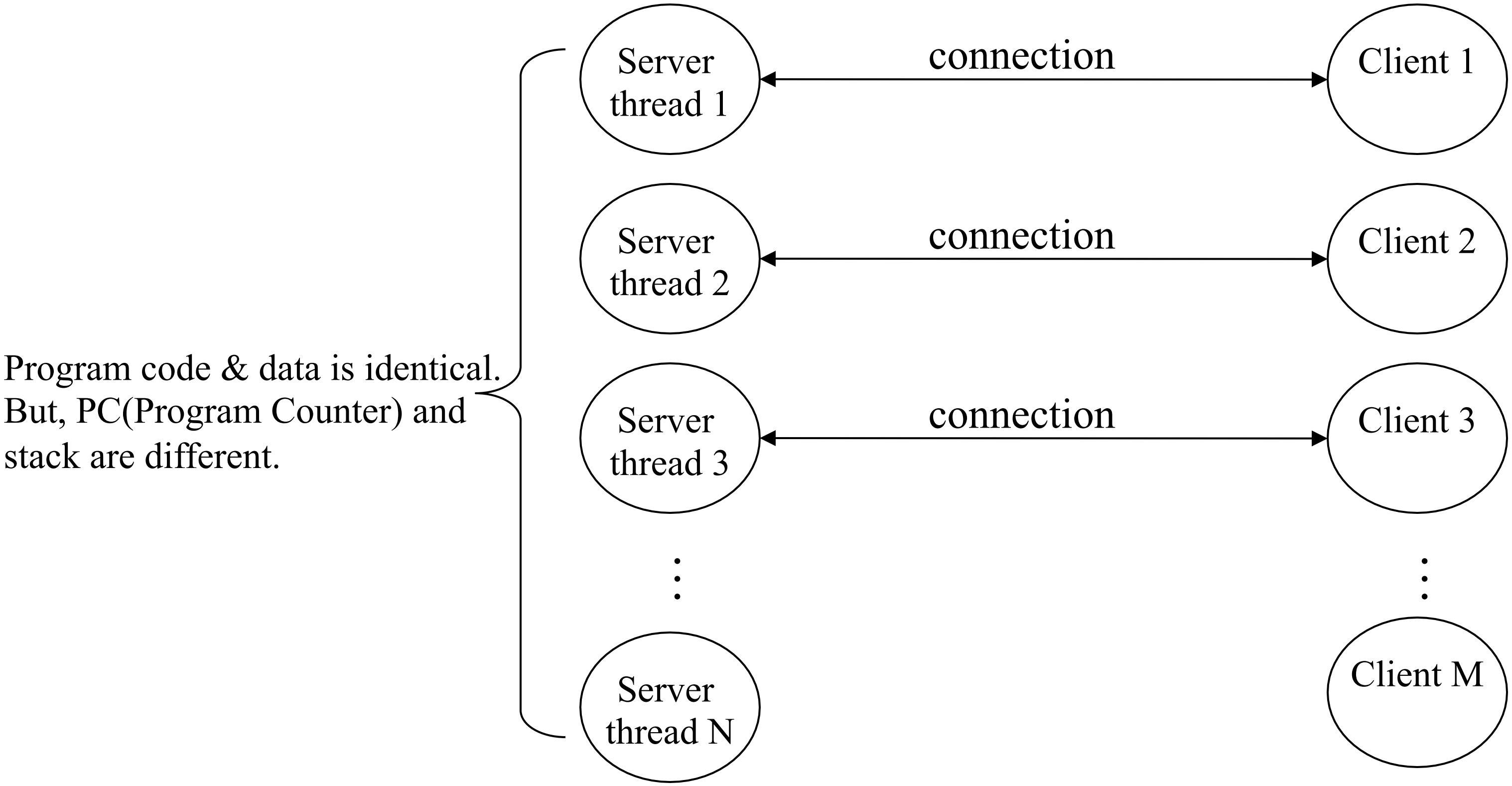
Threads

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

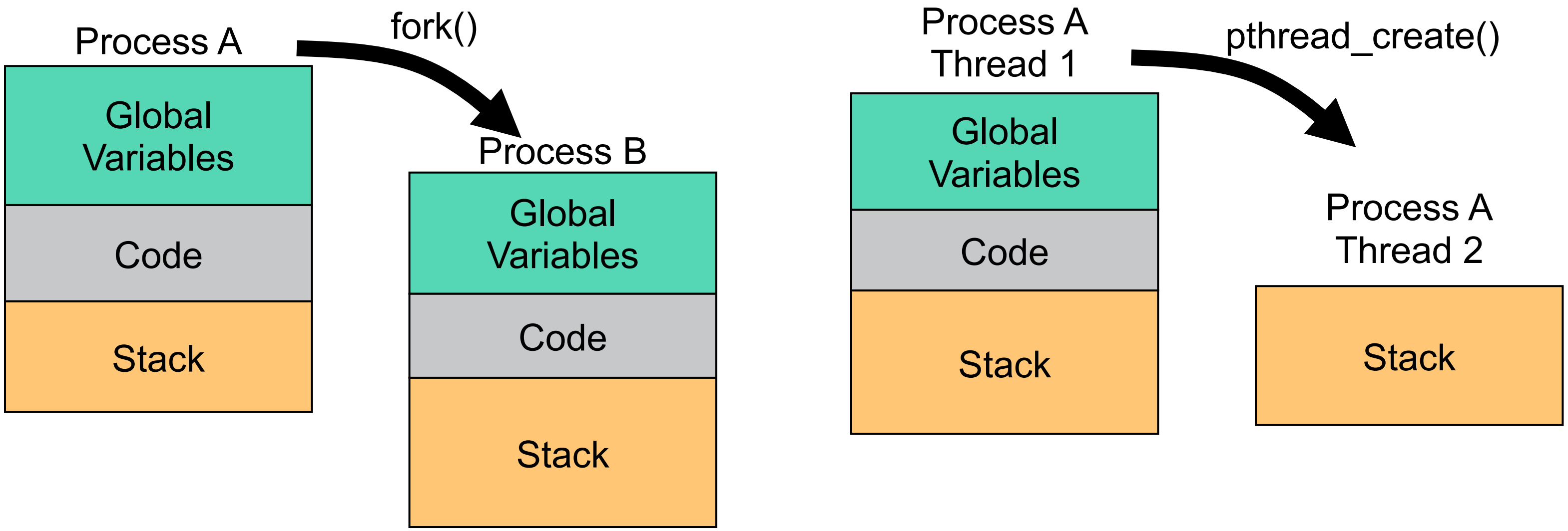


Threads concepts

- 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.

