Processes and Threads

I233 Operating Systems

Chap.2 Processes and Threads

- 2.1 PROCESSES
  - 2.1.1 The Process Model
  - 2.1.2 Process Creation
  - 2.1.3 Process Termination
  - 2.1.4 Process Hierarchies
  - 2.1.5 Process States
  - 2.1.6 Implementation of Processes
  - 2.1.7 Modeling Multiprogramming
- 2.2 THREADS
  - 2.2.1 Thread Usage
  - 2.2.2 The Classical Thread Model
  - 2.2.3 POSIX Threads
  - 2.2.4 Implementing Threads in User Space
  - 2.2.5 Implementing Threads in the Kernel
  - 2.2.6 Hybrid Implementations
  - 2.2.7 Scheduler Activations
  - 2.2.8 Pop-Up Threads
  - 2.2.9 Making Single-Threaded Code Multithreaded

The Process Model

- What is “Process” ?
  - An abstraction of a running program
  - Executed sequentially (sequential processes)
  - Just 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

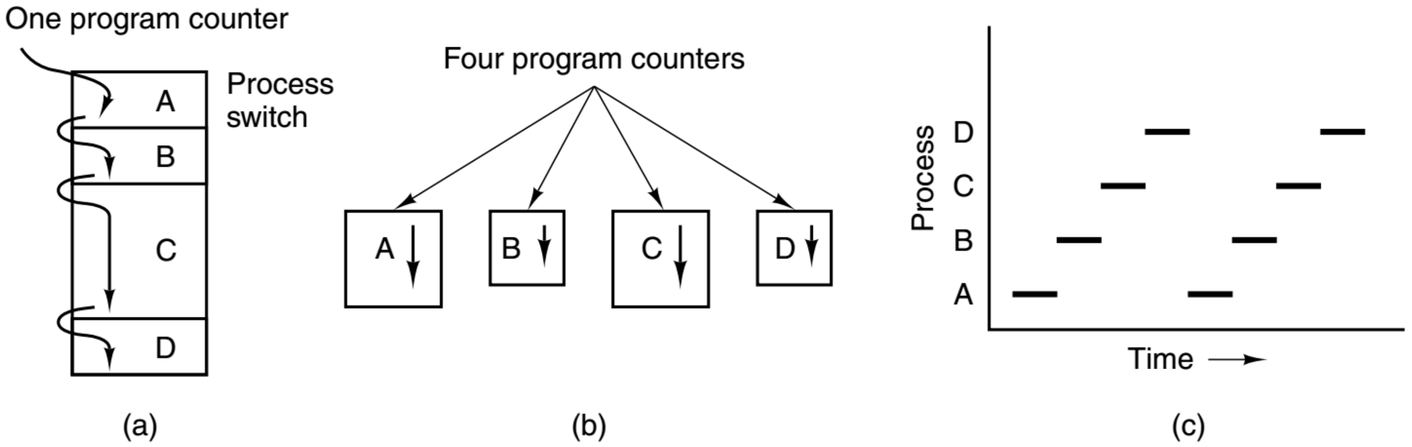


Figure 2-1. (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.87.

The difference between a process and a program

- An analogy on the textbook (a culinary-minded computer scientist)
  - Environment
    - A birthday cake recipe ⇔ Program
    - flour, eggs, sugar, extract of vanilla, etc. ⇔ Input data
    - Computer scientist ⇔ Processor (CPU)
  - The “process” is the activity consisting of
    - our baker reading the recipe, fetching the ingredients, and baking the cake.
  - Interruptions by higher-priority incident
    - Computer scientist’s son has been stung by a bee (Event occurs), and comes running in (Arrival of the event).
    - The computer scientist records where he was in the recipe (the state of the current process is saved),
    - The computer scientist (CPU) gets out a first aid book (Program), and begins following the directions in it.
    - When the bee sting has been taken care of, the computer scientist goes back to his cake, continuing at the point where he left off. (Returning from interrupt)

Process Creation

- Four principal events cause processes to be created:
  - System initialization
    - Typically numerous processes are created.
    - Some of these run in the background and are not associated with particular users, but instead have some specific function.
  - Execution of a process-creation system call by a running process
    - Often a running process will issue system calls to create one or more new processes to help it do its job.
  - A user request to create a new process
    - In interactive systems, users can start a program by typing a command or (double) clicking on an icon.
  - Initiation of a batch job
    - Users can submit batch jobs to the system. When the operating system decides that it has the resources to run another job, it creates a new process and runs the next job from the input queue in it.

