### Lecture 10

 $\circ$ 

**Shared Memory Multiprocessors** 

Institute of Computer Science & Information Technology, Faculty of Management & Computer Sciences, The University of Agriculture, Peshawar, Pakistan.

### Shared Memory Multiprocessors

- Shared Memory Multiprocessors<br>• Any memory location can be accessible by<br>any of the processors. nared Memory Multiproces<br>Any memory location can be acce<br>any of the processors.
- Shared Memory Multiprocessors<br>
 Any memory location can be accessible by<br>
 A single address space exists, meaning that<br>
 A single address space exists, meaning that<br>
 address within a single wage of address each memory location is given a unique Any memory location can be accessible by<br>any of the processors.<br>A single address space exists, meaning that<br>each memory location is given a unique<br>address within a single range of address. The processors.<br>
• A single address space exists, meaning that<br>
each memory location is given a unique<br>
address within a single range of address.<br>
• For small number of processors, common<br>
architecture is the single bus ar A single address space exists, meaning that<br>each memory location is given a unique<br>address within a single range of address.<br>For small number of processors, common<br>architecture is the single bus architecture:
- 

### Shared Memory Multiprocessors (1)



## Programming Alternatives **Programming Alternatives**<br>1. Using a supportive programming language.<br>2. Using library routines with an existing sequential language. **Programming Alternatives**<br>2. Using a supportive programming language.<br>2. Using library routines with an existing sequential language.<br>2. Using a sequential programming language and ask a parallelizing.

- 
- 
- **Programming Alternatives**<br>3. Using a supportive programming language.<br>3. Using library routines with an existing sequential language.<br>3. Using a sequential programming language and ask a parallelizing<br>compiler to convert **Programming Alternatives**<br>Using a supportive programming language.<br>Using library routines with an existing sequential language.<br>Using a sequential programming language and ask a parallelizing<br>compiler to convert it into p 1. Using a supportive programming language.<br>
2. Using library routines with an existing sequential<br>
3. Using a sequential programming language are compiler to convert it into parallel executal<br>
4. UNIX processes.<br>
5. P-Thr
- 
- 5. P-Threads (POSIX thread)
- 9. Using a sequential programming language and ask a parallelizing<br>6. Compiler to convert it into parallel executable code.<br>4. CUNIX processes.<br>5. P-Threads (POSIX thread)<br>6. Using an existing sequential programming langua Using a sequential programming language and ask a parallelizing<br>compiler to convert it into parallel executable code.<br>UNIX processes.<br>P-Threads (POSIX thread)<br>Using an existing sequential programming language<br>supplemented compiler to convert it into parallel executable code.<br>
UNIX processes.<br>
P-Threads (POSIX thread)<br>
Using an existing sequential programming<br>
supplemented with compiler directives for<br>
parallelism., e.g., OpenMP.<br>
mmad Asim,



### Constructs for Parallelism

- Creating Concurrent Processes
- **Constructs for Parallelism**<br>Creating Concurrent Processes<br>• FORK-JOIN was described by Conway in 1963,<br>and was known before 1960. **Constructs for Parallelism**<br>reating Concurrent Processes<br>FORK-JOIN was described by Conway in 1963,<br>and was known before 1960.
- Creating Concurrent Processes<br>
 FORK-JOIN was described by Conway in 1963,<br>
 In the original construct a FORK statement<br>
 In the original construct a FORK statement<br>
generates one new path for a concurrent<br>
process and reating Concurrent Processes<br>FORK-JOIN was described by Conway in 1963,<br>and was known before 1960.<br>In the original construct a FORK statement<br>generates one new path for a concurrent<br>process and the concurrent processes use FORK-JOIN was described by Conway in 1963,<br>and was known before 1960.<br>In the original construct a FORK statement<br>generates one new path for a concurrent<br>process and the concurrent processes use the<br>JOIN statement at their FORK-JOIN was described by Conway in 1963,<br>and was known before 1960.<br>In the original construct a FORK statement<br>generates one new path for a concurrent<br>process and the concurrent processes use the<br>JOIN statement at their



