LINGI 1113: Systèmes informatiques 2

Mission 3: gestion des processus et threads par les systèmes d'exploitation

ÉCOLE POLYTECHNIQUE DE LOUVAIN

UCL

Les Processus

Processus = programmes en cours d'execution

Services nécessaires:

- de la mémoire pour le programme et les données
- un processeur quand on en a besoin
- un répositoire non volatil d'information (système de fichiers)
- accès à des dispositifs d'entrées/sorties sans avoir à se préoccuper de détails hardware

Services offert par: le système d'exploitation

Les données d'un processus

Utilisées par le processus
- Programme
- Données
- Pile

Utilisées par l'OS:
process control bloc ->

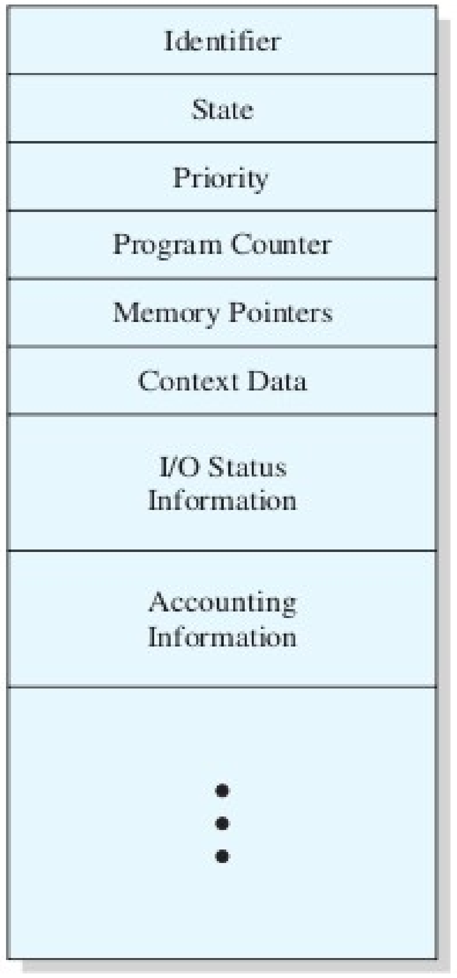


Figure 3.1 **Simplified Process Control Block**

Les états d'un processus

Version simplifiée:

