IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2003

PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING: SECTION A

Wednesday, 11 June 2:00 pm

Time allowed: 1:30 hours

There are THREE questions on this paper.

Answer TWO questions.

Corrected Copy

This exam is OPEN BOOK

Any special instructions for invigilators and information for candidates are in page 1.

Examiners responsible: First Marker(s) G.A. Constantinides
Second Marker(s): Y.K. Demiris

Special information for invigilators: none

Special information for candidates: The notation "Ox" before a number means that the number is expressed using hexidecimal representation.

The Questions

1.

An ARM program is shown below. (SWI 0x02 prints the null-terminated string at address `r0` to the screen).

```
                AREA prog, CODE, READONLY
SWI_Write0      EQU             0x02
SWI_Exit        EQU             0x11

                ENTRY

                ADR             r0,     buffer
                SWI             SWI_Write0
                BL              Jumble
                SWI             SWI_Write0
                SWI             SWI_Exit

Jumble          STMED           r13!, {r0,r1}
JumbleLoop      LDRB            r1, [r0], #1
                CMP             r1, #0
                RSBNE           r1, r1, #('A'+'Z')
                STRNEB          r1, [r0,#-1]
                BNE             JumbleLoop
                LDMED           r13!, {r0,r1}
                MOV             pc, r14

buffer          =               "MYTEXT", 0

                END
```

a) What function does the subroutine `Jumble` perform?

[2]

b) What would you see on the screen if you execute this program?

[2]

When assembled, the address of the entry point to this program is 0x1000. Just before execution, `r0` = 0x0, `r1` = 0x0, and `r13` = 0x0.

c) State the value of `r13` during execution of the `Jumble` subroutine.

[2]

d) Draw a diagram of the stack during execution of the Jumble subroutine, clearly labelling the addresses and values of all entries.

[8]

e) You are required to change the program so that only upper-case letters are modified by the `Jumble` subroutine. All other characters should remain unchanged. Write ARM code for the new version of the subroutine.

[6]

2.

The Fibonacci sequence is: 1, 1, 2, 3, 5, ... and is defined by the following recurrence.

$$F_1 = 1$$
$$F_2 = 1$$
$$F_n = F_{n-1} + F_{n-2}, \text{ for } n > 2$$

Part of an ARM program to generate and print a Fibonacci sequence of length 20 is shown below.

After creating the sequence using a subroutine called `Fibonacci`, the program prints the sequence using a subroutine `PrintNum`, which prints the contents of `r1` to the screen followed by a carriage-return / line-feed combination.

```
                AREA prog, CODE, READONLY
SWI_Exit        EQU    0x11
SeqLength       EQU    20

                ENTRY

                ; Create Fibonacci Sequence
                ADR    r0, buffer
                MOV    r1, #SeqLength
                BL     Fibonacci

                ; Print Sequence
                MOV    r2, r1
PrintLoop       LDR    r1, [r0], #4
                BL     PrintNum
                SUBS   r2, r2, #1
                BNE    PrintLoop

                ; Exit
                SWI    SWI_Exit

; Fibonnacci Sequence Generator
; Input: r0 - pointer to buffer to fill with sequence
;        r1 - desired sequence length (r1 > 2)
; Output: buffer is filled with sequence

        [  SUBROUTINE HERE   ]

buffer          %      4*SeqLength

                END
```

Write the routine `Fibonacci`, to go in the program in the space identified. Comment your code.

[20]

3.

You have been given the task of improving the performance of a computer system. The computer has a 16-bit address bus and a 16-bit data bus, and is byte-addressed (i.e. each byte in memory has a distinct address).

A cache with 8 lines, each of one byte, has been suggested.

In order to evaluate the design, a typical program has been executed, and is found to result in the following memory accesses, ordered by time.