- UNIX Heavyweight Processes<br>• The UNIX system call fork() creates a new process. process.
- UNIX Heavyweight Processes<br>• The UNIX system call fork() creates a new<br>process.<br>• The new process(child process) is an exact copy<br>of the calling process except that it has a unique<br>process ID. UNIX Heavyweight Processes<br>The UNIX system call fork() creates a new<br>process.<br>The new process(child process) is an exact copy<br>of the calling process except that it has a unique<br>process ID. process ID. It has its own copy of the parent's variables.<br>
The new process (child process) is an exact copy<br>
of the calling process except that it has a unique<br>
process ID.<br>
It has its own copy of the parent's variables. The new process(child process) is an exact copy<br>of the calling process except that it has a unique<br>process ID.<br>It has its own copy of the parent's variables.<br>They are assigned the same values as the original<br>variables init The new process(child process) is a<br>of the calling process except that it<br>process ID.<br>It has its own copy of the parent's va<br>They are assigned the same values a<br>variables initially.
- 
- 
- The forked process starts execution at the point of the fork.
- On success, fort() returns 0 to the child process and returns the process ID of the child process to the parent process.
- Processes are 'joined' with the system calls wait() and exit():





### OS Review: Processes

- **OS Review: Processes<br>• processes contain information about program<br>• resources and program execution state, including:<br>– Process ID, process group ID, user ID, and group ID CS Review: Processes**<br>
processes contain information about program<br>
resources and program execution state, including:<br>
Process ID, process group ID, user ID, and group ID<br>
Fragment Working directory Program instructions **COS Review: Processes**<br>
• processes contain information about program<br>
resources and program execution state, including:<br>
— Process ID, process group ID, user ID, and group ID<br>
— Environment, Working directory, Program in **COS Review: Processes**<br>
• processes contain information about program<br>
resources and program execution state, including:<br>
— Process ID, process group ID, user ID, and group ID<br>
— Environment, Working directory, Program in **COS Review: Processes**<br>
• processes contain information about resources and program execution state, includ<br>
– Process ID, process group ID, user ID, and group<br>
– Environment, Working directory, Program instr<br>
– Registers
- 
- 
- 
- File descriptors, Signal actions
- processes contain information about program<br>resources and program execution state, including:<br>— Process ID, process group ID, user ID, and group ID<br>— Environment, Working directory, Program instructions<br>— Registers, Stac processes contain information about program<br>resources and program execution state, including:<br>Process ID, process group ID, user ID, and group ID<br>Environment, Working directory, Program instructions<br>Registers, Stack, Heap<br> resources and program execution sta<br>Process ID, process group ID, user ID,<br>Environment, Working directory, Prog<br>Registers, Stack, Heap<br>File descriptors, Signal actions<br>Shared libraries, Inter-process com<br>(such as message q - Process ID, process group ID, user ID, and group ID<br>
- Environment, Working directory, Program instructions<br>
- Registers, Stack, Heap<br>
- File descriptors, Signal actions<br>
- Shared libraries, Inter-process communication t - Environment, Working directory, Program instru<br>
- Registers, Stack, Heap<br>
- File descriptors, Signal actions<br>
- Shared libraries, Inter-process communicatior<br>
(such as message queues, pipes, semaphor<br>
shared memory).<br>
• – Registers, Stack, Heap<br>
– File descriptors, Signal actions<br>
– Shared libraries, Inter-process communicat<br>
(such as message queues, pipes, semapl<br>
shared memory).<br>
• When we run a program, a process is create<br>
– E.g. ./a.
- 
- 
- 

### **Threads**

- Threads<br>• Threads use, and exist within, the process resources.<br>• Scheduled and run as independent entities.<br>• Duplicate only the bare essential resources that enable them
- 
- **Scheduled and run as independent entities.**<br>Scheduled and run as independent entities.<br>• Duplicate only the bare essential resources that enable them<br>to exist as executable code. Threads<br>
• Threads use, and exist within, the process resources.<br>
• Scheduled and run as independent entities.<br>
• Duplicate only the bare essential resources that enable them<br>
to exist as executable code. Threads<br>Threads use, and exist within, the process resources.<br>Scheduled and run as independent entities.<br>Duplicate only the bare essential resources that enable<br>to exist as executable code.



Threads contain only necessary information, such as a stack (for local variables, function arguments, return values), a copy of the registers, program counter and any thread-specific data to allow them to be scheduled individually. Other data is shared within the process betweena@threads.

