Betriebssysteme und Systemnahe Programmierung

Kapitel 3 • Prozesse

Winter 2016/17

Marcel Waldvogel

The Process Model (1)

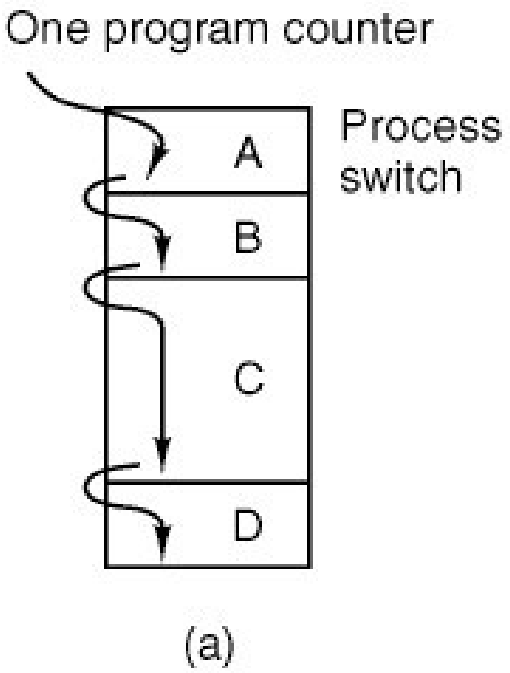


Figure 2-1 (a) Multiprogramming of four programs.

The Process Model (2)

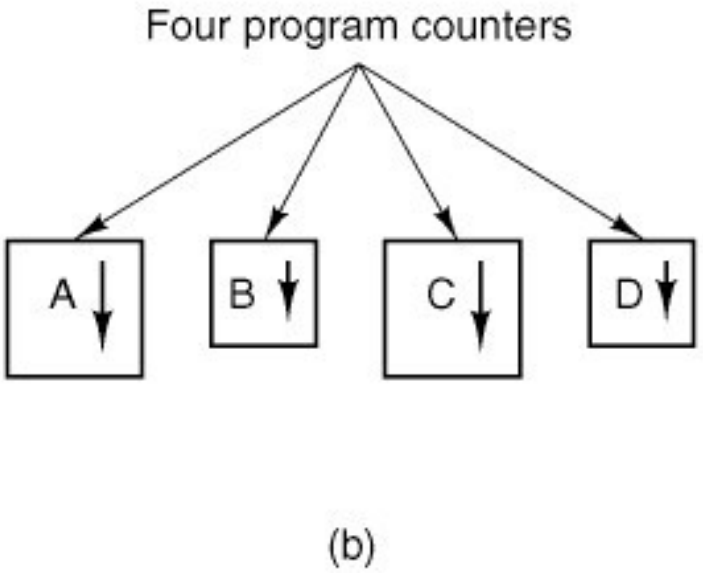


Figure 2-1 (b) Conceptual model of four independent, sequential processes.

The Process Model (3)

