Operating Systems (OS)
IPC and Threads

Prof. Eun-Seok Ryu (esryu@skku.edu)
Multimedia Computing Systems Laboratory
http://mcsl.skku.edu
Department of Immersive Media Engineering
Department of Computer Education (J.A.)
Sungkyunkwan University (SKKU)

학생들의 질문 답변시간

- Term project의 요구사항 Details

- Multi-core와 Threads의 관계 및 비디오처리에서의 Issue정리
  - Thread: Module화 장점 / Multicore에서의 장점
  - Signaling을 이용한 Sync

- Q: 예로 웹서버에 단일 프로세스가 대기하여 Client 요청에 응대 처리할 경우의 문제점?
  - 긴 시간 대기
  - > 별도의 프로세스 생성 > 별도의 threads 생성

MCSLab Multimedia Computing Systems Lab.

SUNG KYUN KWAN UNIVERSITY(SKKU)

Inter-Process Communication(IPC)

Interprocess Communication

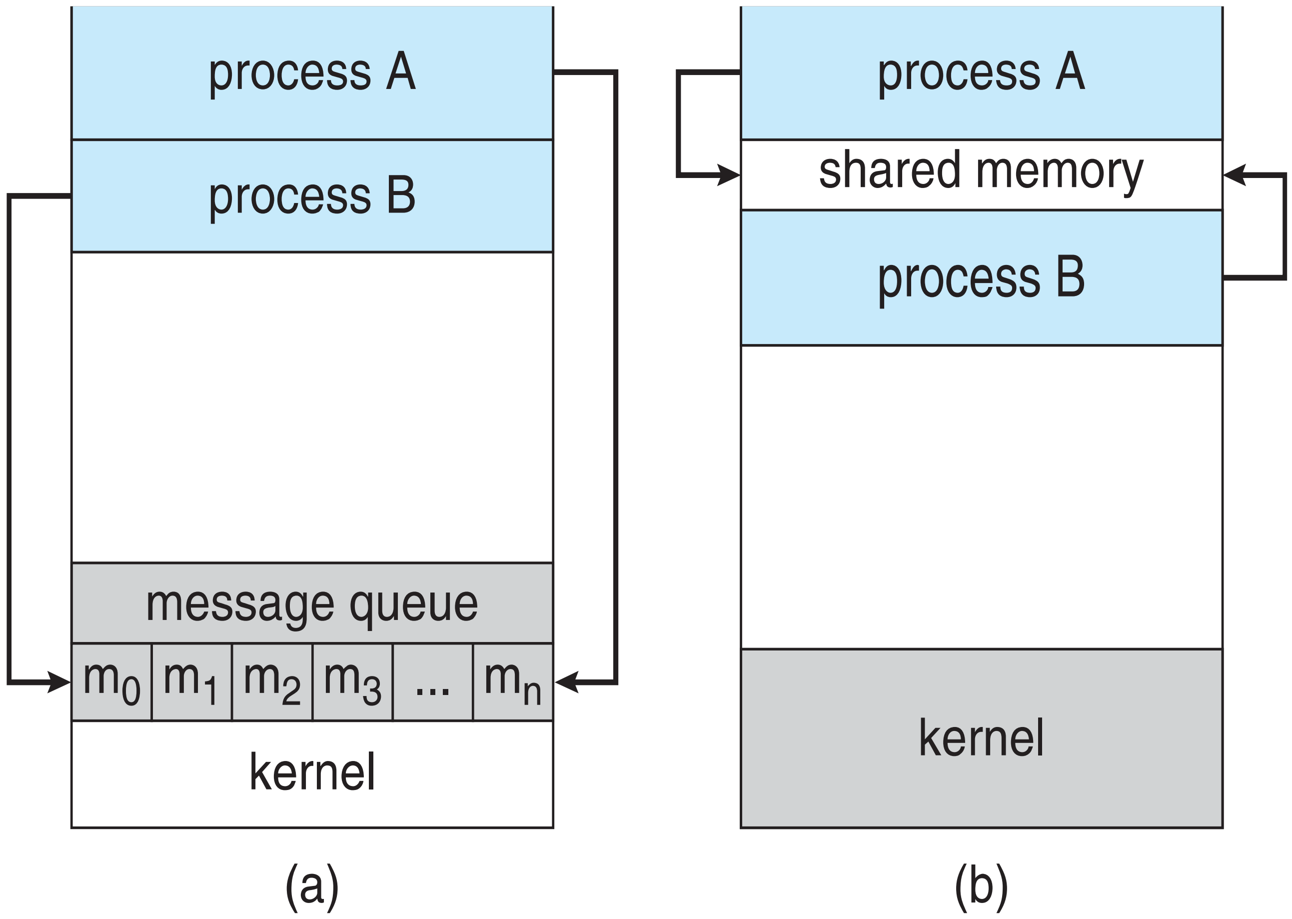
- Processes within a system may be ***independent*** or ***cooperating***
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need **interprocess communication** (**IPC**)
- Two models of IPC
  - **Shared memory**
  - **Message passing**

MCSLab Multimedia Computing Systems Lab.

SUNG KYUN KWAN UNIVERSITY(SKKU)

Communications Models

(a) Message passing. (b) shared memory.



Message Passing v.s. Shared Memory

- 누가 빠른가? 공유메모리 Win
  - Kernel이 중계하는 메시지 전달 방식이 늦음
  - 메시지 전달 방식은 프로세스 A의 User level 메모리의 정보를 Kernel level의 Memory에 넣었다가 다시 프로세스 B의 user level의 메모리에 넣는…중계 방식
  - 공유 메모리 생성은 User level에서의 System call 한번

- 공유메모리 영역은 공유메모리 Segment를 생성하는 Process의 주소공간에 위치 (user-level)
  - Q: 어떻게 다른 Process가 접근 가능?
  - A: 이용하려는 다른 Process는 이 Segment를 자신의 주소 공간에 추가
  - Q: 그렇다면 공유의 위험성/문제점 존재는?
  - A: OS는 한 Process가 다른 Process의 메모리 접근 금지를 풀어주어야 함 + 동시에 같은 위치 안쓰기 (Ring buffer 등으로 in/out 관리)

MCSLab Multimedia Computing Systems Lab.

SUNG KYUN KWAN UNIVERSITY(SKKU)

Message Passing System

- 동일한 주소 공간 이용 X
  - 프로세스들이 통신
  - 두 Process 사이에만. (not 1:many) – next slide에 그림 설명

  - Q: 만일, 1:many를 구현한다면?
    - Mailbox A를 Proc. P1, P2, P3가 공유
    - P1 -> Mailbox A -> P2 Recv
    - |--> P3 Recv 방식
  - Q: 이 경우 문제?
    - 누가 수신하겠는가의 Issue.
    - 최대 1개의 Proc이 수신(Recv) 연산 실행하도록 허용
    - 어느 Process인가는 Algorithm의 문제 (즉, 구현 문제)

MCSLab Multimedia Computing Systems Lab.