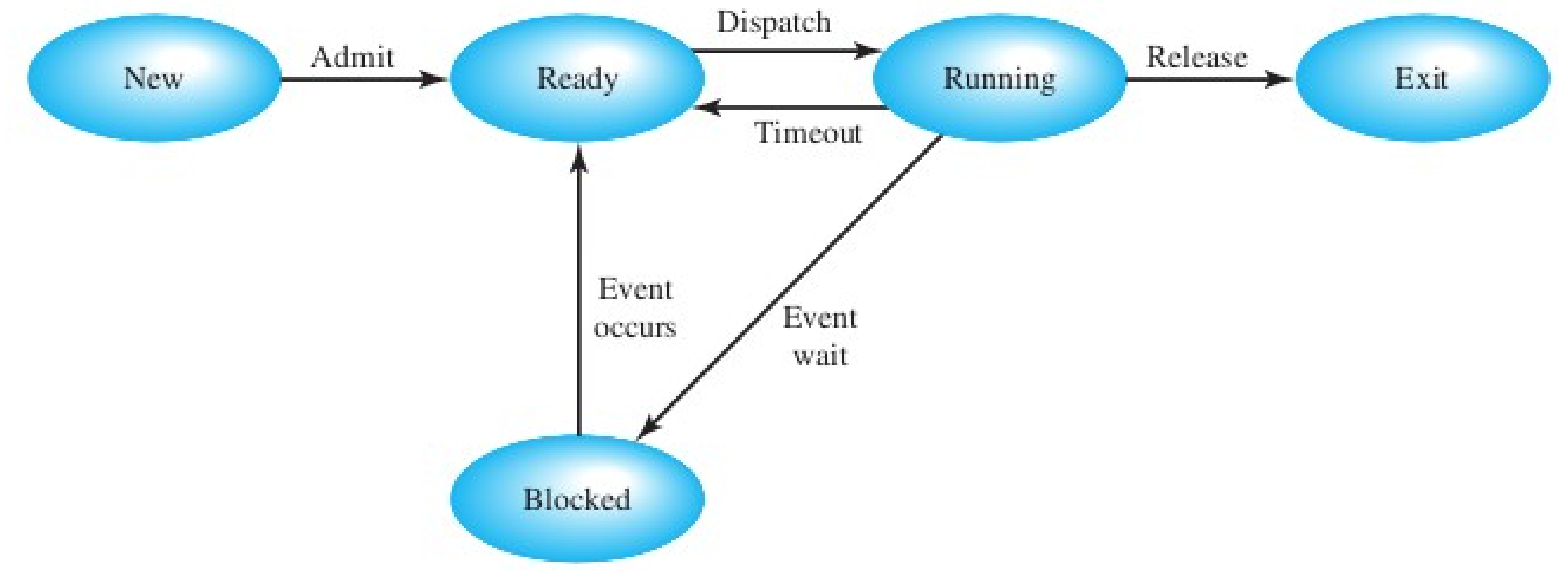


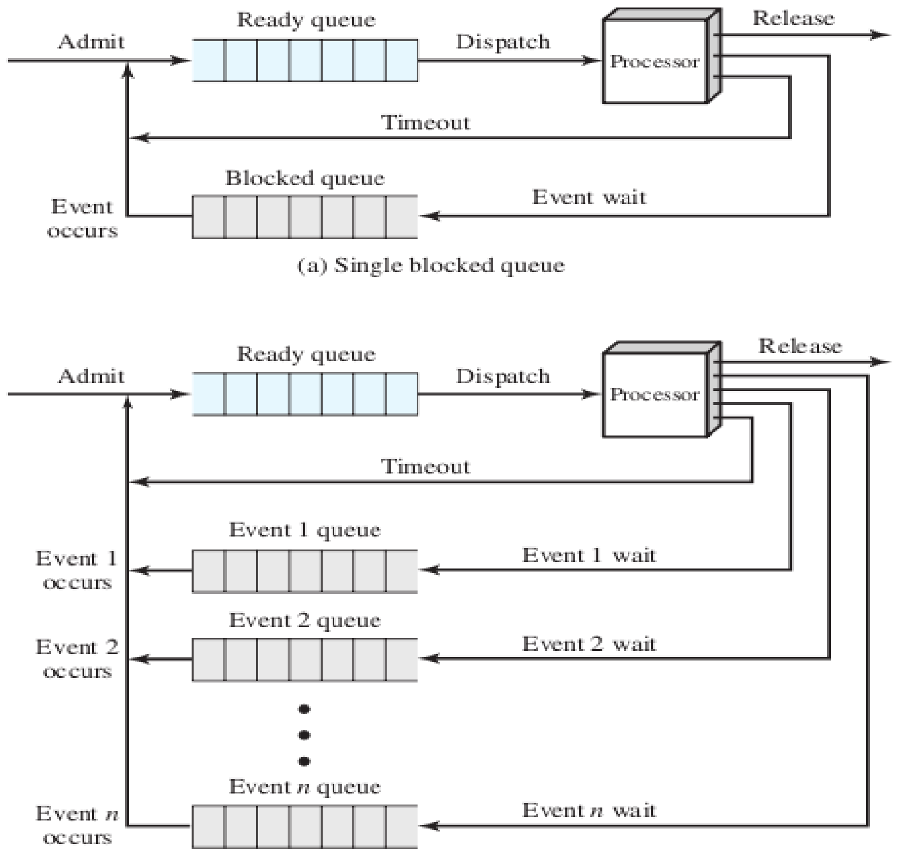
Figure 3.6 **Five-State Process Model**

release: fin ou erreur détectée par processeur (trap) ou l'OS (p.ex: crédit calcul épuisé) ou l'utilisateur (Ctl-C, Ctl-\)

Les états d'un processus

Les structures de données de l'OS pour gérer ces états

•



(b) Multiple blocked queues

Figure 3.8 **Queuing Model for Figure 3.6**

Les états d'un processus

Version plus complète

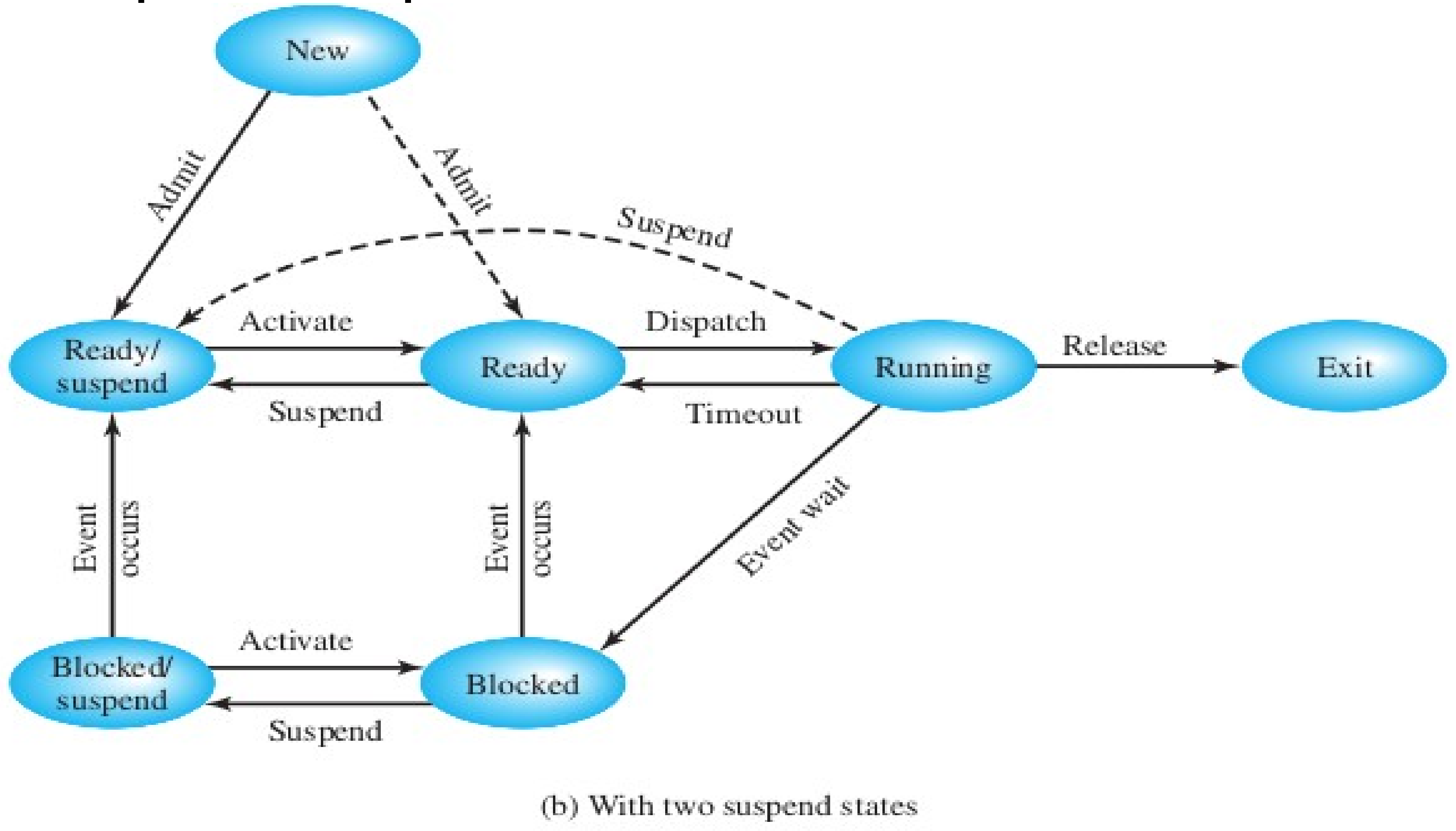


Figure 3.9 Process State Transition Diagram with Suspend States

suspended: pas considéré par l'ordonnanceur (scheduler)

p.ex. « swapped », périodique, Cntl-Z, débuggé

Le process control block

Table 3.5 Typical Elements of a Process Control Block

Process Identification

Numeric identifiers stored in process control block include
• Identifier of this process
• Identifier of the process that created this process (parent process)
• User identifier

Processor State Information

User-Visible Registers (addressable in machine language)

Control and Status Registers

• Program counter: contains the address of the next instruction to be fetched
• Condition codes: result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
• Status information: Includes interrupt enabled/disabled flags, execution mode

Stack Pointers

Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack. One user mode stack per thread, one system stack.

Le process control block

Process Control Information

Scheduling and State Information

Typical items :

• Process state: running, ready, waiting,...

• Priority

• Scheduling-related information:

e.g. amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.

• Event the process is awaiting before it can be resumed.

