Software Construction Laboratory

Week 6 Part 1

Lab 3

Mushi Zhou

Winter 2017

UCLA

Prof. Paul Eggert

Some Help for Assignment 5

For Lab 6,

- Compare the performance of getchar/putchar vs read/write
- Read/write are system calls, i.e. they are unbuffered if you make these calls directly, but you can specify how many bytes to read/write with one call
- Getchar/putchar are buffered so that they read/write everything with one read/write system call to a temporary buffer, and then read/write byte by byte from that buffer

Strace

- Strace, as we have mentioned before, is a debugging tool, that can monitor interactions between processes and Linux kernel, i.e. system calls
- Just do *strace programname* (executable, not your source file)
- By default, strace output all system calls made
- Options:
- -e to show only selected system calls, exp: strace –e open ls
- - t print time stamp of system calls
- - r print relative time of execution
- - c show a statistic report summary

Some Help for Assignment 5

For HW,

- Modify your HW5 to use system calls instead
- Adding the non-case sensitivity option ‘-f’ to comparisons
- Note: the topper() takes in unsigned characters, but the default char is signed, so you’ll have to shift the value before changing to upper case
- Compare execution time of your two HWs in regards to number of comparisons
- What one do you expect to run faster?

Outline for Week 6

- Threads vs Processes
- Multithread Performance
- POSIX Threads
- Thread Synchronizations

What are Process and Thread

Recall from last week:

- A process (or a task) is an executing (i.e., running) instance of a program
- A thread of execution (thread) is the smallest sequence of programmed instructions that can be managed independently in a process
- In most cases a thread is a component of a process
- You can think of threads as child processes that share the parent process resources but execute independently

Multiprocessing vs Multithreading

- These are different!
- Multiprocessing -> in hardware level, if there are more than one processor in your CPU, they are run <u>parallelly </u>for multiple processes
- Having multiple programs running at the same time = *multitasking*
- Multithreading -> is an execution model that allows a single process to have multiple code segments (i.e., *threads*) run <u>concurrently</u> within the context of that process
- Multiple threads of a single process can share the CPU in a single CPU system or (purely) run in parallel in a multiprocessing system

Why Do We Want Multithreading?

- It is necessary!
- For example: You have a GUI application. You request a calculation that needs a fair mount of time to compute
- In a single thread mode, your GUI is going to be unresponsive until that calculation is done. You can perform nothing on the GUI during the period
- If the application is a multithreaded environment, that calculation becomes a thread, and you can perform other tasks while that calculation is running

Multithreaded Performance

- Multithreading makes processing time much sorter!
- You will verify this in assignment 6
- How much faster depends on *implementation* and *the natural of the problem*

- Natural of the problem:

  From embarrassingly parallel

  To inherently serial

  Most of problem fall in between the range

Embarrassingly Parallel Problem

- little or no effort is needed to separate the problem into a number of parallel tasks. The sub-tasks are neutrally parallel.
- For example : a = ? b = ?

  5s a = c + d → These can be done parallelly in

  5s b = e + f → different threads

- The execution time can be reduced from 10s to 5s by computing a and b in parallel
- In this type of problem, multithreading can drastically reduce the processing time

Inherently Serial Problem

- For example: $a + b$?

  $a = c + d$

  $b = a + e$

- It is not easy to parallel these since b depends on a that is been computed
- The multithreaded execution on this type of problem is not likely to yield much faster outcome

How to Achieve Multithreading in a C program?

- POSIX Threads (Pthreads), is an execution model that allows multithreading
- It allows a program to control multiple different flows of work that overlap in time
- Each flow of work is referred to as a thread
- Creation and control over these flows is achieved by making calls to the POSIX Threads API
- When compiling the program, use –lpthread option to link the Pthread libraries

Basic Steps to Use POSIX Threads

- Include <pthread.h>
- Use pthread_create and pthread_join calls to Pthread API

- pthread_create (thread, attr, start_routine, arg)
- This creates a new thread within a process
- The new thread execute the start_routine function with the input argument arg
- The new thread has attributes specified with attr, or default attributes if attr is NULL
- If the pthread_create() routine succeeds it will return 0 and put the new thread id into thread, otherwise an error number shall be returned indicating the error

Basic Steps to Use POSIX Threads

- Function: pthread_join()
- int pthread_join(pthread_t thread, void ** status);

- This function suspends execution of the current process and waits for the target thread to terminate
- On a successful call pthread_join() will return 0
- On failure pthread_join() will return an error number indicating the error

- You only need to use pthread_join and pthread_create for assignment 6

Check Error

There are returned values from the function call

For Pthread_create:

- EAGAIN: The process lacks the resources to create another thread, or the total number of threads in a process would exceed PTHREAD_THREADS_MAX.
- EINVAL: *attr* is invalid.

For Pthread_join

- EINVAL: The value specified by the thread is not a valid thread
- ESRCH: The specified thread is already detached
- EDEADLK: The join would result in a deadlock or the value of thread specifies the calling thread

What Could Go Wrong in Multithreading

- *a is not defined initially
- Thread A -> *a = 5
- Thread B -> *a = 6
- Thread C -> printf(“a = %d”, *a)

- Assume threads a,b,c are started at the same time, what is printed?
- There are three possibilities
- This is called race condition!