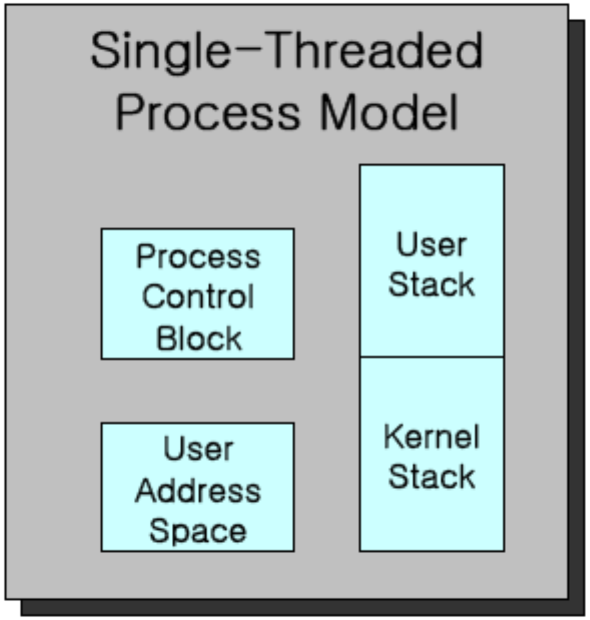


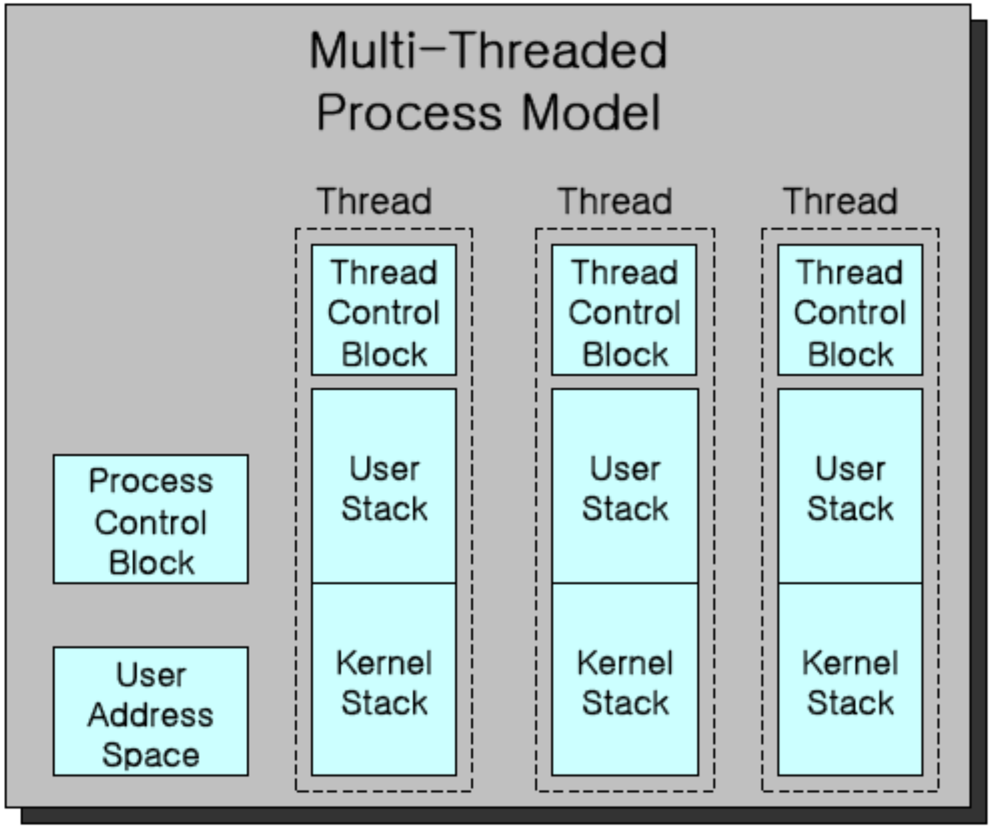
Threads concepts

- 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 -pthread 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);
}
```

What if to use *ntid* instead of *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);   // New thread can run before the old thread terminates.
    exit(0);
}
```

Thread creation

Execution

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
main thread: pid 10043 tid 3480123136 (0xcf6e7700)
new thread:  pid 10043 tid 3471726336 (0xceee5700)
$
```

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 the thread returned from start routine, it contains the return code.
    - If the thread was canceled, it is set to PTHREAD_CANCELED.
    - 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

- execution

```
$ ./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

