CS323 Operating Systems
Threads

Yuanyuan Zhou
Lecture 4
1/29/2003

Content of this lecture

- Administrative announcements
- What is a thread?
- Examples of using threads
- Thread implementations
- Summary

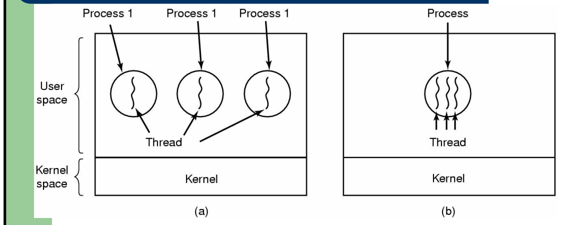
Administrative

- Test quiz due Friday

Process Review

- So What Is A Process?
  - It's one executing instance of a "program"
  - It's separate from other instances
  - It can start ("launch") other processes
  - It can be launched by them
- What's in a process?
  - Code (text), data, stack, heap
  - Process control block
    - Process state, priority, accounting
    - Program counter, register variables, stack pointers, etc
    - Open files and devices

Threads: Lightweight Processes



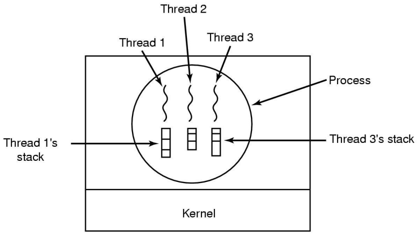
(a) Three processes each with one thread
(b) One process with three threads

The Thread Model

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	

- Threads in the same process share resources
- Each thread execute separately

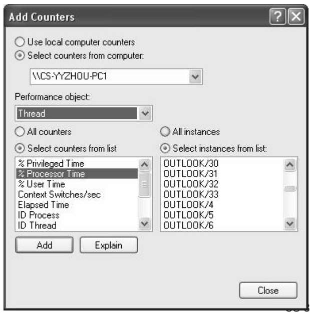
Why each thread has its own stack?



A example program

```
#include ``csapp.h"
void *thread(void *vargp);

int main() {
 phtread_t pid; // stores the new thread ID

 Pthread_create(&tid, NULL, thread, NULL); //create a new thread

 Pthread_join(tid, NULL); //main thread waits for the other thread to terminate
 exit(0); /* main thread exits */
}

void *thread(void *vargp) /*thread routing*/
{
 printf(``Hello, world! \n");
 return NULL;
}
```

Windows Thread Lists from Performance Monitor



○ Use local computer counters
● Select counters from computer:

\\CS-YYZHOU-PC1

Performance object:

Thread

○ All counters
● Select counters from list

% Privileged Time
% Processor Time
% User Time
Context Switches/sec
Elapsed Time
ID Process
ID Thread

○ All instances
● Select instances from list:

OUTLOOK/30
OUTLOOK/31
OUTLOOK/32
OUTLOOK/33
OUTLOOK/4
OUTLOOK/5
OUTLOOK/6

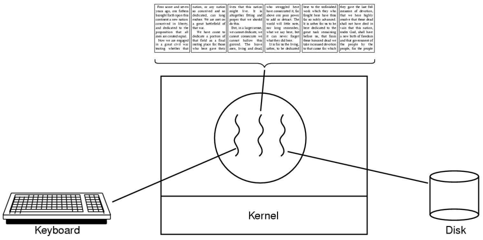
Group discussions (3 minutes)

- Similarity between processes and threads
- Difference between processes and threads
  - Threads share a single address space
- Real life examples?

Real Life Example

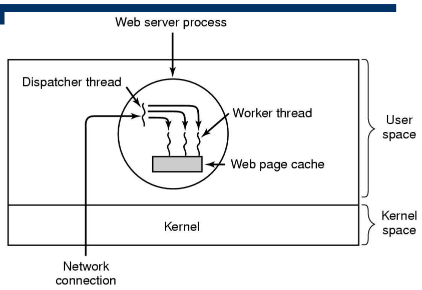
- Process
  - CS323 MP
  - Different from CS321's MP
- Thread
  - CS323 MP1, MP2, MP3, MP4, MP5
  - Each is different
  - Share
    - Nacho
    - Textbook
    - Personnel (TAs, instructors)
  - Affect each other

Thread Usage: word processor

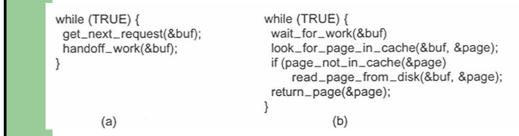


- What if it is single-threaded?

