Computer Networks(CS39006)
Spring Semester (2021-2022)

Lab 4

Prof. Sudip Misra
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Email: smisra@sit.iitkgp.ernet.in
Website: http://cse.iitkgp.ac.in/~smisra/
Research Lab: cse.iitkgp.ac.in/~smisra/swan/

Threading

- Threads are lightweight processes, unlike forking that creates an entire new process (heavyweight), threads are more lightweight **independent processing** units.
- Threading is used when parallelism in code is required - you have to perform multiple tasks simultaneously.
- Performance of threading is improved with more number of physical processors - you achieve parallelism at physical level.
- Threads spawned from a process share data structures such as code section, data section and OS resources with the parent process.
- Only the execution logic is different.
- Context switching between threads much faster
- Level of multithreading also depends on the OS support.
- **Example:-** Your Internet browser is a process but each individual tabs can be handled by threads.

pthreads() - threading in C

- The pthread() library provides you with the user level functions for multithreading in C (POSIX framework for Linux)
  - *pthread_create() - create and define new threads*
  - *pthread_join() - master process waits for all child threads to finish*
  - *pthread_exit() - exit from a thread*
- Various data types and functions based on threading defined on <pthread.h> library
- **pthread_create (thread_id, thread_attr, start_routine, arg)**
  - *thread_id - id of the thread*
  - *thread_attr - attribute of the thread, 'NULL' by default*
  - *start_routine - the function which the thread will execute upon being called*
  - *arg - any argument that you may want to pass to the routine.*

pthreads() - threading in C

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#DEFINE NUM 5
// A normal C function that is executed as a thread
// when its name is specified in pthread_create()
void *Fun(void *tid)
{
    sleep(1);
    printf("This is a thread with id %d\n", (int)tid);
    pthread_exit(NULL);
}

int main()
{
    pthread_t thread_id[NUM];
    printf("Printing before thread\n");
    for (int i=0; i<5; i++){
        pthread_create(&thread_id[i], NULL, Fun, (void *)i); // Call the thread to be created

    }
    pthread_join(thread_id[NUM-1], NULL); //Master thread waits for the thread with id thread_id[NUM-1] to complete
    printf("Printing after thread\n");
    return 0;
}
```

Mutex

- When multiple threads/processes share some common resources, synchronization problem arises.
- Critical resources must not be modified by more than two threads/processes simultaneously
- Mutex locks ensure that synchronization is maintained by competing threads.
- **Very important** to consider synchronization problems during multithreading.
  - *pthread_mutex_t <name> - binary mutex provided by pthread library*
  - *pthread_mutex_lock(&lock) - locks the mutex named 'lock'*
  - *pthread_mutex_unlock(&lock) - unlocks the mutex named 'lock'*

Mutex

```c
pthread_mutex_t lock;
int val;

void some_function()
{
    //Critical section starts
    pthread_mutex_lock(&lock); //lock the mutex before entering critical section
    int i = 0;

    val++; // NOTE that this 'j' is a global variable shared by ALL thye threads
        //This is where synchronization problem arises
    while(i <= 5)
    {
        printf("%d", val);
        sleep(1);

        i++;
    }
    printf("Okay\n");
    pthread_mutex_unlock(&lock); //unlock the mutex before entering critical section
    //Critical section ends
}
```

Thank You!!!