```
#include <pthread.h>

void pthread_cleanup_push(void (*rtn)(void *), void *arg);
void pthread_cleanup_pop(int execute);
```

- A thread can arrange for functions to be called when it exits. (like atexit())
  - The functions are called as thread cleanup handlers.
  - More than one cleanup handler can be established.
  - The handlers are recorded in a stack, so they are executed in the reverse order of the registrations.

Thread termination

- Pthread_cleanup_push
  - Pushes routine onto the top of the stack of cleanup handlers.
  - *rtn*: cleanup handler function
  - *arg*: a single argument

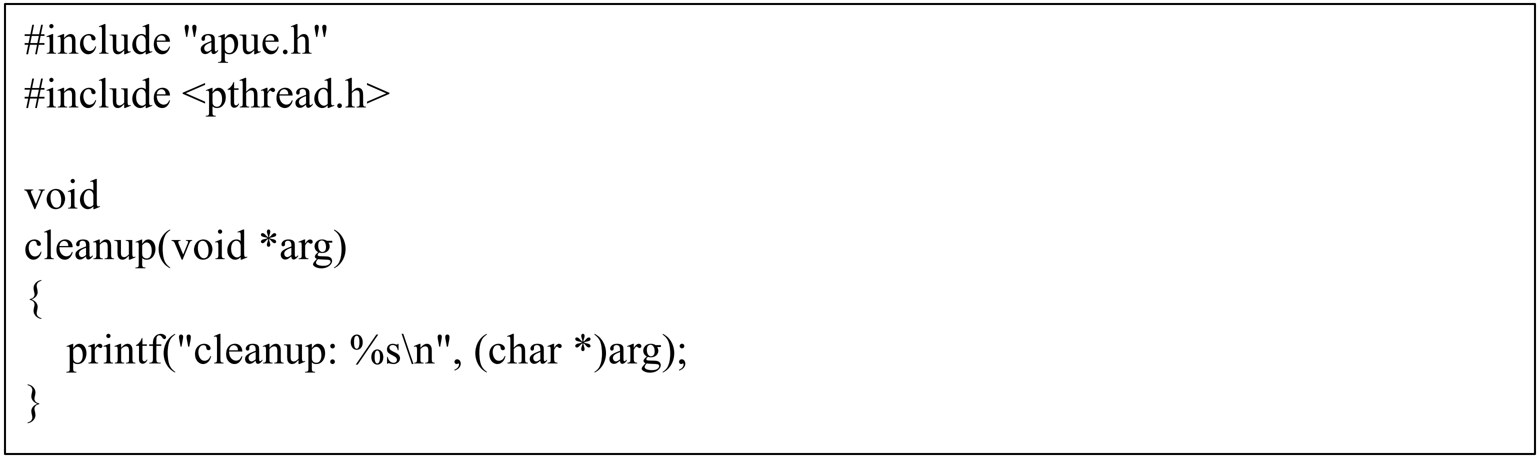
- Pthread_cleanup_pop
  - Pops the top cleanup handler from the current thread's cleanup handler stack.
  - If *execute* is 0, the cleanup handler is not called; it just removes the cleanup handler on top of stack.

