UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

Integrated Computer Science **Trinity Term 2013**
Year 3 Examination

Concurrent Systems II

Wednesday May 15, 2013 **Luce Lower (205)** **09:30–11:30**

Dr Mike Brady

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 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) State Amdahl's Law. [2 marks]

   (b) What are the implications of Amdahl's Law for improving the performance of a program by parallelising it? [3 marks]

   (c) In the pthreads library, what is a *condition variable*? Explain where and when condition variables might be more suitable than mutexes. [4 marks]

   (d) Using facilities provided in the pthreads library, write a program with three threads that simulate a producer and two consumers. You can simulate production by using a 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) Are condition variables *fair?* What does *fair* mean and what are its implications?

   [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) In concurrent programming, a *critical section* is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution. A process can only enter its critical section by executing a *preprotocol*. Upon exit from its critical section, a process executes a *postprotocol* which may be used in conjunction with the preprotocol to ensure that only one process can be in its critical section at any time.

   Write Promela code to model two processes, with appropriate pre- and postprotocols, each running in an endless loop of *non critical section ... critical section ... non critical section* etc. (Note, it's not important whether the scheme you use for mutual exclusion works fully or not—the important thing is to devise a plausible scheme whose properties you can attempt to verify.) [6 marks]

   (d) Write the extra code you'd need to add if you were trying to prove that the scheme of pre- and postprotocol you are using guarantees mututal exclusion. [4 marks]

   (e) It is often stated that critical section code is assumed to *progress*, while non critical section code is not guaranteed to progress. What do the terms *progress* and *non progress* mean? [3 marks]

   (f) Write Promela code to simulate code that is not guaranteed to progress, and indicate how you would use it while attempting to prove that your scheme did not suffer from starvation. [3 marks]

3. (a) Give an overview of how memory is managed in a typical modern operating system. Your answer should deal with address translation, page tables (including multi-level page tables), virtual memory, page faults and the the working set.

[7 marks]

(b) A number of important 'figures of merit' are used when considering operating system schedulers: *average start time, average execution time ratio* and *average turnaround time*. Explain these terms, and discuss their significance. [3 marks]

(c) Given the processes P1 to P5 in the table below, with creation times, CPU time requirements and priorities (1 is the highest priority, 5 is the lowest) as shown, determine the start and ending times of each process assuming priority scheduling with a quantum of 2 units of time. Hence, calculate the averge start time, execution time ratio and turnaround time ratio for each process.

	Creation Time	CPU Time Needed	Priority
P1	0	3	2
P2	2	10	3
P3	5	5	4
P4	6	8	3
P5	9	2	1

[7 marks]

(d) Explain, with reference to the previous part of the question if appropriate, how you would expect different scheduling schemes to affect the figures of merit mentioned above. [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
```