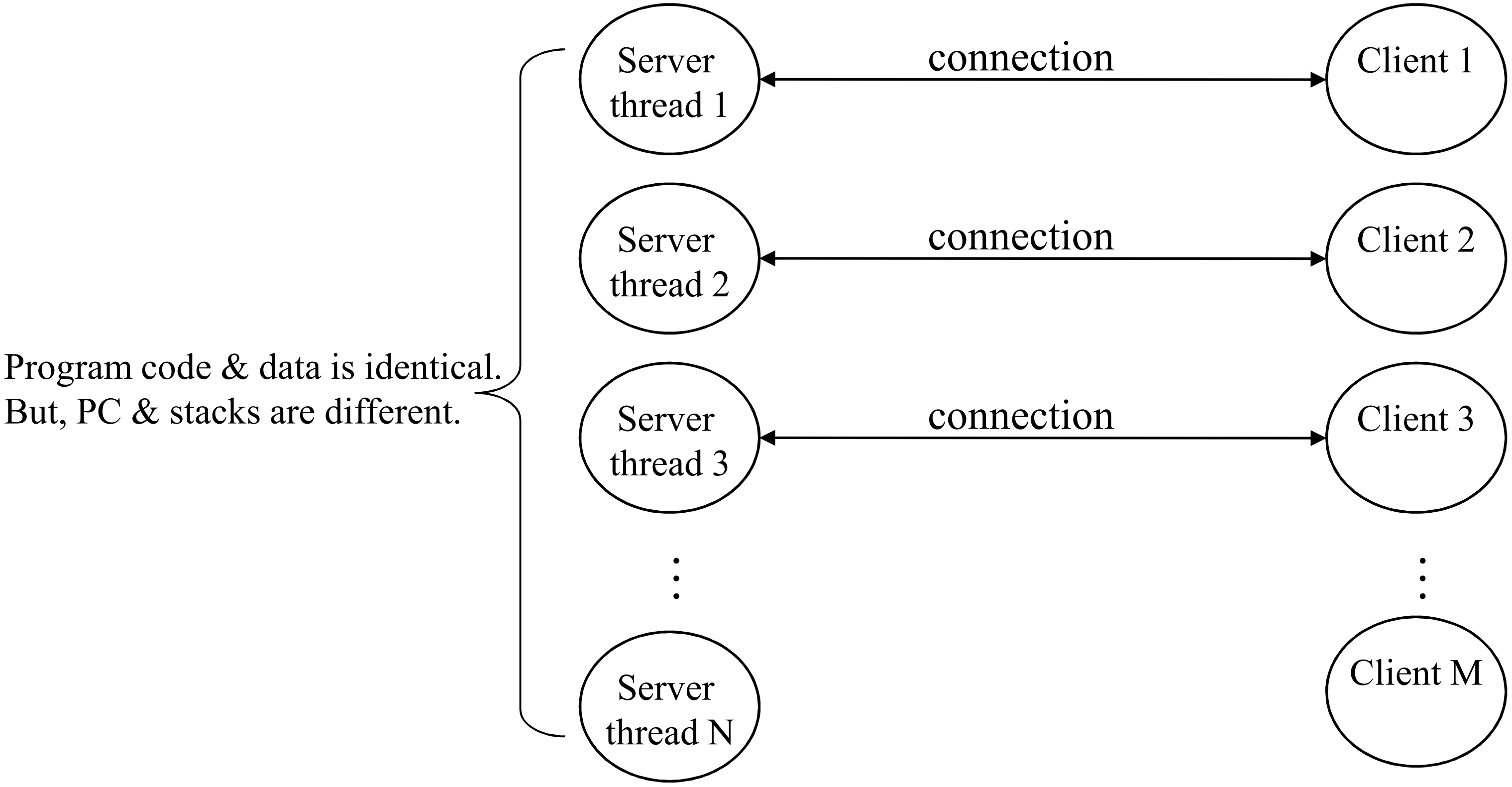
Threads

Threads concepts

- Thread
  - An independent and schedulable execution unit.
  - A process can be divided into two or more running threads.
  - A single thread of control(= a UNIX process)
    - Each process is doing only one thing at a time.
  - Multiple threads of control in a process.
    - The process can do more than one thing at a time.
  - Multithreading is possible on even uni-processor
    - by time-division multiplexing.

