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 2021**
Computer Science and Business
Year 2 Annual Examinations
Integrated Engineering
Year 3 Annual Examinations

Concurrent Systems and Operating Systems

Wednesday, 19th May **Real-time Online** **12:00–14:30**

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.

Your answers should be uploaded as one or more clearly named files, and may be text files and/or scanned images of handwriting.

Text files: txt,doc,odt,pdf or similar.

Handwriting/Images: jpg,png,pdf or similar.

Materials required for this examination:

There is a Reference section at the end of the paper (pp5–7).

Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator in the relevant answer file

1. (a) Different *processes* do not share global memory, while different *threads* in a process all share that processes memory. What advantage do processes have over threads? What advantage do threads have over processes? [2 marks]

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

   (c) Function `trapezoid` computes the area of a trapezoid, for use in a numerical integration algorithm.

```
double trapezoid(double a, double h) {
  return h*(f(a)+f(a+h))/2.0;
}
```

   Here `a` is the current x-value, while `h` is the width of the thin vertical strip whose area is being approximated.

   Assuming that `trapezoid` has been provided, as well as a function `double f(double x)`, write a C *pthreads* program that uses 100 threads to numerically integrate `f` between 0 and 10. [8 marks]

   (d) Consider two threads running concurrently, where `global` is a global variable, while `local` is local to each thread, each of which does the following:

```
local = global;
local = local +1;
global = local;
```

   Assuming that `global` is initialised to 0 initially, describe how the execution could result in `global` having a final value of 1. [4 marks]

2. (a) Explain the concepts in Promela of statement executability and blocking using the following two Promela statements as examples:

```
mybool =  x > y  // statement 1
x > y            // statement 2
```

What is the effect of the above statements when they do execute? [6 marks]

(b) Write a model in Promela for each of the following hardware *atomic* instructions

i. Test-and-Set: read a value from memory; write 1 to that memory location

ii. Exchange: swap the contents of two distinct memory locations.

Your models should clearly show the use of any temporary storage in the CPU. [4 marks]

(c) Consider a model of a proposed solution to the mutual exclusion problem, where the part of the model that describes the critical region has the form:

```
preamble-before-entering-critical-region;
critical_stuff;
postamble-on-leaving-critical-region;
```

Describe how adding a numeric variable plus some form of verification check can be used so that SPIN can check that only one process is ever doing `critical_stuff` at any one time. [4 marks]

(d) Model checkers are usually used to see if all desired properties hold of a model. If they find a property violation, they report an error and return a counter-example that shows a sequence of steps that leads to a property failure. Given a model known to be correct, with a number of valid end-states, explain how to use the SPIN model-checker to generate a path through the model that leads to a given valid end-state.

[6 marks]

3. (a) Explain the lifetime of a typical programme running on a typical computer. What aspect of its behaviour has had the most influence on the design of both hardware and operating system software. [4 marks]

   (b) Describe two hardware features that have been provided to facilitate the implementation of modern operating systems. [6 marks]

   (c) Given a Hard Disk with the following parameters:

Rotational speed (p)	5400 RPM
Seek Time between adjacent Tracks	1mS
Average Seek Time	10mS
Sector Size	512 bytes
Sectors per Track	100

   (i) Explain the terms "Track", "Sector", and "Seek Time".

   (ii) Calculate the time to read 100kbytes if all the sectors are randomly mixed across the disk.

   (iii) Calculate the time to read 100kbytes if all the sectors are in order on two adjacent tracks.

   [5 marks]

   (d) Explain the File-system concepts of "Inode", "Filename", and the distinction between "Open" and "Closed" files. [5 marks]

Reference

Pthread Types and Function Prototypes

Declarations

```
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/Exit/Join a thread

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
                   void *(*thread_function)(void *), void *arg);
void  pthread_exit(void *); // terminate calling thread
int   pthread_join(pthread_t thr, void **retval); // wait for thr to exit.
```

Static Initialisation

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

Mutex locking and unlocking

```
int pthread_mutex_lock(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);
```

Thread Function A thread function (`my_thread_f` say) would look like this:

```
void *my_thread_f(void *args) { ... }
```

Promela Syntax

Datatypes

```
bool - (1 bit)
int - (32* bits signed)
pid
chan
mtype = { name, name, ... }
typedef typename { seq. of declarations }
```

Operators (descending precendence)

```
() [] . ! ~ ++ -- * / % + - << >>
< <= > >= == != & ^ | && ||
( .. -> .. : .. ) conditional expression
```

Predefined

```
Constants - true, false
Variables - (read-only):
     _nr_pr - number of processes
     _pid instance number of executing process
```

Preprocessor

```
inline name (arguments) { ... }
```

Statements

```
type var; - variable declaration
type var[N]; - array declaration
var = expr
assert(expression)
printf, printm - print to standard output
skip - no operation
break - exit innermost do loop
goto - jump to label
Label prefixes with special meaning:
```

```
     accept - accept cycle
     end - valid end state
     progress - non-progress cycle
atomic { ... } - execute without interleaving
```

Guarded commands

```
if :: guard -> statements :: .. fi
do :: guard -> statements :: .. od
else guard - executed if all others are false
```

Processes

```
Declaration - proctype procname (parameters) { ... }
Activate with prefixes - active or active[N]
Explicit process activation - run procname (arguments)
Initial process - init { ... }
```

Channels

```
chan ch = [capacity] of { type, type, ... }
ch ! args  - send
ch ? args  - receive + remove if first message matches
```

Temporal logic

```
!    not               [] always
&&   and               <> eventually
||   or                X  next
->   implies           U  strong until
<->  equivalent to     V  dual, as  pVq <-> !(!pU!q)
```

LTL formula

```
ltl name { ... }
```