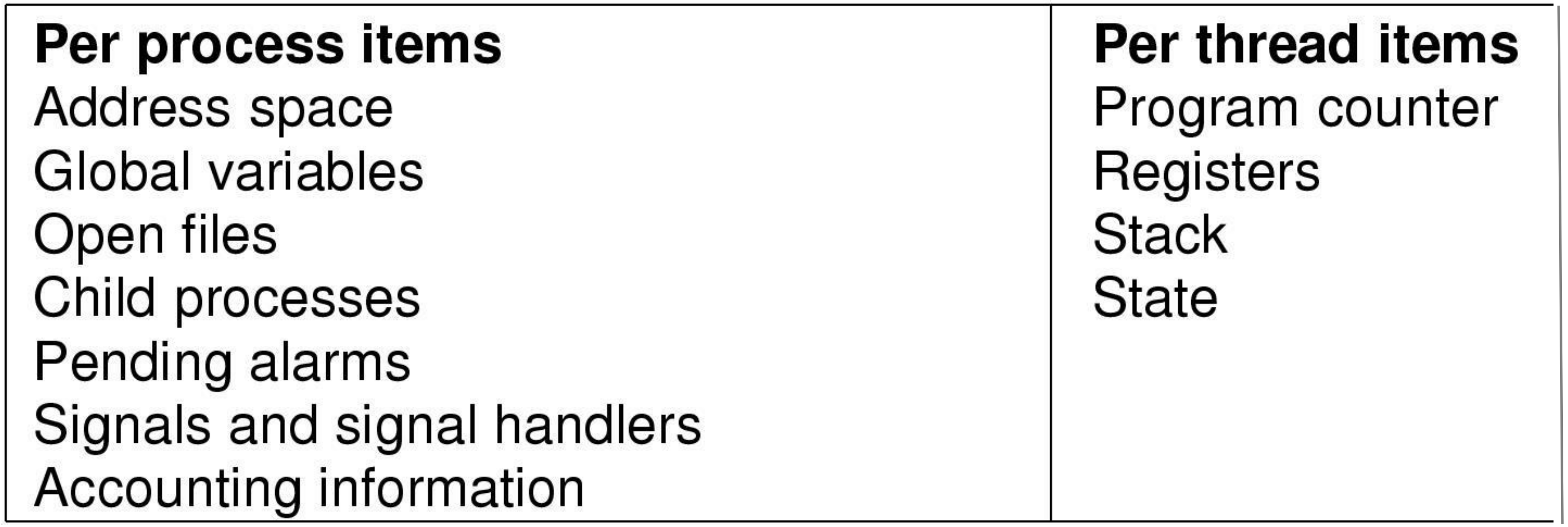


The Thread Model (1)



