Assignment: Concurrency & Control

1)

(a) Three forms of concurrent computation:

- Multiprogramming: the processes multiplex their execution on a single processor
- Multiprocessing: the processes multiplex their execution on tightly coupled processors (processors sharing memory)
- Distributed processing: the processes multiplex their execution on loosely coupled processors (not sharing memory)

(b)

```
semaphore sem = 0
shared int v

taskZ() {
   // Compute value v
   v = compute_v()
   signal(sem)
}

taskX() {
   wait(sem)
   // Use value v
   resultX = use_v(v)
}

taskY() {
   wait(sem)
   // Use value v
   resultY = use_v(v)
}
```

(c)

- Signalling: Semaphores allow tasks to wait for a signal before proceeding and ensure synchronisation without a busy-wait. This reduces the CPU usage as tasks block until the semaphore is signalled
- Concurrency Control: Semaphores are able to control access resources more flexibly than locks, especially when multiple tasks need to be synchronised. In addition, locks are binary, whereas semaphores can manage multiple signals, allowing multiple tasks to proceed when the condition is met

(d)

Pool: A pool is a collection of pre-allocated resources (like threads or connections) that are managed for efficient reuse. Pools also prevent the overhead from creating and destroying resources constantly.
For example, a thread pool reuses a fixed number of threads to execute tasks, which reduces the overhead associated with thread creation and destruction.

Channel: A channel is a communication pathway to pass messages between concurrent tasks. Channels can be used for synchronous or asynchronous message passing. For example, in Go, channels are used to send and receive data between goroutines, enabling communication and synchronisation

(e)

```
counting_semaphore sem = 0
shared int sensor_value

taskZ() {
   while (true) {
      // Read sensor value
      sensor_value = read_sensor()
      signal(sem)
      wait_period() // Wait for the next period
   }
}

taskX() {
   while (true) {
      wait(sem)
      // Compute command using sensor_value
      command = compute_command(sensor_value)
      send_command(command)
   }
}
```

2)

(a)

For first half of the loan:

$$x_{1(k+1)} = (1.04 \times 25000) - 500$$

For second half of the loan:

$$x_{2(k+1)} = (1.08 \times 50000) - 500$$

(b)

For $x_0 = 25000$:

$$x_{1k} = 1.04^k \times 25000 - \sum_{m=0}^{k-1} 1.04^{k-1-m} \times 500$$

$$x_{16} = 1.04^6 \times 25000 - (1.04^5 + 1.04^4 + 1.04^3 + 1.04^2 + 1.04^1 + 1.04^0) \times 1500$$

$$= 28316.50$$

For $x_0 = 50000$:

$$x_{2k} = 1.08^k \times 50000 - \sum_{m=0}^{k-1} 1.08^{k-1-m} \times 1500$$

$$x_{26} = 1.08^6 \times 50000 - (1.08^5 + 1.08^4 + 1.08^3 + 1.08^2 + 1.08^1 + 1.08^0) \times 1500$$

$$= 75675.75$$

$x_{16} + x_{26} = 28316.50 + 75675.75 = 103992.25$
Therefore, the total balance after 6 months is approximately £103,992.25

(c)
Observability of a dynamic system:
It is when you have LTI system with output measurements

$$x_{k+1} = \bar{A}x_k \qquad \dot{x} = \bar{A}x$$

$$y_k = \bar{C}x_k \qquad y = \bar{C}x$$

And unknown state condition x_0. If we can figure out x_0 (assuming we know A and C) from the output measurements, then we can find x_k for all k.

For example, in a discrete time LTI system:

$$y_0 = \bar{C}x_0$$
$$y_1 = \bar{C}x_1 = \bar{C}\bar{A}x_0$$
$$y_2 = \bar{C}x_2 = \bar{C}\bar{A}x_1 = \bar{C}\bar{A}^2x_0$$
$$\vdots$$
$$y_{n-1} = \bar{C}x_{n-1} = \bar{C}\bar{A}^{n-1}x_0$$

$$\Rightarrow \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} \bar{C} \\ \bar{C}\bar{A} \\ \bar{C}\bar{A}^2 \\ \vdots \\ \bar{C}\bar{A}^{n-1} \end{pmatrix}}_{\text{observability matrix } H} x_0$$

Where n denotes order of the system

In addition, for a system to be observable, we solve for x_0. We do this by checking if the determinant of H is not equal to zero and is non-singular.

(d)

$$x_{k+1} = Ax_k + Bu_k$$

$$u_k = k^T x_k \quad , \quad k = (-0.3,\ 0)^T$$

$$A = \begin{pmatrix} 1.04 & 0 \\ 0 & 1.08 \end{pmatrix} \qquad B = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$$

sub u_k into x_{k+1}:

$$x_{k+1} = Ax_k + Bkx_k$$

$$x_{k+1} = (A + BK)x_k$$

$$A + BK = \begin{pmatrix} 1.04 & 0 \\ 0 & 1.08 \end{pmatrix} + \begin{pmatrix} -1 \\ -1 \end{pmatrix}(-0.3 \quad 0)$$

$$= \begin{pmatrix} 1.34 & 0 \\ 0.3 & 1.08 \end{pmatrix}$$

Eigenvalues for stability:

$$\det(A + BK - \lambda I) = 0$$

$$\begin{vmatrix} 1.34 - \lambda & 0 \\ 0.3 & 1.08 - \lambda \end{vmatrix} = 0$$

$$(1.34 - \lambda)(1.08 - \lambda) = 0$$

eigenvalues: $\lambda_1 = 1.34$

$\lambda_2 = 1.08$

For the system to be stable, the eigenvalues must lie within the unit circle (i.e. their magnitudes must be less than 1). However, both values are greater than 1, meaning that the system is unstable under this payment policy.

(e)

Given the instability of the system, Buyer B's payment policy is not affordable. The loan balances will continue to increase rather than decrease, resulting in an unsustainable financial situation. Buyer B would need to adjust the payment policy to ensure that the system remains stable, which would involve ensuring that the eigenvalues of the matrix A+BK lie within the unit circle.