Threads concepts

- A typical example
  - Apache web server



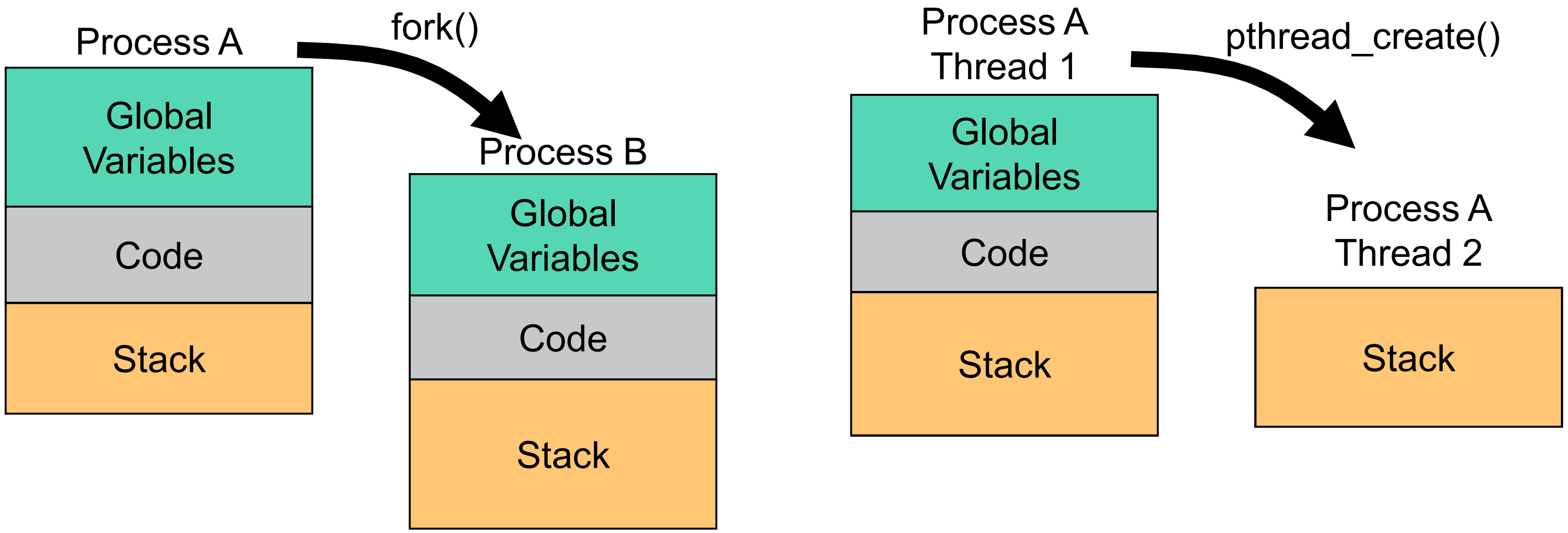
Threads concepts

- Advantages of thread
  - Easy to share information.
    - the memory address space and file descriptors.
  - Throughput can be improved.
    - The processing of independent tasks can be interleaved.
  - More interactive.
    - The separated threads can deal with user input/output.

Threads concepts

Advantages of thread(cont.)

- The cost for creating a new process is low.



