UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2000

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
BEng Honours Degree in Information Systems Engineering Part I
MEng Honours Degree in Information Systems Engineering Part I
BEng Honours Degree in Mathematics and Computer Science Part II
MEng Honours Degree in Mathematics and Computer Science Part II
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER MCM2=C111R=I1.5R

OPERATING SYSTEMS CONCEPTS

Tuesday 9 May 2000, 14:30
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions

1 This question concerns virtual memory and paging. Consider a computer with 64-bit virtual addresses, 40-bit physical addresses, with a page size of 8192 (i.e. 2^{13}) bytes.

a Using a diagram, show how the current process's page table is used to translate a virtual address to a physical address in the computer's main memory.

b What are the advantages of making the page size large?

c What are the advantages of making the page size small?

d What is the disadvantage of making the page size variable?

For the remainder of this question, consider the following program fragment (expressed in an informal pseudo-code):

```
declare type R : record
                  x : byte
                  y : byte
                 end
declare A : array 1..819200 of R
for i = 1 to 819200 do
  A[i].x = A[i].x+1
```

Suppose the computer above has 819200 bytes of available physical memory.

e Explain why this code will take a long time to run

f Suggest a way to improve it substantially

(The 6 parts carry, respectively, 20%, 20%, 20%, 10%, 15% and 15% of the marks).

2 This question concerns concurrency and synchronisation. Consider the following program fragment, which is part of a multi-user room booking application:

```
procedure BookRoom(rm, time)
  if rooms[rm].isFree[time] then
    rooms[rm].isFree[time] := false
    print("Succeeded")
  else print("Sorry, already booked")
  endif
end BookRoom
```

a Explain what might go wrong if two different users try to make the same room booking at the same time.

b Alyssa P Hacker recommends we use a separate lock for each room, as shown below:

```
procedure BookRoom(rm, time)
  LOCK(rooms[rm].lock)
  if rooms[rm].isFree[time] then
    rooms[rm].isFree[time] := false
    print("Succeeded")
  else print("Sorry, already booked")
  endif
  UNLOCK(rooms[rm].lock)
end BookRoom
```

Can you explain why Alyssa might think this would lead to better performance than using just one lock?

c Suppose some users want to book *pairs* of rooms. We provide a new procedure:

```
procedure BookRoomPair(rm1, rm2, time)
  LOCK(rooms[rm1].lock)
  LOCK(rooms[rm2].lock)
  if rooms[rm1].isFree[time] and rooms[rm2].isFree[time] then
    rooms[rm1].isFree[time] := false
    rooms[rm2].isFree[time] := false
    print("Succeeded")
  else print("Sorry, already booked")
  endif
  UNLOCK(rooms[rm2].lock)
  UNLOCK(rooms[rm1].lock)
end BookRoomPair
```

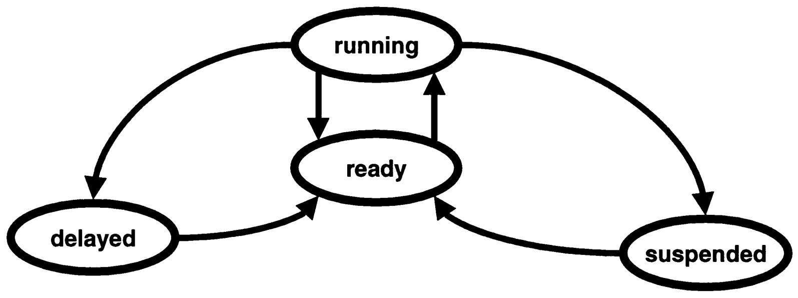
Suppose two users, A and B, attempt to book a pair of rooms at the same time. Using an example, explain what might go wrong, and why.

d Suggest a way to avoid the problem with the procedure in part (c).

(The four parts carry, respectively, 25%, 25%, 25% and 25% of the marks).

3 This question concerns processes in a simple operating system kernel supporting multiple processes, semaphores, interrupts and pre-emptive scheduling, as presented in the lecture course.

a Each process in this simple kernel can be in one of four different states, as shown in the diagram below:



(i) What event would cause a process to change from Running to Ready? Why does this happen?

(ii) What event would cause a process to change from Running to Suspended? Why does this happen?

b Give an example of why an interrupt handler might call the `V` operation.

c What would happen if the `P` operation were called in an interrupt handler?

(The four parts carry, respectively, 25%, 25%, 25% and 25% of the marks).

4 This question concerns linking and loading.

a Consider the following program, written for the very simple computer as introduced in the lecture course. The machine code is shown on the left, the assembler code on the right:

Location	*Contents*		*Assembly code representation*	
0	23		`A`	`.word 23`
2	45		`B`	`.word 45`
4	2	2	`myproc`	`loadm B`
6	6	12		`addc 12`
8	3	2		`storem B`
10	2	0		`loadm A`
12	6	1		`subc 1`
14	3	0		`storem A`
16	13	0		`jmpnz myproc`
18	12	0		`ret`

(i) Write down the symbol table which the assembler would need to construct. Assume (as shown above) that assembly starts at location 0 and that each instruction and each word occupies two bytes.

(ii) Suppose that instead of starting at 0, the program has to start at some other position in memory. List the locations which would require relocation.

b (i) What is an external symbol table used for?

(ii) Why would a label *not* appear in the external symbol table?

(iii) List the symbols in the program above which should be external for the procedure to be useful.

c Is relocation needed in an operating system which uses address translation hardware to support paging? Justify your answer carefully.

(The six parts carry, respectively, 25%, 25%, 10%, 10%, 5% and 25% of the marks).