LECTURE 09/08

RECAP

PROJECT GOAL

- Identify Performance bottleneck (contention bug) in large codebase
- Use an existing solution in large codebase

C- TYPE SYSTEM

```
Typedef [type] [alias]

Typedef struct Point {
      Int x;
      Int y;
 } Point;

Point p;
p1.x = 0;
p2.x = 1;
```

C- MEMORY MANAGEMENT

Malloc/ Free

Malloc([# of bytes])

Free([pointer])

char* x = malloc(3);

Free(x);

C- METAPROGRAMMING

Macro

#[keywords]

#define HEADER

#Ifndef HEADER

#endif

ATOMIC OPERATIONS

- This project uses build in GNU atomic operations rather than C++ 11 <atomic> library
- However, they are almost the same.

ADD FETCH

- __atomic_add_fetch(*type* **ptr, type* *val, int memorder*)

  { *ptr += val; return *ptr; } (done atomically)

e.g:

```
Int x = 0;
__atomic_add_fetch(&x, 1, __ATOMIC_SEQ_CST) ;
Print(x) // 1
```

EXCHANGE

- __atomic_exchange_n (type *ptr, type val, int memorder)
    - {type old = *ptr; *ptr=val; return old;}

e.g

Int x = 0;

Int y = __atomic_exchange_n (&x, 1, __ATOMIC_SEQ_CST)

Print(x) // 1

Print(y) // 0

SPIN LOCK

```
Int locked = 0 // 0 is false, 1 is true
Void lock()
{
   While(__atomic_exchange_n(&locked, 1, __ATOMIC_SEQ_CST))
       While(*locked)
 }

Void unlock()
{
   While(__atomic_exchange_n(&locked, 0, __ATOMIC_SEQ_CST))
}
```

SET UP

- Remote access to CADE with GUI: https://nx.eng.utah.edu
- Physical Location: WEB cade lab floor-2
- SSH (No GUI)

- I use VScode + Clangd
- Open VScode on cade, go to extension and search and install Clangd
- At your project directory, run: bear -- make all

DEMO TIME

Introduction to Basic Tools

Make

Make is a tool that controls generation of executables from source files

With GCC, we might have a long list of commands with many arguments to compile our program:

```
        gcc -g -flto -Ofast -Wall -march=native -pthread  -I ./include -c src/iceberg_table.c -o obj/iceberg_table.o
        gcc -g -flto -Ofast -Wall -march=native -pthread  -I ./include -c src/partitioned_counter.c -o obj/partitioned_counter.o
        g++ -g -flto -Ofast -Wall -march=native -pthread  obj/iceberg_table.o obj/hashutil.o obj/partitioned_counter.o obj/lock.o obj/main.o -o main -lssl
-lcrypto
```

This is hard to remember! It’s also easy to make mistakes!

Make and Makefiles

A **Makefile** lets you define how a project should be built

It can have variables, use environment variables, handle different systems, and bundle together a massive number of files easily.

Once the file is written, all you need to do is evoke a single command to build the whole project:

make

Make Sidenote

Make is a very popular solution for building projects as it is designed to execute shell commands in order, you can experiment with Make for more than C and C++ projects!

However, there are other options and out there to build and deploy software depending on the language.

If you are using C++ and C for your own personal projects, consider using CMake which can even build Makefiles for your project regardless of system and libraries you have installed

Project number one provides a Makefile for you, this slide is for your information only

PERF

Perf is a Linux command/program that is designed to instrument CPU performance.

It is typically included in the Linux kernel, and is installed already on CADE machines

Perf is an event-oriented tool to help you troubleshoot your programs, particularly from a speed standpoint

Some questions you can investigate:

- Where are performance hotspots in my code?
- Where are the most cache misses?
- Why is the kernel using so much compute time? What kernel functions are evoked?
- Is CPU stalled on I/O?

Perf thoughts adapted from P.J. Drongowski: http://sandsoftwaresound.net/perf/perf-tutorial-hot-spots/

PERF

For this lab, we’ll be using perf to investigate the hotspots in the code so that we can implement fixes

The more important perf options we’ll use are for tracking **cpu-clock** and perhaps **cpu-cycles**

Perf is done in two stages:

```
perf stat -e cpu-clock [executable] [program options]

perf report
```

Key questions to ask while using Perf

What parts of the program take the most execution time?

Do the number of software or hardware events indicate an actual performance issue to be fixed?

How can we fix the performance issue?

Perf will not give you the answer, you need to be a detective!