CSE451 Section 3

10/11/07

Aziel Epilepsia
Important dates

• Oct 12, Fri: Project 1 due – 10am
  – Project 2 will be posted this day also
• Oct 15, Mon: HW3 due in class
• Oct 24, Wed: Project 2 part 1 (thread scheduler) due
• Oct 25, Thurs: Midterm review
  – Would students prefer this review in section or in a separate event?
• Oct 26, Fri: Midterm!
Goals of today

• Hand back HW 2
• Review commonly missed problems in HW 2
• Review Memory Leaks (question asked from HW1)
• Pthreads
• Sockets
HW2 Summary

• Average was 9.8 out of 12
• Many points were missed due to inefficient code:
  – i.e. 16-part case statement to handle hex conversion
  – Expectation is to be able to see where code can be optimized by less specific routines
Problematic Problems

• These two problems from the last homework were the most missed:
  • 3.4: (Read program and explain what output will be at Line A)
  • 3.6: (Fibonacci sequence)
Problem 3.4

#include <sys/types.h>
...
int value = 5;
int main()
{
pid_t pid;
    pid = fork();
    if (pid == 0){ /*Child process*/
        value += 15;
    }
else if (pid > 0){ /*parent process */
        wait(NULL);
        printf("PARENT: value = %d",value); /*LINE A*/
        exit(0);
    }
}

• Common mistake is not realizing fork() gives child process a complete copy of parent’s memory. The global variable is NOT shared.
• Result at Line A is that “5” is printed out. Parent’s memory space is independent of the child’s memory space.
Problem 3.6

• Problems with Fibonacci sequence code not compiling or running.
• fork() and wait() were implemented correctly, but many programs did not compile or run as expected.
• Problem asked for students to do error checking on args from command line, some avoided this portion.
• Some didn’t write code that prints the Fibonacci sequence.
Notes from Marissa

• Please submit all homework code, in the future via the online turn-in described in the homework posting.
Topic requested by student

• Memory leaks

• Several students missed the memory leak occurring in queue.c -> queue_remove().

• Whenever memory is allocated, a free() command must be executed on dereferenced data structures.
  – A simple pointer change does not free memory!
Memory leaks

• A memory leak occurs when a program does not free allocated memory that is no longer needed.

• Result is that resources get consumed as program continues executing, and instantiating data structures.

• Memory is a finite resource – when it runs out program will terminate itself or memory segmentation fault.
free()

• free() deallocates memory in the space pointed to by a pointer.

• Result is that memory is made available for future program/data use.

• How to check if free occurred correctly? You can’t read data in that address using gdb.
boolean_t
queue_remove(queue_t q, queue_element_t *e)
{
    queue_link_t oldHead;

    assert(q != NULL);
    if (queue_is_empty(q))
        return FALSE;

    *e = q->head->e;

    oldHead = q->head;
    q->head = q->head->next;
    free(oldHead);
    return TRUE;
}
free() summary

• Pay attention to about your malloc() and free() routines.

• When you are removing a pointer or changing its address to NULL, free the memory it points to first.

• Use gdb to ensure that pointers are being freed properly.
Pthreads and project 2 brief

- Project 2 will involve creating a thread scheduler
- HW3 involves writing a program using the Pthreads API.

- Pthreads refers to the POSIX standard (IEEE 1003.1) defining an API for thread creation and synchronization.
  - It’s a specification for thread behavior
  - Not an implementation (we’ll need to make the implementation)
Pthreads (2)

- All Pthreads programs must include a pthread.h header file.
  - A variant of this will be given to you for project2.
- When compiling Pthreads programs, use the `-pthread` flag in gcc

Some examples from the API:

- Types available in API:
  - `pthread_t tid; // Thread identifier`
  - `pthread_attr_t attr; // Thread attributes`

- Functions available:
  - `pthread_attr_init(&attr); // get default attributes for thread`
  - `pthread_create(); // create the thread`
  - `pthread_join(); // wait for thread to exit`
  - `pthread_exit`
Threads project

• You will need to implement the following:
  – Data structures to represent threads
  – Routine to initialize the data structures
  – Thread creation routine
  – Thread destruction routine
  – Mechanism for a thread to yield, letting another thread run
  – Mechanism for a thread to wait for another to finish
  – Simple non-preemptive thread scheduler

• All these structures and functions are defined in the Pthreads API, we will need to implement them to meet the spec of the project.
Problem 4.9 involves writing a thread program that outputs prime numbers to the screen.

A separate thread will be responsible for calculating and handling the output.

More details on Pthreads?

- man pthreads
Sockets

- Sockets are endpoints for communication.
- Sockets are identified by an IP address concatenated with a port#: 
  - i.e. 127.0.0.1:80
- Uses a client-server architecture
  - Client sends request for connection to server port
  - Server accepts request and completes connection

- Processes communicating over a network possess one socket each.

- All connections consist of a unique pair of sockets.
Communicating via sockets

- A client communicates with the server by creating a socket and connecting to the server’s port.

- Once connection is made, client can read from socket using normal stream I/O statements.
Socket implementation references

• For more details:

• General details: man socket

• Implementation: see sys/socket.h

• GNU C Library Socket Tutorial
  – http://www.cs.utah.edu/dept/old/texinfo/glibc-manual-0.02/library_15.html