//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This is a sample threaded program in C++. The main thread creates

// 4 daughter threads. Each daughter thread simply prints out a message

// before exiting. Notice that I’ve set the thread attributes to joinable and

// of system scope.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <iostream.h>

#include <stdio.h>

#include <pthread.h>

#define NUM\_THREADS 4

void \*thread\_function( void \*arg );

int main( void )

{

int i, tmp;

int arg[NUM\_THREADS] = {0,1,2,3};

pthread\_t thread[NUM\_THREADS];

pthread\_attr\_t attr;

// initialize and set the thread attributes

pthread\_attr\_init( &attr );

pthread\_attr\_setdetachstate( &attr, PTHREAD\_CREATE\_JOINABLE );

pthread\_attr\_setscope( &attr, PTHREAD\_SCOPE\_SYSTEM );

// creating threads

for ( i=0; i<NUM\_THREADS; i++ )

{

tmp = pthread\_create( &thread[i], &attr, thread\_function, (void \*)&arg[i] );

if ( tmp != 0 )

{

cout << "Creating thread " << i << " failed!" << endl;

return 1;

}

}

// joining threads

for ( i=0; i<NUM\_THREADS; i++ )

{

tmp = pthread\_join( thread[i], NULL );

if ( tmp != 0 )

{

cout << "Joing thread " << i << " failed!" << endl;

return 1;

}

}

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// This is the function each thread is going to run. It simply asks

// the thread to print out a message. Notice the pointer acrobatics.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void \*thread\_function( void \*arg )

{

int id;

id = \*((int \*)arg);

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

pthread\_exit( NULL );

}

#include<pthread.h>

#include <stdio.h>

/\* Example program creating thread to compute square of value \*/

int value;/\* thread stores result here \*/

void \*my\_thread(void \*param); /\* the thread \*/

main (int argc, char \*argv[])

{

pthread\_t tid; /\* thread identifier \*/

int retcode;

/\* check input parameters \*/

if (argc != 2) {

fprintf (stderr, "usage: a.out <integer value>\n");

exit(0);

}

/\* create the thread \*/

retcode = pthread\_create(&tid,NULL,my\_thread,argv[1]);

if (retcode != 0) {

fprintf (stderr, "Unable to create thread\n");

exit (1);

}

/\* wait for created thread to exit \*/

pthread\_join(tid,NULL);

printf ("I am the parent: Square = %d\n", value);

} //main

/\* The thread will begin control in this function \*/

void \*my\_thread(void \*param)

{

int i = atoi (param);

printf ("I am the child, passed value %d\n", i);

value = i \* i;

/\* next line is not really necessary \*/

pthread\_exit(0);

}

#include<pthread.h>

#include <stdio.h>

/\* Producer/consumer program illustrating conditional variables \*/

/\* Size of shared buffer \*/

#define BUF\_SIZE 3

int buffer[BUF\_SIZE]; /\*shared buffer \*/

int add=0; /\* place to add next element \*/

int rem=0; /\* place to remove next element \*/

int num=0; /\* number elements in buffer \*/

pthread\_mutex\_t m=PTHREAD\_MUTEX\_INITIALIZER; /\* mutex lock for buffer \*/

pthread\_cond\_t c\_cons=PTHREAD\_COND\_INITIALIZER; /\* consumer waits on this cond var \*/

pthread\_cond\_t c\_prod=PTHREAD\_COND\_INITIALIZER; /\* producer waits on this cond var \*/

void \*producer(void \*param);

void \*consumer(void \*param);

main (int argc, char \*argv[])

{

pthread\_t tid1, tid2; /\* thread identifiers \*/

int i;

/\* create the threads; may be any number, in general \*/

if (pthread\_create(&tid1,NULL,producer,NULL) != 0) {

fprintf (stderr, "Unable to create producer thread\n");

exit (1);

}

if (pthread\_create(&tid2,NULL,consumer,NULL) != 0) {

fprintf (stderr, "Unable to create consumer thread\n");

exit (1);

}

/\* wait for created thread to exit \*/

pthread\_join(tid1,NULL);

pthread\_join(tid2,NULL);

printf ("Parent quiting\n");

}

/\* Produce value(s) \*/

void \*producer(void \*param)