### What is a Thread in Real

- What is a Thread<br>• OS view<br>– An independent stream of instituted to run by the OS. What is a Thread in Real<br>OS view<br>– An independent stream of instructions that can be<br>scheduled to run by the OS.<br>Software developer view What is a Thread in Real<br>S view<br>An independent stream of instructions that can be<br>scheduled to run by the OS.<br>Strare developer view<br>A "precedure" that runs independently from the main **Solution What is a Thread in Real<br>
• OS view**<br>
– An independent stream of instructions that can<br>
scheduled to run by the OS.<br>
• Software developer view<br>
– A "procedure" that runs independently from the mainter program
- 
- What is a Thread in Real<br>
OS view<br>
 An independent stream of instructions that can be<br>
scheduled to run by the OS.<br>
Software developer view<br>
 A "procedure" that runs independently from the main<br>
program<br>
 Imagine multip program **IMAGE IS A THEAG IN KEAT<br>
S view<br>
An independent stream of instructions that can be<br>
scheduled to run by the OS.<br>
Sftware developer view<br>
A "procedure" that runs independently from the main<br>
program<br>
· Imagine multiple su** view<br>
un independent stream of instructio<br>
heduled to run by the OS.<br>
tware developer view<br>
"procedure" that runs independently<br>
ogram<br>
Imagine multiple such procedures of main i<br>
and/or independently<br>
equential program: a US VIEW<br>
- An independent stream of instructions that can be<br>
scheduled to run by the OS.<br>
Software developer view<br>
- A "procedure" that runs independently from the main<br>
program<br>
· Imagine multiple such procedures of main scheduled to run by the OS.<br>
Software developer view<br>
- A "procedure" that runs independently from the main<br>
program<br>
· Imagine multiple such procedures of main run simultaneously<br>
and/or independently<br>
- Sequential progra A "procedure" that runs independently from the main<br>program<br>
• Imagine multiple such procedures of main run simultaneously<br>
and/or independently<br>
Sequential program: a single stream of instructions in a<br>
program.<br>
Multi-th
	-
	- program.
	- streams
		-



## POSIX threads (PThreads)

- **POSIX threads (PThreads)**<br>• Threads used to implement parallelism in shared memory multiprocessor systems, such as SMPs **POSIX threads (PThreads)**<br>Threads used to implement parallelism in shared<br>memory multiprocessor systems, such as SMPs<br>Historically hardware vendors have implemented
- **POSIX threads (PThreads)**<br>• Threads used to implement parallelism in shared<br>memory multiprocessor systems, such as SMPs<br>• Historically, hardware vendors have implemented<br>their own proprietary versions of threads<br>– Portabi **POSIX threads (PThreads)**<br>Threads used to implement parallelism in shared<br>memory multiprocessor systems, such as SMPs<br>Historically, hardware vendors have implemented<br>their own proprietary versions of threads<br>– Portability • Threads used to implement parallelism in shared<br>memory multiprocessor systems, such as SMPs<br>• Historically, hardware vendors have implemented<br>their own proprietary versions of threads<br>– Portability a concern for software Threads used to implement paramelism in snared<br>memory multiprocessor systems, such as SMPs<br>Historically, hardware vendors have implemented<br>their own proprietary versions of threads<br>– Portability a concern for software deve memory mutup rocessor systems, such as siting<br>Historically, hardware vendors have implemented<br>their own proprietary versions of threads<br>– Portability a concern for software developers.<br>For UNIX systems, a standardized C la
- istorically, hardware vendors have implemente<br>eir own proprietary versions of threads<br>Portability a concern for software developers.<br>Dr UNIX systems, a standardized C languag<br>reads programming interface has bee<br>becified by
	-

### The POSIX Thread API

- Commonly referred to as PThreads, POSIX has emerged as the standard threads API, supported by most vendors. The POSIX Thread API<br>Commonly referred to as PThreads, POSIX has<br>emerged as the standard threads API, supported<br>by most vendors.<br>— Implemented with a pthread.h header/include file and<br>a thread library Commonly referred to as PThreads, POSIX has<br>emerged as the standard threads API, supported<br>by most vendors.<br>— Implemented with a pthread.h header/include file and<br>a thread library<br>Functionalities<br>— Thread management, e.g. Commonly referred to as  $P1$  hreadenerged as the standard thread<br>by most vendors.<br>— Implemented with a pthread.h hea<br>a thread library<br>Functionalities<br>— Thread management, e.g. creation a<br>— Thread synchronization primitive
	- a thread library

