CS4254

Computer Network Architecture and Programming

Dr. Ayman A. Abdel-Hamid

Computer Science Department

Virginia Tech

Threads

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

Outline

•Threads (Chapter 26)

➤Introduction

Threads

- ➢Basic Thread Functions
- ➤TCP echo client using threads
- ➤TCP Echo Server using threads
- ➤Thread Synchronization

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

Introduction 1/4

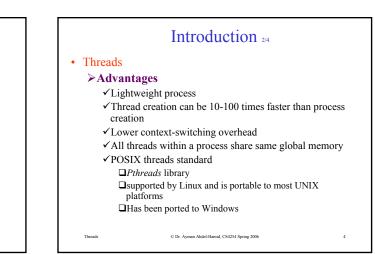
- Problems with fork approach for concurrency
 - ≻ Expensive

Threads

- ✓ Memory copied from parent to child
- ✓ All descriptors are duplicated in the child
- ✓ Can be implemented using *copy-on-write* (don't copy until child needs own copy)
- > IPC (Inter-process communication) required to pass information between parent and child after fork
- Threads can help!!

Threads

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006



	Introduction 3/4	
• Threads	5	
≻Disad	lvantages	
Ina	bal variables are shared between threads \rightarrow dvertent modification of shared variables can be astrous (need for concurrency control)	
	ny library functions are not <i>thread-safe</i> . Library functions that return <i>pointers to internal static</i> <i>arrays</i> are not thread safe	
	To make it thread-safe \rightarrow caller allocates space for the result and passes that pointer as argument to the function	
	k of robustness: If one thread crashes, the whole lication crashes	
Threads	© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006	5

Introduction 4/4

Threads

≻State

Threads within a process share global data: process instructions, most data, descriptors, etc, ...
 File descriptors are shared. If one thread closes a file, all

- other threads can't use the file ✓ Each thread has its own stack and local variables
- \checkmark I/O operations block the calling thread.
- Some other functions act on the whole process. For example, the exit() function terminates the entire process and all associated threads.

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

Basic Thread Functions 1/6

pthread_create function

#include <pthread.h>

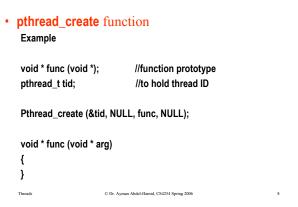
Thread

int pthread_create (pthread_t * tid, const pthread_attr_t *attr, void *(*func) (void*), void *arg);

//Returns 0 if OK, positive Exxx value on error

- When a program is started, single thread is created (main thread). Create more threads by calling pthread_create()
- pthread_t is often an unsigned int. returns the new thread ID
- attr: is the new thread attributes: priority, initial stack size, daemon thread or not. To get default attributes, pass as NULL
- ➤ func: address of a function to execute when the thread starts
- arg: pointer to argument to function (for multiple arguments, package into a structure and pass address of structure to function)

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006



Basic Thread Functions 26

Basic Thread Functions 3/6				
 pthread_join function <pre>#include <pthread.h> int pthread_join (pthread_t tid, void ** status); //Returns 0 if OK, positive Exxx value on error</pthread.h></pre> 				
Wait for a given thread to terminate (similar to waitpid() for Unix processes)				
➤ Must specify thread ID (tid) of thread to wait for				
≻ If status argument non-null				
Return value from thread (pointer to some object) is pointed to by status				
Threads © Dr. Ayman Abdel-Hamid, CS4254 Spring 2006 9				

Basic Thread Functions 4/6 pthread_self function #include <pthread.h> pthread_t pthread_self (void); //Returns thread ID of calling thread > similar to getpid() for Unix processes

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

•

Threads

•

11

Basic Thread Functions 5/6

• pthread detach function #include <pthread.h>

Threads

int pthread_detach (pthread_t tid); //Returns 0 if OK, positive Exxx value on error

- > A thread is either *joinable* (default) or *detached*
- > When a joinable thread terminates \rightarrow thread ID and exit status are retained until another thread calls pthread_join
- > When a detached thread terminates \rightarrow all resources are released and can not be waited for to terminate
- Example : pthread_detach (pthread_self());