Thread termination

- When the thread performs one of the followings, cleanup handlers are executed.
  - Makes a call to pthread_exit().
  - Responds to a cancellation request.
  - Invoke pthread_cleanup_pop() with a nonzero *execute* argument.

Thread termination

Example



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

void
cleanup(void *arg)
{
    printf("cleanup: %s\n", (char *)arg);
}
```

Thread termination

- Example(cont.)

```
void *
thr_fn1(void *arg)
{
   printf("thread 1 start\n");
   pthread_cleanup_push(cleanup, "thread 1 first handler");
   pthread_cleanup_push(cleanup, "thread 1 second handler");
   printf("thread 1 push complete\n");

   if (arg)
      return((void *)1);

   pthread_cleanup_pop(0);
   pthread_cleanup_pop(0);
   return((void *)1);
}
```

We should match calls to pthread_cleanup_pop with the calls to pthread_cleanup_push; Otherwise, the program might not compile.

Thread termination

- Example(cont.)

```
void *
thr_fn2(void *arg)
{
    printf("thread 2 start\n");
    pthread_cleanup_push(cleanup, "thread 2 first handler");
    pthread_cleanup_push(cleanup, "thread 2 second handler");
    printf("thread 2 push complete\n");

    if (arg)
        pthread_exit((void *)2);

    pthread_cleanup_pop(0);
    pthread_cleanup_pop(0);
    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, (void *)1);
    if (err != 0)
        err_quit("can't create thread 1: %s\n", strerror(err));

    err = pthread_create(&tid2, NULL, thr_fn2, (void *)1);
    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);
}
```

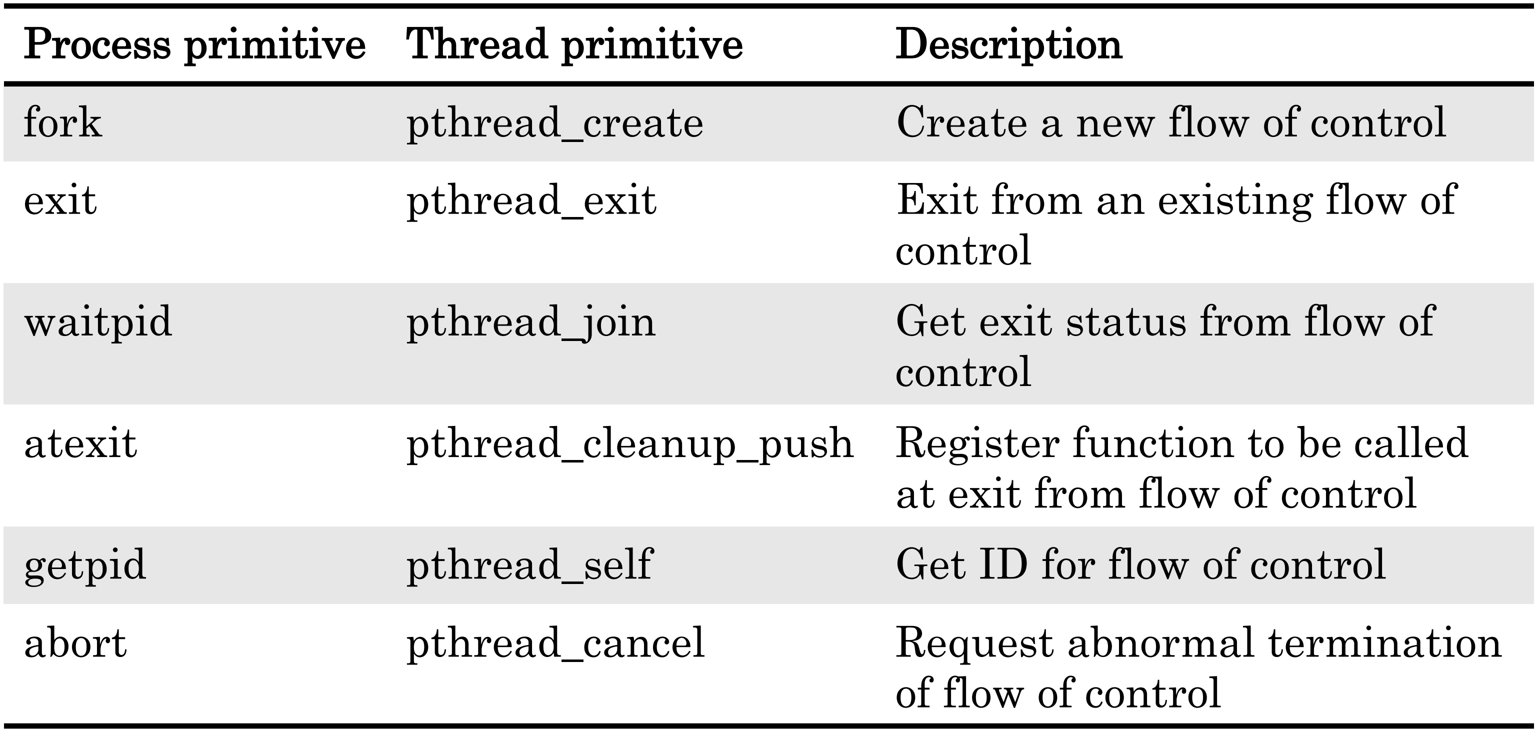
Thread termination

- execution