What is required to create a process?

- = What information constitutes a process?
  - Prepare the program (executable). e.g. lookup the file on filesystem
  - Extraction of the program (executable) onto memory
  - Initialize the stack
  - Initialize the variable area
  - Initialize registers
  - Initialization of input/output (state) if input and output are specified in advance
- How big of a task is this?
  - Structure representing a “process” (struct proc)

Method of Process Creation

- A method of explicitly specifying the attributes of a process
  - JCL (Job Control Language)
    - https://en.wikipedia.org/wiki/Job_Control_Language
  - Example on JCL

```
//IS198CPY JOB (IS198T30500),'COPY JOB',CLASS=L,MSGCLASS=X
//COPY01   EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=OLDFILE,DISP=SHR
//SYSUT2   DD DSN=NEWFILE,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(40,5),RLSE),
//            DCB=(LRECL=115,BLKSIZE=1150)
//SYSIN  DD DUMMY
```

- Method to create a copy of the parent process.
  - fork() system-call on UNIX

Process Termination

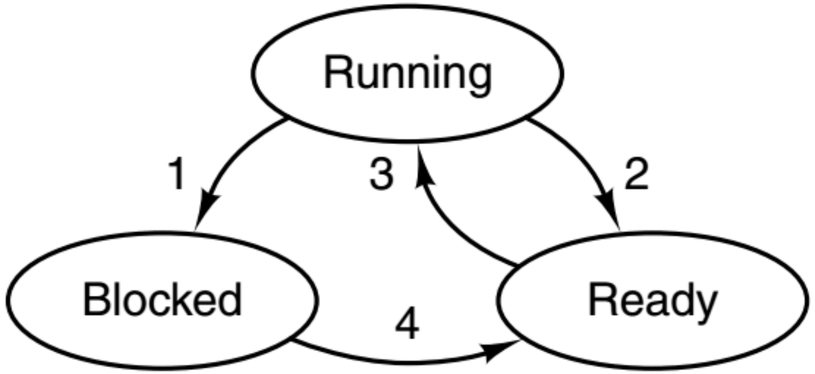
- Causes of process termination
  - Normal exit (voluntary)
  - Error exit (voluntary)
  - Fatal error (involuntary, system detects)
  - Killed by another process (involuntary)
- Handling of process termination
  - Release of resources
  - Notification of final results related to process execution

Process Hierarchies (association)

- Cases where there is a relationship between processes
  - Parent-child relationship
  - UNIX-like systems
  - All the processes in the whole system belong to a single tree
- Cases where there is no relationship between processes
  - Windows
  - Batch systems

Process States

- A “state” that focuses on the state of CPU allocation for a given process.
- A key concept: “Blocked” = Waiting for input/output to finish. (Waiting for resources to be available.)



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.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.93.

Processes and Scheduler

- Scheduler: sched()
- Called when a system call issued by a process becomes “waiting for input/output”.
- Algorithm
  - Saves the current process state and sets the state to “Blocked”.
  - Select one process from the set of processes in the "Ready" state. (Selection based on scheduling policy)
  - Restore the waiting process state so that the selected process can run, and set the state to “Running”.

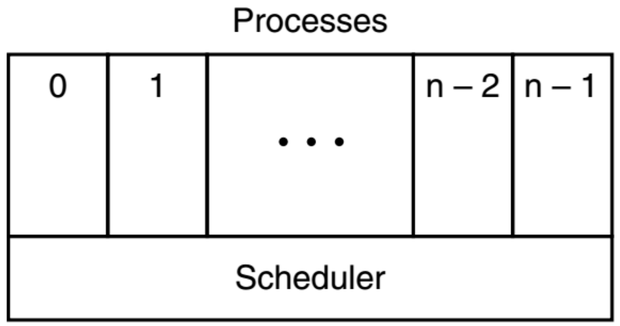


Figure 2-3. The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.94.

Implementation of Processes

Process management	**Memory management**	**File management**
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	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		

Figure 2-4. Some of the fields of a typical process-table entry.