{

int i;

for (i=1; i<=20; i++) {

/\* Insert into buffer \*/

pthread\_mutex\_lock (&m);

if (num > BUF\_SIZE) exit(1); /\* overflow \*/

while (num == BUF\_SIZE) /\* block if buffer is full \*/

pthread\_cond\_wait (&c\_prod, &m);

/\* if executing here, buffer not full so add element \*/

buffer[add] = i;

add = (add+1) % BUF\_SIZE;

num++;

pthread\_mutex\_unlock (&m);

pthread\_cond\_signal (&c\_cons);

printf ("producer: inserted %d\n", i); fflush (stdout);

}

printf ("producer quiting\n"); fflush (stdout);

}

/\* Consume value(s); Note the consumer never terminates \*/

void \*consumer(void \*param)

{

int i;

while (1) {

pthread\_mutex\_lock (&m);

if (num < 0) exit(1); /\* underflow \*/

while (num == 0) /\* block if buffer empty \*/

pthread\_cond\_wait (&c\_cons, &m);

/\* if executing here, buffer not empty so remove element \*/

i = buffer[rem];

rem = (rem+1) % BUF\_SIZE;

num--;

pthread\_mutex\_unlock (&m);

pthread\_cond\_signal (&c\_prod);

printf ("Consume value %d\n", i); fflush(stdout);

}

}

This simple example code demonstrates the use of several Pthread condition variable routines. The main routine creates three threads. Two of the threads perform work and update a "count" variable. The third thread waits until the count variable reaches a specified value.

#include <pthread.h>

#include <stdio.h>

#define NUM\_THREADS 3

#define TCOUNT 10

#define COUNT\_LIMIT 12

int count = 0;

int thread\_ids[3] = {0,1,2};

pthread\_mutex\_t count\_mutex;

pthread\_cond\_t count\_threshold\_cv;

int main (int argc, char \*argv[])

{

int i, rc;

pthread\_t threads[3];

pthread\_attr\_t attr;

/\* Initialize mutex and condition variable objects \*/

pthread\_mutex\_init(&count\_mutex, NULL);

pthread\_cond\_init (&count\_threshold\_cv, NULL);

/\* For portability, explicitly create threads in a joinable state \*/

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

pthread\_create(&threads[0], &attr, inc\_count, (void \*)&thread\_ids[0]);

pthread\_create(&threads[1], &attr, inc\_count, (void \*)&thread\_ids[1]);

pthread\_create(&threads[2], &attr, watch\_count, (void \*)&thread\_ids[2]);

/\* Wait for all threads to complete \*/

for (i=0; i<NUM\_THREADS; i++) {

pthread\_join(threads[i], NULL);

}

printf ("Main(): Waited on %d threads. Done.\n", NUM\_THREADS);

/\* Clean up and exit \*/

pthread\_attr\_destroy(&attr);

pthread\_mutex\_destroy(&count\_mutex);

pthread\_cond\_destroy(&count\_threshold\_cv);

pthread\_exit(NULL);

} /\* of main /

void \*inc\_count(void \*idp)

{

int j,i;

double result=0.0;

int \*my\_id = idp;

for (i=0; i<TCOUNT; i++) {

pthread\_mutex\_lock(&count\_mutex);

count++;

/\* Check the value of count and signal waiting thread when condition is

reached. Note that this occurs while mutex is locked. \*/

if (count == COUNT\_LIMIT) {

pthread\_cond\_signal(&count\_threshold\_cv);

printf("inc\_count(): thread %d, count = %d Threshold reached.\n", \*my\_id, count);

}

printf("inc\_count(): thread %d, count = %d, unlocking mutex\n",\*my\_id, count);

pthread\_mutex\_unlock(&count\_mutex);

/\* Do some work so threads can alternate on mutex lock \*/

for (j=0; j<1000; j++) result = result + (double)random();

}

pthread\_exit(NULL);

}

void \*watch\_count(void \*idp)

{

int \*my\_id = idp;

printf("Starting watch\_count(): thread %d\n", \*my\_id);

/\*

Lock mutex and wait for signal. Note that the pthread\_cond\_wait

routine will automatically and atomically unlock mutex while it waits.

Also, note that if COUNT\_LIMIT is reached before this routine is run by

the waiting thread, the loop will be skipped to prevent pthread\_cond\_wait

from never returning.