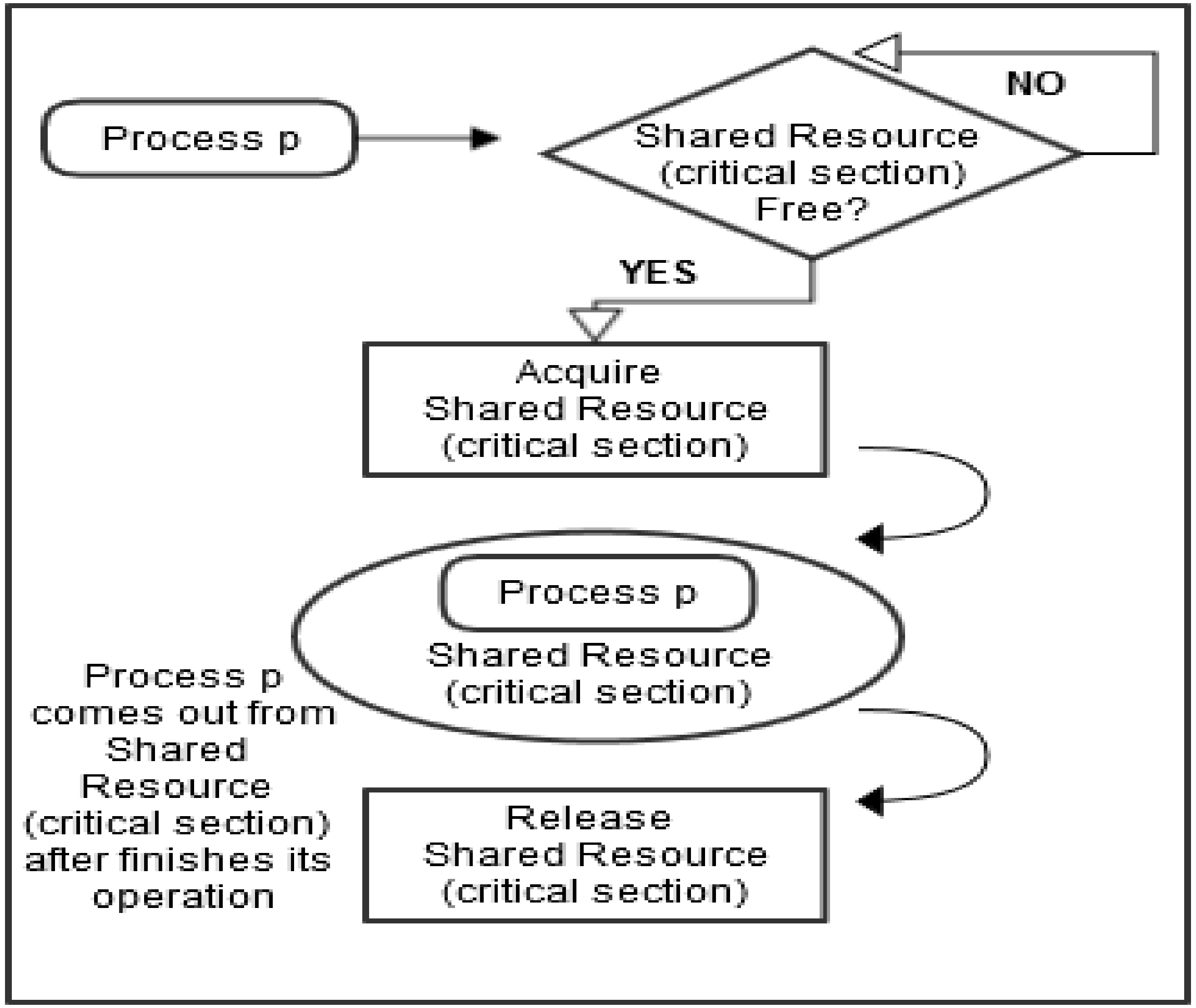
Race Condition

- A **race condition** is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly
- In other words, it is the behavior of an program where the output is dependent on the sequence or timing of other uncontrollable events
- So we need to deal with this kind of problem
- Sychronization

Synchronization in Multithreading

- A mechanism which ensures that two or more concurrent threads do not simultaneously execute some particular program segment known as critical section
- Processes' access to critical section is controlled by using synchronization techniques
- When one thread starts executing the critical section (serialized segment of the program) the other thread should wait until the first thread finishes

Illustration of Synchronization


Process p
Shared Resource (critical section) Free?
NO
YES
Acquire Shared Resource (critical section)
Process p
Shared Resource (critical section)
Process p comes out from Shared Resource (critical section) after finishes its operation
Release Shared Resource (critical section)

The action of getting to the shared resource is commonly referred as “acquiring the lock” of the resource

More on Synchronization

- Synchronization needs to implemented carefully so that it does not lead to deadlock -> where all threads are looping forever waiting to acquire the lock to the shared resource where the lock is actually held by one of the waiting thread

- Pthread API provides synchronization by:
- Function: pthread_mutex_lock()
- Function: pthread_mutex_trylock()
- Function: pthread_mutex_unlock()
- Etc.

More Resources on Multithreading

- Pthread API

http://www.cs.wm.edu/wmpthreads.html

- Common Synchronization Problems

https://en.wikipedia.org/wiki/Synchronization_(computer_science)

- Multithreading vs Multiprocessing vs Multitasking

https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/multiprogramming-multiprocessing-multitasking-multithreading/