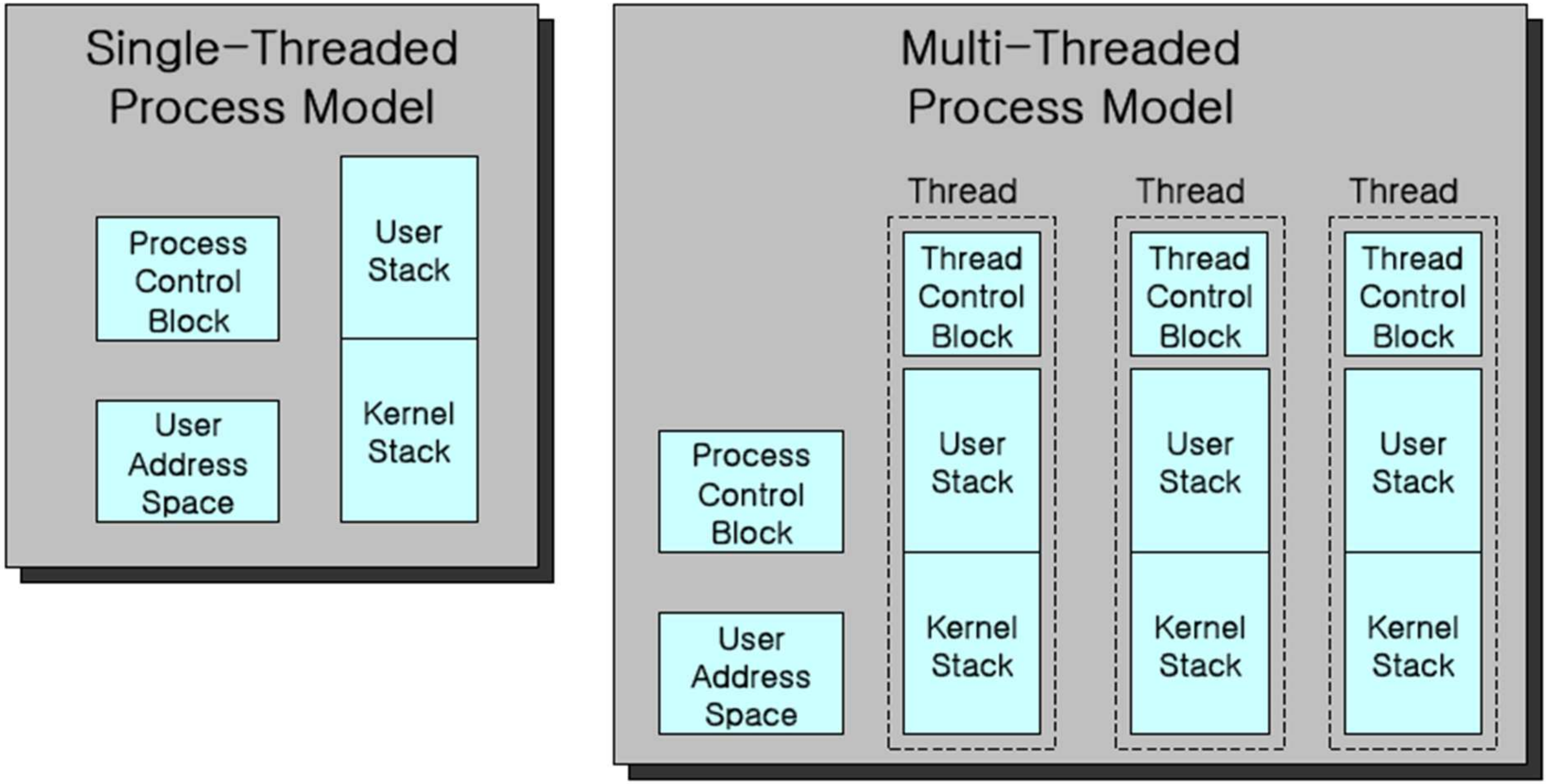
Threads concepts

- A thread-specific information
  - Thread ID
  - Register values
  - Stack
  - Scheduling priority
  - A signal mask

- Sharable information among threads in a process
  - Text section
  - Global data
  - Heap
  - File descriptor

Threads concepts

- Process vs. thread



Posix thread

- What is pthread?
  - IEEE POSIX 1003.1c standards

- Pthread naming convention
  - pthread_

- Compiling pthread program
  - $ **gcc** **-lpthread** **xxx.c**

Thread identification

- Thread ID
  - Identifier of thread (similar to process ID.)
  - A thread ID is represented by pthread_t data type.
    - Unsigned long integer in Linux.
    - A pointer to the pthread structure in FreeBSD.

