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 Computer Science **Semester 2 2019**
Year 2 Annual Examinations

Concurrent Systems and Operating Systems

Thursday, 25th April **RDS Simmonscourt** **17:00–19:00**

Dr Andrew Butterfield

Instructions to Candidates:

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

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

Materials required for this examination:

There is a Reference section at the end of the paper (pp5–6).

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 the main difference between a *process* and a *thread*? [2 marks]

   (b) Explain the operation of the `pthread` library functions `pthread_mutex_lock`, `pthread_mutex_unlock`, and `pthread_cond_wait`. [6 marks]

   (c) Function `is_perfect` returns true if a number is perfect (the sum of all its factors), with signature:

   ```
   int is_perfect(int n);
   ```

   It returns `1` if `n` is perfect, and `0` otherwise.

   Using this function, write a multi-threaded C program to count the number of perfect numbers in the interval $1 \ldots N$. [8 marks]

   (d) What does the phrase *embarassingly parallel* or *massively parallel* mean in relation to a parallel program? Would you consider the program in Q1(c) above to be *embarassingly parallel*? Explain your answer. [4 marks]

2. (a) SPIN is described as a *Model Checker*. What does that mean, precisely? Why might such a checker be useful? [2 marks]

   (b) Explain the terms *deadlock*, *livelock*, *starvation*, *fairness*. [4 marks]

   (c) SPIN has two main modes of operation — interpretation and verification. Explain the difference between them, and the importance of that difference. [4 marks]

   (d) What is the Dining Philosophers Problem, and why is it of such interest in the context of concurrent systems? [2 marks]

   (e) Write a Promela description of a reduced version of the Dining Philosophers Problem which has two (identical) philosophers at a table for two with just two forks. [4 marks]

   (f) Show (with examples) how the system you describe can suffer from deadlock, livelock and/or starvation. [4 marks]

3. (a) What is the difference between a *physical address* and a *virtual address*? [2 marks]

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

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

[9 marks]

(c) Virtual memory is normally not used in applications where assured real-time response is required. Why is that? [3 marks]

(d) Explain, with diagrams, the operation of the *scheduler* in a conventional operating system. In your answer, explain the concept of *fairness* and explain how it might be managed in a scheduler. [6 marks]

Reference

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 semap
int sem_wait(sem_t *sp); // wait
int sem_post(sem_t *sp); // post
int sem_destroy(sem_t * sem); // delete
```