Data Structuring

• pointers for the queue associated to the processa state

• identification of parents and children

Interprocess Communication

flags, signals, and messages

Process Privileges

e.g. Access to other procvesses memory for debuggers

Memory Management

Resource Ownership and Utilization

UCL

Le process control block

Les process control blocks sont les structures de données les plus importantes d'un système d'exploitation

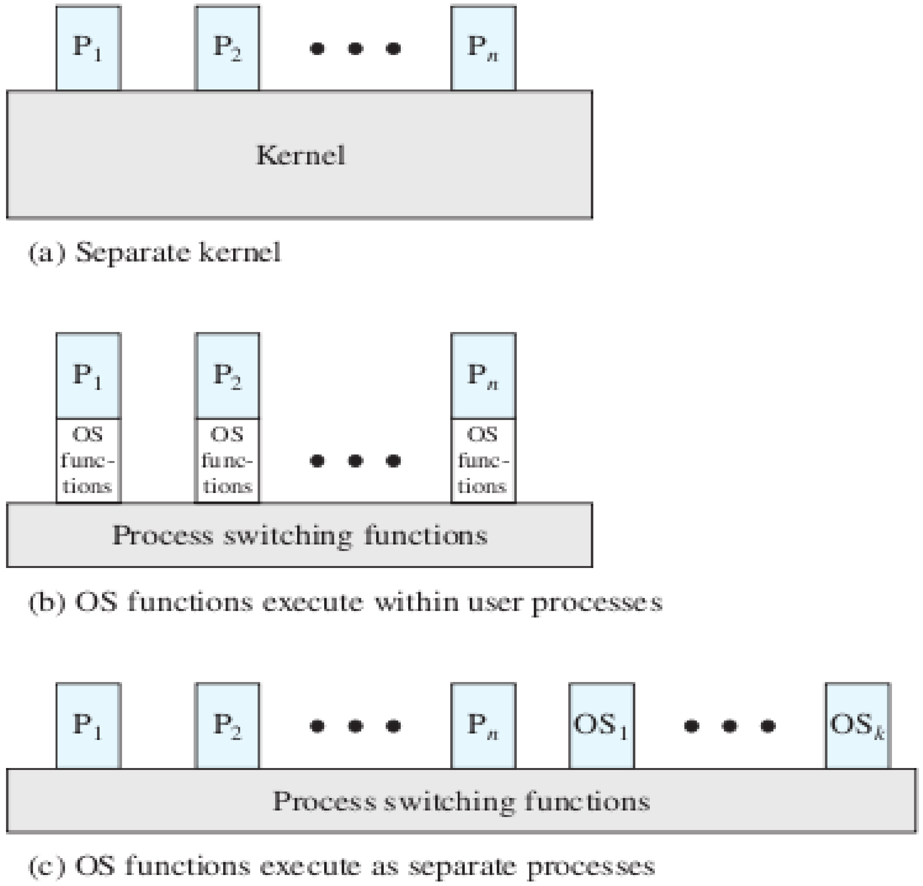
- Contiennent toutes les informations sur le processus dont l'OS a besoin
- Doivent être lus et/ou modifiés par de nombreuses parties de l'OS
  - ordonnancement (scheduling),
  - allocation de ressources,
  - gestion des interruptions,
  - monitoring et analyse des performances.
- **=> R I S QU E s**
  - Peut être corrompu en cas de bug dans le sytème
  - Si on en modifie la structure, modifications à apporter dans tout le système
  - => en faire des objets dont les attributs sont privés et ne peuvent être manipulés que par les méthodes de l'objet... ou
    être très prudent lorsqu'on y touche en programmant dans l'OS

Les processus et le noyau

Noyau hors de tout processus

Chaque processus peut fonctionner en mode utilisateur et en mode noyau

L'OS est un ensemble de processus et un micro-noyau



P1
P2
Pn
Kernel
(a) Separate kernel
OS func-tions
Process switching functions
(b) OS functions execute within user processes
OS1
OSk
(c) OS functions execute as separate processes