(a) Three processes each with one thread
(b) One process with three threads

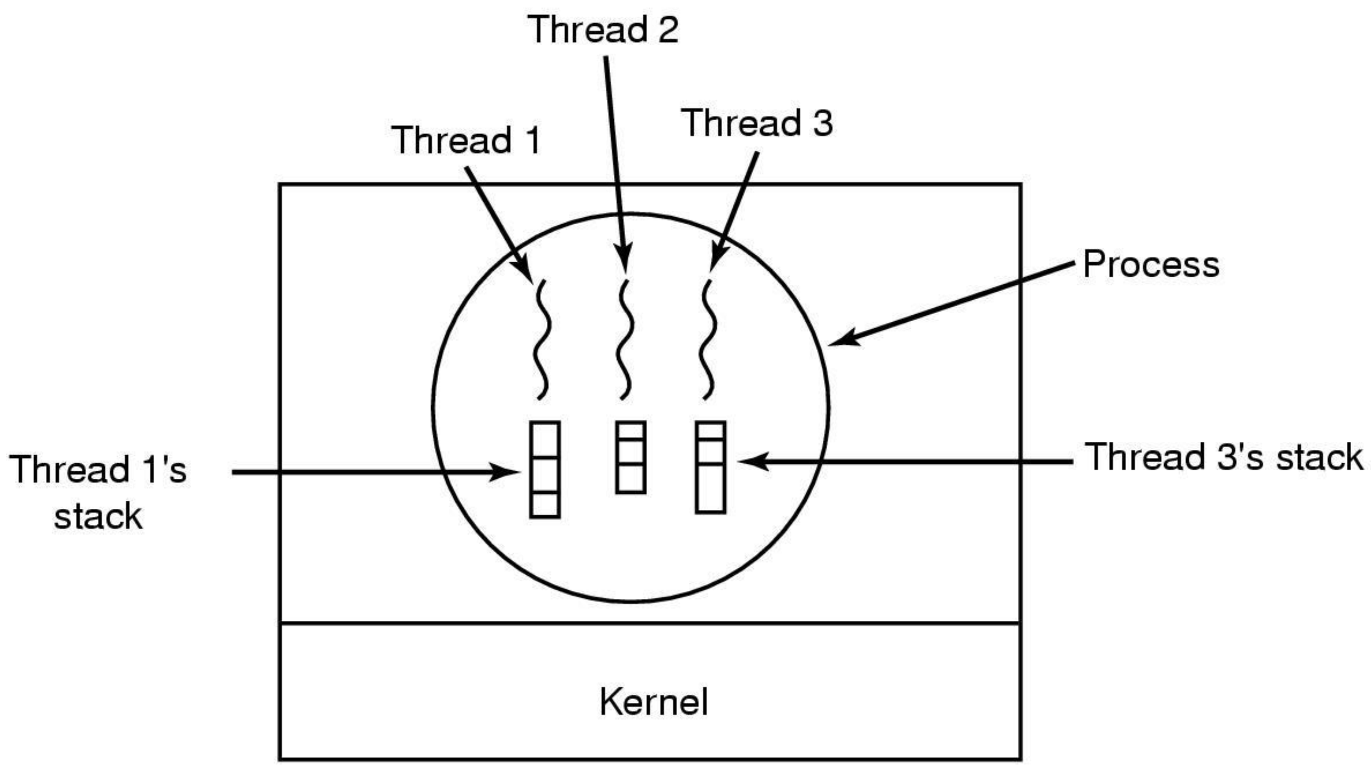
The Thread Model (2)



Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

- Items shared by all threads in a process
- Items private to each thread

The Thread Model (3)



