TEXAS TECH UNIVERSITY

# CS4379: Parallel and Concurrent Programming
# CS5379: Parallel Processing

## Lecture 16


**Guest Lecture by Wei Zhang**

X-Spirit.Zhang@ttu.edu

**Dr. Yong Chen**

**Associate Professor**

**Computer Science Department**

**Texas Tech University**

# Course Info

- **Lecture Time**: TR, 12:30-1:50
- **Lecture Location**: ECE 217
- **Sessions**: CS4379-001, CS4379-002, CS5379-001, CS5379-D01
- **Instructor:** Yong Chen, Ph.D., Associate Professor
- **Email:** yong.chen@ttu.edu
- **Phone:** 806-834-0284
- **Office**: Engineering Center 315
- **Office Hours**: 2-4 p.m. on Wed., or by appointment
- **TA:** Mr. Ghazanfar Ali, Ghazanfar.Ali@ttu.edu
- **TA Office hours:** Tue. and Fri., 2-3 p.m., or by appointment
- **TA Office:** EC 201 A
- **More info:**
  - http://www.myweb.ttu.edu/yonchen
  - http://discl.cs.ttu.edu; http://cac.ttu.edu/; http://nsfcac.org

# Outline

- Questions?

- Overview of programming models and thread basics
- The POSIX Threads
- Synchronization Primitives in Pthreads
  - Mutual Exclusion for Shared Variables
  - Producer-Consumer problem
  - Condition Variables for Synchronization
  - Controlling Thread and Synchronization Attributes
  - Composite Synchronization Constructs
- Compiling and running Pthreads program

# Controlling Thread and Synchronization Attributes

- The Pthreads API allows a programmer to change the default attributes of entities using attributes objects.
- An attributes object is a data-structure that describes entity (thread, mutex, condition variable) properties.
- Once these properties are set, the attributes object can be passed to the method initializing the entity.
- Separates semantics and implementation
- Enhances modularity, readability, and ease of modification.

# Attribute Objects for Threads

- Use `pthread_attr_init` to create an attribute object.
- Individual properties associated with the attributes object can be changed using the following functions:

  `pthread_attr_setdetachstate`,
  `pthread_attr_setstacksize`,
  `pthread_attr_setinheritsched`,
  `pthread_attr_setschedpolicy`, and
  `pthread_attr_setschedparam`
- Provide fine control

# Attribute Objects for Mutexes

- Initialize the attributes object using function:

  `pthread_mutexattr_init.`

- The function `pthread_mutexattr_settype` can be used for setting the type of mutex specified by the mutex attributes object.

```
pthread_mutexattr_settype (
pthread_mutexattr_t *attr,
int type);
```

- Here, `type` specifies the type of the mutex and can take one of:
  - `PTHREAD_MUTEX_NORMAL_NP`
  - `PTHREAD_MUTEX_RECURSIVE_NP`
  - `PTHREAD_MUTEX_ERRORCHECK_NP`

# Composite Synchronization Constructs

- By design, Pthreads provide support for a basic set of operations.
- Higher level constructs can be built using basic synchronization constructs
- We discuss two such constructs - read-write locks and barriers.

# Read-Write Locks

- In many applications, a data structure is read frequently but written infrequently. For such applications, we should use read-write locks.
- Example: http://ianfinlayson.net/class/cpsc425/notes/08-read-write
- Initially unlocked, read/write lock can be obtained
- If read locked
  - Read locks granted, reader counter incremented
  - Write lock performs a condition wait, pending writer counter incremented
- If write locked
  - Read/write locks perform condition wait, pending writer counter incremented
- Reading: https://en.wikipedia.org/wiki/Readers–writer_lock

# Read-Write Locks

- The lock data type `mylib_rwlock_t` holds the following:
  - A count of the number of readers, writers
  - A condition variable `readers_proceed` that is signaled when readers can proceed,
  - A condition variable `writer_proceed` that is signaled when one of the writers can proceed,
  - A count `pending_writers` of pending writers, and
  - A mutex `read_write_lock` associated with the shared data structure