Figure 3.15 Relationship between Operating System and User Processes

Les processus de Unix

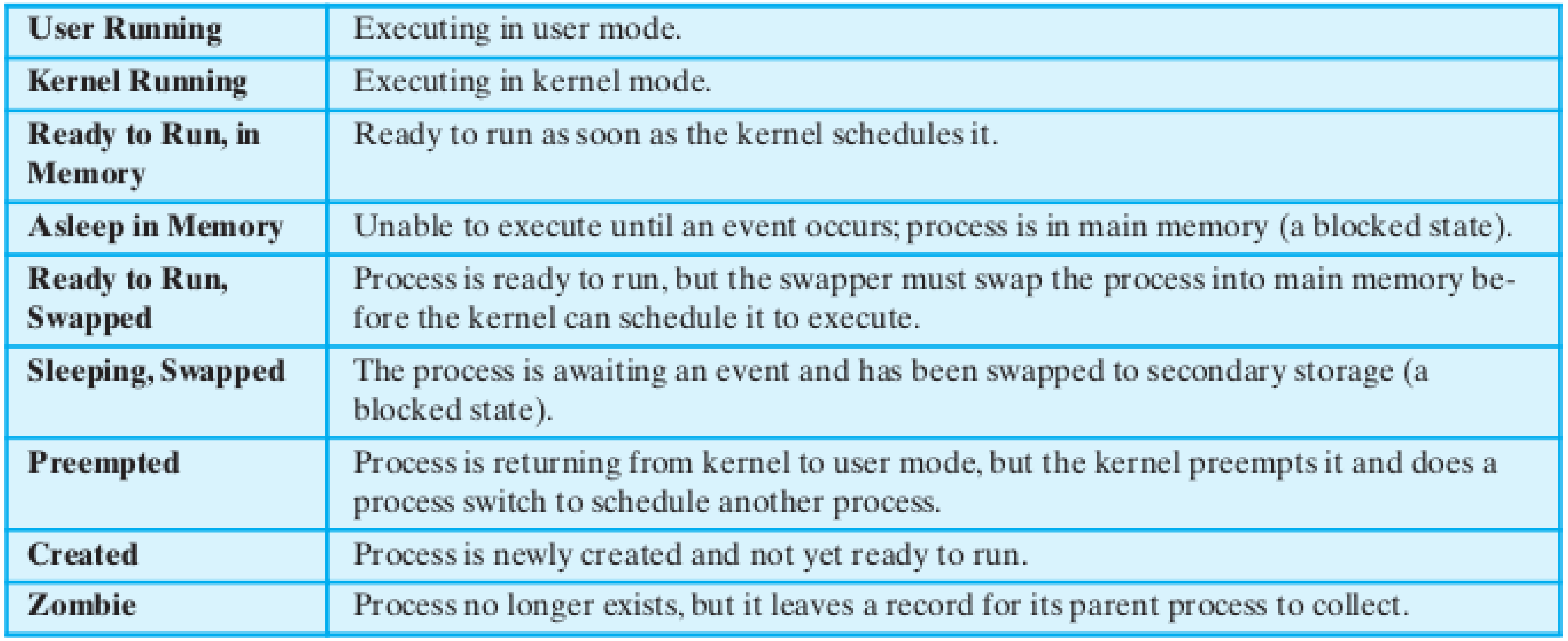


Table 3.9 UNIX Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

Les processus de Unix

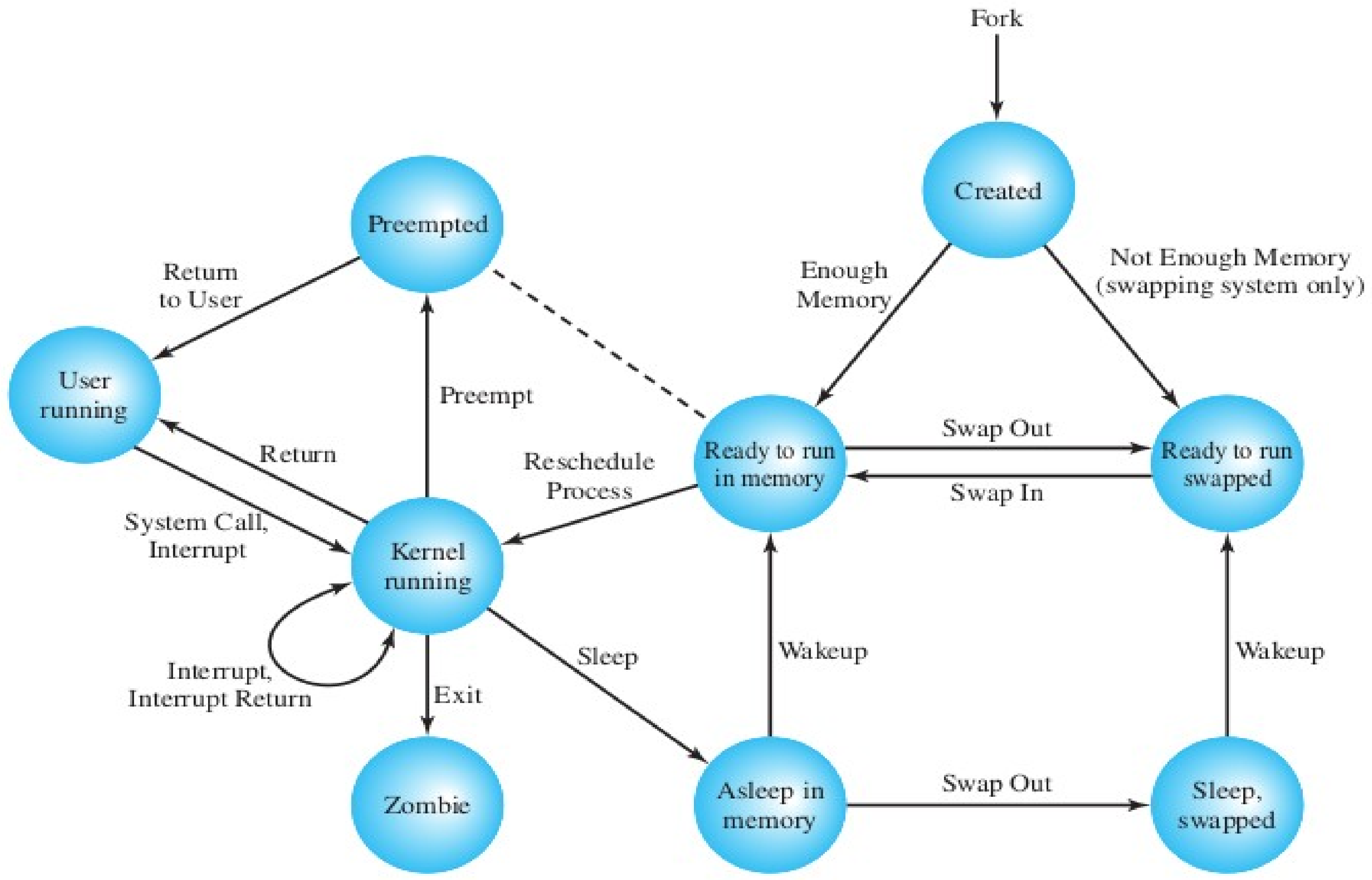


Figure 3.17 **UNIX Process State Transition Diagram**

Les processus de Unix

UNIX Process Image

<u>User level context</u>

Text - Data - Stack - Shared memory

<u>Register context</u>

PC - PSW - SP - GPR

<u>System level context</u>

Process table entry: ce dont l'OS peut toujours avoir besoin

U(ser) area: ce dont l'OS a besoin quand il est le processus en mode noyau

Per process region table: correspondance adresses virtuelles/adresses physiques + permisions (R/W/E)

Kernel stack

Les processus de Unix

Process table entry

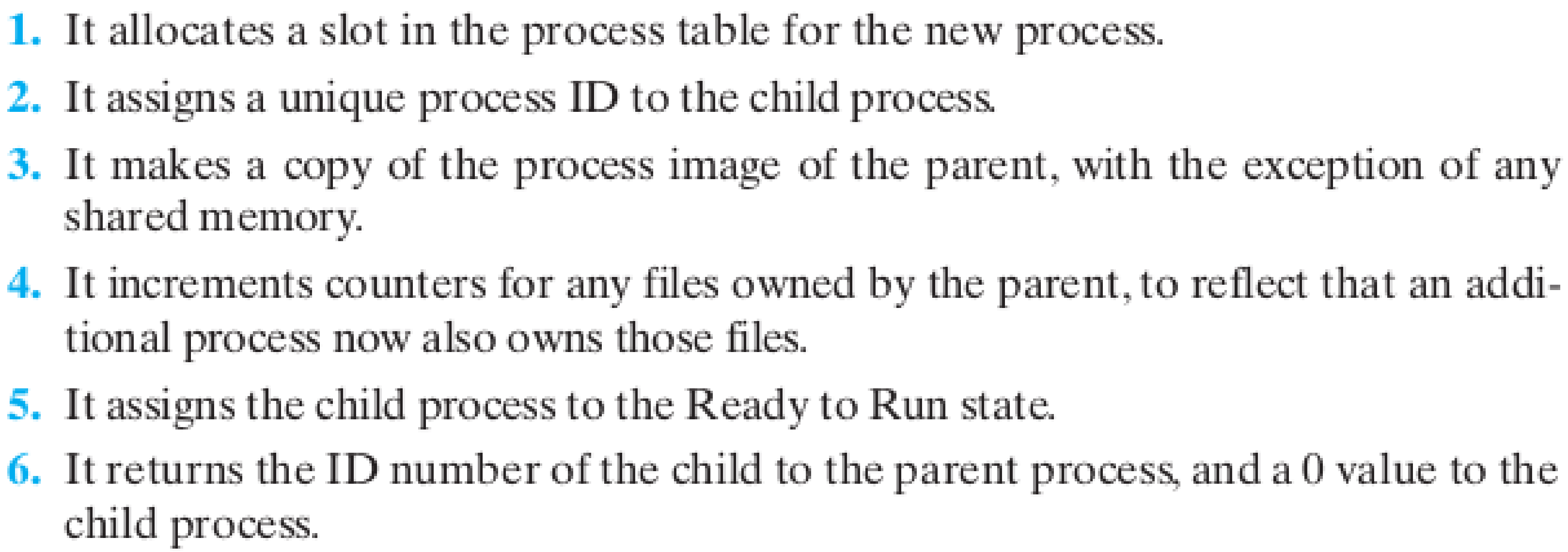
- état du processus
- pointeurs vers txt, data;stack et Uarea
- UID, EID
- PID
- Event descriptor (when sleeping)
- Priority
- Pending Signals
- Timers (process execution time, resource utilization, user set timers)
- Pointer to next process in ready queue
- Memory status (in, out, locked in, may be swapped)

