Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes d'exploitation

Processus et Threads

Jacques Supcik | 2018/2019 | T-2adfg

l03-handouts | 2018-09-20

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz

Les processus

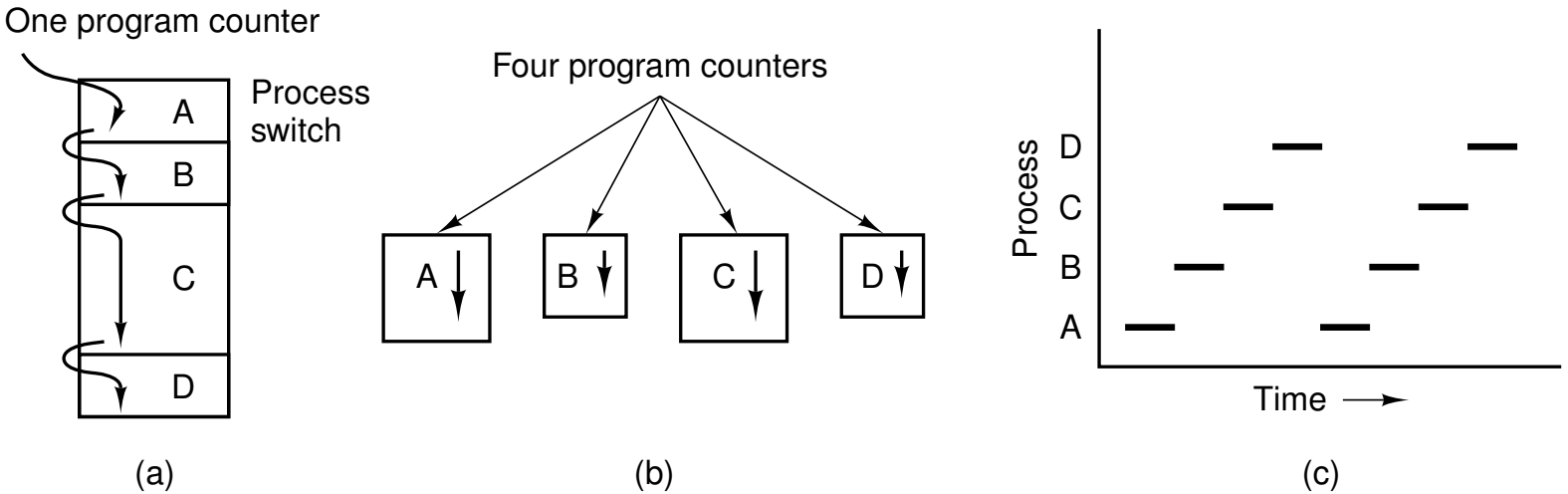
- Concept central d'un système d'exploitation
- Abstraction d'un programme en cours d'exécution
- Concurrence (pseudo parallélisme) même avec un seul processeur

Les processus

Différence entre un processus, un processeur et un programme

- Le programme est la recette de cuisine (algorithme)
- Le processeur est le cuisinier qui suit la recette
- Le processus est l'action de cuisiner (activité)
- Si le cuisiner se brûle, il remplace le livre de cuisine par celui des premiers secours et se soigne

Le modèle de processus



- (a) Multiprogrammation de 4 programmes
- (b) Modèle conceptuel de 4 processus séquentiels indépendants
- (c) Un seul programme est actif à un instant donné

La création de processus

Evénements provoquant la création de processus :