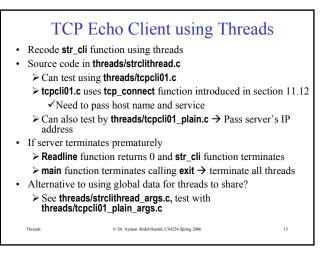
© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

Basic Thread Functions 66 pthread_exit function #include <pthread.h> void pthread_exit (void * status); //Does not return to caller > If thread not detached, thread ID and exit status are retained for a later **pthread_join** by another thread in the calling process > Other ways for a thread to terminate ✓ Function that started the thread terminates, with its return value

- being the exit status of the thread
- ✓ main function of process returns or any thread calls exit,. In such case, process terminates including any threads

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

12



TCP Echo Server using Threads 1/2 · One thread per client instead of one child process per client Source code in threads/tcpserv01.c Uses tcp_listen function introduced in section 11.13 Main processing loop for (;;) { len = addrlen; connfd = Accept(listenfd, cliaddr, &len); Pthread_create(&tid, NULL, &doit, (void *) connfd); } > The casting (void*) connfd is OK on most Unix implementations (size of an integer is <= size of a pointer) ▶ Is there an alternative approach? © Dr. Ayman Abdel-Hamid, CS4254 Spring 2006 14 Thread

•	TCP Echo Server using Threads 2/ If we pass the address of connfd , what can go wrong?	2
	for (;;) {	
	len = addrlen;	
	connfd = Accept(listenfd, cliaddr, &len);	
	Pthread_create(&tid, NULL, &doit, &connfd);	
	}	
•	A more correct approach would be to allocate space for the connected descriptor every time before the call to accept	
	for (; ;) {	
	len = addrlen;	
	iptr = Malloc(sizeof(int));	
	*iptr = Accept(listenfd, cliaddr, &len);	
	Pthread_create(&tid, NULL, &doit, iptr);	
	} // source code in threads/tcpserv02.c	
	Threads © Dr. Ayman Abdel-Hamid, CS4254 Spring 2006	15

Thread Synchronization: Mutex

- How can a thread ensure that access/updates to shared variables are atomic?
- How can a thread ensure that it is the only thread executing some critical piece of code?
 - Need a mechanism for thread coordination and synchronization
 - semaphores and mutex calls
- Mutex: Mutual Exclusion
 - Threads can create a mutex and initialize it. Before entering a critical region, lock the mutex.
 - > Unlock the **mutex** after exiting the critical region
- See examples in threads/example01.c and threads/example02.c Threads
 C Dr. Ayman Abdel-Hamid, CS4254 Spring 2006
 16

Thread Synchronization: Semaphores A mutex allows one thread to enter a critical region A semaphore can allow some N threads to enter a critical region Used when there is a limited (but more than 1) number of copies of a shared resource Can be dynamically initialized Thread calls a semaphore wait function before it enters a critical region Semaphore is a generalization of a mutex

© Dr. Ayman Abdel-Hamid, CS4254 Spring 200

Thread Synchronization: Condition Variables 1/2

- · A condition variable is only needed where
 - A set of threads are using a mutex to provide mutually exclusive access to some resource
 - Once a thread acquires the resource, it needs to wait for a particular condition to occur
- · If no condition variables are available
 - Some form of *busy waiting* in which thread repeatedly acquires the **mutex**, tests the condition, and then releases the **mutex** (wasteful solution)

18

• A condition variable allows a thread to release a **mutex** and block on a condition atomically

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

Threads

Thread Synchronization: Condition Variables 2/2 • Acquire a mutex

- Call **pthread_cond_wait** specifying both a condition variable and the mutex being held
- · Thread blocks until some other thread signals the variable
- Two forms of signaling

Thread

Threads

Allow one thread to proceed, even if multiple threads are waiting on the signaled variable

- \checkmark OS simultaneously unblocks the thread and allows the thread to reacquire the mutex
- Allow all threads that are blocked on the variable to proceed
- Blocking on a condition variable does not prevent others from proceeding through the critical section, another thread can acquire the mutex

© Dr. Ayman Abdel-Hamid, CS4254 Spring 2006

19