Each thread has its own stack

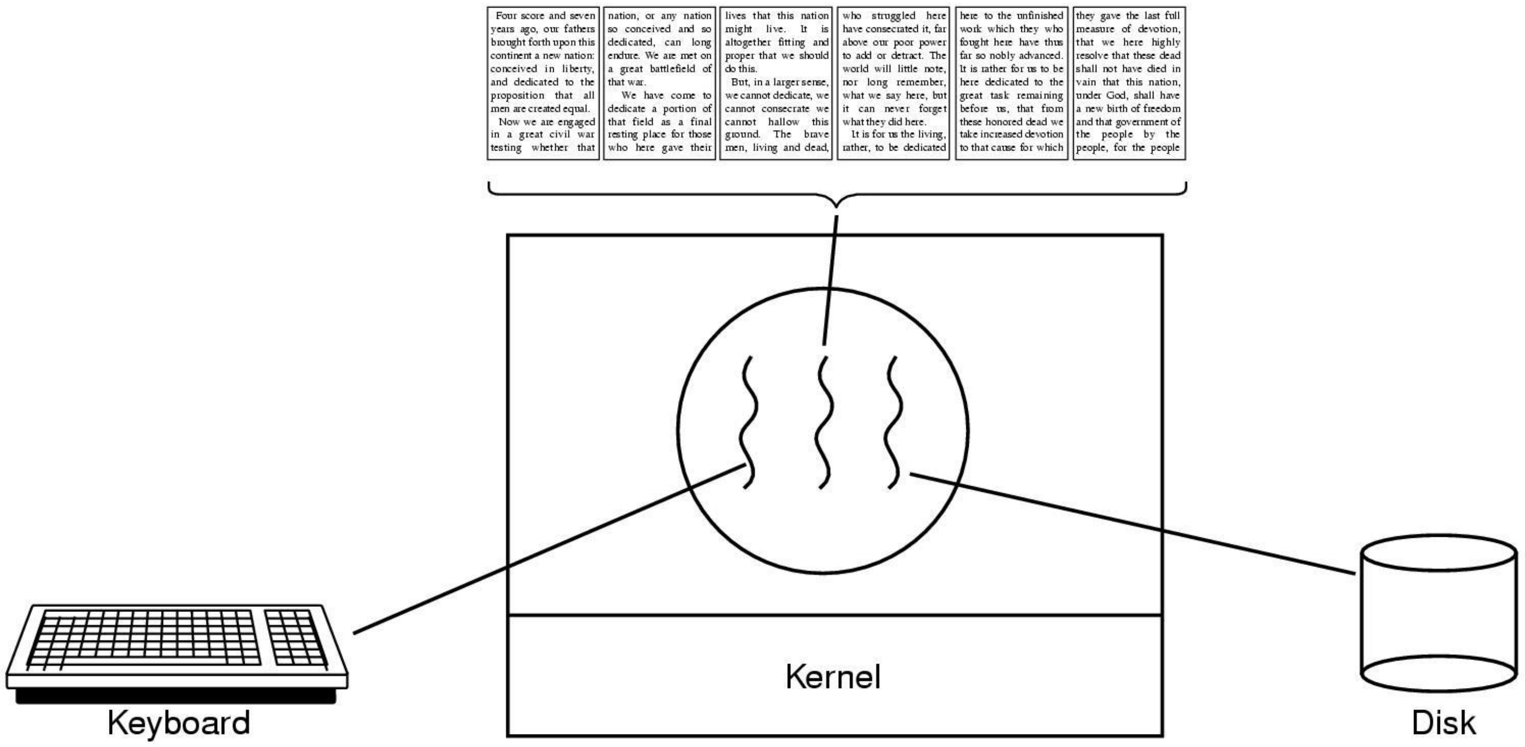
Thread Usage

- Why do you use threads?
  - Responsiveness: Multiple activities can be done at same time. They can speed up the application.
  - Resource Sharing: Threads share the memory and the resources of the process to which they belong.
  - Economy: They are easy to create and destroy.
  - Utilization of MP (multiprocessor) Architectures: They are useful on multiple CPU systems.

- Example - Word Processor, Spreadsheet
  - One thread interacts with the user.
  - One formats the document (spreadsheet).
  - One writes the file to disk periodically.

  Why not three processes?

Thread Usage (1)


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 these 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 by the people, for the people

Figure 2-7. A word processor with three threads

Thread Usage(2)

- Example – Web server
  - One thread, the dispatcher, distributes the requests to a worker thread.
  - A worker thread handles the requests.

Thread Usage (2)