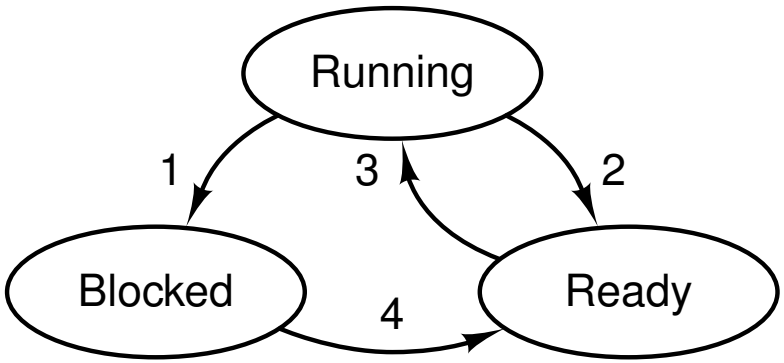
- Initialisation du système
- Exécution d'un appel système de création de processus en cours d'exécution (fork)
- Requête utilisateur sollicitant la création de nouveau processus
- Lancement d'un travail en traitement par lots (batch)

La fin de processus

Evénements provoquant la fin d'un processus :

- Arrêt normal (volontaire)
- Arrêt pour erreur (volontaire)
- Arrêt pour erreur fatale (involontaire)
- Le processus est arrêté par un autre processus (involontaire)

Les états des processus



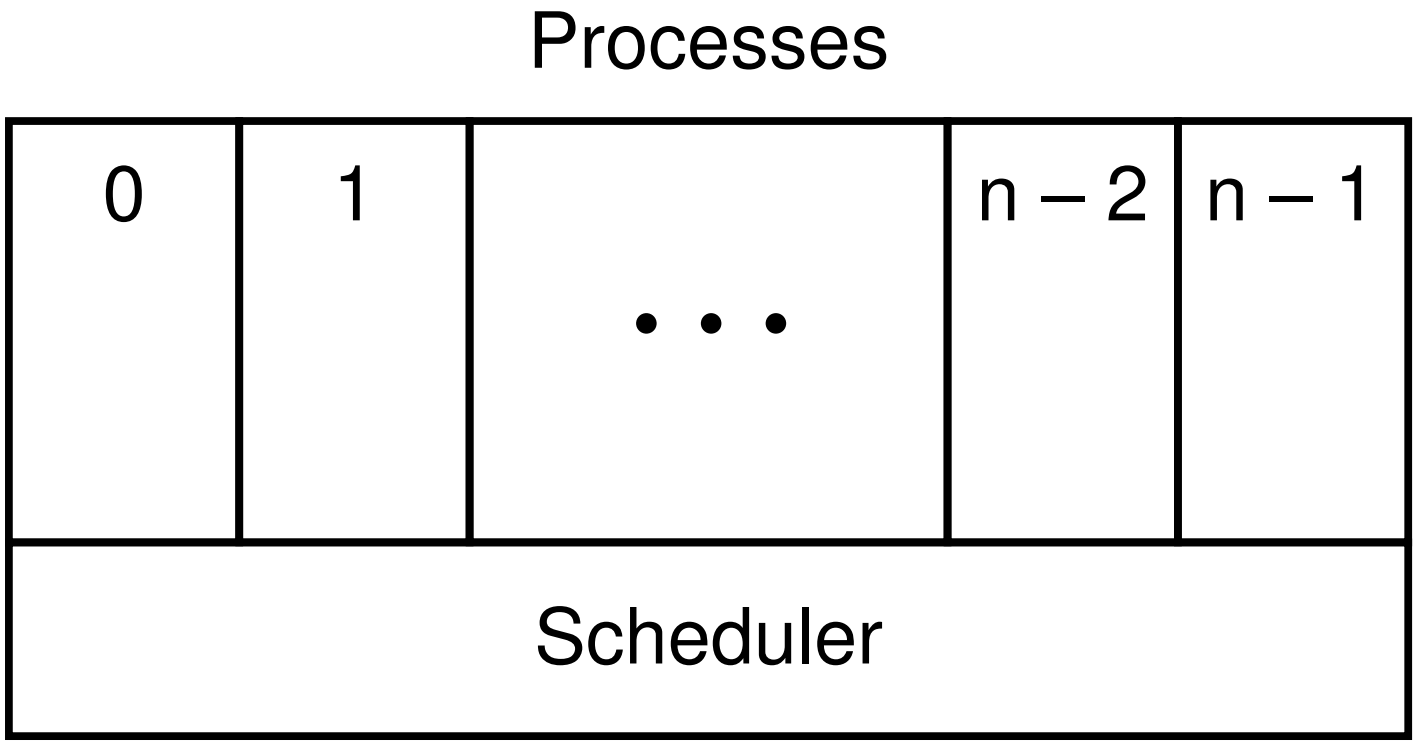
1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