Source: Andrew S. Tanenbaum. "Modern Operating Systems", 4th ed., global ed., pp.95.

Interrupt Handling

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 decides which process is to run next.
7. C procedure returns to the assembly code.
8. 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.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.96.

Interrupts, Exceptions, Modes, Upper/Lower half

- Interrupts and Exceptions
  - Interrupts (Hardware interrupts)
  - Exceptions (Software interrupts)
- Modes
  - User mode
  - Kernel (System) mode
- Lower half v.s. Upper half
  - Lower half: The part executed by hardware interrupt
  - Upper half: The part executed by software interrupt
  - Where do they intersect?

Threads

- Review the “Process”
  - Process = resource grouping + thread of execution
  - In “Thread” model:
    - Process = resource grouping
    - Thread = thread of execution

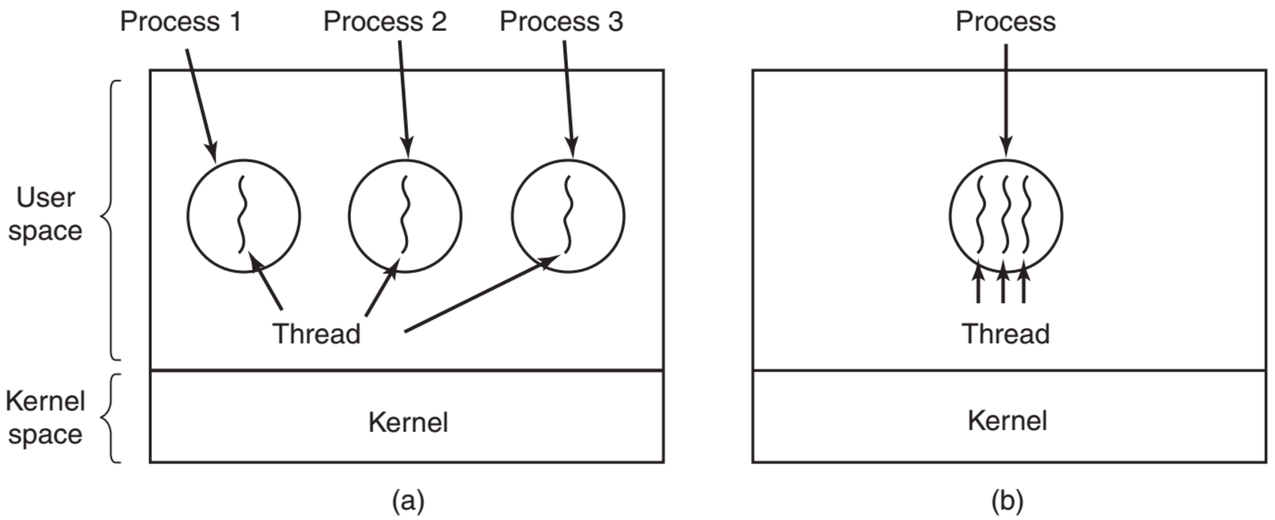


Figure 2-11. (a) Three processes each with one thread. (b) One process with three threads.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.104.

Resource grouping and Execution

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 2-12. The first column lists some items shared by all threads in a process. The second one lists some items private to each thread.