1. Read value 0x01 from address 0x0000
2. Read value 0x01 from address 0x0001
3. Write value 0x02 to address 0x0000
4. Read value 0x02 from address 0x0002
5. Read value 0x03 from address 0x0003
6. Write value 0x06 to address 0x0001
7. Read value 0x05 from address 0x0004
8. Read value 0x08 from address 0x0005
9. Write value 0xA0 to address 0x0002

a) State the principles of spatial and temporal locality

[4]

b) Draw a diagram illustrating the cache contents after the access sequence above has been completed. For each cache line, include the tag, the valid bit, and the data.

[8]

c) State the number of cache misses caused by the above sequence.

[2]

d) An alternative design is to use a 4-line cache with two bytes per line. Which of the memory accesses in the above sequence result in cache misses, when applied to the alternative design?

[6]

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2003

PRINCIPLES OF COMPUTERS AND SOFTWARE ENGINEERING: SECTION B

Wednesday, 11 June 2:00 pm

Time allowed: 1:00 hour

Corrected Copy

There are TWO questions on this paper.

Answer ONE question.

Any special instructions for invigilators and information for candidates are in page 1.

Examiners responsible: First Marker(s) Y.K. Demiris
Second Marker(s): M.P. Shanahan

Section B
Use a separate answer book for each section.

Answer ONLY ONE of the following two questions

QUESTION 1:

(a) Describe the MQS (Multi-level Queue Scheduling) process scheduling algorithm and list its advantages and disadvantages [4]

(b) Consider the following set of processes, with their corresponding duration and arrival times:

Process	Arrival time (ms)	Duration (ms)
A	0	2
B	2	3
C	3	6
D	4	5

Show the order of execution (including timing information) of the processes if the scheduler implements the following scheduling algorithms:

(i) Shortest job first (SJF) [3]
(ii) RR (round robin) with a time quantum of 4ms [3]
(iii) RR with a time quantum of 1ms [4]

For each of the algorithms calculate the average waiting time, and the average turnaround time.

(c) Describe the memory management scheme known as *paging*, and list its advantages and disadvantages. [6]

QUESTION 2:

(a) Describe *banker's algorithm* for dynamic deadlock avoidance [3]

(b) In the context of a memory paging system, consider the following scenario:

- You have three available frames
- The reference string is 4-3-1-3-5-5-6-3-1

Starting with empty frame contents, show the sequence of frame contents after each request, and count the number of page faults for each of the following page replacement algorithms:

(i) Optimal page replacement [3]
(ii) FIFO (first-in, first out) page replacement [2]
(iii) LRU (Least Recently Used) page replacement [2]

(c) In the "readers-writers" problem, a set of processes are accessing a block of shared data (e.g. a library catalogue): the reader processes only read shared data, while writer processes only write shared data. The following conditions are in place:

- A writer process can write items in the shared data block only if there are no other writer processes that are accessing the block.

- A reader process can read the shared data only if no writer is accessing them; more than one reader can read the shared data at the same time.
- Readers have priority: once a single reader has started accessing the shared data, readers can retain control of the shared data block as long as there is at least one reader in the act of reading.

Using semaphores to ensure that the conditions above hold, provide Pascal procedures for the reader and writer processes. Declare and properly initialise all semaphores and other variables that you will use. The data type *Semaphore*, and the standard semaphore primitives *init(Sem, number)*, *wait(Sem)*, and *signal(Sem)* are available. You may assume that the following procedures are also available: produce_item, write_item, read_item, consume_item. [10]

The Answers

Answer 1

a) Jumble replaces each uppercase character in the string by its alphabet-reversed version, i.e. A -> Z, B -> Y, …, Z -> A. The effect on non-uppercase characters has no special interpretation.