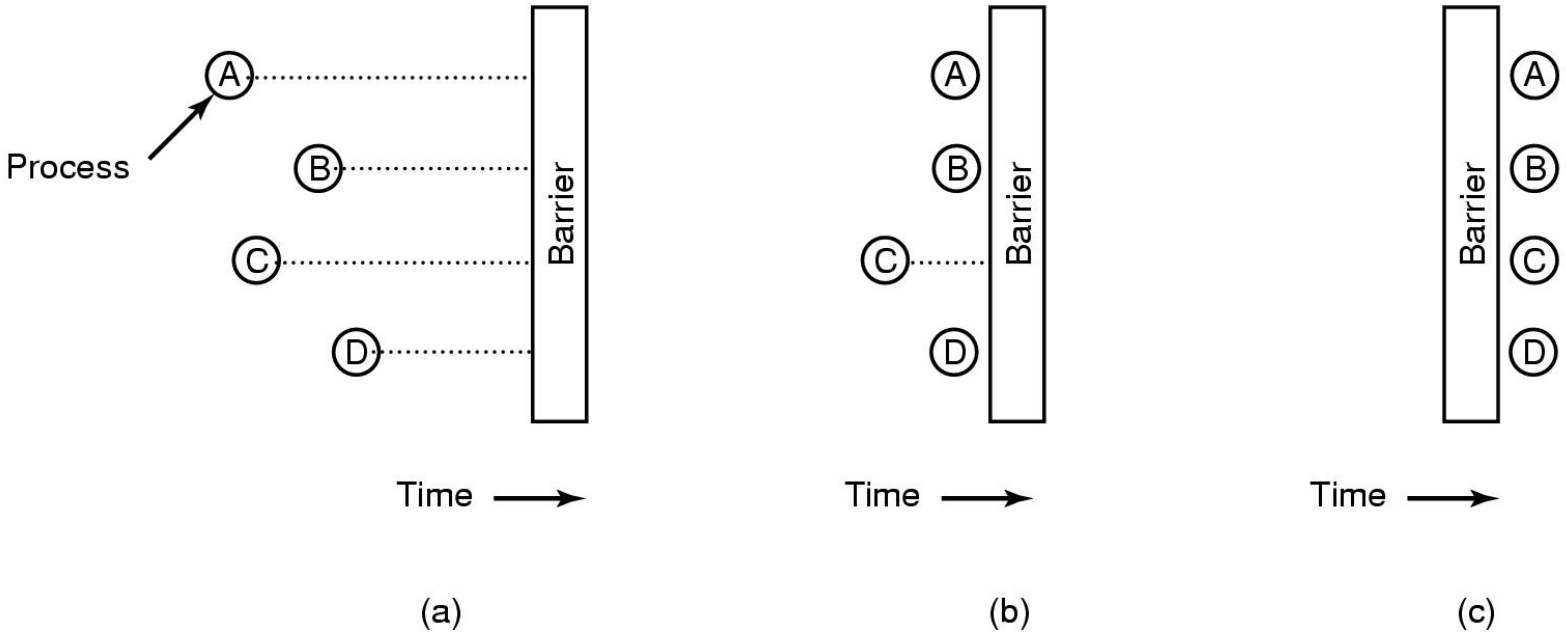
# Read-Write Locks

```
void mylib_rwlock_rlock(mylib_rwlock_t *l) {
     if there is a write lock or pending writers, perform
     condition wait.. else increment count of readers and grant
     read lock
}

void mylib_rwlock_wlock(mylib_rwlock_t *l) {
     if there are readers or writers, increment pending writers
     count and wait. On being woken, decrement pending writers
     count and increment writer count
}

void mylib_rwlock_unlock(mylib_rwlock_t *l) {
   if there is a write lock then unlock, else if there are read
   locks, decrement count of read locks. If the count is 0 and
   there is a pending writer, let it through, else if there are
   pending readers, let them all go through
```

# Barriers (1)

- One commonly used synchronization mechanism in IPC
- A process/thread is blocked at barrier till all processes/threads participating in the barrier reach it
- Useful for applications with phases and have the rule that no process/thread may proceed into the next phase until all processes/threads are ready to proceed
  - E.g. iterations of calculations with many time steps in scientific apps

# Barriers (2)




Use of a barrier. (a) Processes approaching a barrier.
(b) All processes but one blocked at the barrier.
(c) When the last process arrives at the barrier, all of them are let through.