User area

- Pointeur vers process table entry
- UID, EID
- Timers (time spent by proc in user mode & kernel mode)
- Signal handler array
- Control terminal, if any
- Error field for returning syscalls
- Sys call return value
- I/O parameters (for read & write)
- File parameters (current dir, etc.)
- Process file table
- Max size of proc and created files)
- Mode mask for files created by proc

Les processus de Unix

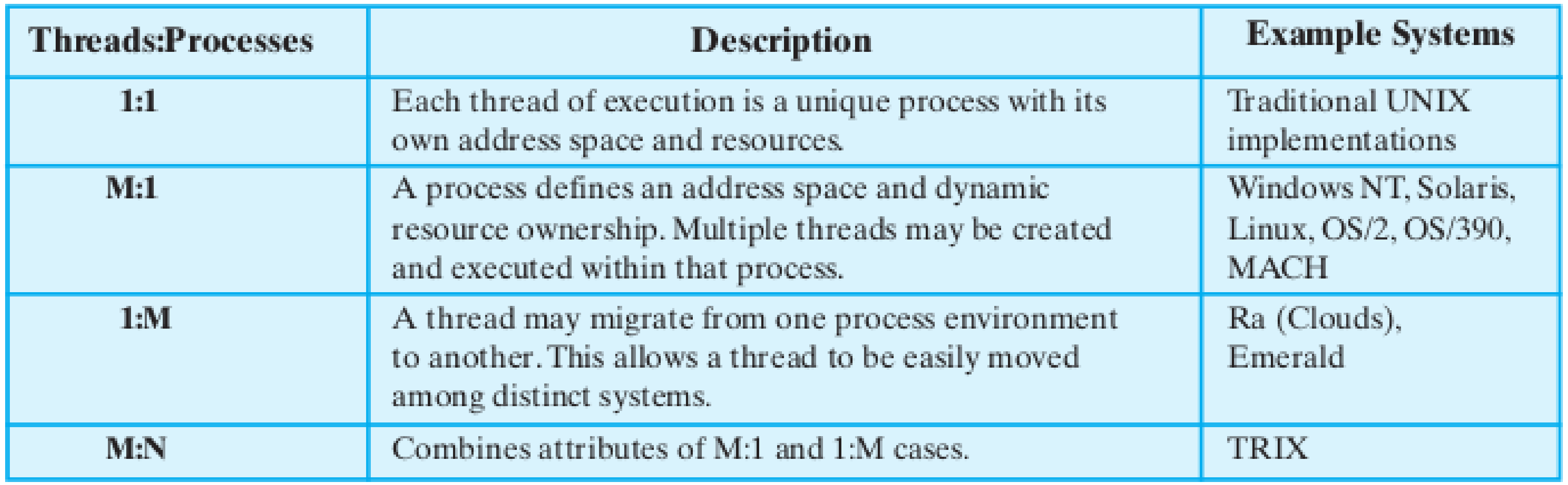
Process creation by the kernel (fork)



1. It allocates a slot in the process table for the new process.
2. It assigns a unique process ID to the child process.
3. It makes a copy of the process image of the parent, with the exception of any shared memory.
4. It increments counters for any files owned by the parent, to reflect that an additional process now also owns those files.
5. It assigns the child process to the Ready to Run state.
6. It returns the ID number of the child to the parent process, and a 0 value to the child process.

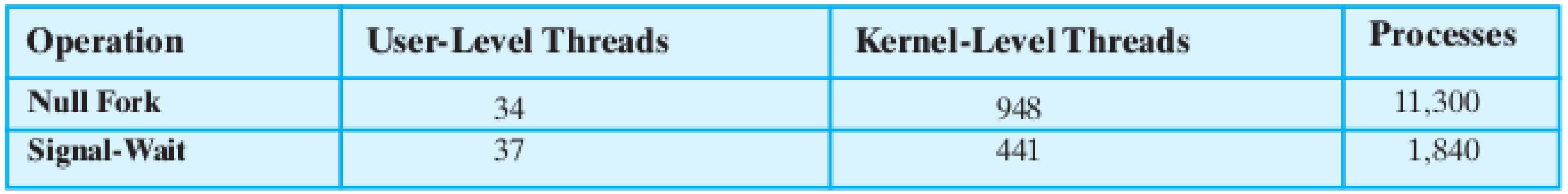
Les processus et les threads

Table 4.2 Relationship between Threads and Processes



Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Table 4.1 Thread and Process Operation Latencies (μs)



Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal-Wait	37	441	1,840

Les processus et les threads

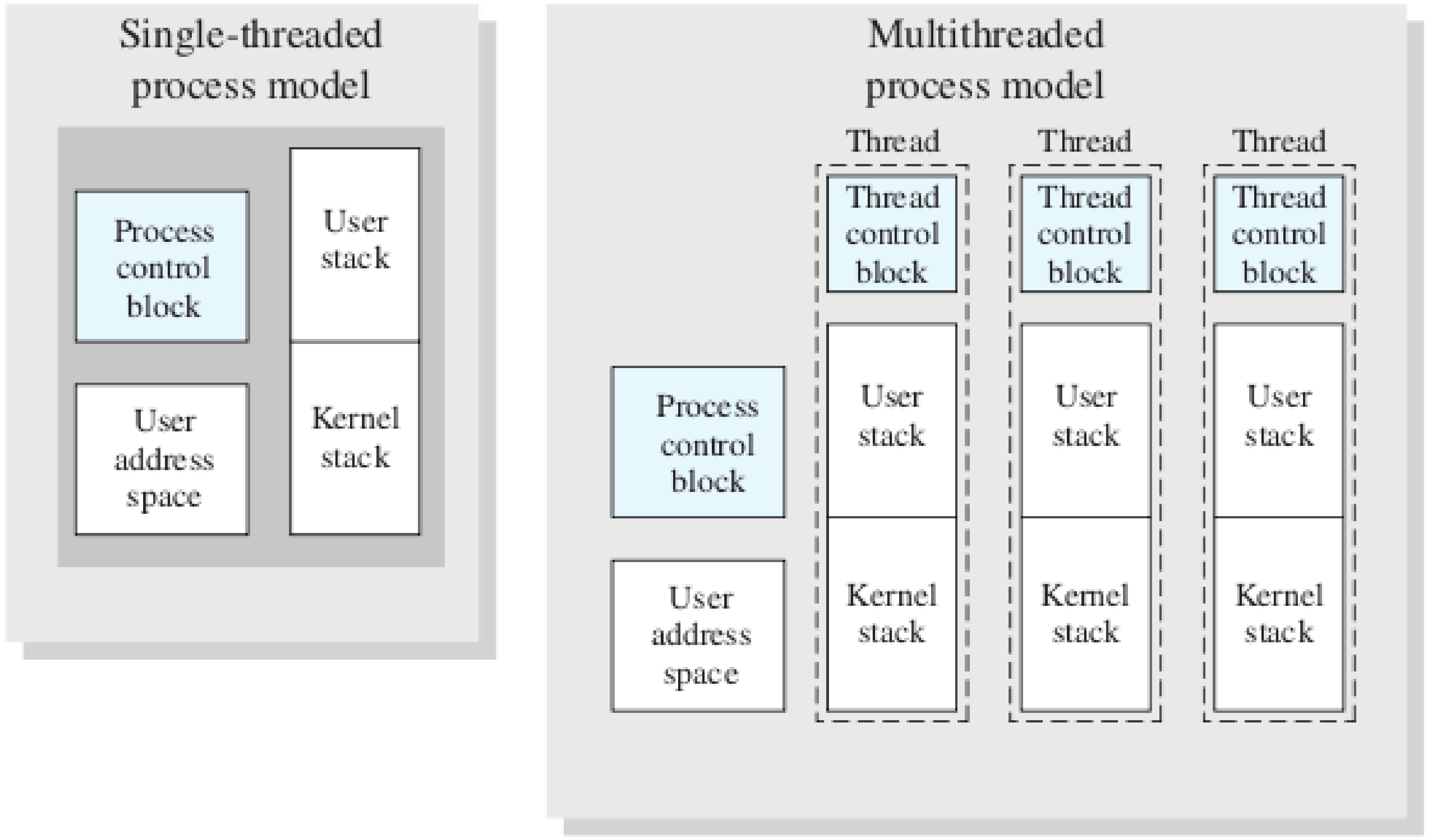


Figure 4.2 **Single Threaded and Multithreaded Process Models**

Les processus et les threads

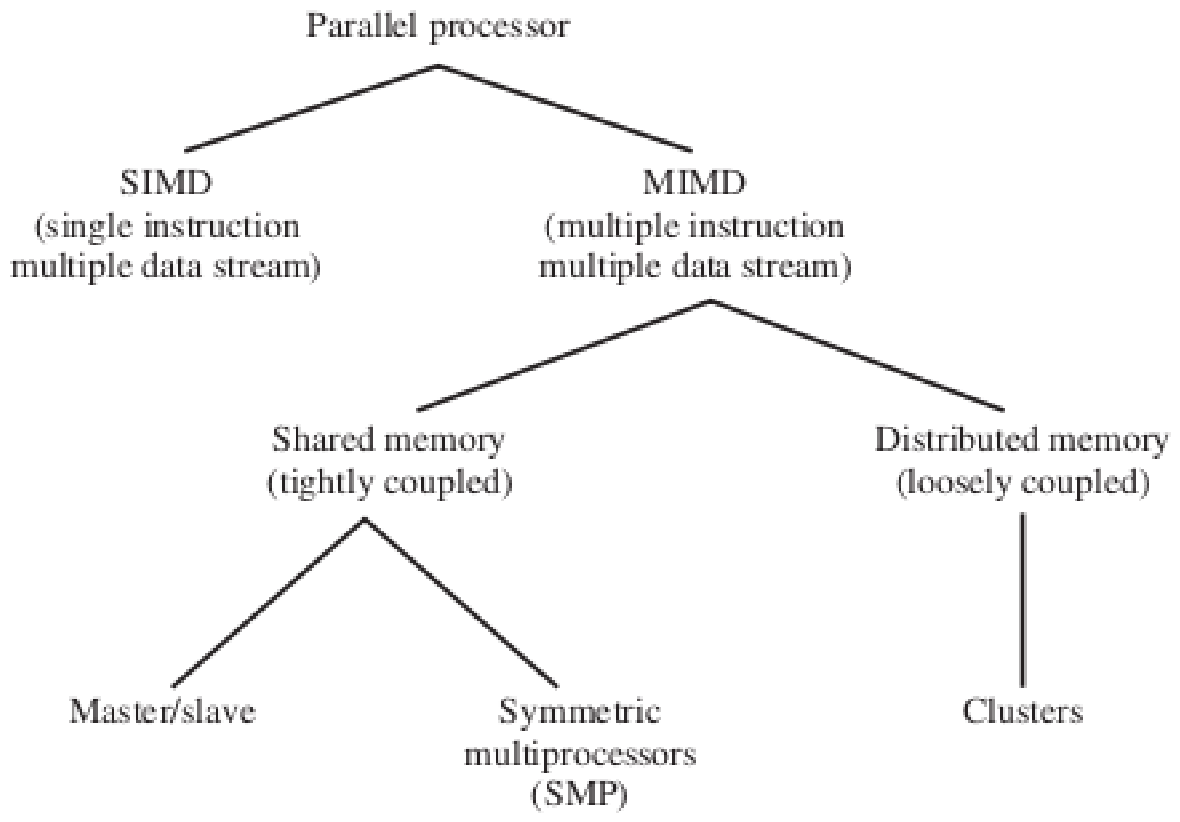

Parallel processor
SIMD (single instruction multiple data stream)
MIMD (multiple instruction multiple data stream)
Shared memory (tightly coupled)
Distributed memory (loosely coupled)
Master/slave
Symmetric multiprocessors (SMP)
Clusters