```
$ ./a.out
thread 1 start
thread 1 push complete
thread 2 start
thread 2 push complete
cleanup: thread 2 second handler
cleanup: thread 2 first handler
thread 1 exit code 1
thread 2 exit code 2
$
```

Thread termination

- Comparison of process and thread primitives

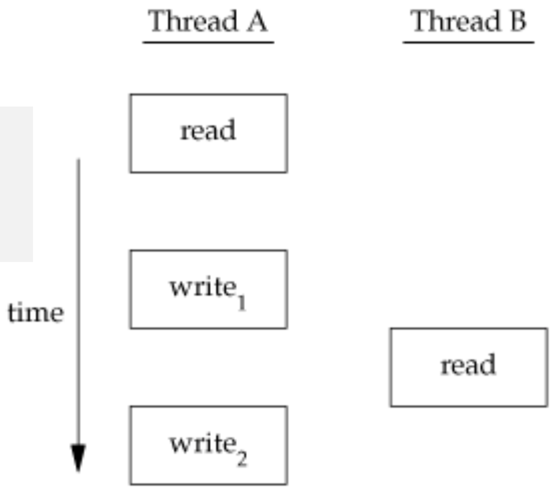


Process primitive	Thread primitive	Description
fork	pthread_create	Create a new flow of control
exit	pthread_exit	Exit from an existing flow of control
waitpid	pthread_join	Get exit status from flow of control
atexit	pthread_cleanup_push	Register function to be called at exit from flow of control
getpid	pthread_self	Get ID for flow of control
abort	pthread_cancel	Request abnormal termination of flow of control

Thread synchronization

- Why thread synchronization?
  - When one thread modify a variable that other threads read or modify, inconsistency problem exists.
  - When multiple threads share the same memory, each thread must see a consistent view of its data.

Assume that
the write operation takes
two memory cycles.



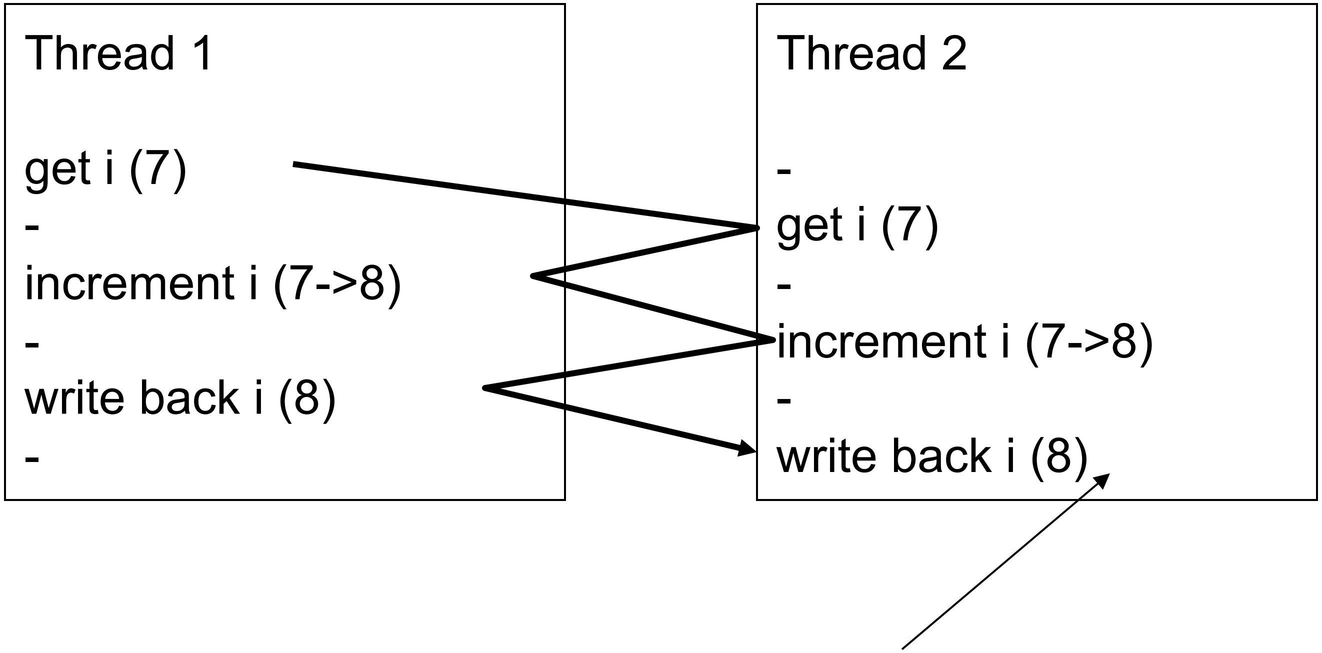
Interleaved memory cycles with two threads.

Thread synchronization

- Even a simple increment operation is broken down into three steps.
  - Read the memory location into a register.