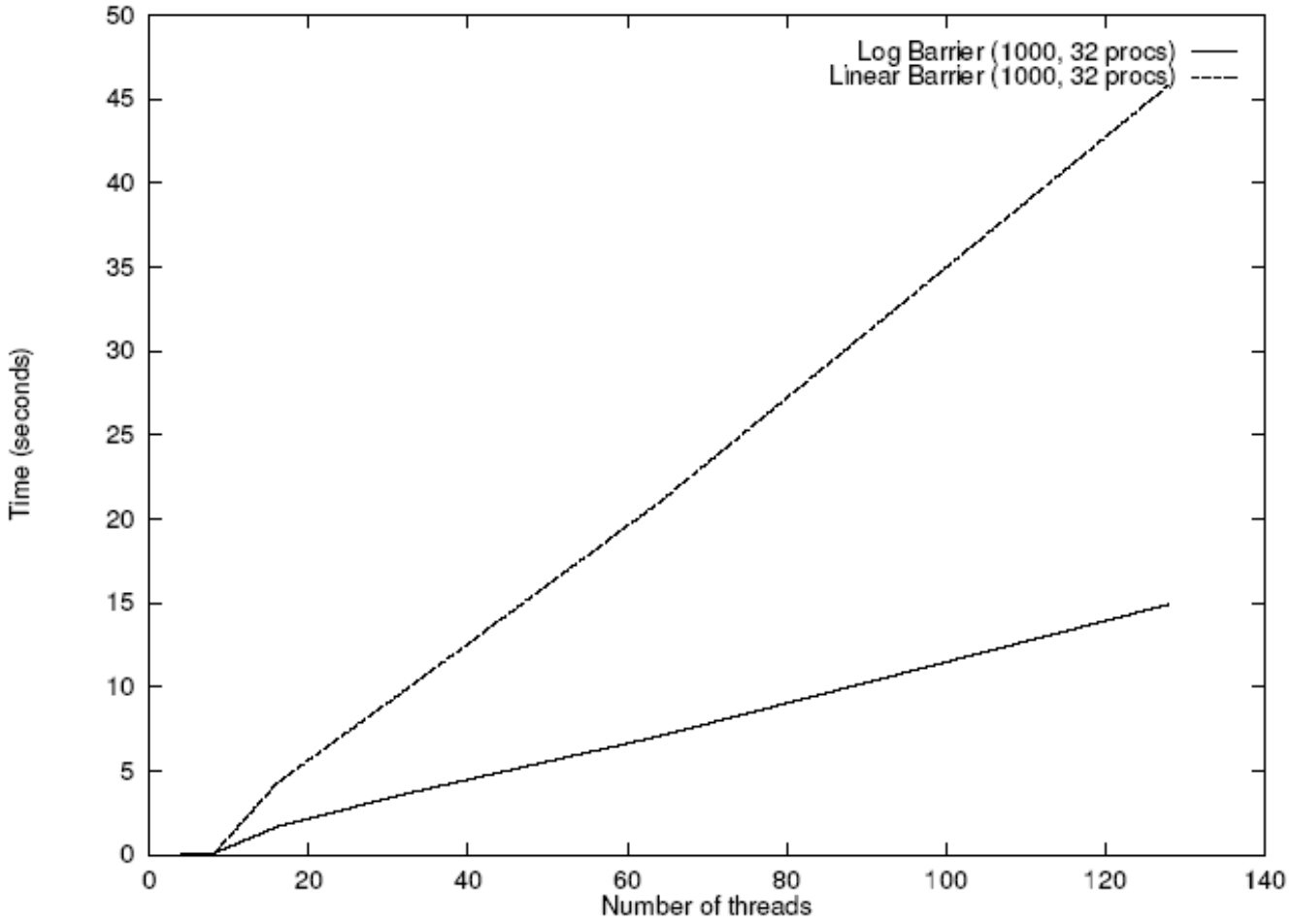
# Barriers Implementation: a Linear Barrier

- Barriers can be implemented using a counter, a mutex and a condition variable.
- A single integer is used to keep track of the number of threads that have reached the barrier.
- If the count is less than the total number of threads, the threads execute a condition wait.
- The last thread entering (and setting the count to the number of threads) wakes up all the threads using a condition broadcast.

# Barriers Implementation: a Logarithmic Barrier

- In the prior implementation, threads enter the barrier and stay until the broadcast signal releases them, and threads released one by one
- A logarithmic/tree barrier implementation pairs threads and uses n/2 condition variables, more efficient than a linear barrier
- See the reference book (ITPC) for detailed codes

# Log Barrier v.s. Linear Barrier




- Execution time of 1000 sequential and logarithmic barriers as a function of number of threads on a 32 processor SGI Origin 2000.