SUNG KYUN KWAN UNIVERSITY(SKKU)

Inter-Process Communication (IPC)

- Direct/Indirect Communications
  - Direct communication:
    - Processes must name each other explicitly:
    - **send**(*P*, *message*) – send a message to process P
    - **receive**(*Q*, *message*) – receive a message from Q

```
Process A                      Process B
while (TRUE) {                 while (TRUE) {
    produce an item                receive ( A, item )
    send ( B, item )               consume item
}                              }
```

  - Indirect communication (more convenient):
    - messages are sent to a shared mailbox which consists of a queue of messages.
    - senders place messages in the mailbox, receivers pick them up.
    - **send**(*A*, *message*) – send a message to mailbox A.
    - **receive**(*A*, *message*) – receive a message from mailbox A.

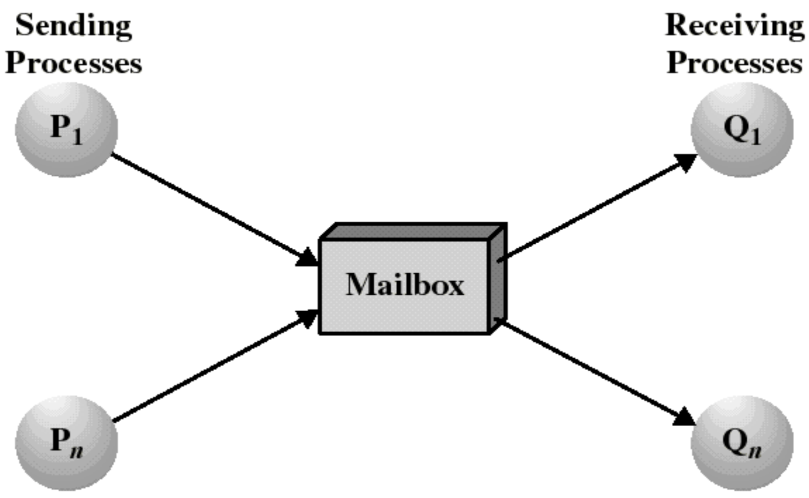
Example: Message Queue

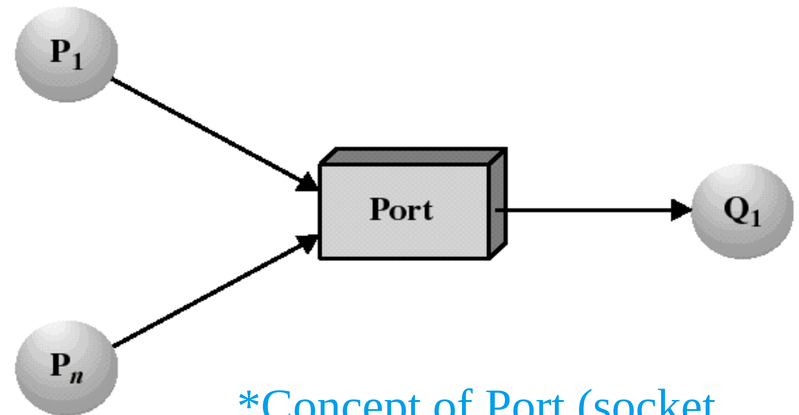
```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
int msgget (key_t key, int msgflg) //MSG creat
int msgsnd (int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg) // MSG sending
ssize_t msgrcv (int msqid, struct  msgbuf  *msgp,  size_t msgsz, long msgtyp, int msgflg)  // MSG receaving
```

- Example:
  - key_id = msgget((key_t)1234, IPC_CREAT|0666);
  - msgsnd(key_id, (void *)&mybuf, sizeof(struct msgbuf), IPC_NOWAIT)

Example: Mailboxes and Ports

- A mailbox can be private to one sender/receiver pair.
- The same mailbox can be shared among several senders and receivers:
  - the OS may then allow the use of message types (for selection).
- Port: is a mailbox associated with one receiver and multiple senders
  - used for client/server applications: the receiver is the server.





*Concept of Port (socket programming)

MCSLab Multimedia Computing Systems Lab.

SUNG KYUN KWAN UNIVERSITY(SKKU)

Issues

- Is the message safe?
  - Lost messages
  - long transmission delays can cause problems
    - something delayed a message beyond its time limit but still in transit

- Message queuing order
  - FCFS / Priority system / User selection

Signaling

- examples of Linux signal types:
  - SIGINT : interrupt from keyboard
  - SIGFPE : floating point exception
  - SIGKILL : terminate receiving process
  - SIGCHLD: child process stopped or terminated
  - SIGSEGV : segment access violation

  - to set up a signal handler:

```
#include <signal.h>
#include <unistd.h>
    void (*signal(int signum, void (*handler)(int)))(int);
```

  - signal is a call which takes two parameters
    - signum : the signal number
    - handler : a pointer to a function which takes a single integer parameter and returns nothing (void)
  - return value is itself a pointer to a function which:
    - takes a single integer parameter and returns nothing

Threads

Benefits of Threads

- **Responsiveness –** may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing –** threads share resources of process, easier than shared memory or message passing
- **Economy –** cheaper than process creation, thread switching lower ov erhead than context switching
- **Scalability –** process can take advantage of multiprocessor architectu res

MCSLab Multimedia Computing Systems Lab.

SUNG KYUN KWAN UNIVERSITY(SKKU)

Multicore Programming

- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**

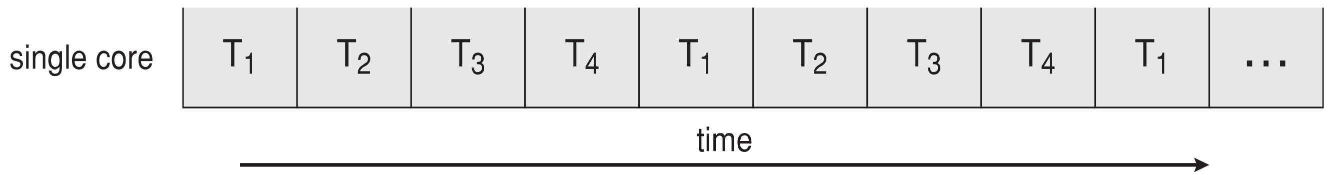
비디오 예제

- ***Parallelism*** implies a system can perform more than one task simultaneously
- ***Concurrency*** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

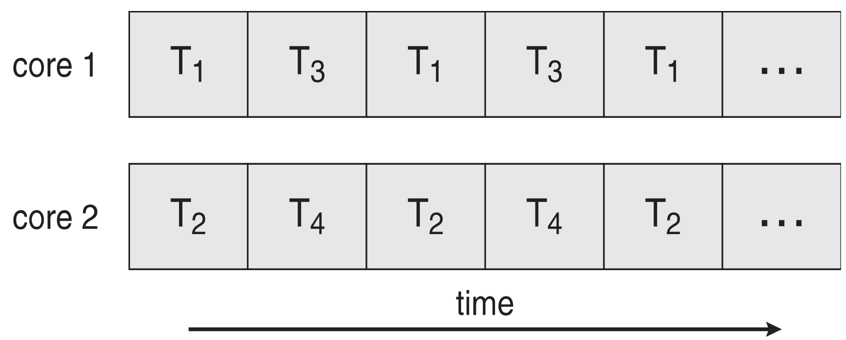
- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation

Concurrency vs. Parallelism

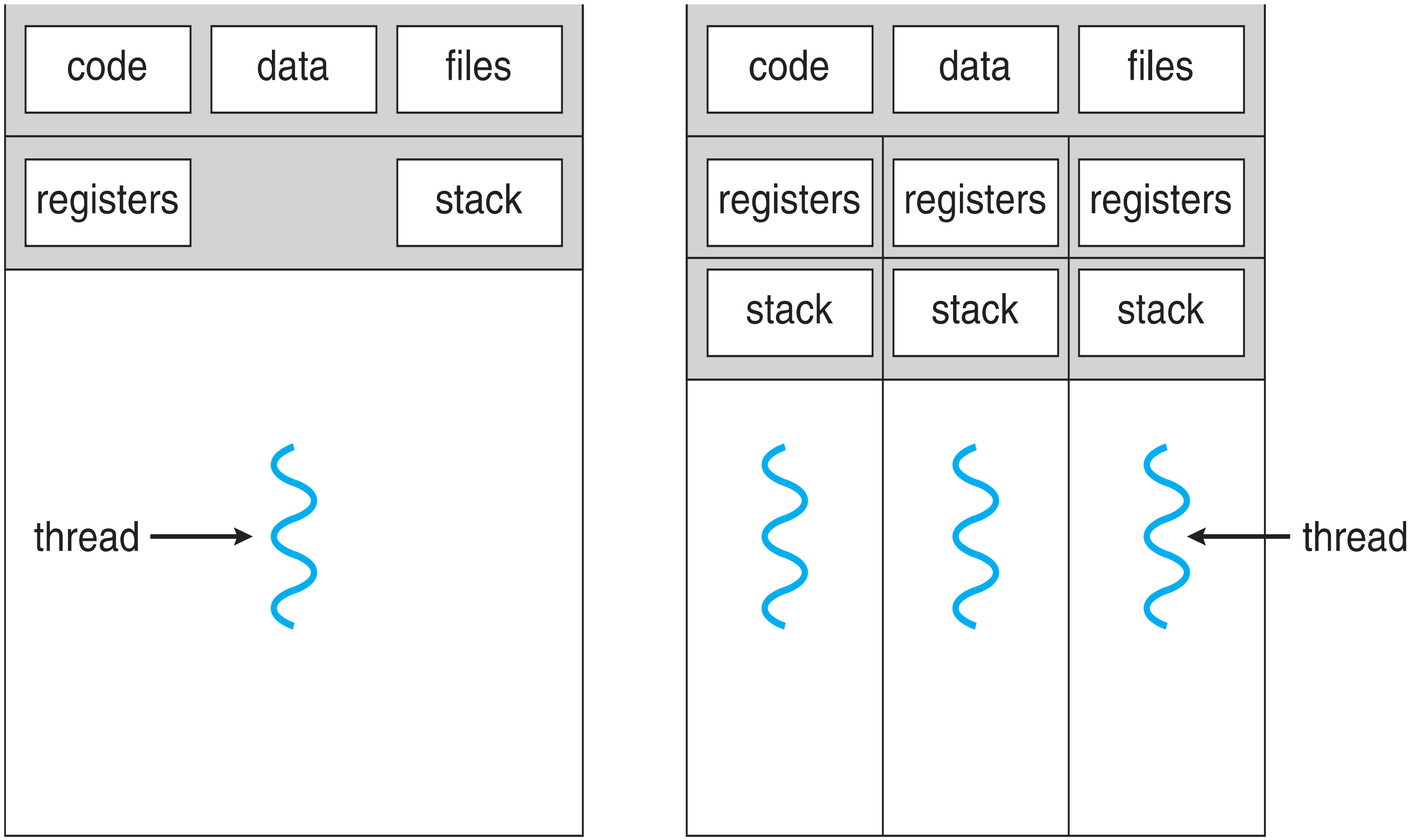
- Concurrent execution on **single-core** system:



- **Parallelism** on a **multi-core** system:



Single and Multithreaded Processes

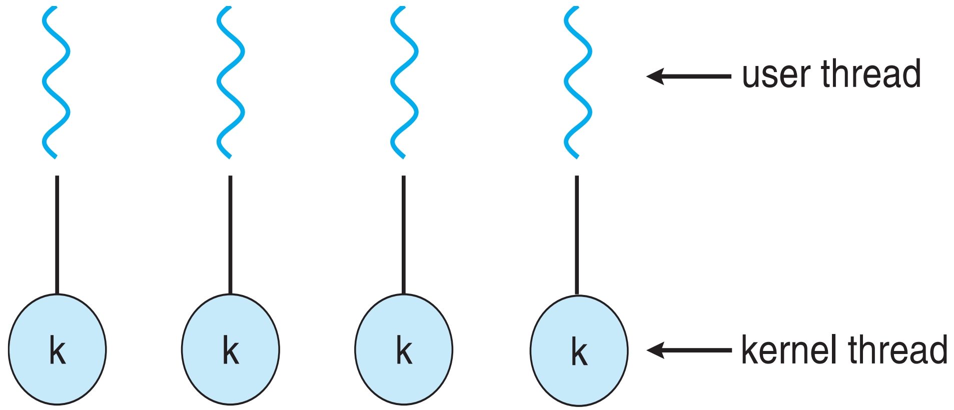


single-threaded process

multithreaded process

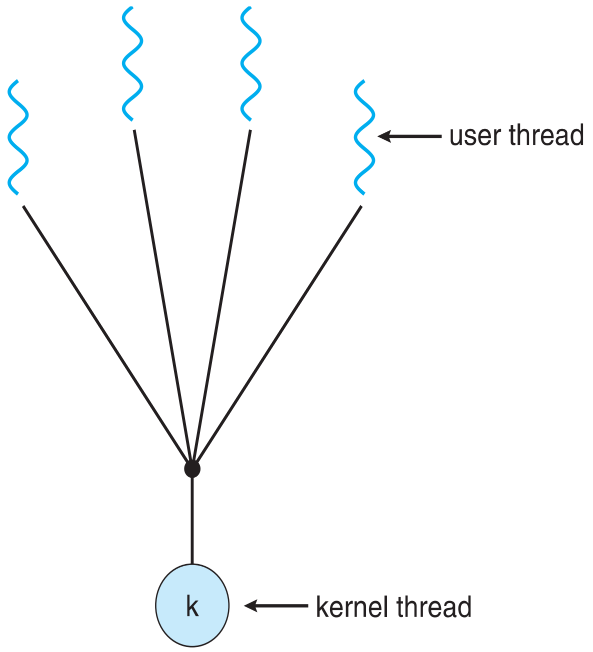
Kernel Threads and User Threads

- One-to-One model (In most cases)
  - Each user-level thread maps to kernel thread
  - Creating a user-level thread creates a kernel thread
  - More concurrency than many-to-one
  - Number of threads per process sometimes restricted due to overhead
  - Examples
    - Windows
    - Linux
    - Solaris 9 and later