### **• Functionalities**

- 
- -
	- Condition variables
	- • Reader/writer locks
- –Thread-specific data

# PThread API<br><sub>read.h></sub>

### • #include <pthread.h>



### Thread Creation

• Initially, main() program comprises a single, default thread Thread Creat<br>
Initially, main() program comprises a single, default<br>
– All other threads must be explicitly created<br>
int pthread\_create(<br>
pthread t \*thread

```
Thread Creat<br>
initially, main() program comprises a single, defaul<br>
-All other threads must be explicitly created<br>
int pthread_create(<br>
pthread_t *thread,<br>
const pthread_attr_t *attr,
          Thread Creation<br>ally, main() program comprises a single, default thread<br>I other threads must be explicitly created<br>pthread_create(<br>pthread_t *thread,<br>const pthread_attr_t *attr,<br>void *(*start_routine)(void *),
           Thread Creation<br>ally, main() program comprises a single, default thread<br>other threads must be explicitly created<br>pthread_create(<br>const pthread_ttr_t *attr,<br>void *(*start_routine)(void *),<br>void * arg);
         void *(*start_routine)(void *), 
          void * arg);
```
- thread: An opaque, unique identifier for the new thread returned by the subroutine
- attr: An opaque attribute object that may be used to set thread attributes You can specify a thread attributes object, or NULL for the default values
- start routine: the C routine that the thread will execute once it is created
- arg: A single argument that may be passed to start routine. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

Opaque object: A letter is an opaque object to the mailman, and sender and receiver know the information.

### Thread Creation

- pthread\_create creates a new thread and makes it executable, i.e. run immediately in theory<br>– can be called any number of times from anywhere within your code **Thread Creation**<br>
• pthread\_create creates a new thread and makes it endomediately in theory<br>
– can be called any number of times from anywhere with<br>
Once created, threads are peers, and may create other in Thread Creation<br>
• pthread\_create creates a new thread and makes it executable, i.e. run<br>
— can be called any number of times from anywhere within your code<br>
Once created, threads are peers, and may create other threads<br>
T **Thread Creation**<br>
• pthread\_create creates a new thread and makes it executable, i.e. run<br>
immediately in theory<br>
– can be called any number of times from anywhere within your code<br>
• Once created, threads are peers, and **Thread Creation**<br>
• pthread\_create creates a new thread and makes it executable, i.e. run<br>
immediately in theory<br>
– can be called any number of times from anywhere within your code<br>
• Once created, threads are peers, and
	-
- 
- 



### Terminating Threads

- 
- **Ferminating Threads**<br>• pthread\_exit is used to explicitly exit a thread<br>- Called after a thread has completed its work and is no<br>longer required to exist **Ferminating Threads**<br>
pthread\_exit is used to explicitly exit a thread<br>
- Called after a thread has completed its work and is no<br>
longer required to exist<br>
If main()finishes before the threads it has created **Terminating Threads**<br>
Aread\_exit is used to explicitly exit a thread<br>
Called after a thread has completed its work and<br>
longer required to exist<br>
main()finishes before the threads it has created<br>
If exits with othread exi **If the Threads**<br> **If the thread** exit is used to explicitly exit a thread<br>
- Called after a thread has completed its work and is no<br>
longer required to exist<br>
• If main()finishes before the threads it has created<br>
- If ex **Ferminating Threads**<br>
pthread\_exit is used to explicitly exit a thread<br>
– Called after a thread has completed its work and is no<br>
longer required to exist<br>
If main()finishes before the threads it has created<br>
– If exits w **Terminating Threads**<br>pthread\_exit is used to explicitly exit a thread<br>– Called after a thread has completed its work and is no<br>longer required to exist<br>If main()finishes before the threads it has created<br>– If exits with p
- -
	- main()finishes
- pthread\_exit is used to explicitly exit a thread<br>
 Called after a thread has completed its work and is no<br>
longer required to exist<br>
 If main()finishes before the threads it has created<br>
 If exits with pthread\_exit(), - Called after a thread has completed its work and is no<br>longer required to exist<br>If main()finishes before the threads it has created<br>- If exits with pthread\_exit(), the other threads will continue<br>to execute<br>- Otherwise, longer required to exist<br>
If main()finishes before the threads it has created<br>  $-$  If exits with pthread\_exit(), the other threads will continue<br>
to execute<br>  $-$  Otherwise, they will be automatically terminated wher<br>  $\frac{\$ • If main()finishes before the threads it has created<br>
— If exits with pthread\_exit(), the other threads will continue<br>
to execute<br>
— Otherwise, they will be automatically terminated when<br>
main()finishes<br>
• The programmer to execute<br>Otherwise, they will be automatically termi<br>main()finishes<br>ne programmer may optionally specify a t<br>atus, which is stored as a void pointer for<br>at may join the calling thread<br>leanup: the pthread\_exit()routine do
- -