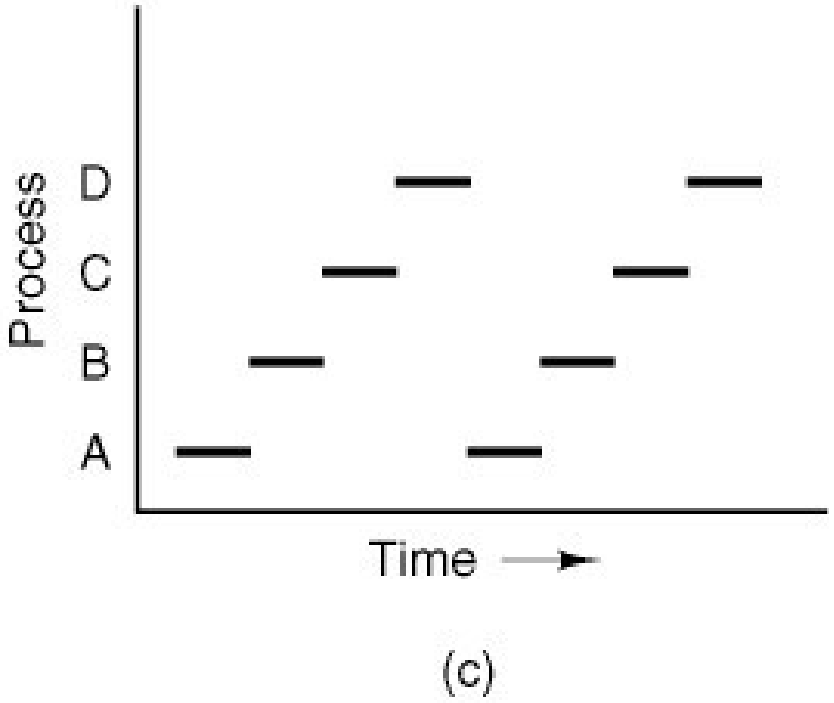


Figure 2-1 (c) Only one program is active at any instant.

Process Creation

Principal events that cause processes to be created:

1. System initialization.
2. Execution of a process creation system call by a running process.
3. A user request to create a new process.
4. Initiation of a batch job.

Process Termination

Conditions that cause a process to terminate:

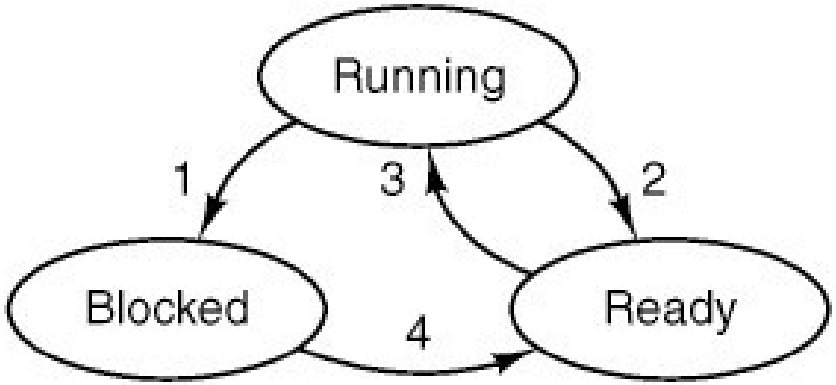
1. Normal exit (voluntary).
2. Error exit (voluntary).
3. Fatal error (involuntary).
4. Killed by another process (involuntary).

Process States (1)

Possible process states:

1. Running
   (actually using the CPU at that instant).
2. Ready
   (runnable; temporarily stopped to let another process run).
3. Blocked
   (unable to run until some external event happens).

Process States (2)



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

Figure 2-2 A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Process States (3)

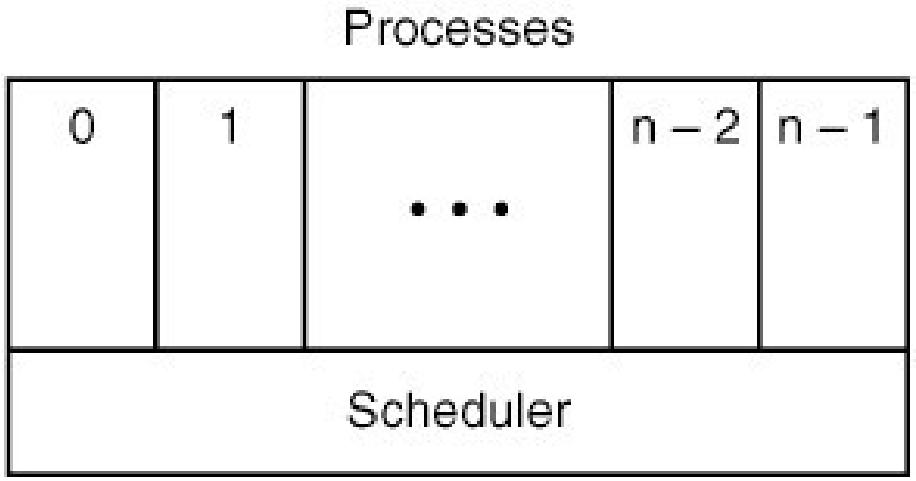


Figure 2-3 The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

Implementation of Processes

Process management	Memory management	File management
Registers	Pointer to text segment	UMASK mask
Program counter	Pointer to data segment	Root directory
Program status word	Pointer to bss segment	Working directory
Stack pointer	Exit status	File descriptors
Process state	Signal status	Effective uid
Time when process started	Process id	Effective gid
CPU time used	Parent process	System call parameters
Children's CPU time	Process group	Various flag bits
Time of next alarm	Real uid	
Message queue pointers	Effective uid	
Pending signal bits	Real gid	
Process id	Effective gid	
Various flag bits	Bit maps for signals	
	Various flag bits	

Figure 2-4. Some of the fields of the MINIX 3 process table. The fields are distributed over the kernel, the process manager, and the file system.