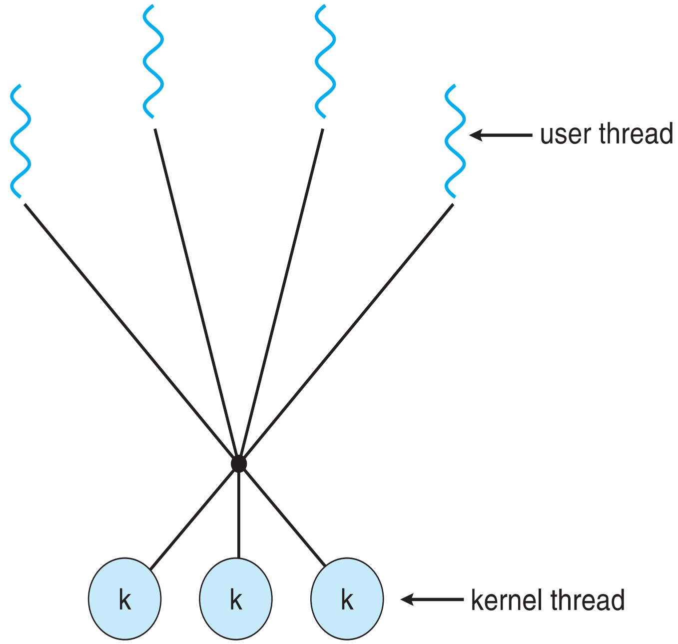
Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



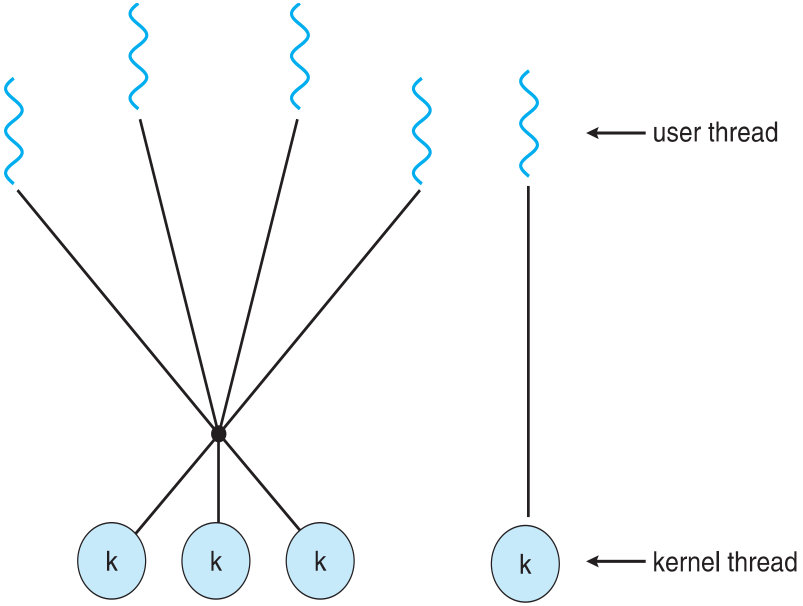
Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *Thread Fiber* package



Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

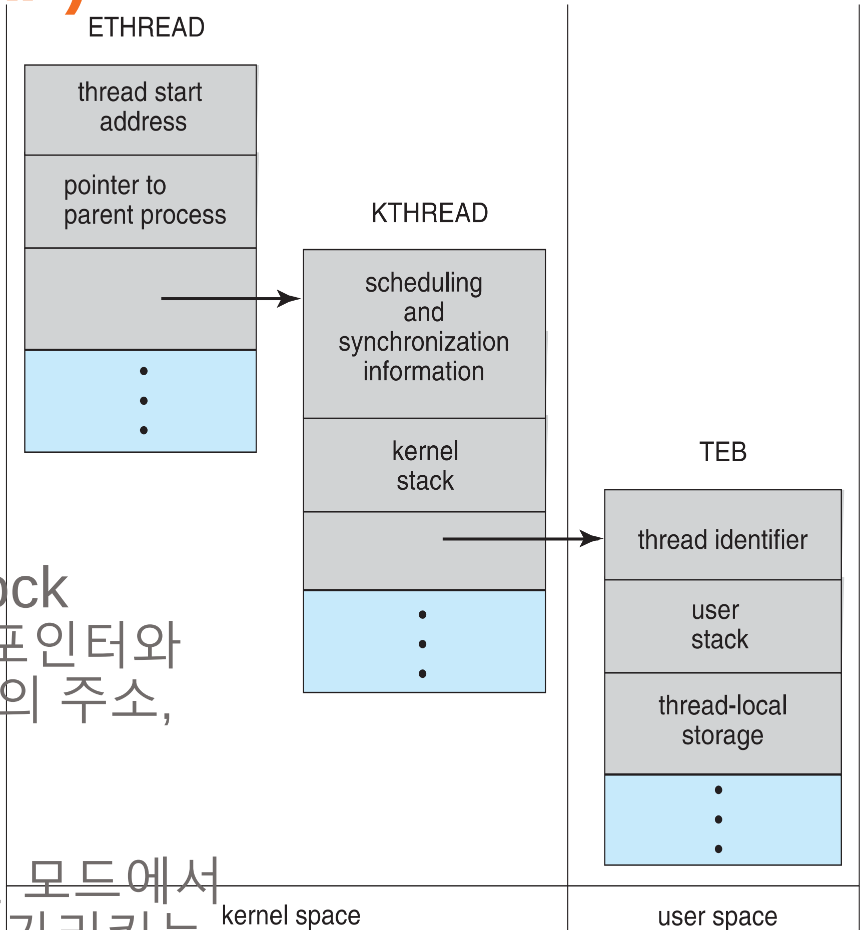


Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

Threads 구현 사례 (Windows XP)

- Windows XP
  - Thread의 일반적 구성요소
    - Thread ID
    - Register집합
    - 실행 위치에 따른 사용자 Stack, Kernel Stack
    - 개별 데이터 저장 영역



- ETHREAD: Executive Thread Block
  - 쓰레드가 속한 프로세스를 가리키는 포인터와 그 쓰레드가 실행을 시작해야 할 루틴의 주소, KTHREAD에 대한 포인터.
- KTHREAD: Kernerl Tread Block
  - 쓰레드 스케쥴링 및 동기화 정보, 커널 모드에서 실행될 때 사용되는 커널 스택과 TEB 가리키는 포인터.
- TEB: Thread Environment Block
  - 유저 모드에서 실행될 때 접근되는 유저 공간 자료 구조. 쓰레드 ID, 데이터 저장 공간 등