[2 marks]

b) The output from this program would be "MYTEXTNBGVCG"

[2 marks]

c) `r13 = 0xFFFFFFF8`

[2 marks]

d)

Address	Data
0x00000000	0x00000000
0xFFFFFFFC	0x00001034

[8 marks]

e)

```
Jumble          STMED     r13!, {r0,r1}
JumbleLoop      LDRB      r1, [r0], #1
                CMP       r1, #0
                BE        Done
                CMP       r1, #'A'
                BLO       JumbleLoop
                CMP       r1, #'Z'
                BHI       JumbleLoop
                RSB       r1, r1, #('A'+'Z')
                STRNEB    r1, [r0,#-1]
                BNE       JumbleLoop
Done            LDMED     r13!, {r0,r1}
                MOV       pc, r14
```

[6 marks]

Answer 2

```
Fibonacci       STMED   r13!, {r0-r4}
                MOV     r2, #1              ; )
                STR     r2, [r0], #4        ; ) starts with 1,1,
                MOV     r3, #1              ; )
                STR     r3, [r0], #4        ; )

                SUB     r1, r1, #2          ; number of items left to calculate

FibLoop         ADD     r4, r2, r3          ; ) calculate and store next number
                STR     r4, [r0], #4        ; )

                MOV     r3, r2              ; ) advance to next operands
                MOV     r2, r4              ; )

                SUBS    r1, r1, #1          ; ) end of for loop
                BNE     FibLoop             ; )

                LDMED   r13!, {r0-r4}
                MOV     pc, r14
```

[20 marks]

Mark breakdown:

Pushing and popping… [2 marks]
… the appropriate registers [2 marks]
Correctly labelling [2 marks] and returning from [2 marks] the subroutine
Correctly constructing a loop, with appropriate loop bound [4 marks]
Only requiring one memory access per loop iteration [2 marks]
Other correct functioning [6 marks]

A direct recursive implementation is exponential-time & therefore does not deserve full marks. In this case, do not give marks for correct loop construction and loop bound, but no other penalty.

Answer 3

a)

Spatial Locality: If an item is referenced, nearby items are likely to be referenced

Temporal Locality: If an item is referenced, it is likely to be referenced again soon

[4 marks]

b)

Cache Line	Valid	Tag	Data
0x0	Y	0x00	0x02
0x1	Y	0x00	0x06
0x2	Y	0x00	0xA0
0x3	Y	0x00	0x03
0x4	Y	0x00	0x05
0x5	Y	0x00	0x08
0x6	N	-	-
0x7	N	-	-

[8 marks]

c) 6 misses (all the reads).

[2 mark]

d) 3 misses (1st read of every pair)

1. 0x0000 => byte 0, line 0, tag 0, miss
2. 0x0001 => byte 1, line 0, tag 0, hit
3. 0x0000 => byte 0, line 0, tag 0, hit
4. 0x0002 => byte 0, line 1, tag 0, miss
5. 0x0003 => byte 1, line 1, tag 0, hit
6. 0x0001 => byte 1, line 0, tag 0, hit
7. 0x0004 => byte 0, line 2, tag 0, miss
8. 0x0005 => byte 1, line 2, tag 0, hit
9. 0x0002 => byte 0, line 1, tag 0, hit

[6 marks]

E1.9 – section B: Operating Systems
Model answers to exam questions 2003

Question 1

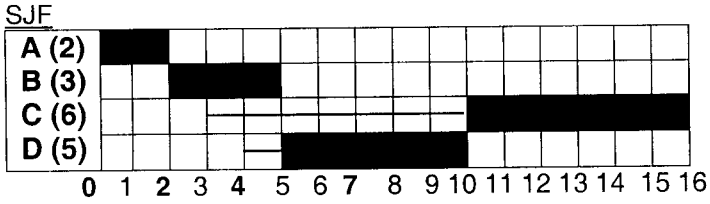
(i) [bookwork] The MQS scheduling algorithm divides ready processes into separate queues (for example, separate queues for foreground (interactive) and background (batch) processes); each queue has a priority associated with it, and queues with higher priority are given more time-slices for each of their processes. Each queue can have a separate algorithm for scheduling within the queue.