### Thread Attribute

Thread Attril<br>int pthread\_create(<br>pthread\_t \*thread,<br>const pthread\_attr\_t \*attr, Thread Attribute<br>
pthread\_create(<br>
pthread\_t \*thread,<br>
const pthread\_attr\_t \*attr,<br>
void \*(\*start\_routine)(void \*), Thread Attribute<br>
pthread\_create(<br>
thread\_t \*thread,<br>
const pthread\_attr\_t \*attr,<br>
oid \*(\*start\_routine)(void \*),<br>
void \* arg); void \*(\*start\_routine)(void \*), void  $*$  arg); Thread Attribute<br>
int pthread\_t \*thread,<br>
const pthread\_attr\_t \*attr,<br>
void \*(\*start\_routine)(void \*),<br>
void \* arg);<br>
• Attribute contains details about<br>
– whether scheduling policy is inherited or explicit<br>
– scheduling p int pthread\_create(<br>
pthread\_t \***thread**,<br>
const pthread\_attr\_t \***attr,**<br>
void \*(\***start\_routine**)(void \*),<br>
void \* **arg**);<br>
Attribute contains details about<br>
— whether scheduling policy is inherited or explicit<br>
— schedul Int pthread\_t \*thread,<br>
const pthread\_attr\_t \*attr,<br>
void \*(\*start\_routine)(void \*),<br>
void \* arg);<br>
Attribute contains details about<br>
— whether scheduling policy is inherited or explicit<br>
— scheduling policy, scheduling p

- -
	-
	-
- const pthread\_attr\_t \*attr,<br>void \*(\*start\_routine)(void \*),<br>void \* arg);<br>• Attribute contains details about<br>– whether scheduling policy is inherited or explicit<br>– scheduling policy, scheduling priority<br>– stack size, stack void \*(\*start\_routine)(void \*),<br>void \* arg);<br>Attribute contains details about<br>– whether scheduling policy is inherited or explicit<br>– scheduling policy, scheduling priority<br>– stack size, stack guard region size<br>pthread\_att • Attribute contains details about<br>
– whether scheduling policy is inherited or explicit<br>
– scheduling policy, scheduling priority<br>
– stack size, stack guard region size<br>
• pthread\_attr\_init and pthread\_attr\_destroy are u Attribute contains details about<br>
— whether scheduling policy is inherited or explicit<br>
— scheduling policy, scheduling priority<br>
— stack size, stack guard region size<br>
pthread\_attr\_init and pthread\_attr\_destroy are<br>
initi
- 