Figure 4.8 Parallel Processor Architectures

Les processus et les threads

Exemple SOLARIS

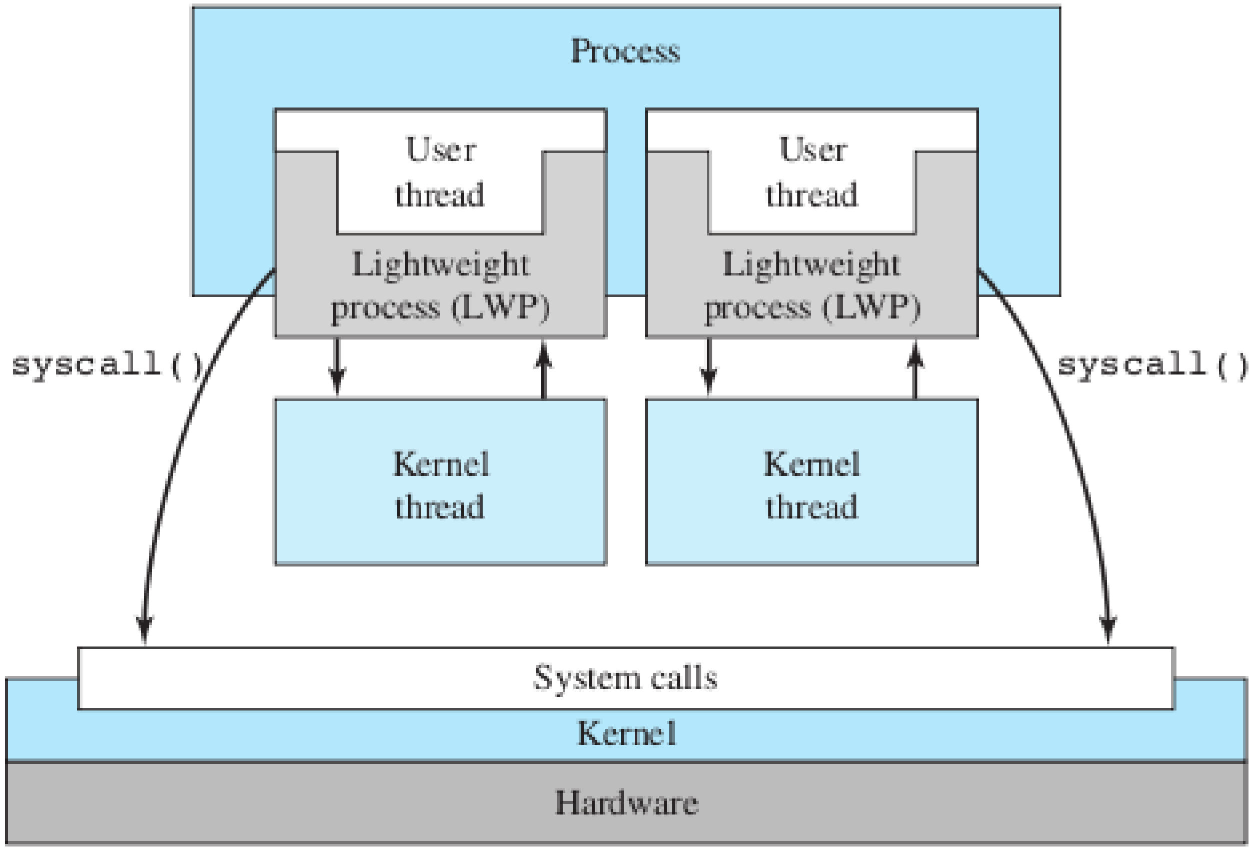
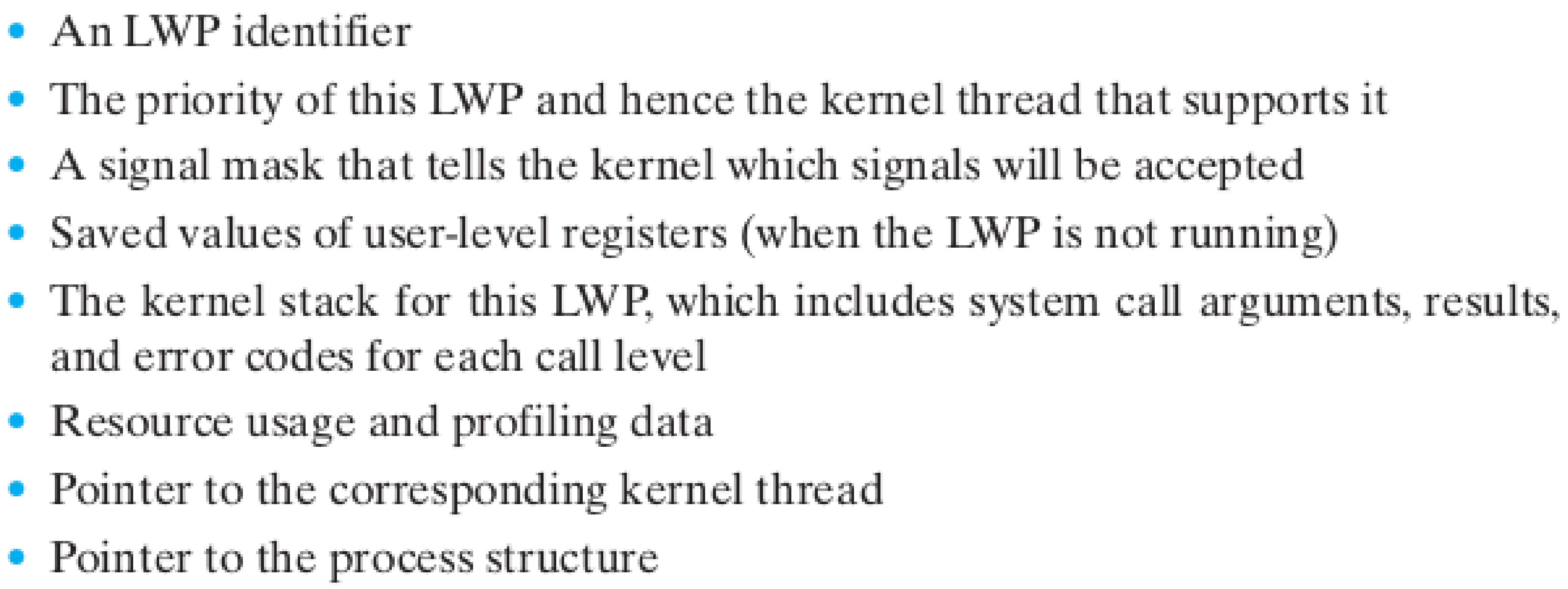


Figure 4.15 **Processes and Threads in Solaris [MCDO07]**

Les processus et les threads

Exemple SOLARIS

La stucture de données LWP



- An LWP identifier
- The priority of this LWP and hence the kernel thread that supports it
- A signal mask that tells the kernel which signals will be accepted
- Saved values of user-level registers (when the LWP is not running)
- The kernel stack for this LWP, which includes system call arguments, results, and error codes for each call level
- Resource usage and profiling data
- Pointer to the corresponding kernel thread
- Pointer to the process structure