Un processus peut être :

- en cours d'exécution (running)
- bloqué (blocked)
- prêt (ready)

Les transitions entre les états apparaissent dans cette figure

Implémentation des processus



La couche inférieure d'un système d'exploitation structuré en processus gère des interruptions et prend en charge l'ordonnancement (scheduler). Viennent s'y superposer les processus séquentiels.

Implémentation des processus

Quelques champs d'une entrée type de la table des processus

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		

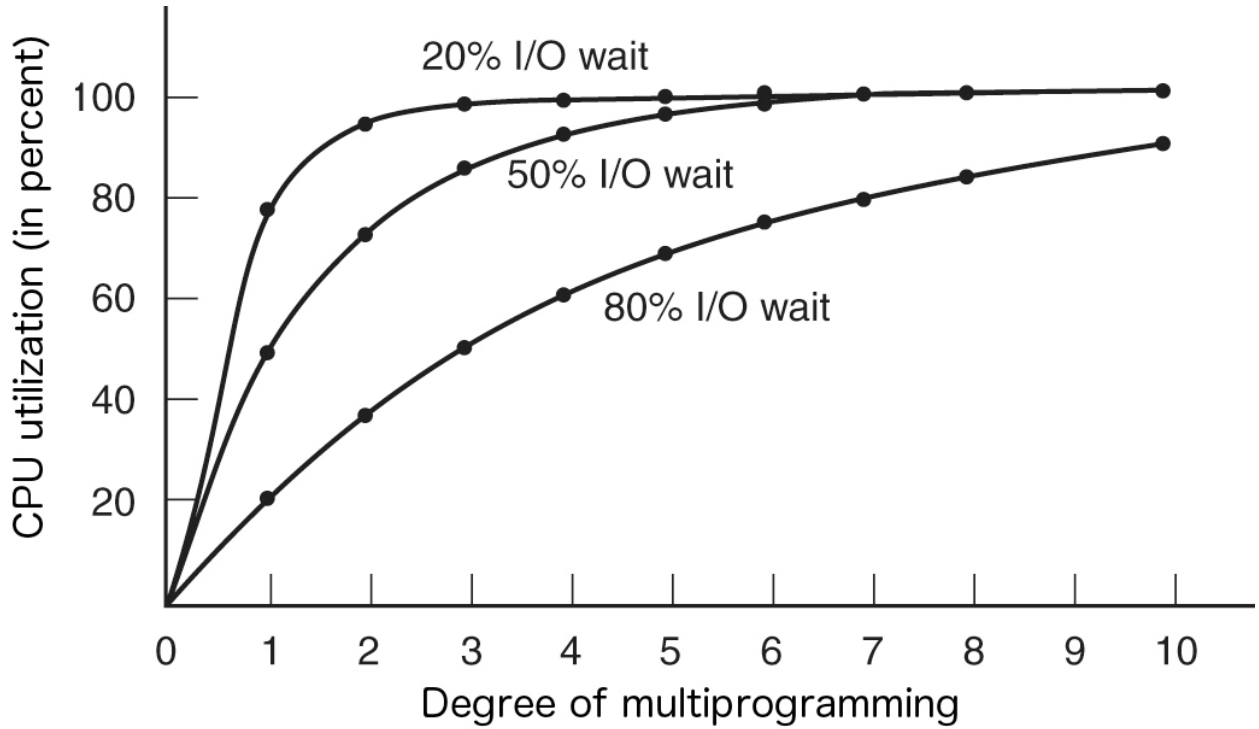
Implémentation des processus

Traitement «bas niveau» des interruptions

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.

