# UNIVERSITY OF LONDON
## IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

## EXAMINATIONS 1997

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing 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*

PAPER 2.5

OPERATING SYSTEMS II

Wednesday, April 30th 1997, 2.00 - 3.30

*Answer THREE questions*

For admin. only: paper contains 4 questions

1a Describe in a few lines the effect of the Unix system calls `fork` and `exec` and how they interact.

b What are the essential differences between *threads* and *processes.*

c Explain in brief terms why the *server paradigm* solves the problem of mutual exclusion.

d What is the main rôle of the scheduler? Give the situations in which it will be invoked in an operating system that uses a message passing mechanism for communication between processes, and a round-robin scheduling algorithm.

The four parts carry each 25% of the marks

2a In MINIX, asynchronous sends are used for message passing, and only one message can be stored for each process. Although no process can take the scheduler if a task is running, and user processes cannot send messages to each other, explain why in some situations messages can get lost.

b In the process administration of the Unix-like operating system MINIX, process 0 (zero) corresponds to the pseudo-process `HARDWARE`, that serves as the source of the messages which are sent as the direct result of an interrupt. Although this pseudo-process has a number, it does not have a slot in the Process Table (hence the predicate 'pseudo'). Explain.

c In MINIX, systemcalls are implemented using `SendRec`, forcing the user process to explictly wait for the result from the requested action to arrive. Kernel processes, however, can send messages without having to wait explicitly for an answer. Although this sounds reasonable in the communication between user processes and the kernel, it is as such not a safe protocol, since deadlocks can occur. Explain why this creates no problem in the communication between user processes and the kernel, and how the MINIX scheduler avoids getting into a deadlock.

The three parts carry, respectively, 20%, 40% and 40% of the marks

Turn over/...

3a Assume that an operating system supports the following message passing primitives:

`send (P, msg)` - an asynchronous send, where the sender continues after sending message `msg` to destination process `P`.

`receive (S, msg)` - the receiver is blocked waiting for a message from process source `S`, if the message is not available

`source := receiveany (msg)` - the receiver is blocked waiting for a message, from any process. `source` indicates the sender of the message.

The system also provides a call `install (inthandler, vector)` to allow a procedure to handle interrupts from a specified vector.

A driver process for an analogue-to-digital (A to D) converter device starts a conversion by setting a device control register, when a request to read a value is received. When the device has completed the conversion it generates an interrupt. The driver reads the digital value from the device data registers and writes it into an address specified in the request.

Give an outline implementation of the A to D driver process and its interrupt handler. Assume the driver handles only one request at a time and does not perform retries for errors.

b Describe the 3 main states of a process in an operating system and the additional process states needed to support swapping of processes between main memory to backing store. This may occur when the system is short of main memory to run all the user processes.

Give a state diagram to show the main state changes and what actions typically cause state changes to take place.

The two parts carry, respectively, 60% and 40% of the marks

4 A kernel of an operating system implements asynchronous message passing and provides the following message primitives using ports. A port is a message queue belonging to the recipient, i.e. only a single process can receive messages from a particular port, but a process may have multiple ports from which it receives messages at different times. Multiple processes can send messages to a port.

```
receive (portid,  msg1)
```

where `portid` is the unique identifier for a port belonging to the receiver process and `msg1` is a pointer to a variable in the receiver into which the message is to be copied. The receiver blocks if there are no messages queued on the port, else the first queued message is received and removed from the port.

```
send (portid,  msg2)
```

where `portid` is the identifier of the message queue for a destination process and `msg2` is a pointer to the message to be sent. The sender continues irrespective of the state of the receiver.

a Explain why the kernel needs to provide buffering for messages.

b Give an outline implementation for the `send` primitive. Assume the kernel executes with interrupts disabled.

c How can a process receive a response to a query? A printer process which receives requests from user processes to use the printer and then receives a page at a time as a single message to print. Give a psuedocode outline of the printer process using the above message primitives showing how it interacts with the user process and indicating what ports it uses. Assume it prints a page by means of a call print (page).

The three parts carry, respectively, 20%, 40% and 40% of the marks

End of paper