UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science Programme **Trinity Term 2014**
Year 2 Annual Examinations

Concurrent Systems and Operating Systems

Tuesday May 13, 2014 **Drawing Office** **09:30–11:30**

Dr Mike Brady

Instructions to Candidates:

Attempt **two** questions. All questions carry equal marks. Each question is marked out of a total of 20 marks.

You may not start this examination until you are instructed to do so by the Invigilator.

Materials permitted for this examination:

A two-page document, entitled *"Pthread Types and Function Prototypes"* accompanies this examination paper.

Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.

1. (a) What is a pthreads *mutex*? How would you use one? Give an example of a situation where a mutex would be useful. [2 marks]

   (b) What is a pthreads *condition variable*? How would you use one? Give an example of a situation where a condition variable would be useful. [2 marks]

   (c) Explain what a *semaphore* is and how you use one. Explain the differences between the different kinds of semaphores. [5 marks]

   (d) Without using the semaphores provided in the pthreads library, write a program that simulates a producer and two consumers connected by a first-in-first-out queue. You can simulate production by using, say, a pseudo-random sleep delay to simulate the time taken to produce the item and you can simulate adding the item to a queue by *safely* adding 1 to a global variable. Similarly, removal of an item from a queue can be simulated by safely decrementing the global variable by 1 and consumption can be simulated using a sleep delay.

   Write a brief note to justify your choice of facilities from the pthreads library—what functions and data structures are you using, and why have you chosen them? [9 marks]

   (e) Is your program *fair?* If it is, explain how you know that. If it isn't, explain how you would make it fair. [2 marks]

2. (a) SPIN is used in the formal verification of parallel systems. What does that mean, and why is SPIN and Promela different from conventional programming languages and tools? [2 marks]

   (b) What is the difference in using SPIN in *interpreter* mode and in *verification* mode? [2 marks]

   (c) Write the Promela code to implement the concurrent system described by the following diagram, which is designed to increment the value of the global variable n to 20. (Notice that the two processes are slightly different.)

Example: Concurrent Counting Algorithm	
integer n ← 0;	
p	**q**
integer temp	integer temp
p1: do 10 times	q1: do 10 times
p2: temp ← n	q2: n ← n + 1
p3: n ← temp + 1	

   [8 marks]

   (d) Is the system correct or faulty? Explain your reasoning.
   Show, with examples of code or commands, how you would use SPIN in an attempt to *prove* that your conclusions were correct. [8 marks]

3. (a) List the main purposes and roles of a modern operating system. [2 marks]

   (b) List the major components of an operating system you know, such as Linux, Mac OS X, Windows or OSP 2. [2 marks]

   (c) Explain what scheduling is in the context of an operating system. [2 marks]

   (d) Draw a diagram of the lifecycle of a process or thread and explain the difference between cooperating and preemptive scheduling. [2 marks]

   (e) Give examples, with timing diagrams and explanations, of different scheduling policies. [4 marks]

   (f) Give an overview of memory management in a virtual-memory-based operating system. Describe and appraise the features and policies involved. [5 marks]

   (g) Using a hard disk as a virtual memory backing store with an average access time of, say, 5 milliseconds (a millisecond is 10^{-3} seconds) , main memory with an access time of 10 nanoseconds (a nanosecond is 10^{-9} second) and a page fault rate of 1 per 100,000, calculate the average virtual memory access time. [3 marks]

Pthread Types and Function Prototypes

Definitions

```
pthread_t;  //this is the type of a pthread;
pthread_mutex_t; //this is the type of a mutex;
pthread_cond_t; // this is the type of a condition variable
```

Create a thread

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void *(*thread_function)(void *), void *arg);
```

Static Initialisation

Mutexes and condition variables can be initialized to default values using the `INITIALIZER` macros. For example:

```
pthread_mutex_t count_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t count_cond = PTHREAD_COND_INITIALIZER;
```

Dynamic Initialisation

Mutexes, condition variables and semaphores can be initialized dynamically using the following calls:

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void *(*thread_function)(void *), void *arg);
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *mutexattr);
int pthread_cond_init(pthread_cond_t *cond,
                      pthread_condattr_t *cond_attr);
int pthread_attr_init(pthread_attr_t *attr);
```

Deletion

```
int   pthread_mutex_destroy(pthread_mutex_t *);
int   pthread_cond_destroy(pthread_cond_t *);
```

Thread Function

The thread_function prototype would look like this:

```
void *thread_function(void *args);
```

Thread Exit & Join

```
void  pthread_exit(void *); // exit the thread i.e. terminate the thread
int   pthread_join(pthread_t, void **); // wait for the thread to exit.
```

Mutex locking and unlocking

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Pthread Condition Variables

```
int pthread_cond_wait(pthread_cond_t *cond,
                      pthread_mutex_t *mutex);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
```

Semaphores

```
sem_t; // this is the type of a semaphore
int sem_init(sem_t *sem, int pshared, unsigned int value); // pshared = 0 for semaphores
int sem_wait(sem_t *sp); // wait
int sem_post(sem_t *sp); // post
int sem_destroy(sem_t * sem); // delete
```