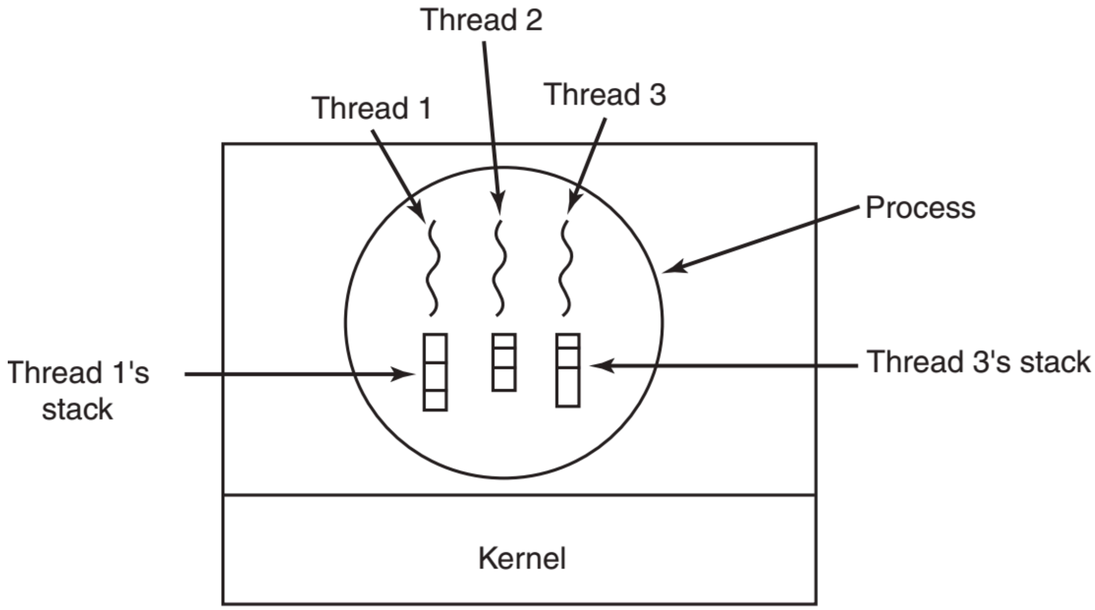


Figure 2-13. Each thread has its own stack.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.104, 105.

Why “threads” ?

- Threads can be switched faster than processes
  - No need for switching regarding the state of resource use
  - …
- May be able to write programs naturally
  - In case multiple inputs and outputs have to be processed simultaneously
  - …

日本語資料

Chap.2 プロセス・スレッド

- 2.1 プロセス
  - 2.1.1 プロセスモデル
  - 2.1.2 プロセス生成
  - 2.1.3 プロセスの終了
  - 2.1.4 プロセスの階層
  - 2.1.5 プロセスの状態
  - 2.1.6 プロセスの実現
  - 2.1.7 マルチプログラミングのモデリング
- 2.2 スレッド
  - 2.2.1 スレッドのモデル
  - 2.2.2 古典的なスレッドモデル
  - 2.2.3 POSIX スレッド
  - 2.2.4 ユーザ空間におけるスレッドの実現
  - 2.2.5 カーネル内でのスレッドの実現
  - 2.2.6 ハイブリッドな実現
  - 2.2.7 スケジューラアクティベーション
  - 2.2.8 ポップアップスレッド
  - 2.2.9 単一スレッド用コードのマルチスレッド化

プロセスモデル

- プロセスとは
  - 実行中のプログラムを表す抽象。
  - 逐次的に実行される(逐次プロセス)
  - 命令コードの集合 + 変数、プログラムカウンタ、レジスタの現在の値を持つ
    → 各プロセスは自身の CPU を持っているように見える。

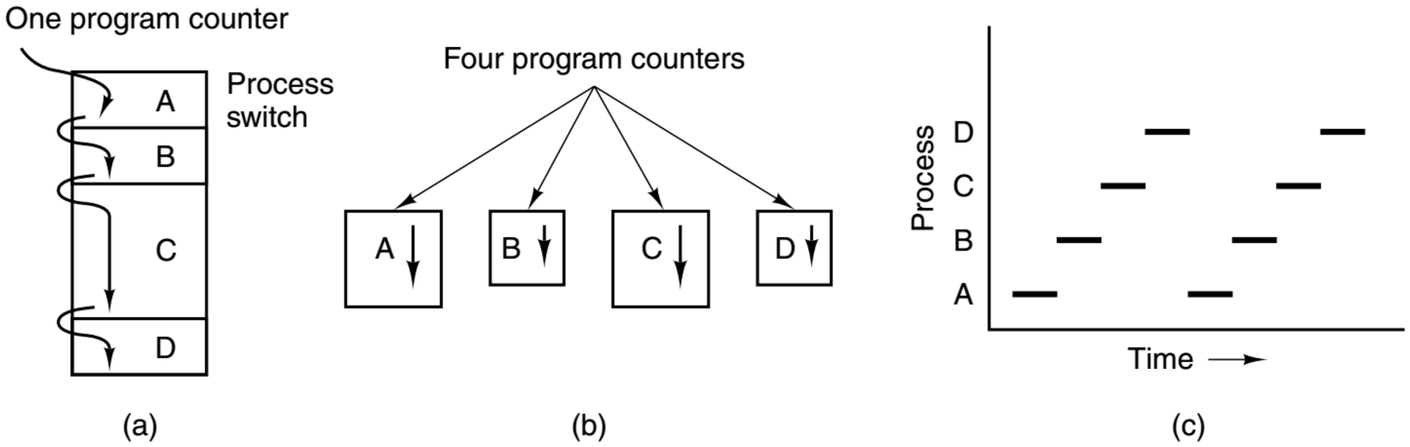


Figure 2-1. (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.87.