Thread Usage: Web Server



Web Server



```
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)

- Rough outline of code for previous slide
  (a) Dispatcher thread
  (b) Worker thread

Tradeoffs

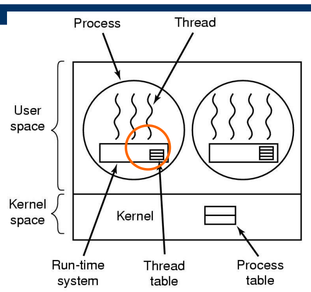
Three ways to construct a server

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

Blocking Vs. non-blocking System Calls

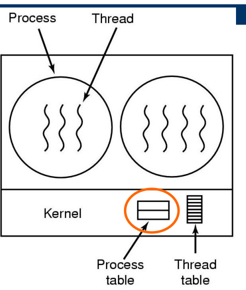
- Blocking system call
  - Usually I/O related: read(), fread(), getc(), write()
  - Doesn't return until the call completes
  - The process/thread is switched to blocked state
  - When the I/O completes, the process/thread becomes ready
  - Simple
  - Real life example: attending a lecture
- Using non-blocking system call for I/O
  - Asynchronous I/O
  - Complicated
  - The call returns once the I/O is initiated, and the caller continue
  - Once the I/O completes, an interrupt is delivered to the caller
  - Real life example: apply for job

Implementing Threads in User Space (old Linux)



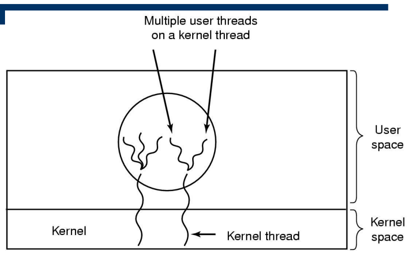
A user-level threads package

Implementing Threads in the Kernel (Windows 2000/XP)



A threads package managed by the kernel

Hybrid Implementations (Solaris)

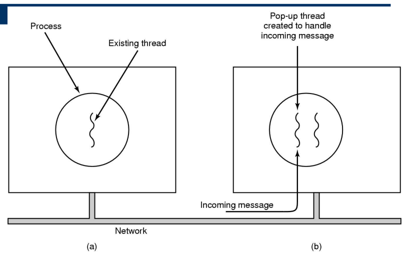


Multiplexing user-level threads onto kernel-level threads

Scheduler Activations

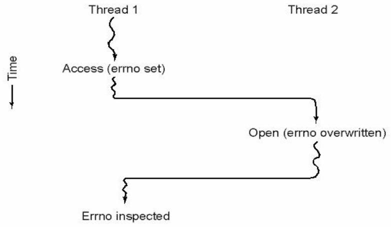
- User-level
- Goal – mimic functionality of kernel threads
  - gain performance of user space threads
- Avoids unnecessary user/kernel transitions
- Kernel assigns virtual processors to each process
  - lets runtime system allocate threads to processors
- Problem:
  Fundamental reliance on kernel (lower layer)
  calling procedures in user space (higher layer)

Pop-Up Threads



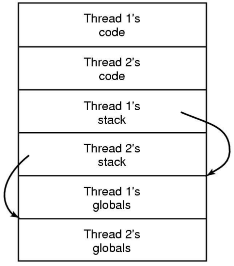
- Creation of a new thread when message arrives
  (a) before message arrives
  (b) after message arrives

A Challenge: Making Single-Threaded Code Multithreaded



Conflicts between threads over the use of a global variable

A solution: Private Global Variables



Thread 1's code

Thread 2's code

Thread 1's stack

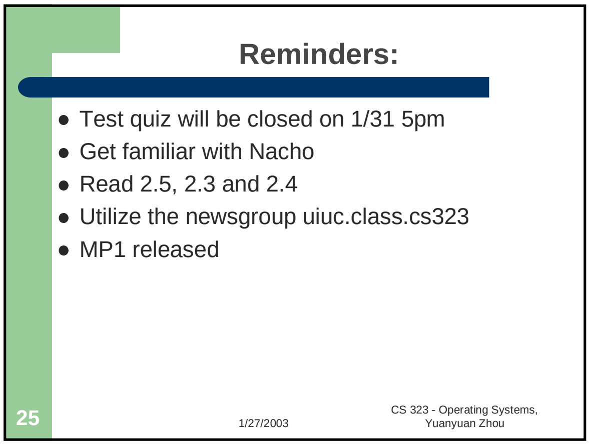
Thread 2's stack

Thread 1's globals

Thread 2's globals

Summary

- What is thread? Why need threads?
- Difference between process and thread
- Thread Implementations and their tradeoffs
  - User-level
  - Kernel-level
  - Hybrid
  - Service activation
  - Pop-up threads
- More details on threads will be discussed in “distributed system” (CS318) class
- Next lecture: Scheduling



Reminders:

- Test quiz will be closed on 1/31 5pm
- Get familiar with Nacho
- Read 2.5, 2.3 and 2.4
- Utilize the newsgroup uiuc.class.cs323
- MP1 released

25

1/27/2003

CS 323 - Operating Systems,
Yuanyuan Zhou