Modélisation de la multiprogrammation

Utilisation du CPU en fonction des processus en mémoire



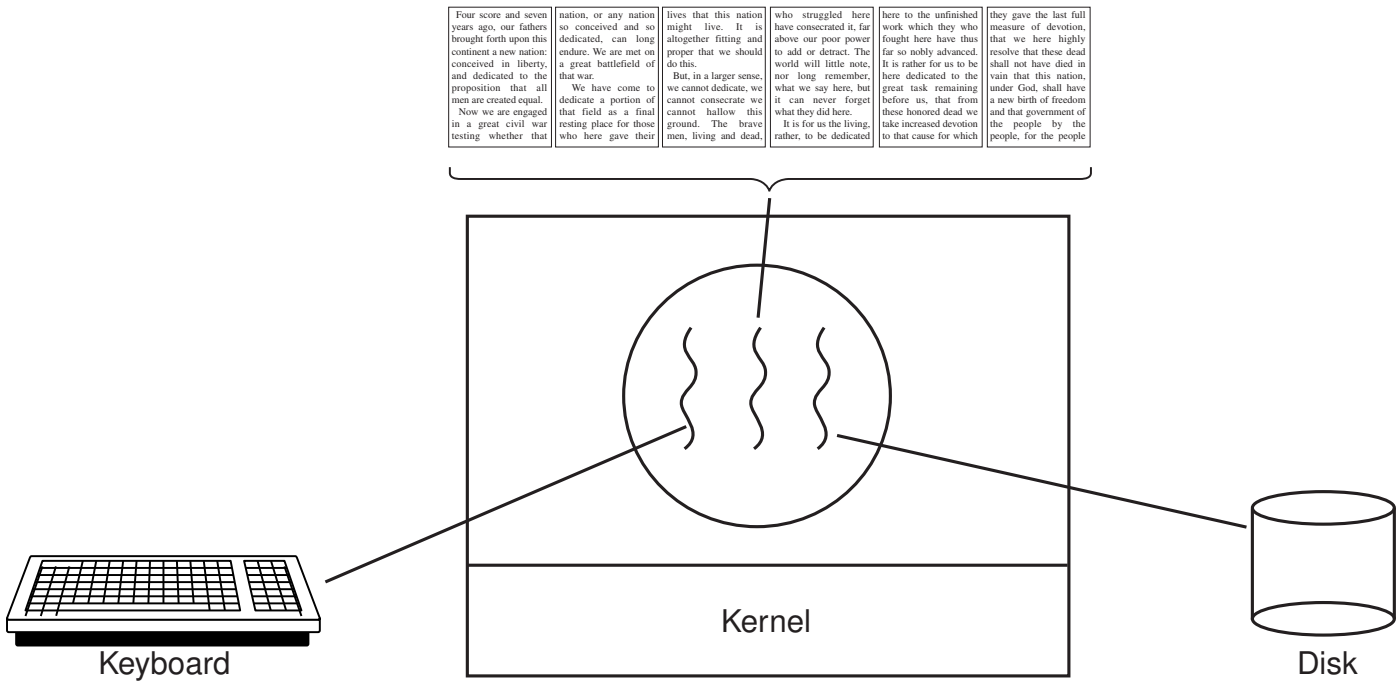
$$\text{Taux d'utilisation du CPU} = 1 - p^n$$

p est la fraction du temps que le CPU attend sur des I/O et *n* est le degré de multiprogrammation[1].