Thread identification

```
#include <pthread.h>

pthread_t pthread_self(void);
                                        Returns: the thread ID of the calling thread
```

- Obtain its own thread ID.

```
#include <pthread.h>

int pthread_equal(pthread_t tid1, pthread_t tid2);
                                        Returns: nonzero if equal, 0 otherwise
```

- Compare two thread IDs

Thread creation

```
#include <pthread.h>

int pthread_create(pthread_t *tidp,
                   const pthread_attr_t *attr,
                   void *(*start_rtn)(void), void *arg);
                                  Returns: 0 if OK, error number on failure
```

- Create a new thread.
  - *tidp* is the ID of the newly created thread.
  - Execute *start_rtn* with *arg* as its argument.
  - *attr* is set to NULL for the default attributes.

Thread creation

Example

```
#include "apue.h"
#include <pthread.h>

pthread_t ntid;

void printids(const char *s)
{
    pid_t       pid;
    pthread_t  tid;

    pid = getpid();
    tid = pthread_self();
    printf("%s pid %u tid %u (0x%x)\n", s, (unsigned int)pid,
      (unsigned int)tid, (unsigned int)tid);
}
```

Thread creation

- Example(cont.)

```
void *thr_fn(void *arg)
{
    printids("new thread: ");
    return((void *)0);
}

int main(void)
{
    int     err;

    err = pthread_create(&ntid, NULL, thr_fn, NULL);
    if (err != 0)
        err_quit("can't create thread: %s\n", strerror(err));
    printids("main thread:");
    sleep(1);
    exit(0);
}
```

Thread creation

- 실행

In Solaris

```
$ ./a.out
main thread: pid 7225 tid 1 (0x1)
new thread:  pid 7225 tid 4 (0x4)
$
```

When a thread is created, there is no guarantee which runs first.

In FreeBSD

```
$ ./a.out
main thread: pid 14954 tid 134529024 (0x804c000)
new thread:  pid 14954 tid 134530048 (0x804c400)
$
```

FreeBSD uses a pointer to the thread data structure for its thread ID.

In Linux

```
$ ./a.out
new thread:  pid 6628 tid 1026 (0x402)
main thread: pid 6626 tid 1024 (0x400)
$
```

Linux does not have a separate system call for thread creation. It uses clone().

Thread termination

- If any thread within a process call exit()?
  - The entire process terminates.

- A single thread can exit without terminating the entire process.
  - The thread can simply return from the start routine.
  - The thread can be canceled by another thread in the same process.
  - The thread can call pthread_exit().

Thread termination

```
#include <pthread.h>

void pthread_exit(void *rval_ptr);
```

- Terminates a calling thread.
  - *rval_ptr* is available to other threads in the process calling the pthread_join().

Thread termination

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **rval_ptr);
                                        Returns: 0 if OK, error number on failure
```

- Suspends execution of the calling thread until the target thread terminates.
  - It is similar to wait().
  - *rval_ptr* argument
    - If not NULL, it contains the exit status of the target thread.
    - If we’re not interested in a return value, it is set to NULL.

Thread termination

- Example

```
#include "apue.h"
#include <pthread.h>

void *thr_fn1(void *arg)
{
   printf("thread 1 returning\n");
   return((void *)1);
}

void *thr_fn2(void *arg)
{
   printf("thread 2 exiting\n");
   pthread_exit((void *)2);
}
```

Thread termination

- Example(cont.)

```
int main(void)
{
    int         err;
    pthread_t   tid1, tid2;
    void        *tret;

    err = pthread_create(&tid1, NULL, thr_fn1, NULL);
    if (err != 0)
        err_quit("can't create thread 1: %s\n", strerror(err));

    err = pthread_create(&tid2, NULL, thr_fn2, NULL);
    if (err != 0)
        err_quit("can't create thread 2: %s\n", strerror(err));
```

Thread termination

- Example(cont.)