  - Increment the value in the register.
  - Write the new value back to the memory location.

Thread synchronization

- If two thread try to increment the same variable at almost the same time?



Thread 1

get i (7)
-
increment i (7->8)
-
write back i (8)
-

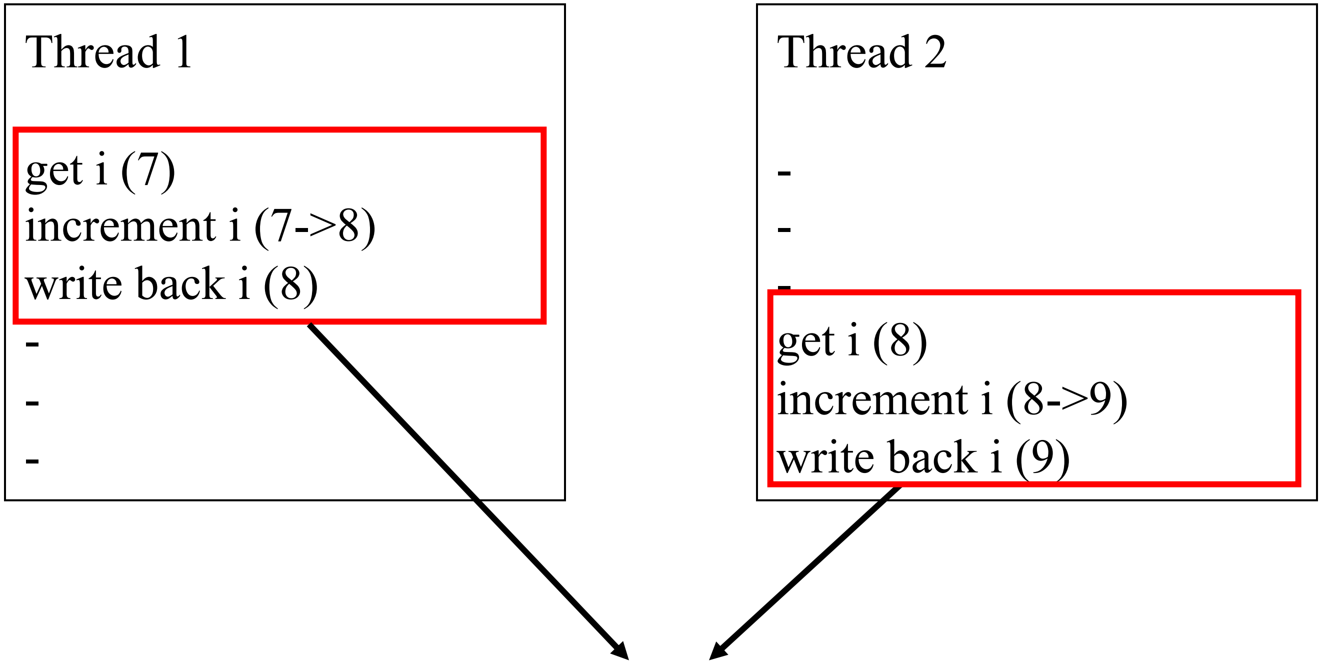
Thread 2

-
get i (7)
-
increment i (7->8)
-
write back i (8)

value of i is 8. (error)

Thread synchronization

- Synchronization is required.
  - mutex, condition variable, …



Thread 1

get i (7)
increment i (7->8)
write back i (8)
-
-
-

Thread 2

-
-
-
get i (8)
increment i (8->9)
write back i (9)

Operation should be atomic.
(The thread should have a lock to access the shared variable.)

Mutexes

- Mutex (mutual exclusion)
  - A lock that we set before accessing a shared resource and release when we’re done.
  - While it is set, any other thread that tries to set it will block until it is released.
  - When the mutex is released, all threads blocked on the lock will be unblocked.
  - Finally, one of threads can set the lock, and the others will block again.

Mutexes

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
                                        Both return: 0 if OK, error number on failure
```