Advantages: Flexible – allows fine control of scheduling

Disadvantages: Does not allow for the possibility of processes that change requirements throughout their execution (e.g. a process that started with a long CPU burst, but requires interaction later)

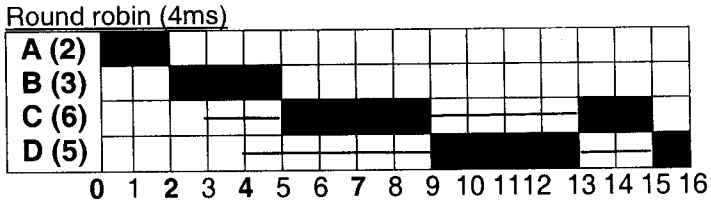
(ii) [new computed example]

SJF



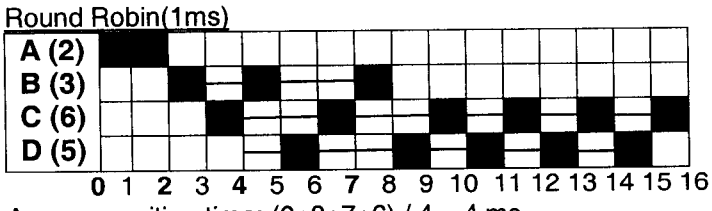
Average waiting time: (0+0+7+1) / 4 = 2 ms
Average turnaround time: (2+3+13+6) / 4 = 24 / 4 = 6ms

Round robin (4ms)



Average waiting time: (0+0+8+0) / 4 = 2 ms
Average turnaround time: (2+5+9+5) / 4 = 21 / 4 = 5.25 ms

Round Robin(1ms)



Average waiting time: (0+3+7+6) / 4 = 4 ms
Average turnaround time: (2+6+13+11) / 4 = 8 ms

(iii) [Bookwork]

Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous. Physical memory is divided into fixed-sized blocks called *frames*. Logical memory is broken into blocks (of the same size as frames) called pages. Addresses generated by the CPU are now divided into two parts: a page number (p) and a page offset (d): the page number is used as an index into a page table, containing the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address

Advantages: No external fragmentation – any free frame can be allocated to a process that needs it.
Disadvantages: internal fragmentation still possible, since frames are allocated as units.

Question 2:

(a) [bookwork] Banker's algorithm is used to dynamically avoid deadlocks. When a process requests a resource, the algorithm first determines whether granting the request will lead to an unsafe state – if doesn't it grants the request, otherwise the decision is postponed until a process releases some of its resources. To check whether the state is safe, the algorithm:

(1) checks whether it has some resources to satisfy some process
(2) The resources of that process are presumed released and added to the available resources
(3) steps 1 and 2 are repeated until we find that all current processes can be satisfied.

(b) [new computed example]

Optimal page replacement algorithm (5 page faults)

		4	**3**	**1**	**3**	**5**	**5**	**6**	**3**	**1**
Frame1	-	4	4	4		5		6		
Frame2	-	-	3	3		3		3		
Frame3	-	-	-	1		1		1		

FIFO page replacement algorithm (7 page faults)

		4	**3**	**1**	**3**	**5**	**5**	**6**	**3**	**1**
Frame1	-	4	4	4		5		6	6	1
Frame2	-	-	3	3		3		5	5	5
Frame3	-	-	-	1		1		1	3	3

LRU (Least recently used) page replacement algorithm (6 page faults)

		4	**3**	**1**	**3**	**5**	**5**	**6**	**3**	**1**
Frame1	-	4	4	4		5		5		1
Frame2	-	-	3	3		3		3		3
Frame3	-	-	-	1		1		6		6

(c) [bookwork]

Readers-writers problem

```
var mutex, wrt: Semaphore;
var read_count: int;

init(mutex, 1); init(wrt,1);
read_count=0;
```

Writer process:

```
procedure writer()
begin
  while(TRUE) do
    begin
      produce_item;
      wait(wrt);
      write_item;
      signal(wrt);
    end;
end;
```

Reader process:

```
procedure reader()
begin
  while(TRUE) do
    begin
        Wait(mutex);
        read_count=read_count+1;
        if read_count = 1 then wait(wrt);
        signal(mutex);
        get_item;
        wait(mutex);
        read_count = read_count – 1;
        if read_count = 0 then signal(wrt);
        signal(mutex);
        consume_item;
    end;
end;
```