```
    err = pthread_join(tid1, &tret);
    if (err != 0)
        err_quit("can't join with thread 1: %s\n", strerror(err));
    printf("thread 1 exit code %d\n", (int)tret);

    err = pthread_join(tid2, &tret);
    if (err != 0)
        err_quit("can't join with thread 2: %s\n", strerror(err));
    printf("thread 2 exit code %d\n", (int)tret);

    exit(0);
}
```

Thread termination

- 실행

```
$ ./a.out
thread 1 returning
thread 2 exiting
thread 1 exit code 1
thread 2 exit code 2
$
```

Thread termination

```
#include <pthread.h>

int pthread_cancel(pthread_t tid);
                                        Returns: 0 if OK, error number on failure
```

- Cancel another thread in the same process.
  - Cause the thread with *tid* to behave as if it had called pthread_exit().
  - It doesn’t wait for the thread to terminate; it merely makes the request.

Thread termination

- Comparison of process and thread primitives

Process primitive	Thread primitive	Description
fork	pthread_create	Create a new flow of control
exit	pthread_exit	Exit from an existing flow of control
waitpid	pthread_join	Get exit status from flow of control
getpid	pthread_self	Get ID for flow of control
abort	pthread_cancel	Request abnormal termination of flow of control

Thread synchronization

- A synchronization example
  - Two threads try to modify the same variable simultaneously.
    - i is initialized to 5.
    - Two threads perform “i++;”, respectively.

Thread A

i++;

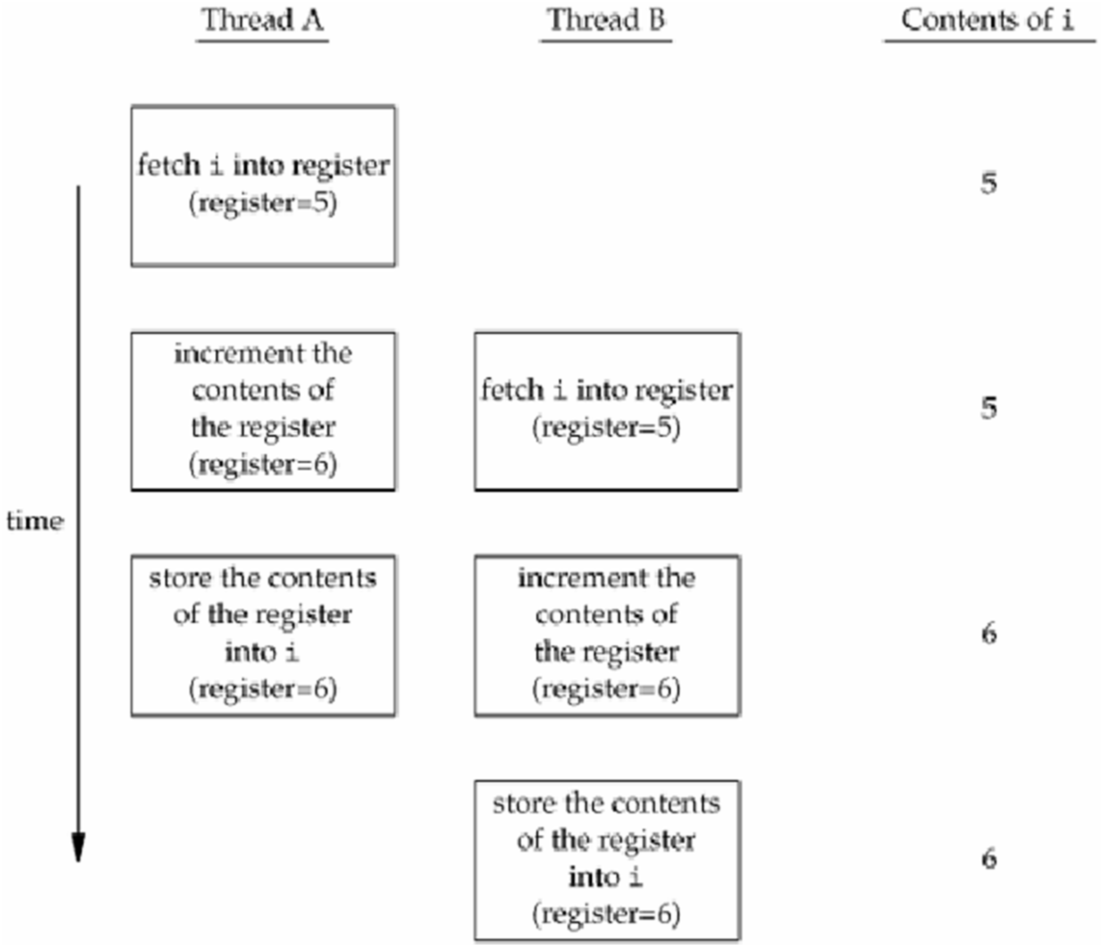
Thread B

i++;

  - Simple statement “i++;” consists of several instructions.
    - Read the memory location into a register.
    - Increment the value in the register.
    - Write the new value back to the memory location.

Thread synchronization

- A synchronization example



Thread A	Thread B	Contents of i
fetch i into register (register=5)		5
increment the contents of the register (register=6)	fetch i into register (register=5)	5
store the contents of the register into i (register=6)	increment the contents of the register (register=6)	6
	store the contents of the register into i (register=6)	6

Mutex

- Mutex
  - It is a lock that we set (lock) before accessing a shared resource and release (unlock) when we're done.
  - While it is set, any other thread that tries to set it will block until it is released.
  - If more than one thread is blocked when the mutex is unlocked, then only one will be able to set the lock.

Mutex

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *restrict mutex,
                       const pthread_mutexattr_t *restrict attr);
                                Return: 0 on success; otherwise, an error number
```