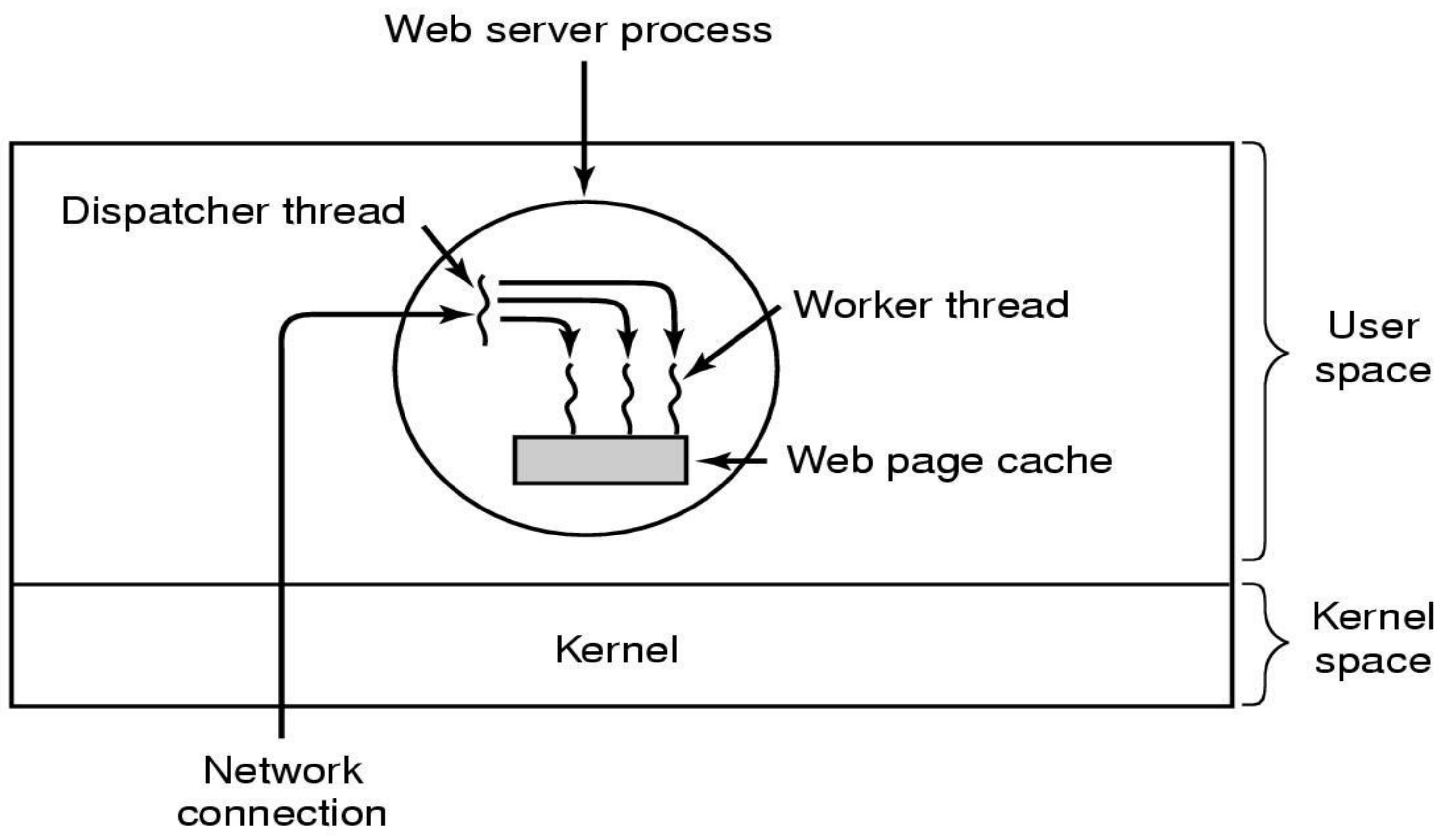
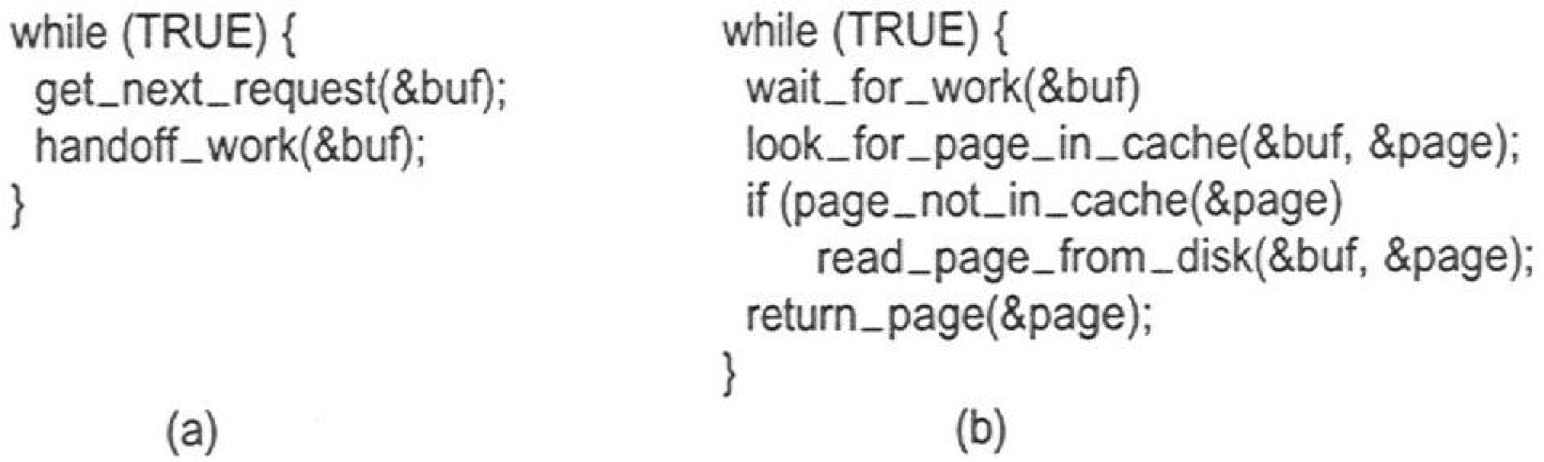