Windows Multithreaded C Program

```c
#include <windows.h>
#include <stdio.h>
DWORD Sum; /* data is shared by the thread(s) */

/* the thread runs in this separate function */
DWORD WINAPI Summation(LPVOID Param)
{
   DWORD Upper = *(DWORD*)Param;
   for (DWORD i = 0; i <= Upper; i++)
      Sum += i;
   return 0;
}

int main(int argc, char *argv[])
{
   DWORD ThreadId;
   HANDLE ThreadHandle;
   int Param;

   if (argc != 2) {
      fprintf(stderr,"An integer parameter is required\n");
      return -1;
   }
   Param = atoi(argv[1]);
   if (Param < 0) {
      fprintf(stderr,"An integer >= 0 is required\n");
      return -1;
   }

  /* create the thread */
  ThreadHandle = CreateThread(
    NULL, /* default security attributes */
    0, /* default stack size */
    Summation, /* thread function */
    &Param, /* parameter to thread function */
    0, /* default creation flags */
    &ThreadId); /* returns the thread identifier */

  if (ThreadHandle != NULL) {
     /* now wait for the thread to finish */
    WaitForSingleObject(ThreadHandle,INFINITE);

    /* close the thread handle */
    CloseHandle(ThreadHandle);

    printf("sum = %d\n",Sum);
  }
}
```

타 교재에서의 전통적인 UNIX Kernel 설명

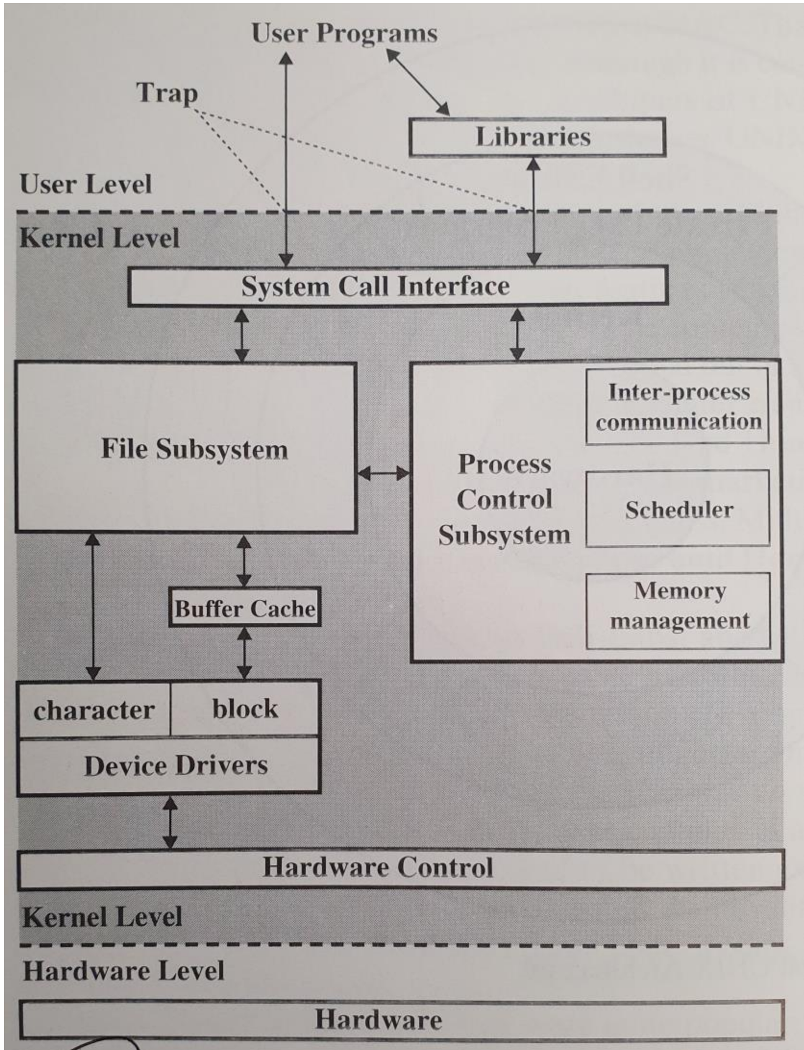


Figure 2.16 Traditional UNIX Kernel [BACH86]