[1] `http://bit.ly/1iZBT28`

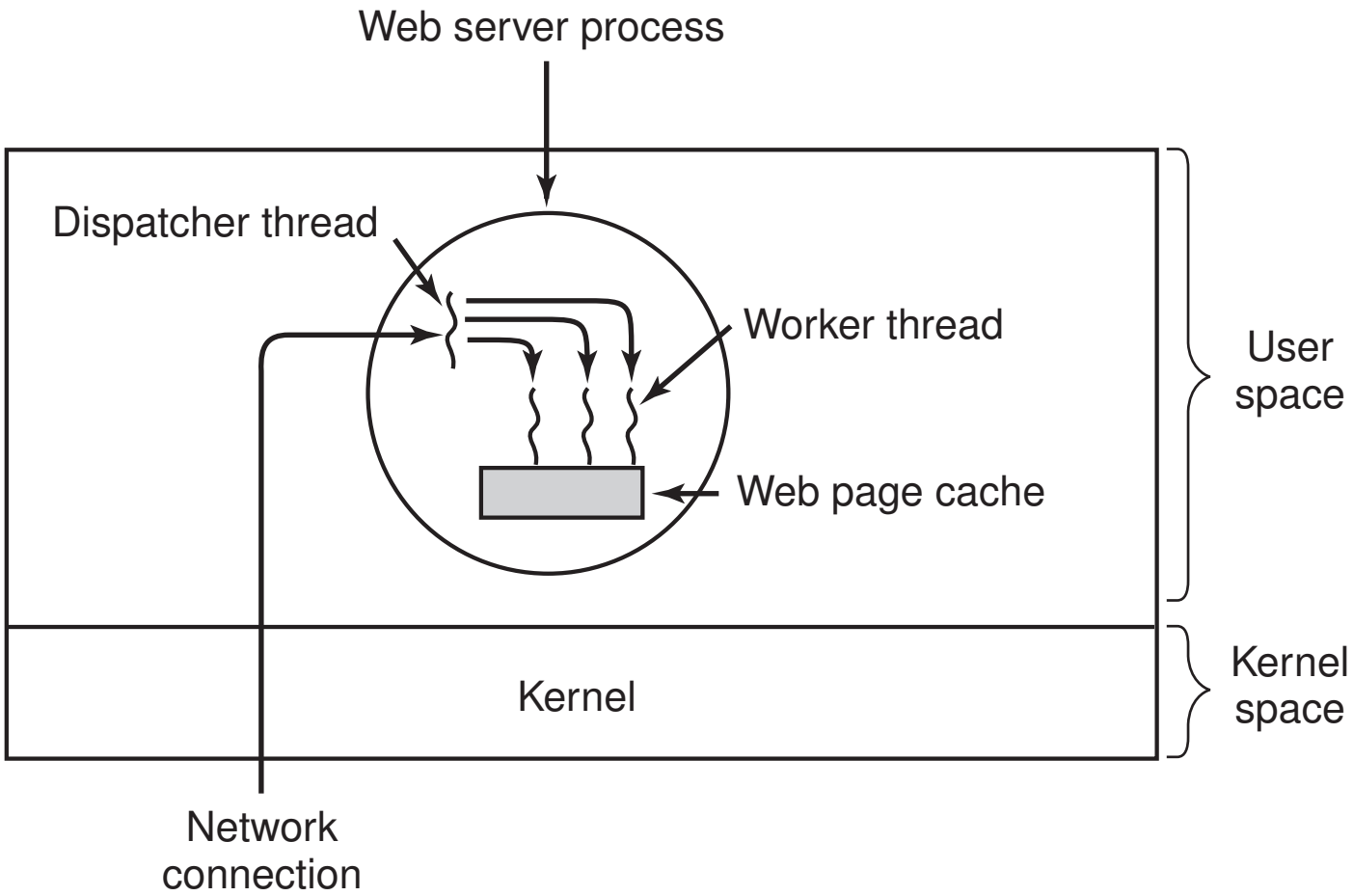
L'utilisation des threads

Un logiciel de traitement de texte avec 3 threads



L'utilisation des threads

Un serveur web multithread



L'utilisation des threads

Aperçu du code pour le serveur web multithread

```
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)

- (a) Thread dispatcher
- (b) Thread worker

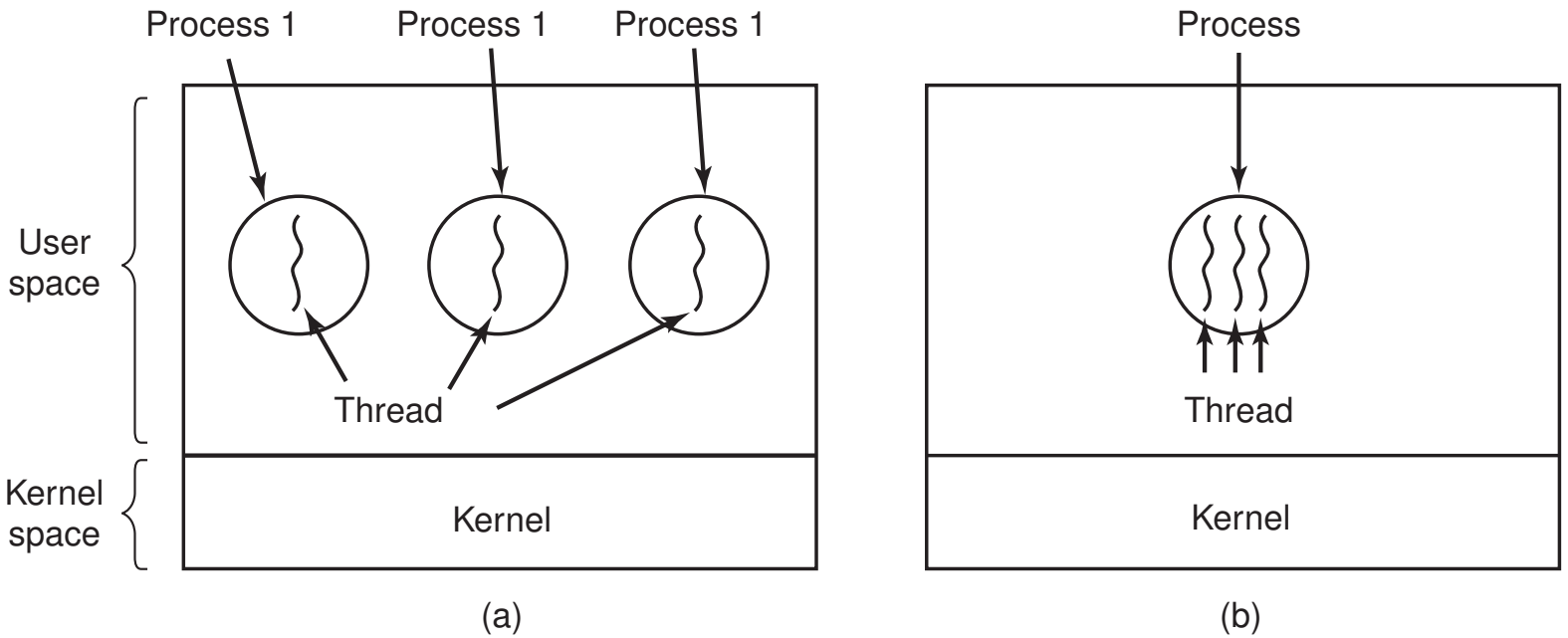
L'utilisation des threads

Les 3 manières de construire un serveur

