Modern Computer Architecture

Lecture 9
Multi-Core: pthreads, OpenMP
Multi- and Many-Core Systems I
  • Tightly Coupled
  • Shared Memory
  • Multi-Threading
  • Example: pthreads, OpenMP

Multi- and Many-Core Systems II
  • Loosely Coupled
  • No Shared Memory
  • Message Passing
  • Example: Unix Processes, MPI, Cloud

Not part of this Lecture.
See, e.g., Parallel Programming
Thread
- represents independent control flow
In hardware:
  - Program Counter
  - Registers
  - Thread Local Storage (TLS)
    - __thread int variable;
    - __declspec(thread) int variable; (Visual Studio)
  - State for communication (signals, semaphores, etc.)

Process
- represents a program
  managed by OS:
  - memory map
  - heap
  - table of open files
  - table of open network sockets
  - shared libraries
  - environment variables
  - security credentials (user name etc.)

traditionally: each process had one thread
today: multiple parallel threads
Unix Processes

• Creation of processes is expensive
• A lot of state has to be duplicated
• Happens e.g. in web-server
• Cumbersome for different processes to communicate
• Slow to check security credentials at inter-process communication

• many programs create threads as “light-weight processes”
• problem: many library functions were not designed to run in multi-threaded environment (e.g. use global variables)
• property: “thread-safe”, usually pass global state in additional parameters

```c
[one process]
ppid = fork ();
if (ppid < 0) {
    my_handle_error ();
} else if (ppid == 0) {
    [child process]
    my_child_function ();
} else {
    [parent process]
    my_parent_function ();
}
```
Posix Thread API

- `pthread_create (thread, attr, start_routine, arg)`
- `pthread_exit (status)`
- `pthread_cancel (thread)`
- `pthread_attr_init (attr)`
- `pthread_attr_destroy (attr)`
Thread Attributes

- Detached or joinable state
- Scheduling inheritance
- Scheduling policy
- Scheduling parameters
- Scheduling contention scope
- Stack size
- Stack address
- Stack guard (overflow) size

Linux allows to use different scheduling policies:
- maximum throughout (server)
- minimum latency (desktop PC)
- real-time programs
• First In – First Out (FIFO), First-Come First-Served (FCFS)
• Shortest-Job-Next (SJN), Shortest Job First (SJF), Shortest Processing Time (SPT)
• Shortest-Remaining-Time (SRT) oder Shortest-Remaining-Processing-Time (SRPT).
• Earliest Due Date (EDD), Earliest Deadline First (EDF)
• Priority Scheduling
  – Rate Monotonic Scheduling (RMS)
  – Deadline Monotonic Scheduling (DMS)
  – Multilevel Feedback Queue Scheduling, Shortest-Elapsed-Time (SET)
• Round Robin, Time Slice, Weighted Round Robin (WRR)
```c
#include <pthread.h>
#include <stdio.h>

#define NUM_THREADS 5

void *
PrintHello(void *threadid)
{
    long tid; tid = (long)threadid;
    printf("Hello World! It's me, thread #%ld!\n", tid);
    pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc; long t;
    for(t=0; t<NUM_THREADS; t++) {
        printf("In main: creating thread %ld\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello,
                            (void *)t);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    /* Last thing that main() should do */
    pthread_exit(NULL);
}
```
Thread Joining and Detaching

- `pthread_join (threadid,status)`
  Waits for child processes to exit.

- `pthread_detach (threadid)`
  Note: it’s not possible to re-attach a detached thread.

For newly created threads:
- `pthread_attr_setdetachstate (attr,detachstate)`
- `pthread_attr_getdetachstate (attr,detachstate)`
Pthreads Stack Management

- `pthread_attr_getstacksize (attr, stacksize)`
- `pthread_attr_setstacksize (attr, stacksize)`
- `pthread_attr_getstackaddr (attr, stackaddr)`
- `pthread_attr_setstackaddr (attr, stackaddr)`
Mutex Variables

- `pthread_mutex_init (mutex,attr)`
- `pthread_mutex_destroy (mutex)`
- `pthread_mutex_lock (mutex)`
- `pthread_mutex_trylock (mutex)`
- `pthread_mutex_unlock (mutex)`
GDB:

- info threads

  3 process 35 thread 27 0x34e5 in sigpause ()
  2 process 35 thread 23 0x34e5 in sigpause ()
  * 1 process 35 thread 13 main (argc=1, argv=0x7fffffff8)
      at threadtest.c:68

- break **filename**:line thread **number**

- Challenge: reproduce sequence of events that lead to program failure
- “Parallel Chaos”
• Fast Thread Creation and Synchronization
• Variable number of threads

#define N 100000
int main(int argc, char *argv[])
{
    int i, a[N];
    #pragma omp parallel for
    for (i = 0; i < N; i++) {
        a[i] = 2 * i;
    }
    return 0;
}
```c
#include <omp.h>

int main(void)
{
    int id, i;
    omp_set_num_threads(4);

    #pragma omp parallel for private(id)
    for (i = 0; i < 4; ++i) {
        id = omp_get_thread_num();

        printf("Hello World from thread %d\n", id);
    }

    #pragma omp barrier
    if (id == 0) {
        printf("There are %d threads\n", omp_get_num_threads());
    }

    return 0;
}
```
$ gcc-4.2 -fopenmp -o pg pg.c
$ ./pg

Hello World from thread 3
Hello World from thread 0
Hello World from thread 1
Hello World from thread 2

There are 4 threads

#pragma omp parallel
{
    ...  
}

arbitrary order

barrier
```c
#pragma omp parallel
{
    #pragma omp for private(j)
    for(i=0; i<100; i++)
    {
        for(j=0; j<10; j++)
        {
            T[omp_get_thread_num()]++;
            #pragma omp critical
            NumOfIters++;
            #pragma omp end critical
        }
    }
}
```
```c
#pragma omp parallel
{
    #pragma omp for private(j) reduction(+: NumOfIters)
    for(i=0; i<100; i++)
    {
        for(j=0; j<10; j++)
        {
            T[omp_get_thread_num()]++;
            NumOfIters++;
        }
    }
}
```
Performance Optimization

Software Development Relevant for
• Game Consoles
• Creative Applications
• Super-Computers

Architectural Template
• Modern super-scalar pipeline
• Memory hierarchy
• SIMD extensions (MMX, SSE, AVX)
• GPU Parallelism (CUDA, OpenCL)
• Shared-Memory Parallelism (OpenMP, pthreads)
• Message Passing Parallelism (MPI, MapReduce)