Les processus et les threads

Exemple SOLARIS

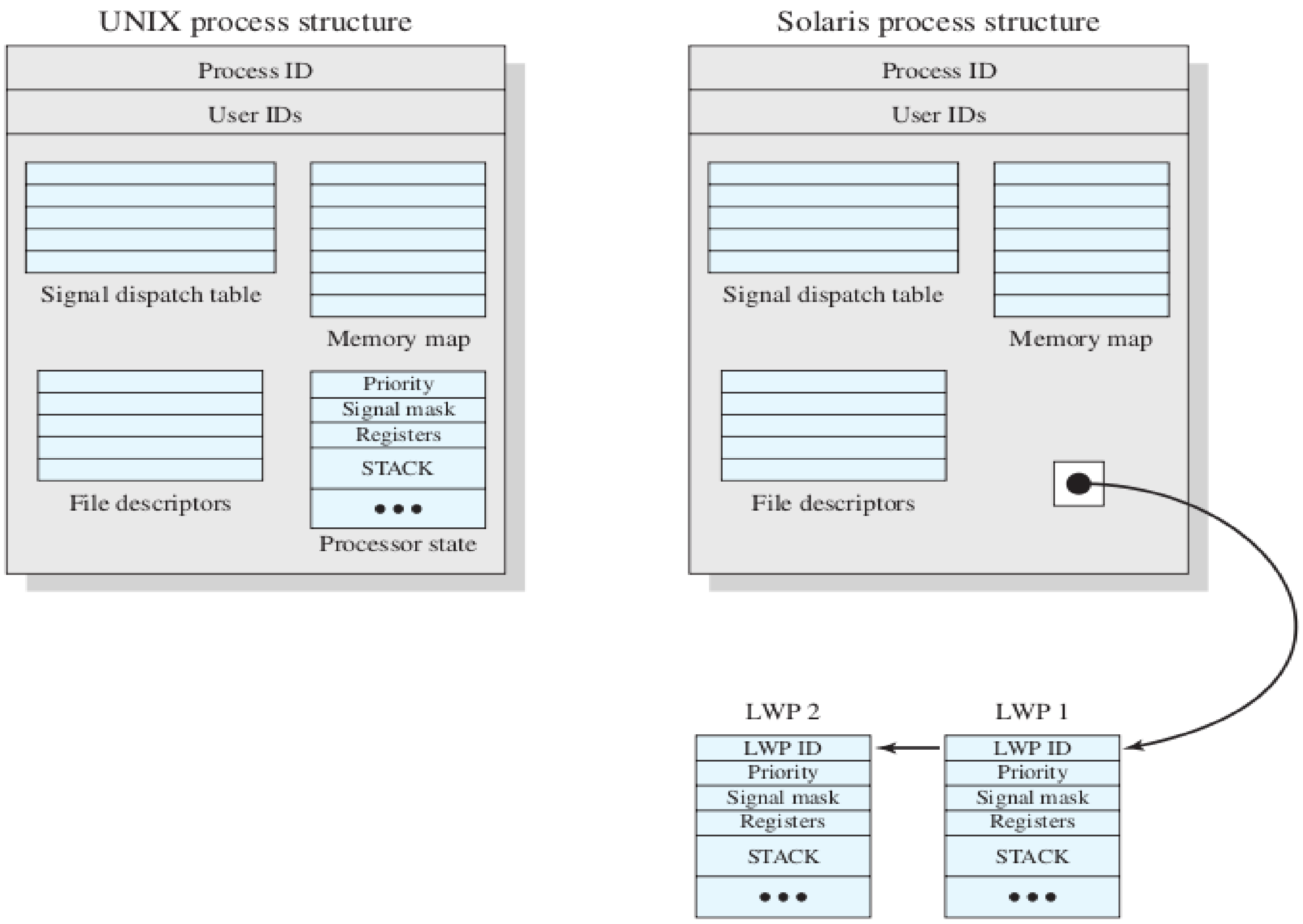


Figure 4.16 **Process Structure in Traditional UNIX and Solaris [LEWI96]**

Les processus et les threads

Exemple SOLARIS

Thread states

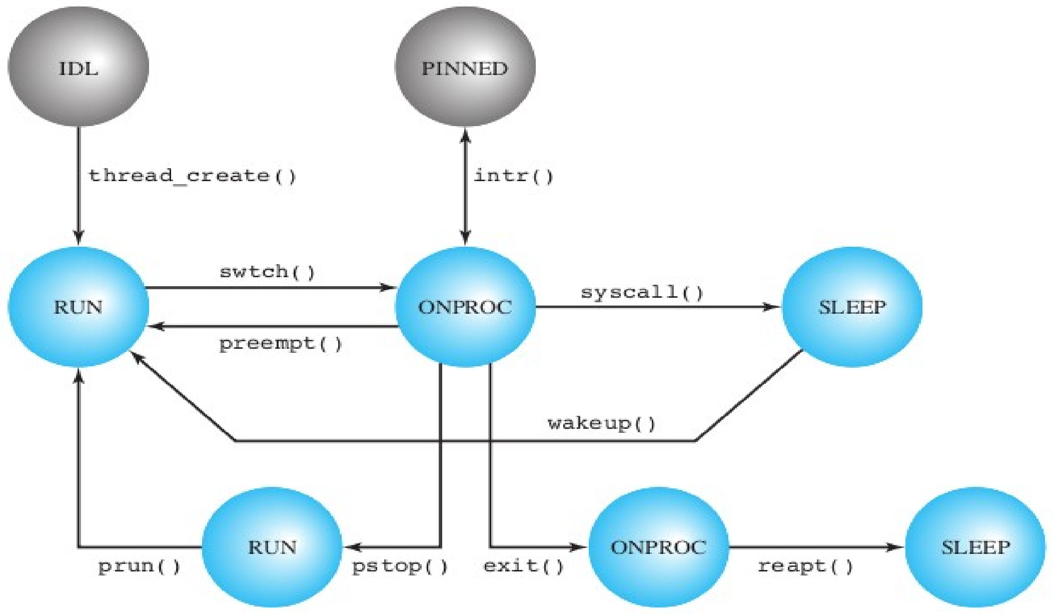


Figure 4.17 Solaris Thread States [MCDO07]

- **RUN:** The thread is runnable; that is, the thread is ready to execute.
- **ONPROC:** The thread is executing on a processor.
- **SLEEP:** The thread is blocked.
- **STOP:** The thread is stopped.
- **ZOMBIE:** The thread has terminated.
- **FREE:** Thread resources have been released and the thread is awaiting removal from the OS thread data structure.

Les processus et les threads

Exemple SOLARIS

Thread dans le noyau

- Le Noyau est multithreadé et préemptible
- Routines d'interruptions minimales, traitements dans des threads

Les processus et les threads

Exemple LINUX

- Les threads sont des processus qui partagent la mémoire et appartiennent au même « process group » (vieux concept UNIX venant de BSD: procs qui ont le même terminal de contrôle)
- Créés par « clone », une généralisation de « fork »: selon les paramètres, crée un processus ou un thread