プロセスとプログラムの違い・割り込み

- 教科書のアナロジー (Cooking Computer Scientist)
  - 環境
    - ケーキのレシピ ⇔ プログラム
    - 小麦粉・卵・砂糖・バニラなど ⇔ 入力
    - コンピュータ科学者 ⇔ Processor (CPU)
  - 実際のケーキ作り
    - レシピを読み(指示に従いながら)小麦粉、卵、砂糖、バニラ(入力)などを適切に調理(処理)し、ケーキを作る(出力)
  - 優先度の高い事件の発生
    - 息子が蜂に刺され(イベントの発生)、コンピュータ科学者のところへ来た(イベントの到着)
    - コンピュータ科学者は、現在の仕事を中断する(割り込みの発生)
    - コンピュータ科学者(CPU)は、応急処置の本(プログラム)を読み、適切な医薬品(入力)で息子に治療を施す(出力)
    - もとの作業(ケーキ作り)に戻る(割り込みからの復帰)

プロセスの生成

- プロセスが生成される要因
  - システムの初期化
    - 通常、1 つ以上(多数)のプロセスが作成される。
    - これらの中には、バックグラウンドで実行され、特定のユーザーとは関係なく、何か特定の機能を持つものもある。
  - 実行中のプロセスから発行されたプロセス生成のシステムコールの実行
    - 実行中のプロセスが、新しい仕事を検出し、新しいプロセスにその処理を任せることを決定した場合。
  - ユーザから新しいプロセスの生成要求があった場合
    - 会話型システムにおいて、ユーザがプログラムの実行を指示した場合。
  - バッチジョブの開始時
    - ユーザが与えたジョブ(プログラム+入力データ)は待ち行列に入っており、システムがジョブを取り出して処理する。

プロセスの生成には何が必要か

- = プロセスを構成している情報とは何か
  - プログラム (実行形式) の準備。e.g. ファイルシステム上での検索など
  - プログラム (実行形式) のメモリ上への展開
  - スタックの初期化
  - 変数領域の初期化
  - レジスタ類の初期化
  - 入出力(状態)の初期化 (入力・出力があらかじめ指定されている場合)
- これがどれくらい大変な作業か?
  - プロセスを表す構造体 (struct proc)

プロセス生成の方法

- プロセスの属性(プロセス構造体の内容)を明示的に指定する方法
  - JCL (Job Control Language)
    - https://en.wikipedia.org/wiki/Job_Control_Language
  - JCL の例

```
//IS198CPY JOB (IS198T30500),'COPY JOB',CLASS=L,MSGCLASS=X
//COPY01   EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=OLDFILE,DISP=SHR
//SYSUT2   DD DSN=NEWFILE,
//            DISP=(NEW,CATLG,DELETE),
//            SPACE=(CYL,(40,5),RLSE),
//            DCB=(LRECL=115,BLKSIZE=1150)
//SYSIN  DD DUMMY
```