타 교재에서의 Process / Thread Object 설명

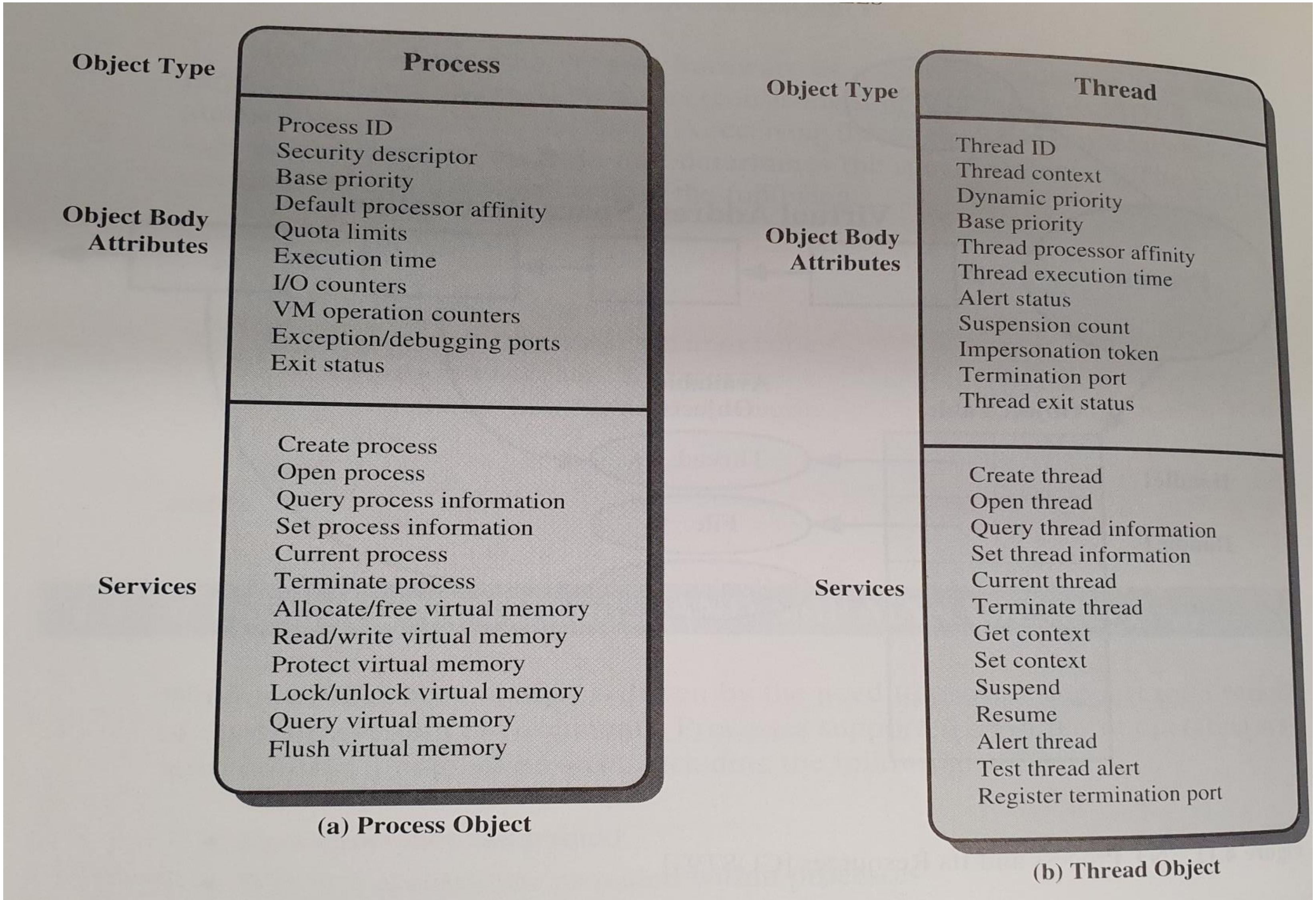
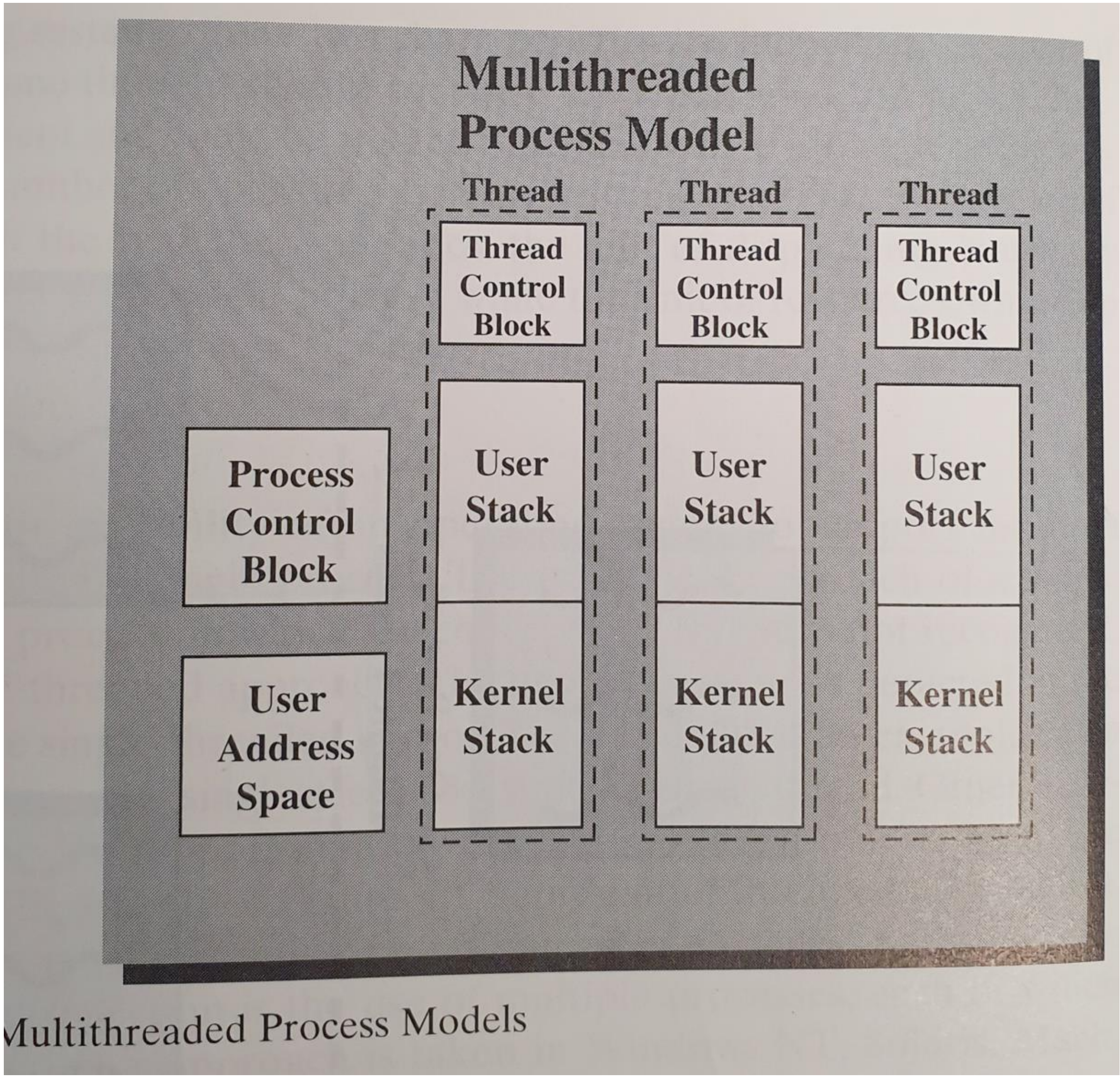