## Passing Arguments to Threads

- Passing Arguments to Threads<br>• The pthread\_create() routine permits the programmer to pass one argument to the thread start routine **Passing Arguments to Threads**<br>The pthread\_create() routine permits the programmer to<br>pass one argument to the thread start routine<br>For cases where multiple arguments must be passed: **Passing Arguments to Threads**<br>• The pthread\_create() routine permits the programmer to<br>pass one argument to the thread start routine<br>• For cases where multiple arguments must be passed:<br>- Create a structure which contains Passing Arguments to Threads<br>The pthread\_create() routine permits the programmer to<br>pass one argument to the thread start routine<br>For cases where multiple arguments must be passed:<br>- Create a structure which contains all o **Passing Arguments to Threads**<br>The pthread\_create() routine permits the programmer to<br>pass one argument to the thread start routine<br>For cases where multiple arguments must be passed:<br>— Create a structure which contains all **Passing Arguments to I hreads**<br>The pthread\_create() routine permits the programmer to<br>pass one argument to the thread start routine<br>For cases where multiple arguments must be passed:<br>- Create a structure which contains al • The pthread\_create() routine permits the programmer to pass one argument to the thread start routine<br>
• For cases where multiple arguments must be passed:<br>
– Create a structure which contains all of the arguments<br>
– The
- -
- pass one argument to the thread start routine<br>
For cases where multiple arguments must be passed:<br>
 Create a structure which contains all of the arguments<br>
 Then pass a pointer to the object of that structure in the<br>
pth For cases where multiple arguments must be passed:<br>  $-$  Create a structure which contains all of the arguments<br>  $-$  Then pass a pointer to the object of that structure in the<br>
pthread\_create()routine.<br>  $-$  All arguments m r cases where multiple arguments must<br>Create a structure which contains all of the a<br>Then pass a pointer to the object of the<br>pthread\_create()routine.<br>All arguments must be passed by reference are<br>ake sure that all passed
	-
- -
	- -

# Shared Memory and Threads Shared Memory and Threads<br>• All threads have access to the same global, shared memory<br>• Threads also have their own private data<br>• Programmers, are, responsible, for, synchronizing, access

- 
- 
- **Shared Memory and Threads**<br>• All threads have access to the same global, shared memory<br>• Threads also have their own private data<br>• Programmers are responsible for synchronizing access (protecting) globally shared data. **Shared Memory and Threads**<br>• All threads have access to the same global, shared memory<br>• Threads also have their own private data<br>• Programmers are responsible for synchronizing access (protecting) globally shared data. **Shared Memory and Threads**<br>All threads have access to the same global, shared memory<br>Threads also have their own private data.<br>Programmers are responsible for synchronizing access<br>(protecting) globally shared data.



# Thread Consequences **Fhread Consequen**<br>• Shared State!<br>- Accidental changes to global variables can<br>- Changes made by one thread to shared

- -
- Thread Consequences<br>• Shared State!<br>– Accidental changes to global variables can be fatal.<br>– Changes made by one thread to shared system resources<br>(such as closing a file) will be seen by all other threads – **Thread Consequences**<br>– Shared State!<br>– Accidental changes to global variables can be fatal.<br>– Changes made by one thread to shared system resources<br>(such as closing a file) will be seen by all other threads<br>– Two pointe Thread Consequences<br>
Shared State!<br>
Accidental changes to global variables can be fatal.<br>
Changes made by one thread to shared system resources<br>
(such as closing a file) will be seen by all other threads<br>
Two pointers havi – **Thread Consequences**<br>– Accidental changes to global variables can be fatal.<br>– Changes made by one thread to shared system resources<br>(such as closing a file) will be seen by all other threads<br>– Two pointers having the sa **Thread Consequences**<br>
• Shared State!<br>
— Accidental changes to global variables can be fatal.<br>
— Changes made by one thread to shared system resources<br>
(such as closing a file) will be seen by all other threads<br>
— Two poi
	-
	- possible
- Shared State!<br>• Accidental changes to global variables can be fatal.<br>– Changes made by one thread to shared system resources<br>(such as closing a file) will be seen by all other threads<br>– Two pointers having the same value programmer - Accidental changes to global variables can be fatal.<br>
- Changes made by one thread to shared system resources<br>
(such as closing a file) will be seen by all other threads<br>
- Two pointers having the same value point to the (such as closing a file) will be seen by all other<br>Two pointers having the same value point to th<br>Reading and writing to the same memory<br>possible<br>Therefore requires explicit synchronizati<br>programmer<br>any library functions a - Two pointers having the same value poin<br>
- Reading and writing to the same me<br>
possible<br>
- Therefore requires explicit synchre<br>
programmer<br>
• Many library functions are not thread<br>
- Library Functions that return pointer

- Reading and writing to the same memory locations is<br>possible<br>– Therefore requires explicit synchronization by the<br>programmer<br>Many library functions are not thread-safe<br>– Library Functions that return pointers to static i
- -

