

Concurrent and Distributed Systems - 2022–2023

CS0: Get-started Questions (DJ Greaves) (Rev A)

The questions on this preliminary sheet are mostly open-ended and their purpose is mainly for discussion in the first supervision. So do not worry about generating complete answers in advance.

* Star denotes optional/advanced exercise.

Q0 Parallel Programming

List or tabulate the essential similarities and/or differences between parallel programming and distributed systems.

Can you think of an application or algorithm where the shared memory is **not** just being used for some form of ‘message passing’?

Q1 Non-deterministic Scheduling

Why might the output from a concurrent program vary on different runs? What is one advantage of allowing this? What is one disadvantage? [Do not consider programs that read the RTC (real-time clock), use random number generators or read a different input data in different runs!]

Q2 Operating System Fundamental Abstractions.

Early versions of the Windows Operating System (before circa 1998) lacked most features that would be expected to be found in an operating system: it was essentially a GUI-controlled command shell. List the minimal abstractions expected from a proper operating system. Windows did provide some basic, non-preemptive threads (using co-routines). What benefit did having threads bring and what problem arose from them being non-preemptive? What is the essential difference between a thread and a process? Name four segments typically associated with a normal/minimal process. When further threads are added, what might happen to the number of segments?

Q3 Atomic Hardware Operations

This is a question to think about at the start of this CC/DS course and which you should probably be able to answer with confidence by start of Lent term!

Which of the following operations can be considered atomic on a modern digital computer: Store of a character? Store of a 32-bit word? Store of a 64-bit word? Atomic compare-and-swap? Write of a disk sector? A system call? A floating-point division? Sending a network packet? Signalling a semaphore? An inter-processor interrupt (IPI) aka inter-core interrupt (ICI) ?

Make sure you understand the difference between *cache consistency* and *sequential consistency* by the end of the Computer Design course. You may find this book helpful: ‘*Modern SoC Design on Arm*’ [2021, DJ Greaves].

Q4 CBMC Example (*)

List the state variables required to model a 'Beer Fridge Stocking problem' as lectured with, say, 2 housemates. Separate state variables are likely required for the state of the fridge, each housemate and the note. Give a state transition graph for each participant showing its effects on shared state variables. Is the system finite state? Will their product sensibly be synchronous or asynchronous? Advanced: see if you can do anything useful with the CBMC model checker are your abstraction.

Q5 Pthreads C Example from L1.

If you already have some C experience, you can try this now. Otherwise, do leave it until you are more in the swing of C/C++.

```
// l1d1.c CCDS L1 D1: compile with something like: gcc -g l1d1.c -lpthread
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>

#define NUMTHREADS 4

char *threadstr = "Thread";

// Do we even need the random call to get random results?
void *threadfn(void *arg)
{
    long int threadnum = (long int)(arg);
    sleep(rand() % 2); // Sleep 0 or 1 seconds
    printf(" %s %ld\n", threadstr, threadnum);
    return 0;
}

int main(void)
{
    printf("Starting\n");

    pthread_t threads[NUMTHREADS]; // Thread control blocks.

    for (long int i = 0; i < NUMTHREADS; i++)
        pthread_create(&threads[i], 0, threadfn, (void *)i);

    for (int i = 0; i < NUMTHREADS; i++) pthread_join(threads[i], 0);

    return 0;
}
// eof
```

©2019-22 - DJ Greaves.