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

Le modèle de thread classique

Les threads sont parfois qualifiés de «processus légers» (lightweight process)



- (a) 3 processus avec 1 thread chacun
- (b) 1 processus avec 3 threads

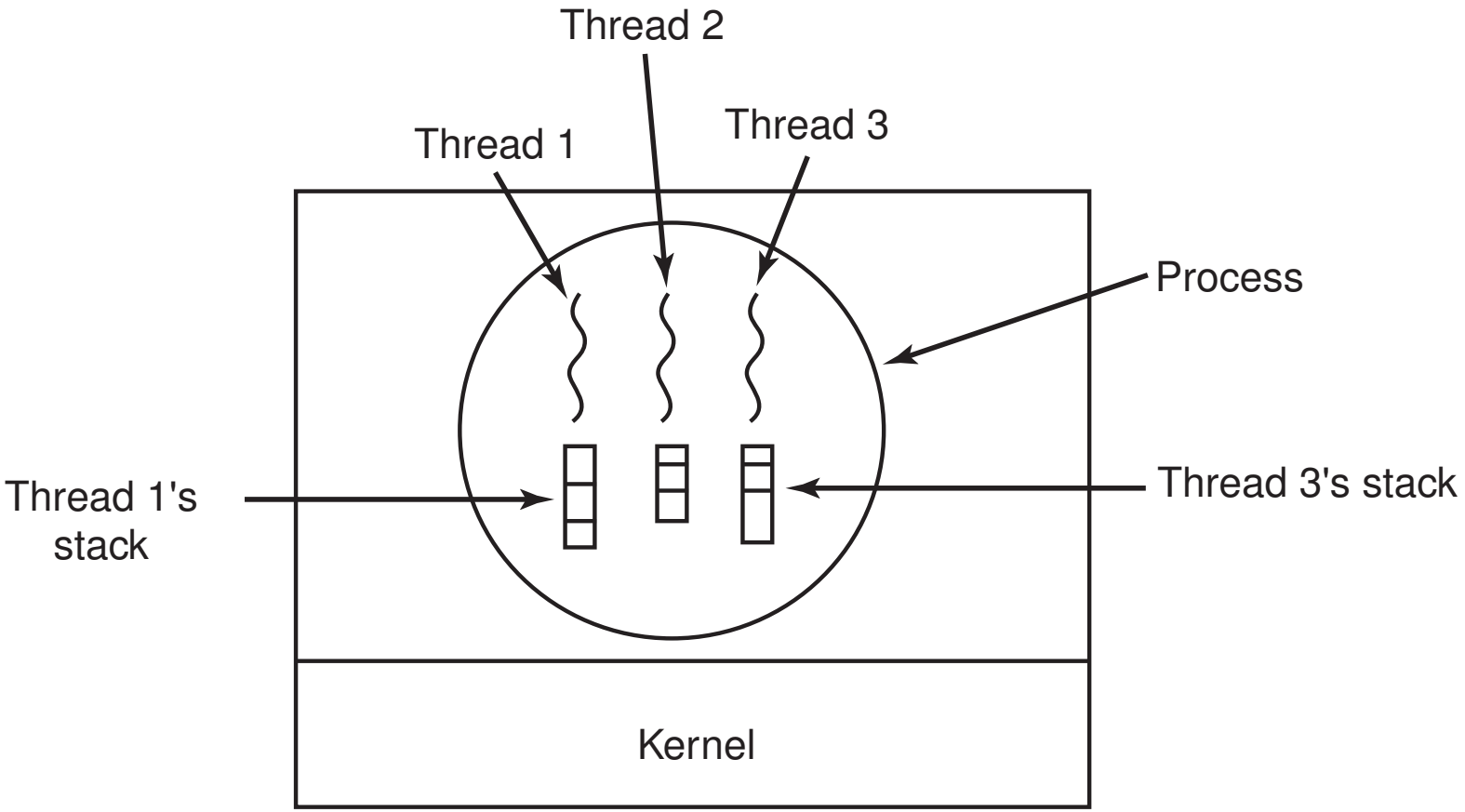
Le modèle de thread classique

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	

La première colonne montre certains éléments partagés par tous les threads d'un processus. La seconde présente certains éléments privés de chaque thread.