- Initialize a mutex.
  - initialize the mutex referenced by *mutex* with attributes specified by *attr*.
  - attr is generally set to NULL.

- Static initialization
  - pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

Mutex

```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
                                        Return: 0 on success; otherwise, an error number
```

- Destroy a mutex.
  - destroy the mutex object referenced by *mutex*.

Mutex

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
                                        Return: 0 on success; otherwise, an error number
```

- Lock and unlock a mutex.
  - If the mutex is already locked, the calling thread shall block until the mutex becomes available.

Mutex

- Example (mutex_test.c)

```
#include    <pthread.h>
#define     NLOOP 50

int counter;                                // This is incremented by two threads
pthread_mutex_t counter_mutex = PTHREAD_MUTEX_INITIALIZER;
void *doit(void *);

int main(int argc, char **argv)
{
            pthread_t tidA, tidB;
            pthread_create(&tidA, NULL, &doit, NULL);
            pthread_create(&tidB, NULL, &doit, NULL);

            // wait for both threads to terminate
            pthread_join(tidA, NULL);
            pthread_join(tidB, NULL);

            exit(0);
}
```

Mutex

Example

```
void *doit(void *vptr)
{
        int  i, val;

        for (i = 0; i < NLOOP; i++)
        {
                pthread_mutex_lock(&counter_mutex);
                val = counter;
                printf("%d: %d\n", pthread_self(), val + 1);
                counter = val + 1;
                sleep(1);
                pthread_mutex_unlock(&counter_mutex);
                sleep(1);
        }
        return(NULL);
}
```

Mutex

- 실행

```
[tskim@oslab test]$ gcc –o mutex_test –lpthread mutex_test.c
[tskim@oslab test]$ ./mutex_test
1093278016: 1
1103767872: 2
1093278016: 3
1103767872: 4
1093278016: 5
1103767872: 6
1093278016: 7
1103767872: 8
1093278016: 9
1103767872: 10
1093278016: 11
…


1093278016: 99
1103767872: 100
[tskim@oslab test]$
```