\*/

pthread\_mutex\_lock(&count\_mutex);

if (count<COUNT\_LIMIT) {

pthread\_cond\_wait(&count\_threshold\_cv, &count\_mutex);

printf("watch\_count(): thread %d Condition signal received.\n", \*my\_id);

}

pthread\_mutex\_unlock(&count\_mutex);

pthread\_exit(NULL);

}

inc\_count(): thread 0, count = 1, unlocking mutex

Starting watch\_count(): thread 2

inc\_count(): thread 1, count = 2, unlocking mutex

inc\_count(): thread 0, count = 3, unlocking mutex

inc\_count(): thread 1, count = 4, unlocking mutex

inc\_count(): thread 0, count = 5, unlocking mutex

inc\_count(): thread 0, count = 6, unlocking mutex

inc\_count(): thread 1, count = 7, unlocking mutex

inc\_count(): thread 0, count = 8, unlocking mutex

inc\_count(): thread 1, count = 9, unlocking mutex

inc\_count(): thread 0, count = 10, unlocking mutex

inc\_count(): thread 1, count = 11, unlocking mutex

inc\_count(): thread 0, count = 12 Threshold reached.

inc\_count(): thread 0, count = 12, unlocking mutex

watch\_count(): thread 2 Condition signal received.

inc\_count(): thread 1, count = 13, unlocking mutex

inc\_count(): thread 0, count = 14, unlocking mutex

inc\_count(): thread 1, count = 15, unlocking mutex

inc\_count(): thread 0, count = 16, unlocking mutex

inc\_count(): thread 1, count = 17, unlocking mutex

inc\_count(): thread 0, count = 18, unlocking mutex

inc\_count(): thread 1, count = 19, unlocking mutex

inc\_count(): thread 1, count = 20, unlocking mutex

Main(): Waited on 3 threads. Done.

Given badcnt.c, which is **An Improperly Synchronized Threaded Program, as below**

**unsigned int cnt = 0; /\* shared \*/**

**int main() {**

**pthread\_t tid1, tid2;**

**Pthread\_create(&tid1, NULL,**

**count, NULL);**

**Pthread\_create(&tid2, NULL,**

**count, NULL);**

**Pthread\_join(tid1, NULL);**

**Pthread\_join(tid2, NULL);**

**if (cnt != (unsigned)NITERS\*2)**

**printf("BOOM! cnt=%d\n",**

**cnt);**

**else**

**printf("OK cnt=%d\n",**

**cnt);**

**}**

## Execution flow:

**linux> ./badcnt**

**BOOM! cnt=198841183**

**linux> ./badcnt**

**BOOM! cnt=198261801**

**linux> ./badcnt**

**BOOM! cnt=198269672**

**-------------**

**cnt should be**

**200,000,000.**

**What went wrong?!**

## Use Semaphores to correct the codes

/\* Semaphore s is initially 1 \*/

/\* Thread routine \*/

void \*count(void \*arg)

{

int i;

for (i=0; i<NITERS; i++) {

P(s);

cnt++;

V(s);

}

return NULL;

}

*/\* Initialize semaphore sem to value \*/*

*/\* pshared=0 if thread, pshared=1 if process \*/*

void Sem\_init(sem\_t \*sem, int pshared, unsigned int value) {

if (sem\_init(sem, pshared, value) == -1)

unix\_error("Sem\_init");

}

*/\* P operation on semaphore sem \*/*

void P(sem\_t \*sem) {

if (sem\_wait(sem) == -1)

unix\_error("P");

}

*/\* V operation on semaphore sem \*/*

void V(sem\_t \*sem) {

if (sem\_post(sem) == -1)

unix\_error("V");

}

/\* goodcnt.c - properly sync’d

counter program \*/

#include "csapp.h"

#define NITERS 10000000

unsigned int cnt; /\* counter \*/

sem\_t sem; /\* semaphore \*/

int main() {

pthread\_t tid1, tid2;

Sem\_init(&sem, 0, 1); /\* sem=1 \*/

/\* create 2 threads and wait \*/

...

if (cnt != (unsigned)NITERS\*2)

printf("BOOM! cnt=%d\n", cnt);

else

printf("OK cnt=%d\n", cnt);

exit(0);

}

/\* thread routine \*/

void \*count(void \*arg)

{

int i;

for (i=0; i<NITERS; i++) {

P(&sem);

cnt++;

V(&sem);

}

return NULL;

}