Figure 4.12 Windows NT Process and Thread Objects

타 교재에서의 Multithreaded Process 모델 설명



1개의 프로세스 안에 Multithreads가 존재할 때의 모델

타 교재에서의 프로세스 및 쓰레드 관계 설명

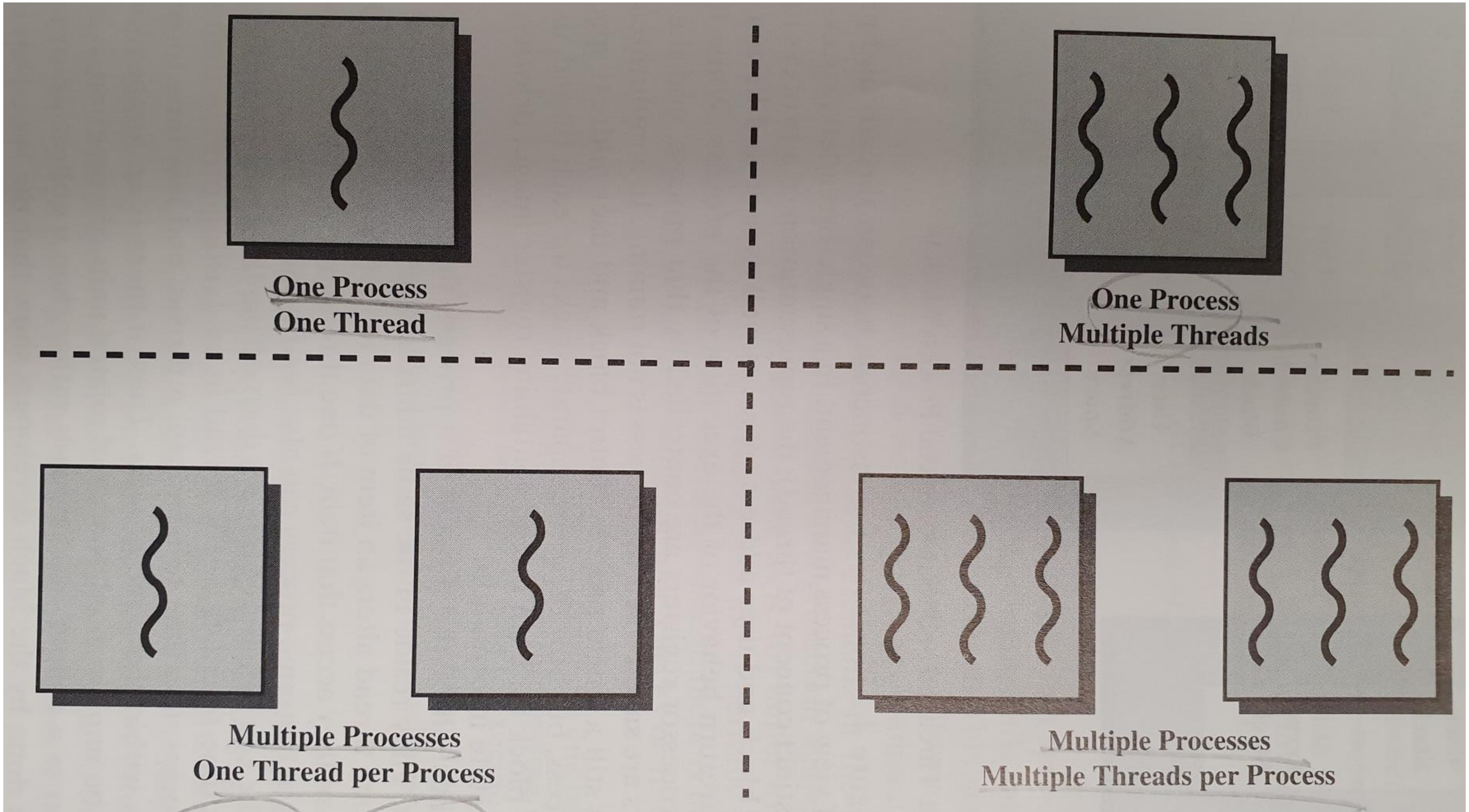


Figure 4.1 Threads and Processes [ANDE97]

