1a
--

Lecture 2. slide 32

convenient: no need for rebooting for newly added modules
efficient: no need for message passing unlike microkernel
flexible: any module can call any other module unlike layered model

1b
--

Lecture 3 slides 11 and 12

The act of switching the CPU from one process to another
is called a context switch.

lecture 8 slide 2

Choosing which process to run next is called scheduling

scheduling is a policy, context switching is a mechanism

1c
--

lecture 5 slide 7

A program has a race condition (data race) if the result of an
executing depends on timing i.e. is non-deterministic

1d
--

Deadlock may occur with all lefties due to circular wait. If each
picks up his left fork, then the right fork will be held by another
philosoher who waits such that the fork will never be released This
leads to a circular wait and deadlock. The same is true for all righties.

Assume the case with all lefties and one rightie. The philosopher to
the left (L) of the rightie (R) picks up his left fork while the
rightie attemps to pick up his right fork. If R fails to do this, R
will wait with neither fork held. L is then free to pick up his right
fork and continue.

If R succeeds in acquiring his right fork, then both L and R will then
attempt to hold the same fork. One of them will succeed and
continue. This means that aleast one philosepher will be able to
progress. There is no circular wait and no deadlock. The same argument
hold for more than one rightie. As long as there is diversity, then
there is no deadlock.

1e)i)

Circular wait deadlock occurs when one thread holds a lock and wishes to aquire
a second one while another thread holds the second lock and wishes to aquire the first

Let x mean that a thread hold lock x, (x) means that it attempts to
hold lock x

so if one thread has the form x(y) and the other has the form y(x)
then we have deadlock, otherwise we do not

thread 1 has the following behaviour
(a), a, a(b), ab, b, b(c), bc, b,

while thread 2 has

(b), b, b(c), bc, c, c(a), ca,c

the patter x(y), y(x) deos not occur so there is no deadlock

1e)ii)
---

t1 has the pattern

(h), h, h(i), hi, hi(j), hij, hi,i,i(g),ig,i

t2 has the pattern

(g), g,g(i),gi,g,

As ti has the pattern i(g) and t2 g(i) then there is deadlock possible if
t1 tries to acquire g after t2 has acquired g and tries to acquire i.

As the update in thread is to i and g, then swapping the lock aquire release
around will give the same answer but prevent dealock:

```
lock(i)
lock(g)
i=g+g
unlock(g)
unlock(i)
```