Le modèle de thread classique

Chaque thread a sa propre pile (stack)



Les threads de POSIX

Quelques appels de fonctions Pthreads

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

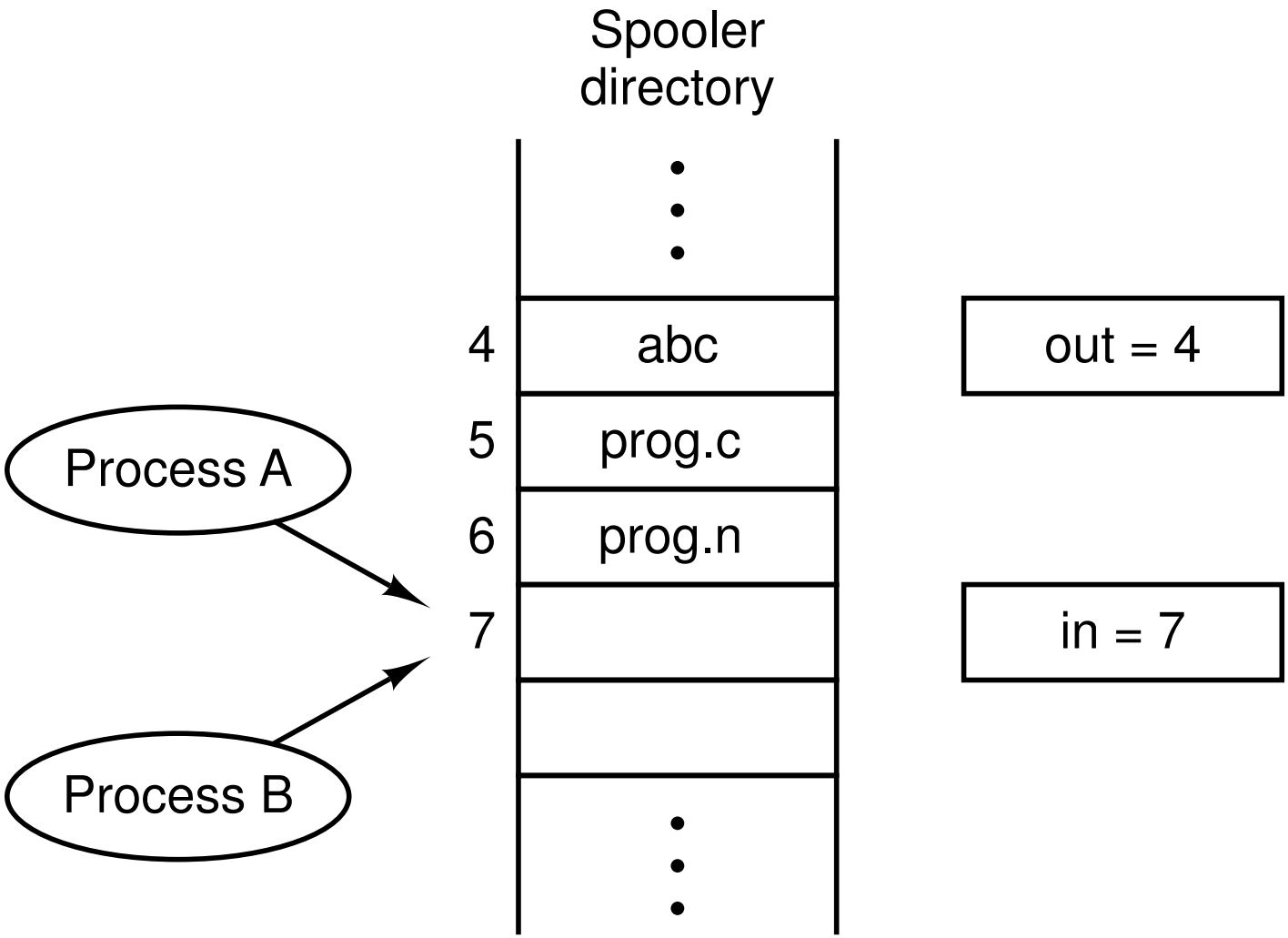
Les threads de POSIX

Exemple de programme utilisant des 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);
}
```

Les conditions de concurrence (race condition)

2 processus veulent accéder à la même adresse mémoire en même temps



Les sections critiques

Conditions pour éviter les conditions de concurrence

- 2 processus ne doivent pas se trouver simultanément dans leurs sections critiques
- Il ne faut pas faire de supposition quant à la vitesse ou au nombre des processeurs mis en œuvre
- Aucun processus s'exécutant à l'extérieur de la section critique ne doit bloquer d'autres processus
- Aucun processus de doit attendre indéfiniment pour pouvoir entrer dans sa section critique

Les sections critiques

Exclusion mutuelle à l'aide de section critiques