### Thread-safeness

- Thread-safeness<br>• Thread-safeness: in a nutshell, refers an<br>application's ability to execute multiple threads<br>simultaneously without "clobbering" shared data Thread-safeness<br>Thread-safeness: in a nutshell, refers an<br>application's ability to execute multiple threads<br>simultaneously without "clobbering" shared data<br>or creating "race" conditions **Thread-safeness**<br>Thread-safeness: in a nutshell, refers an<br>application's ability to execute multiple threads<br>simultaneously without "clobbering" shared data<br>or creating "race" conditions<br>Example: an application creates se Thread-safeness<br>Thread-safeness: in a nutshell, refers<br>application's ability to execute multiple thre<br>simultaneously without "clobbering" shared<br>or creating "race" conditions<br>Example: an application creates several thre<br>ea Thread-safeness<br>
• Thread-safeness: in a nutshell, refers an<br>
application's ability to execute multiple threads<br>
simultaneously without "clobbering" shared data<br>
or creating "race" conditions<br>
• Example: an application cre Thread-safeness<br>Thread-safeness: in a nutshell, refers an<br>application's ability to execute multiple threads<br>simultaneously without "clobbering" shared data<br>or creating "race" conditions<br>Example: an application creates seve Thread-safeness: in a nutshell, refers an<br>application's ability to execute multiple threads<br>simultaneously without "clobbering" shared data<br>or creating "race" conditions<br>Example: an application creates several threads,<br>eac hread-safeness: in a nutshell, refers an<br>pplication's ability to execute multiple threads<br>multaneously without "clobbering" shared data<br>c creating "race" conditions<br>xample: an application creates several threads,<br>ach of wh application's ability to execute multiple threads<br>simultaneously without "clobbering" shared data<br>or creating "race" conditions<br>Example: an application creates several threads,<br>each of which makes a call to the same librar
- each of which makes a call to the same library<br>routine: multaneously without "clobbering" shared data<br>
c creating "race" conditions<br>
stample: an application creates several threads,<br>
ach of which makes a call to the same library<br>
outine:<br>
This library routine accesses/modifies maraneously whenever electroning shared at<br>
r creating "race" conditions<br>
xample: an application creates several threa<br>
ach of which makes a call to the same libra<br>
buttine:<br>
This library routine accesses/modifies a glo<br>
s – In the right contract the same library<br>
Example: an application creates several threads,<br>
each of which makes a call to the same library<br>
routine:<br>
– This library routine accesses/modifies a global<br>
structure or location
	-
	-
	- Example: an application creates several threads,<br>
	such of which makes a call to the same library<br>
	butine:<br>
	This library routine accesses/modifies a global<br>
	structure or location in memory.<br>
	As each thread calls this routin incomposition. This library routine accesses/modifies<br>structure or location in memory.<br>As each thread calls this routine it is possible<br>may try to modify this global structur<br>location at the same time.<br>If the routine does

## Why PThreads (not processes)?<br>The primary motivation<br>To realize actoratial argamen acuteurosas spins Why PThreads (not processes)?<br>The primary motivation<br>-To realize potential program performance gains VVhy PThreads (not processes)?<br>The primary motivation<br>— To realize potential program performance gains<br>Compared to the cost of creating and managing a process

- **VVhy PI hreads (not promotive The primary motivation**<br>To realize potential program performance gain<br>Compared to the cost of creating and mar<br> $-A$  thread can be created with much less OS of<br>Managing threads requires fewer
- Why PThreads (not processes)?<br>• The primary motivation<br>—To realize potential program performance gains<br>• Compared to the cost of creating and managing a process<br>—A thread can be created with much less OS overhead Why PThreads (not processes)?<br>The primary motivation<br>—To realize potential program performance gains<br>Compared to the cost of creating and managing a process<br>—A thread can be created with much less OS overhead<br>Managing thre • The primary motivation<br>
– To realize potential program performance gains<br>
• Compared to the cost of creating and managing a process<br>
– A thread can be created with much less OS overhead<br>
• Managing threads requires fewer Inter-thread communication is more efficient and, in many cases, easier to use than inter-process communication is more efficient and, in many cases, easier to use than inter-process communication is more efficient and in
- Managing threads requires fewer system resources than
- 
- Compared to the cost or creating and managing a process<br>
 A thread can be created with much less OS overhead<br>
Managing threads requires fewer system resources than<br>
managing processes<br>
All threads within a process share t