Figure 2-8. A multithreaded Web server

Thread Usage (2)



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

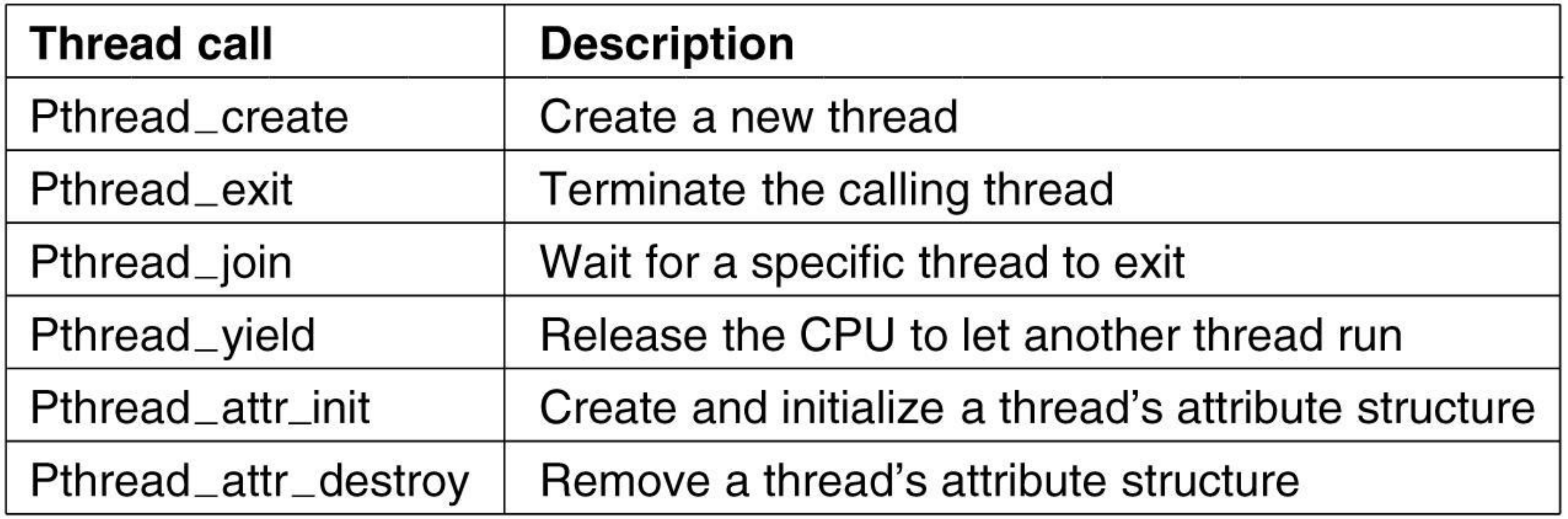
(a)

```
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)

Figure 2-9. Rough outline of code for previous slide
(a) Dispatcher thread (b) Worker thread

POSIX Threads



Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread’s attribute structure
Pthread_attr_destroy	Remove a thread’s attribute structure

Figure 2-14. Some of the Pthreads function calls.

POSIX Threads

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

#define NUMBER_OF_THREADS     10

void *print_hello_world(void *tid)
{
        /* This function prints the thread’s identifier and then exits. */
        printf("Hello World. Greetings from thread %d0, tid);
        pthread_exit(NULL);
}

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);
}
```

Figure 2-15. An example program using threads.

Implementation of Threads

- Three mechanism
  - Implementing threads in user space
  - Implementing threads in kernel space
  - Hybrid Implementations

User Threads(ULT)