- Pthread_mutex_init
  - Mutex variable is represented by *pthread_mutex_t* data type.
  - Before using a mutex, we must first initialize it.
  - To initialize a mutex with the default attributes, *attr* is set to NULL.
- Pthread_mutex_destroy
  - Destroy the mutex.

Mutexes

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
                                        All return: 0 if OK, error number on failure
```

- Pthread_mutex_lock
  - Lock a mutex.
  - If the mutex is already locked, the calling thread will block until the mutex is unlocked.
- Pthread_mutex_trylock
  - Nonblocking version of pthread_mutex_lock().
- Pthread_mutex_unlock
  - Unlock a mutex.

Mutexes

- Example

```
#include <stdlib.h>
#include <pthread.h>

struct foo {
    int             f_count;
    pthread_mutex_t f_lock;
    /* ... more stuff here ... */
};
```

Mutexes

- Example(cont.)

```
struct foo *foo_alloc(void)                    /* allocate the object */
{
    struct foo *fp;

    if ((fp = malloc(sizeof(struct foo))) != NULL) {
        fp->f_count = 1;
        if (pthread_mutex_init(&fp->f_lock, NULL) != 0) {
            free(fp);
            return(NULL);
        }
        /* ... continue initialization ... */
    }
    return(fp);
}
```

Mutexes

- Example(cont.)

```
void foo_hold(struct foo *fp)                   /* add a reference to the object */
{
    pthread_mutex_lock(&fp->f_lock);
    fp->f_count++;
    pthread_mutex_unlock(&fp->f_lock);
}