타 교재에서의 Two-state Process 모델 설명

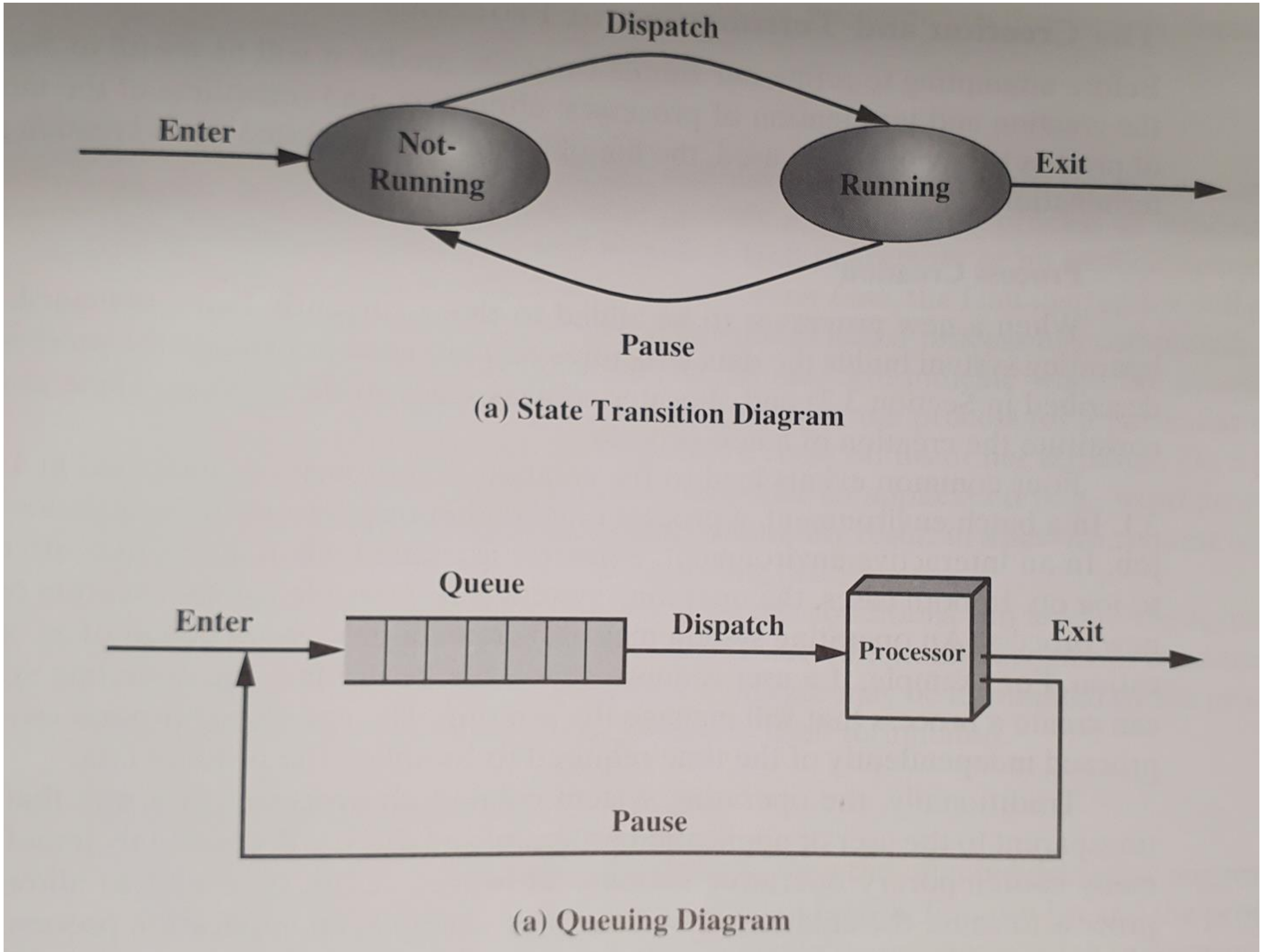


Figure 3.4 Two-state Process Model

타 교재에서의 단일 프로세서 에서의 Multithreading 예

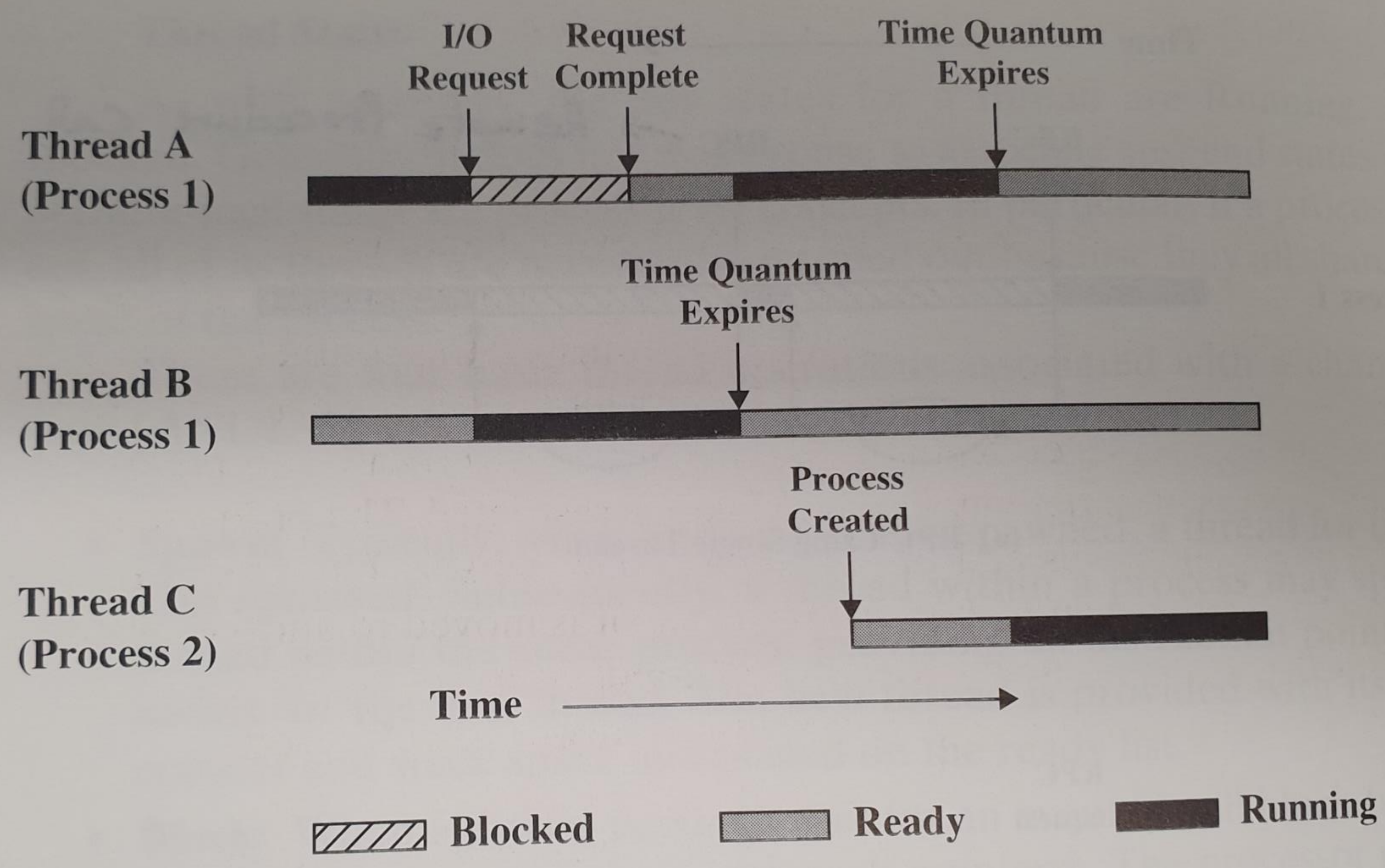
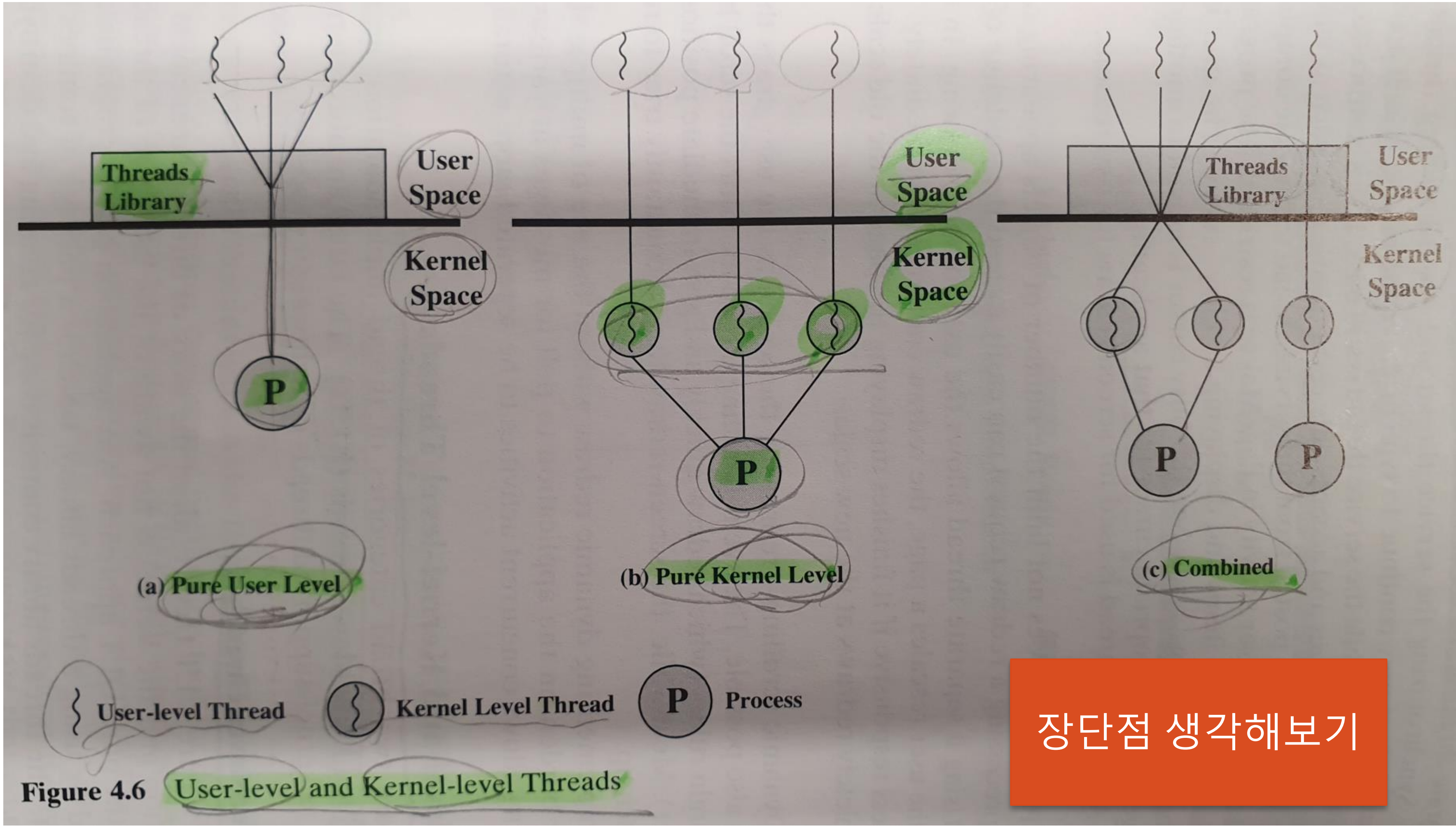


Figure 4.4 Multithreading Example on a Uniprocessor

타 교재에서의 User-level / Kernel-level Threads 설명



장단점 생각해보기