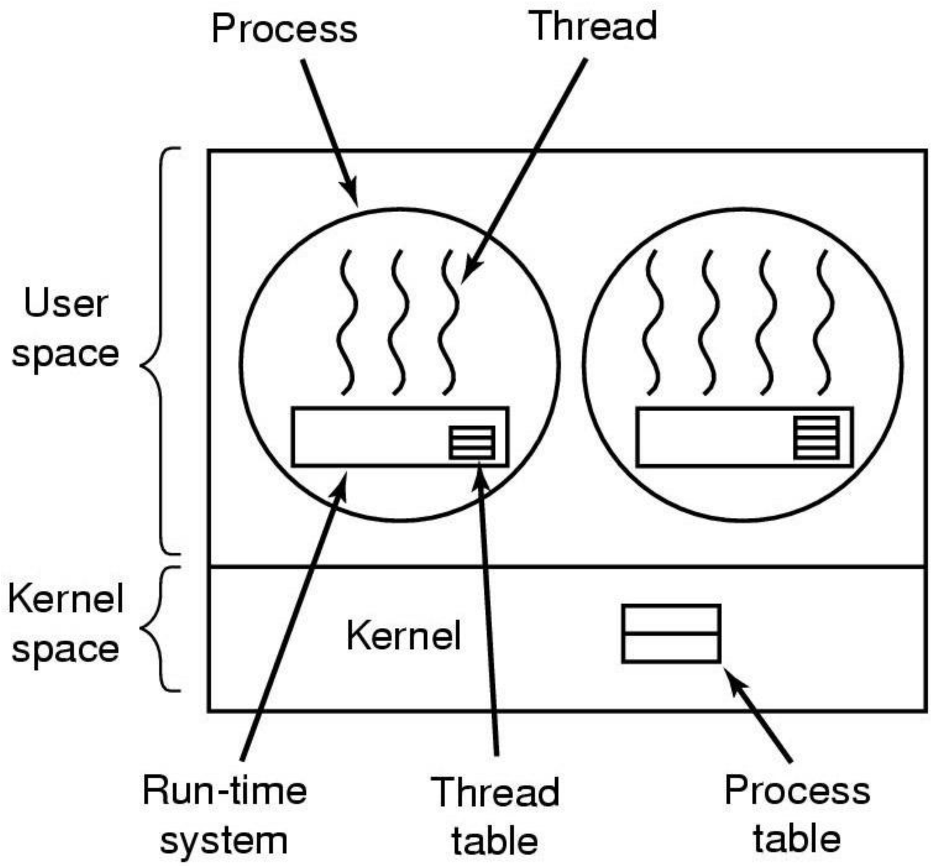
- The threads package entirely in user space, kernel doesn’t know thread
- Threads management done by user-level threads library
- Thread switch don’t need kernel privilege, user-level threads are fast to create and manage.
- Problem: If the kernel is single-threaded, then any user-level thread performing a blocking system call will cause the entire process to block.
- Examples of threads library

  - POSIX *Pthreads*

  - Mach *C-threads*

  - Solaris UI-*threads*

Implementing Threads in User Space



A user-level threads package

Kernel Threads(KLT)

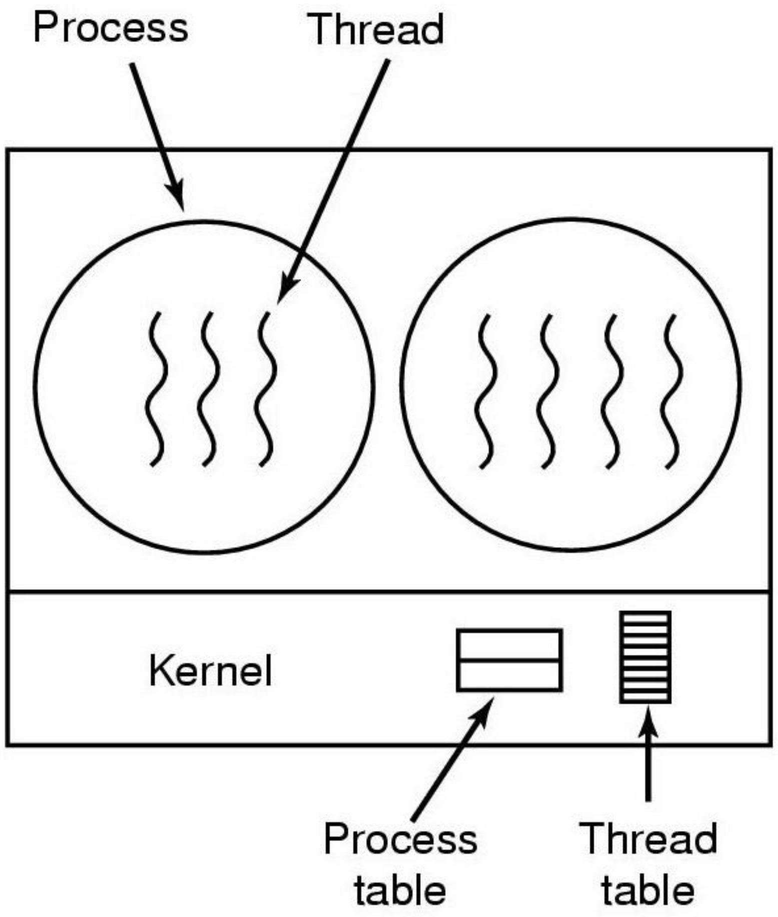
- Supported by the Kernel: The kernel performs thread creation, scheduling, and management in kernel space.
- No thread library, kernel provide API
- Kernel maintain context of process and thread
- Thread switch need kernel
- Thread is the basic unit of Scheduler .