void foo_rele(struct foo *fp)                   /* release a reference to the object */
{
    pthread_mutex_lock(&fp->f_lock);
    if (--fp->f_count == 0) {                   /* last reference */
        pthread_mutex_unlock(&fp->f_lock);
        pthread_mutex_destroy(&fp->f_lock);
        free(fp);
    } else {
        pthread_mutex_unlock(&fp->f_lock);
    }
}
```

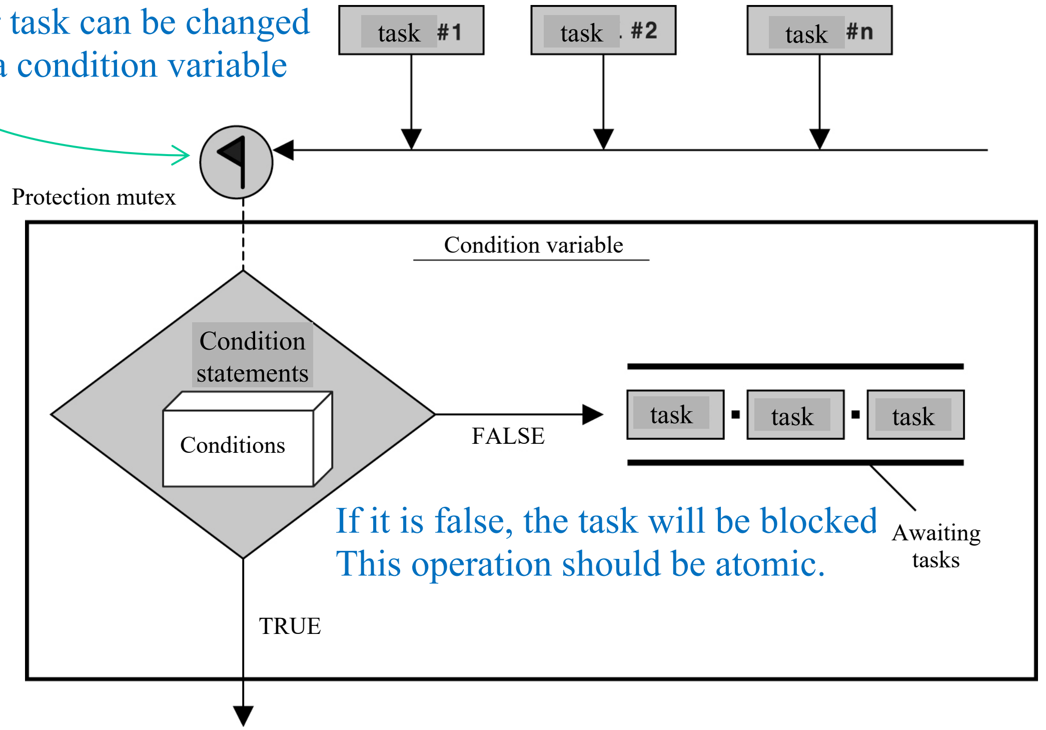
Condition variables

- Condition variables
  - Another synchronization mechanism
  - Provides a place for threads to rendezvous.
    - When used with mutexes, it allows threads to wait for arbitrary condition to occur.
  - Condition itself is protected by a mutex.
    - A thread must first lock the mutex to change the condition state.
    - Other threads will not notice the change until they acquire the mutex.

Condition variables

- Condition variables
  - One task can wait for other task to create a desired condition in the shared resource.

Because the status of another task can be changed while checking the status of a condition variable



If it is false, the task will be blocked
This operation should be atomic.

Condition variables

```
#include <pthread.h>

int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t * attr);
int pthread_cond_destroy(pthread_cond_t *cond);
                                        Both return: 0 if OK, error number on failure
```

- Pthread_cond_init
  - Initializes the condition variable referenced by *cond* with attributes referenced by *attr*.
  - Condition variable is represented by the pthread_cond_t data type.
- Pthread_cond_destroy
  - Destroy the given condition variable specified by *cond*.

Condition variables

```
#include <pthread.h>

int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t * mutex);
int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex,
                           const struct timespec *timeout);
                                 Both return: 0 if OK, error number on failure
```

- Pthread_cond_wait
  - Wait for a condition to be true.
- Pthread_cond_timedwait
  - Same with pthread_cond_wait.
  - An error is returned if the specified time passes.

Condition variables

```
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
                                        Both return: 0 if OK, error number on failure
```

- Notify threads that a condition has been satisfied.
- Pthread_cond_signal
  - Wake up one thread waiting on a condition.
- Pthread_cond_broadcast
  - Wake up all threads waiting on a condition.

Condition variables

- Example

```
#include <pthread.h>

struct msg {
    struct msg *m_next;
    /* ... more stuff here ... */
};

struct msg *workq;
pthread_cond_t qready = PTHREAD_COND_INITIALZER;
pthread_mutex_t qlock = PTHREAD_MUTEX_INITIALZER;
```

Condition variables

Example (cont’d)

```
void process_msg(void) {
    struct msg *mp;
    for (;;) {
          pthread_mutex_lock(&qlock);
          while (workq == NULL) pthread_cond_wait(&qready, &qlock);
          mp = workq;
          workq = mp->m_next;
          pthread_mutex_unlock(&qlock);
     }
};
void enqueue_msg(struct msg *mp) {
      pthread_mutex_lock(&qlock);
      mp->m_next = workq;
      workq = mp;
      pthread_mutex_unlock(&qlock);
      pthread_cond_signal(&qready);
}
```