# Outline

- Questions?
- Overview of programming models and thread basics
- The POSIX Threads
- Synchronization Primitives in Pthreads
- Compiling and running Pthreads program

# Sample Source Codes

- Once you log in HPCC cluster, you can check out source codes with executing the following command:

git clone https://discl.cs.ttu.edu/gitlab/yongchen/cs4379cs5379.git

# Compiling Pthreads programs

- Load GNU Compiler module
  - `$ module load gnu7/7.3.0`
- Compiling
  - `$ gcc -lpthread` (or `g++ -lpthread`)
  - E.g. `$ gcc -Wall -o gauss_pthreads -O2 -lpthread gauss_pthreads.c`
  - Or, `$ gcc -o hello-pthread -lpthread hello-pthread.c`
- Automate compiling using Makefile
  - Edit a Makefile
  - `$ make`

# Makefile – Automating Compilation

- Makefile:
  - A simple way to organize code compilation
  - `$@` says to put the output of the compilation in the file named on the left side of the :
  - `$^` means right side of :
  - `clean:` tells make to clean the object files

constant

```
CC=gcc
OBJ=gauss_pthreads.o

%.o:%.c
	$(CC) -c -o $@ $^

gauss_pthreads:$(OBJ)
	$(CC) -o $@ $^
clean:
	rm -f *.o
```

tab

# Makefile – Automating Compilation

- For how to write a Makefile, such as write rules and targets, please see:
  - http://www.gnu.org/software/make/manual/make.pdf or
  - http://www.gnu.org/software/make/manual/html_node/index.html
  - https://makefiletutorial.com
  - https://www.tutorialspoint.com/makefile/index.htm
- More info: http://www.gnu.org/software/make/

# Debugging Pthread Program

- Prerequisite: set –g option for gcc
- GDB/CGDB: (-ggdb)

```
set print thread-events    # prints out thread start and exit events
info threads               # list all existing threads in program
                           # the gdb threadno is the first value listed
                           # the thread that hit the break point is *'ed
thread threadno            # switch to thread threadno's context
                           # (see its stack when type where, for example)
break [where] thread [threadno] # set a breakpoint at [where] just for
                                # thread threadno

# apply the gdb command to all or a subset of threads
thread apply [threadno|all] command
```

# Running Pthreads programs

- Create job submission script, e.g.

```
#!/bin/sh
#$ -V
#$ -cwd
#$ -S /bin/bash
#$ -N Pthreads_Test_Job
#$ -o $JOB_NAME.o$JOB_ID
#$ -e $JOB_NAME.e$JOB_ID
#$ -q omni
#$ -pe sm 36
#$ -l h_vmem=5.3G
#$ -l h_rt=48:00:00
#$ -P quanah

./hello-pthreads
```

- Submit job
  - **qsub** <job submission script>
- Check job status
  - Command: **qstat**

# Checking Output and Debugging Failed Jobs

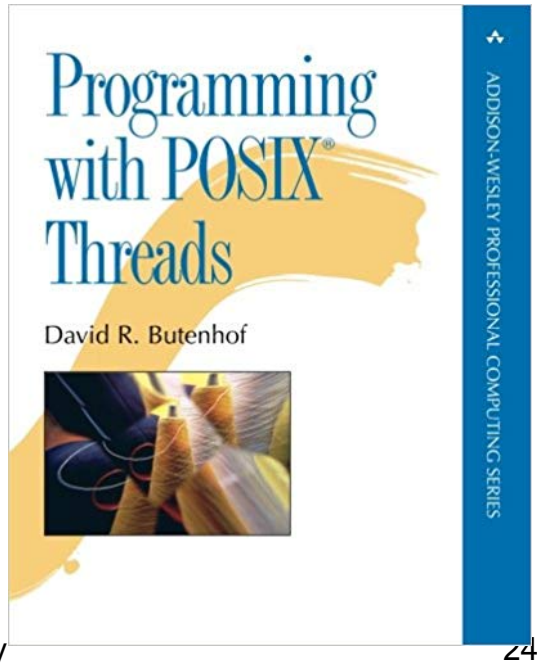
- Job output
  - Standard: $JOB_NAME.o$JOB_ID
  - Error: $JOB_NAME.e$JOB_ID

- When debugging:
  - Check the output files for errors
  - Check the output of qacct –j <job_ID>
    - failed
    - exit_status
    - maxvmem
    - start_time & end_time (<runtime limit)
    - low

# More about Pthreads

- IEEE Std 1003.1, 2004 Edition
- http://pubs.opengroup.org/onlinepubs/009695399/basedefs/pthread.h.html

- "Programming with POSIX Threads", by David R. Butenhof, ISBN-10: 0201633922, ISBN-13: 978-0201633924



# Readings

- Reference book ITPC – Chapter 7
- POSIX Threads Programming, by Blaise Barney, Lawrence Livermore National Laboratory: https://computing.llnl.gov/tutorials/pthreads/

# Questions?

## Questions/Suggestions/Comments are always welcome!

Write me: yong.chen@ttu.edu
Call me: 806-834-0284
See me: ENGCTR 315

*If you write me an email for this class, please start the email subject with [CS4379] or [CS5379].*