- Disadvantage: high cost

- Examples

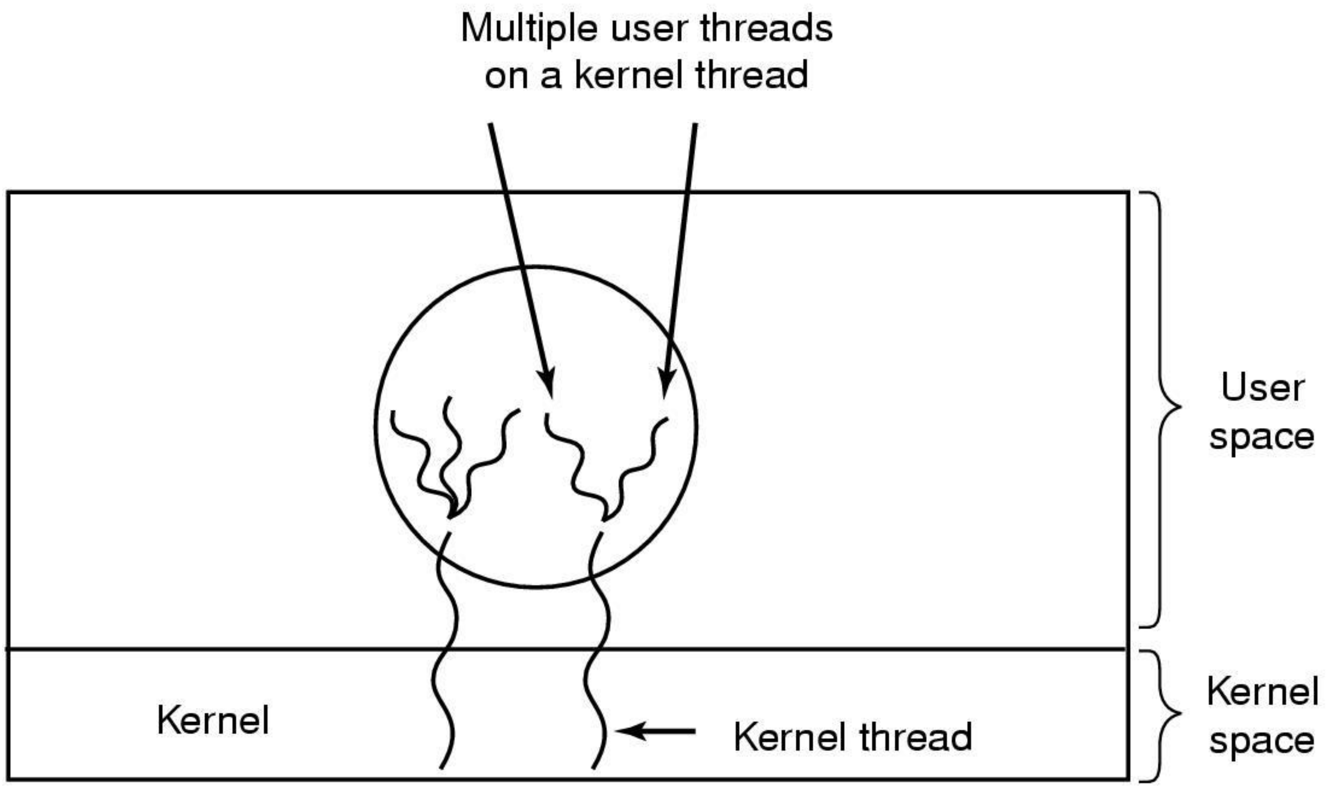
  - *Windows 95/98/NT/2000/XP*

Implementing Threads in the Kernel



A threads package managed by the kernel

Hybrid Implementations



Multiplexing user-level threads onto kernel-level threads (SUN solaris)