- 親プロセスのコピーを生成する方法
  - Unix の fork() システムコール

プロセスの終了

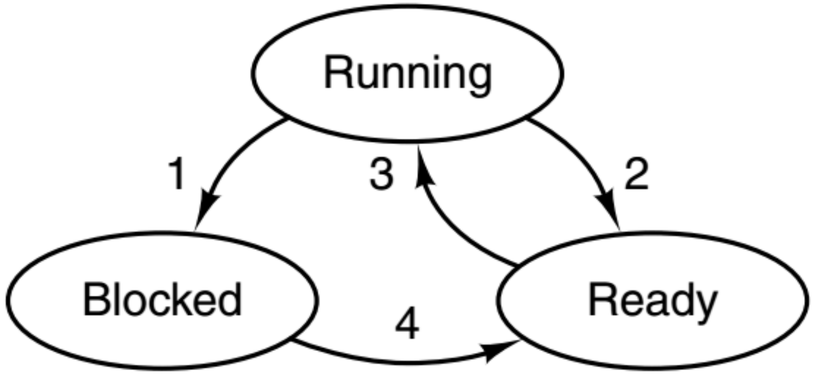
- プロセス終了の要因
  - 正常終了 (自発的)
  - エラーで終了 (自発的)
  - 致命的なエラー (システムが検出、非自発的)
  - 他プロセスによる強制終了 (非自発的)
- プロセス終了の処理
  - 資源の解放
  - プロセス実行に関わる最終的な結果の通知

プロセスの階層 (プロセスの関係)

- プロセス間に関係がある場合
  - 親子の関係
  - UNIX ライクなシステム
  - 木構造(プロセスツリー)を構成する
- プロセス間に関係がないシステム
  - Windows
  - バッチシステム

プロセスの状態

- プロセスに対する CPU の割り当て状態に注目した「状態」
- 重要な概念: ブロック = 入出力の終了待ち (+資源の空き待ち)



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.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.93.

プロセスとスケジューラ

- スケジューラ: sched()
- プロセスが発行したシステムコールの処理中に「入出力待ち」になった場合に呼び出される。
- アルゴリズム
  - 現在のプロセス状態を保存し、状態を BLOCKED にする。
  - READY 状態にあるプロセスの集合の中からプロセスを 1 つ選ぶ(スケジューリングプロシーに基づく選択)。
  - 選んだプロセスが実行可能なように、待避されているプロセス状態を復帰させ、状態を RUNNING にする。

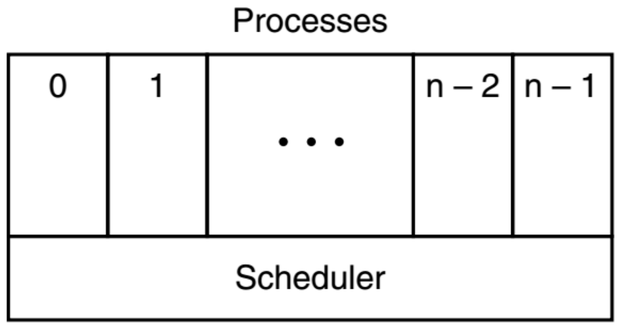


Figure 2-3. The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.94.

プロセスの実現

Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	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		

Figure 2-4. Some of the fields of a typical process-table entry.

Source: Andrew S. Tanenbaum. "Modern Operating Systems", 4th ed., global ed., pp.95.

割り込み処理

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 decides which process is to run next.
7. C procedure returns to the assembly code.
8. 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.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.96.

割り込み・割り出し・モード・upper/lower half

- 割り込みと割り出し
  - 割り込み (ハードウェア割り込み)
  - 割り出し (ソフトウェア割り込み)
- モード
  - ユーザモード
  - カーネル (システム) モード
- Lower half v.s. Upper half
  - Lower half: ハードウェア割り込みで実行される範囲
  - Upper half: ソフトウェア割り込みで実行される範囲
  - どこで交わるのか?

スレッド

- プロセス再訪
  - プロセス = 実行状態 + 資源の利用状態
  - スレッドモデルでは
    - プロセス = 資源の利用状態
    - スレッド = 実行状態

