Coláiste na Tríonóide, Baile Átha Cliath
Trinity College Dublin
Ollscoil Átha Cliath | The University of Dublin

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Engineering **Trinity Term 2016**
Year 3 Annual Examinations

3D4 Operating Systems and Concurrent Systems

20 May 2016 **Goldsmith Hall** **14:00 – 16:00**

Dr Mike Brady

Instructions to Candidates:

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

Attempt **two** questions.

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

Materials permitted for this examination:

A list of `pthread` functions and prototypes (two pages) 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) Write a brief note on *threads* — what they are, what they are for, how they differ from *processes*, how they interact and what advantages and shortcomings they have. [8 marks]

   (b) What does the term *massively parallel* or *embarassingly parallel* mean in relation to a parallel program? [1 marks]

   (c) Write a simple C/C++ function to check if a number is prime. Here is the function's signature:

   ```
   int is_prime(int n);
   ```

   The function should return `1` if the number `n` is prime and `0` otherwise.

   Using this function, write a multi-threaded C/C++ program to count the number of prime numbers in the interval $1..n$. [8 marks]

   (d) With an eye to efficiency and core utilisation, what kinds of problems would you see with this program? How could you fix them? [3 marks]

2. (a) SPIN is said to be a *Model Checker*. What exactly does that mean? Why might it be useful? [2 marks]

   (b) In the context of Promela, SPIN, and concurrent processes, define *scenario*. Give a brief example. [3 marks]

   (c) Develop a Promela model of the well-known *Dining Philosophers* problem:

   - Five philosophers sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers. Thus there are five philosophers and five forks.
   - Each philosopher alternately thinks and eats. However, a philosopher can only eat spaghetti when he or she has both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher.
   - After a philosopher finishes eating, he or she puts down both forks so they become available to others.
   - A philosopher can take the fork on the right or the one on the left as they become available, but cannot start eating before getting both of them.
   - Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed.

   What do you add to the Promela model to detect *deadlock*, *livelock* and *starvation*?

   [15 marks]

3. (a) List and briefly describe the main components of a typical desktop or laptop operating system. Why would such an operating system be unsuitable for real-time operation?

[4 marks]

(b) Explain how *virtual memory* works and give an account of its main components and their operation. [8 marks]

(c) Given a demand-paged virtual memory system with the following parameters:

Page Fault Probability (p)	0.000001
Memory Access Time	10 nS
Average time to read a page from disk	5 mS

(i) Calculate the effective access time.

(ii) What is the slowdown due to the use of demand paging (as a factor of memory access time)?

(iii) What would the page-fault probability have to be to have an overhead of less than 50%. (i.e. so that the average access time would be not more than 150% of the memory access time)?

[8 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
```