Interrupts

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler marks waiting task as ready.
7. Scheduler decides which process is to run next.
8. C procedure returns to the assembly code.
9. Assembly language procedure starts up new current process.

Figure 2-5 Skeleton of what the lowest level of the operating system does when an interrupt occurs.

Threads (1)

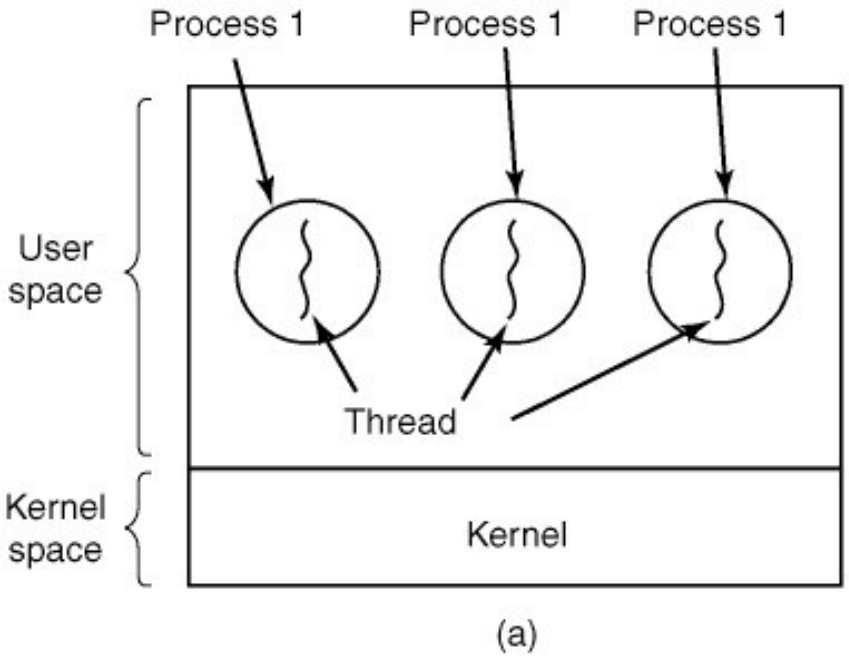


Figure 2-6 (a) Three processes each with one thread.

Threads (2)

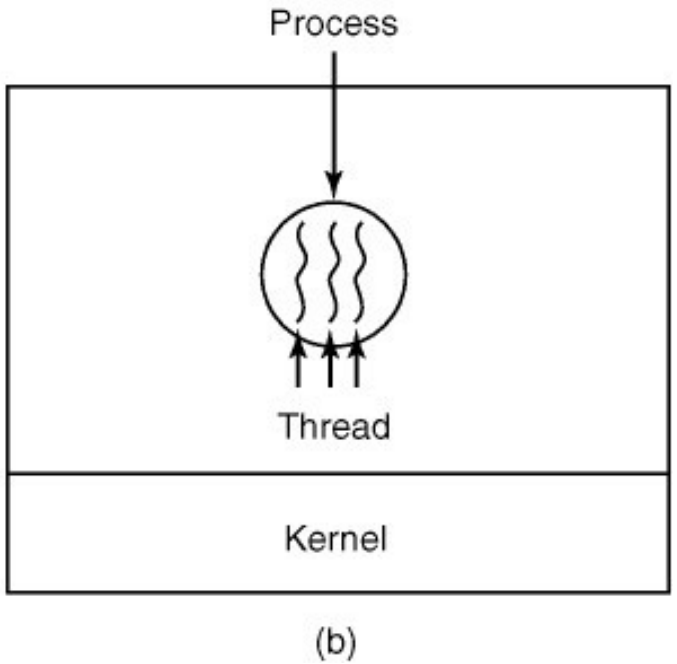


Figure 2-6 (b) One process with three threads.

Threads (3)

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	

Figure 2-7. The first column lists some items shared by all threads in a process. The second one lists some items private to each thread.