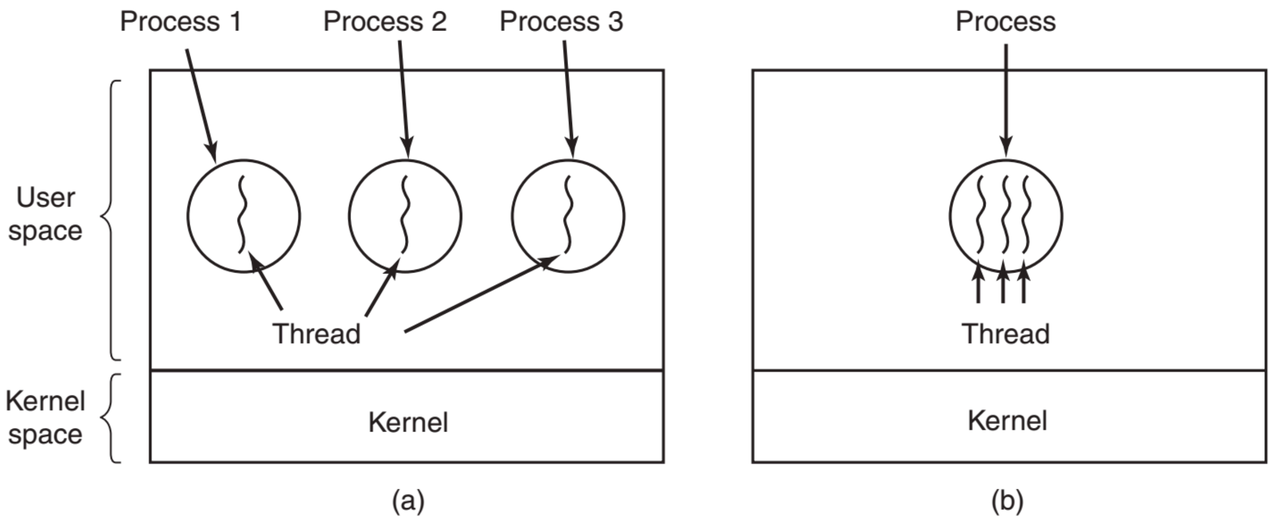


Figure 2-11. (a) Three processes each with one thread. (b) One process with three threads.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.104.

実行管理と資源管理

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 2-12. The first column lists some items shared by all threads in a process. The second one lists some items private to each thread.

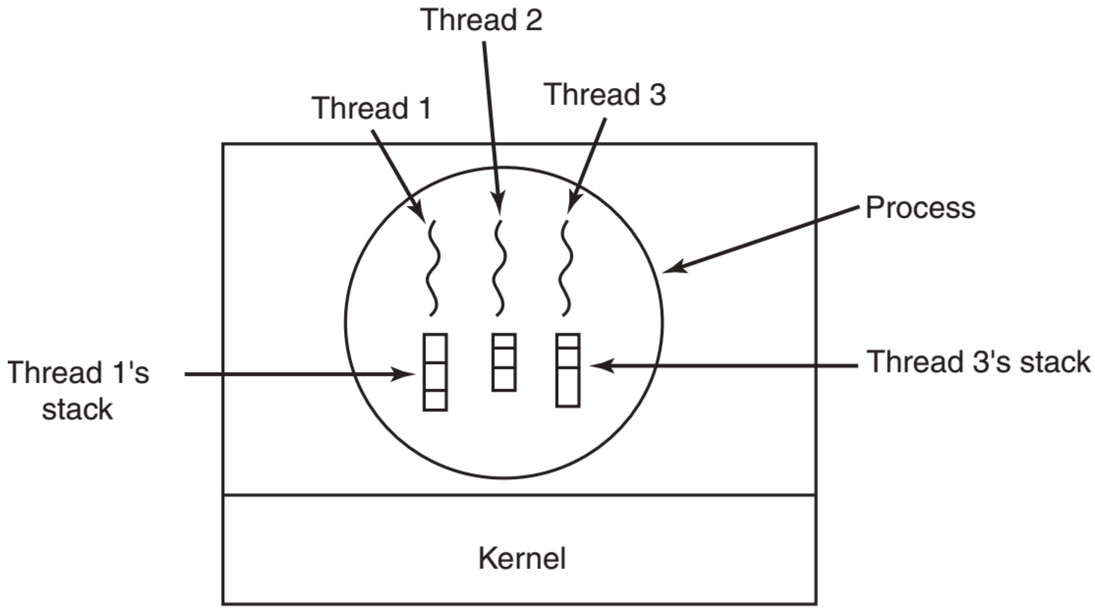


Figure 2-13. Each thread has its own stack.

Source: Andrew S. Tanenbaum. “Modern Operating Systems”, 4th ed., global ed., pp.104, 105.

なぜスレッドか

- スレッドはプロセスに比べて高速に切り替えができる
  - 資源の利用状態に関する入れ替えの必要がない。
  - …
- 自然なプログラムが書ける場合がある
  - 複数の入出力を同時に処理(ケア)しなければならない場合。
  - …