# CSE410 Project 4
# Spring 2018

03/16/2018

- 100 points
- Deadline
  - **Thursday, April 5, 2018 at 11:59 pm**
  - Handin
    - http://secure.cse.msu.edu/handin/
- Compile & Run on “cse410”
  -

# Tasks

- Assignment Notes #2
  - DotProduct::GenerateValues()
  - DotProduct::Producer()
  - DotProduct::Consumer()
  - DotProduct::~DotProduct()
  - DotProduct::NormalDot()
  - DotProduct::MultiProcessInitialize()
  - DotProduct::MultiProcessDot()
  - DotProduct::ProcessDotOperation()
  - DotProduct::MultiThreadDot()
  - DotProduct::ThreadEntry()
  - DotProduct::ThreadDotOperation()
  - DotProduct::Print()

# Work flow

- Generate numbers for 2 vectors
  - Producer*1 (generates one value at a time)
  - Consumers*2 (consumes one value at a time)
- Do “dot product” operations
  - Single Process/Thread
  - Multi-thread (2 threads)
  - Multi-process (2 processes)
- Print result on screen

# “dot product”

- 2 vectors
  - A = [ 1, 2, 3, 4, 5, 6]
  - B = [ 2, 3, 4, 5, 6, 7]
- A·B
  - (1*2 + 2*3 + 3*4 + 4*5 + 5*6 + 6*7)
  - 2 + 6 + 12 + 20 + 30 + 42
  - 112

# Generate numbers

- Ranges: 0 – 100
- Producer*1 (in 1 thread)
  - Only produces a new value when the value generated previously was consumed by either of the consumers
- Consumer*2 (in 2 threads)
  - Only consumes a value whenever there is a value produced by the producer

- Functions
  - Condition Variable (for synchronizing the Producer and Consumers)
    - pthread_cond_init()
    - pthread_cond_wait()
    - pthread_cond_signal()
    - pthread_cond_destroy()
  - Mutex (for multi-thread “dot product” operation)
    - pthread_mutex_init()
    - pthread_mutex_lock()
    - pthread_mutex_unlock()
    - pthread_mutex_destroy()
  - Thread
    - pthread_create()
    - pthread_exit()
    - pthread_join()
- Variables (DotProduct.h)
  - std::vector<int> **mGeneratedNumber** ← this is what to be protected by mutex
  - int mNumberOfValuesPerVector
  - int mTotalNumberOfValues
  - pthread_cond_t mNotEmpty ← condition variable
  - pthread_cond_t mEmpty ← condition variable
  - pthread_mutex_t mMutex1 ← the mutex you have to use
  - std::vector< std::vector<int> > mVectors ← vectors storing generated values

# “dot product” operations

- Single Process/Thread
- Multi-thread (2 threads)
  - Functions
    - Mutex (for multi-thread “dot product” operation)
      - pthread_mutex_init()
      - pthread_mutex_lock()
      - pthread_mutex_unlock()
      - pthread_mutex_destroy()
    - Thread
      - pthread_create()
      - pthread_exit()
      - pthread_join()
  - Variable (DotProduct.h)
    - int* **mProduct** ← this is what to be protected by mutex
    - pthread_mutex_t mMutex2 ← the mutex you have to use

- Multi-process (2 processes)
  - Functions
    - Shared Memory (for mProduct and mSem)
      - shmget()
      - shmat()
      - shmdt()
      - shmctl()
    - Semaphore (for multi-process “dot product” operation)
      - sem_init()
      - sem_wait()
      - sem_post()
      - sem_destroy()
    - Process
      - fork()
      - exit()
      - waitpid()
  - Variables
    - int* **mProduct** ← this is what to be protected by semaphore
    - sem_t* mSem ← the semaphore you have to use
    - unsigned int mShmProductId ← id returned from shmget()
    - unsigned int mShmSemId ← id returned from shmget()

# pthread

```
#include <pthread.h>
#include <stdio.h>

static void *ThreadEntry(void *arg)
{
     ClassName* obj = (ClassName*)arg;
     obj->ThreadOpeartion(); // invoke pthread_exit(NULL) in ThreadOpeartion()
     return NULL;
}

void MultiThread()
{
     pthread_t threads[2];
     for(unsigned int i = 0; i<2; i++)
     {
          int create = pthread_create(&threads[i], NULL, ThreadEntry,(void*)this);
          if(create != 0)
          {
               cerr<<"pthread_create failed";
          }
     }
     for(unsigned int i = 0; i<2; i++)
     {
          int join = pthread_join(threads[i], NULL);
          if(join != 0)
          {
               cerr<<"pthread_join failed";
          }
     }
}
```

# condition variable

```
pthread_cond_t cond;
pthread_cond_init(&cond, NULL);
..
/* in thread 1 */
{
        while(/* shared variable is not in state we want */)
        {
                pthread_cond_wait(&cond, &mtx);
        }
}
/* in thread 2 */
{
        pthread_cond_signal(&cond);
}
..
pthread_cond_destroy(&cond);
```

# mutex

```
pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);
int counter=0; // a shared variable
..
/* Function C */
void functionC()
{
        pthread_mutex_lock(&mutex);
        counter++;
        pthread_mutex_unlock(&mutex);
}
..
pthread_mutex_destroy(&mutex);
```

# shared memory

```
/* before fork() */
int* shared = NULL;

shmID = shmget(IPC_PRIVATE, sizeof(int)*1, IPC_CREAT |
                                           SHM_R | SHM_W);

shared = static_cast<int *>(shmat(shmID, 0, 0));

..

/* after finishing all the job */
shmdt(shared);
shmctl(shmID, IPC_RMID, NULL);
```

# semaphore

```
sem_t sem;
int ret = sem_init(&sem, 1, 1);
int counter = 0; // a shared variable
..
/* Function C */
void functionC()
{
        sem_wait(&sem);
        counter++;
        sem_post(&sem);
}
..
sem_destroy(&sem);
```

# Deliverables

- **yourNetID_project4**.zip
  - *.cpp
  - *.h
  - Makefile (g++ -Wall -**lpthread**…)
  - Readme (optional)
- Handin system
- Compile & Run on “cse410”

# References

- Useful Links
  - http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
  - http://www.csc.villanova.edu/~mdamian/threads/posixsem.html
  - http://www.cs.cf.ac.uk/Dave/C/node27.html
- Figures
  - http://www.receptionist.org/outsourcing/4-tasks-virtual-receptionist-can-effectively-do-for-your-business/
  - http://www.sunpack.com/blog/2011/03/rivalry-business/
  - http://photo-dictionary.com/phrase/2495/knitting-thread.html#b