University of Wolverhampton
School of Mathematics and Computer Science

Student Number:
Name:

6CS005 High Performance Computing Week 2 Workshop

POSIX Thread and Semaphores

Tasks – Threads and messaging

You may need to refer to the Week 3 lecture sides in order to complete these tasks.

1. The following program demonstrates 3 thread sending string messages to each other, using a global array. The messages are sent meant to be sent in the following order:

    a. Thread 0 sends Thread 1 a message
    b. Thread 1 receives the message
    c. Thread 1 sends Thread 2 a message
    d. Thread 2 receives the message
    e. Thread 2 sends Thread 0 a message
    f. Thread 0 receives the message
    g. This then repeats from (a) 10 times

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>

char *messages[3] = {NULL, NULL, NULL};

void *messenger(void *p)
{
  long tid = (long)p;
  char tmpbuf[100];

  for(int i=0; i<10; i++)
  {
    /* Sending a message */
    long int dest = (tid + 1) % 3;
    sprintf(tmpbuf,"Hello from Thread %ld!", tid);
    char *msg = strdup(tmpbuf);
    messages[dest] = msg;
    printf("Thread %ld sent the message to Thread %ld\n",tid, dest);

    /* Receiving a message */
    printf("Thread %ld received the message '%s'\n",tid, messages[tid]);
    free(messages[tid]);
    messages[tid] = NULL;
  }
  return NULL;
}

void main()
{
```

```
    pthread_t thrID1, thrID2, thrID3;

    pthread_create(&thrID1, NULL, messenger, (void *)0);
    pthread_create(&thrID2, NULL, messenger, (void *)1);
    pthread_create(&thrID3, NULL, messenger, (void *)2);
    pthread_join(thrID1, NULL);
    pthread_join(thrID2, NULL);
    pthread_join(thrID3, NULL);
}
```

Enter and run the program. Does it work as planned?

2. Using the technique of "busy-waiting" to correct the program, and establishing the correct order of messages.

3. Use pthread "mutex" to correct the program in (1). You will need multiple mutexes.

4. Use semaphores to correct the program in (1).