Pop-Up Threads

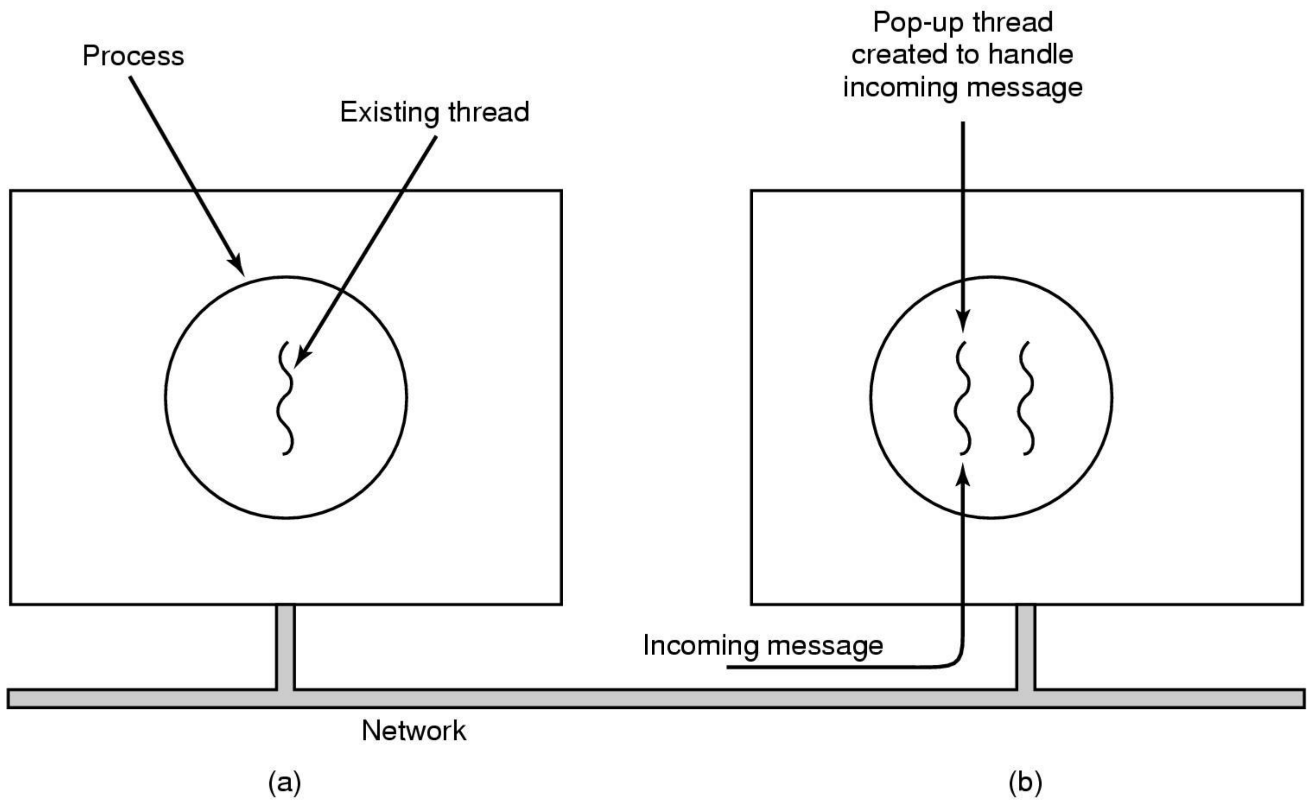


Figure 2-18. Creation of a new thread when message arrives
(a) before message arrives (b) after message arrives

Summary(2)

- What is thread? Why need threads?
- Difference between process and thread
- Thread Usage
- Thread Implementations and their tradeoffs
  - User-level
  - Kernel-level
  - Hybrid
  - Pop-up threads

- Next lecture: Inter-process Communication

Homework

- 1, 12, 16, 18