1. **Diga o que entende por:**
    a. **Programa;**
    b. **Linguagem de Programação;**
    c. **Processos;**
    d. **Multi-programação;**
    e. **Multi-Processamento;**

    a) É uma sequência de instruções a serem seguidas e/ou executadas, na manipulação, redireccionamento ou modificação de um dado/informação ou acontecimento.
    b) Uma linguagem de programação consiste em uma série de instruções para que o processador execute denominadas tarefas.
    c) Um processo é basicamente um programa em execução, sendo este constituído pelo código executável e respectivos dados, pilha de execução, contador de programa, valor do apontador da pilha, valores dos registadores de hardware, alem do conjunto de outras informações necessárias à execução do programa.
    d) Divide-se a memória em diversas partes com um Job alocado em cada uma delas, enquanto um Job espera a conclusão da sua operação I/O, um outro Job pode estar a utilizar o processador, a ideia é manter na memória simultaneamente uma quantidade de jobs suficiente para ocupar o processador a 100% e aproveitar ao máximo.
    e) Capacidade de um sistema operacional executar simultaneamente dois ou mais processos. Pressupõe a existência de dois ou mais processadores. Difere da multitarefa, pois esta simula a simultaneidade, utilizando-se de vários recursos, sendo o principal o compartilhamento de tempo de uso do processador entre vários processos.

2. **Analise as diferenças entre os dados que são tipicamente registados para cada Processo e os dados registados para cada Thread. Como é que essa diferença influencia a gestão de Processos/Threads e a Gestão de memória dum Sistema Operativo.**

A cada novo processo é necessário alocar um espaço de endereçamento, um contador do programa, variáveis. No caso de uma nova thread, não é preciso alocar espaço de endereçamento, sendo apenas necessário um contador e uma linha de controlo já que o escalonamento das threads é feito pelo processo que a cria.

3. Explique para que diferentes fins podem servir as várias técnicas de Multi-Programação que estudou (semáforos,mutexes,monitores,etc…). Exemplifique, com o auxílio de pseudo-código, a utilização de semáforos para solucionar o problema conhecido como Barbeiro Dorminhoco (Sleeping Barber), explicando o seu funcionamento.

Semáforos: E uma variável inteira, pode ter valor 0, indicando que não há nenhum sinal armazenado, ou pode ter valor positivo, indicando o número de sinais armazenados

Barreira: Mecanismo de sincronização que é usado por grupos de processos. Algumas aplicações são divididas em fases e têm por regra que nenhum processo poderá prosseguir para a próxima fase ate que todos os processos estejam prontos para a próxima fase. Este procedimento e alcançado colocando uma barreira no final de cada fase.

Pipes: Um pipe permite a comunicação num só sentido entre dois processos, os dois processos e a pipe devem ter um ancestral em comum. Ambos os processos podem ler ou escrever do pipe. Cabe ao programador definir o sentido da comunicação. Comunicação bidireccional precisa de dois pipes. Leitura de uma mensagem de um pipe vazio bloqueia o processo até que lá seja colocada uma mensagem.

Mutex: Assegura que o consumidor e o produtor não acedem ao buffer em simultâneo. É incializado a 1.

FIFOS: Permite a comunicação entre processos sem qualquer relação de parentesco. Comunicação realizada por um canal de comunicação permanente e acessível a qualquer processo, suportado por um ficheiro especial.

Barbeiro Adormecido: Usa-se três semáforos, costumers, que conta o número de clientes à espera, barbers, o número de barbeiro livres à espera de clientes e o mutex usado para exclusão mutua, e uma variável wating que também conta os clientes que estão à espera. Não há loop para o processo cliente, mas o barbeiro deve estar em loop tentando apanhar o próximo cliente.

```
#define CHAIRS 5                    /* # chairs for waiting customers */

typedef int semaphore;              /* use your imagination */

semaphore customers = 0;            /* # of customers waiting for service */
semaphore barbers = 0;              /* # of barbers waiting for customers */
semaphore mutex = 1;                /* for mutual exclusion */
int waiting = 0;                    /* customers are waiting (not being cut) */

void barber(void)
{
     while (TRUE) {
          down(&customers);         /* go to sleep if # of customers is 0 */
          down(&mutex);             /* acquire access to 'waiting' */
          waiting = waiting - 1;    /* decrement count of waiting customers */
          up(&barbers);             /* one barber is now ready to cut hair */
          up(&mutex);               /* release 'waiting' */
          cut_hair( );              /* cut hair (outside critical region) */
     }
}


void customer(void)
{
     down(&mutex);                  /* enter critical region */
     if (waiting < CHAIRS) {        /* if there are no free chairs, leave */
          waiting = waiting + 1;    /* increment count of waiting customers */
          up(&customers);           /* wake up barber if necessary */
          up(&mutex);               /* release access to 'waiting' */
          down(&barbers);           /* go to sleep if # of free barbers is 0 */
          get_haircut( );           /* be seated and be serviced */
     } else {
          up(&mutex);               /* shop is full; do not wait */
     }
}
```

4. Questão
    a. Defina Deadlock ou impasse (interblocagem).
    Um deadlock é quando os processos bloqueiam e não conseguem desbloquear. Ocorre tipicamente quando dois ou mais processos que precisam de informação um do outro, e tal informação ainda não está disponível. Logo é criado um ciclo infinito, que faz com que os recursos em uso pelos processos em questão não sejam libertos.
5. Surge de recursos non-preemptable.

    a. Considerando as várias estratégias possíveis para solucionar um impasse num Sistema Operativo, qual seria a estratégia a adoptar se tivesse o objectivo de anular a hipótese de ocorrer um impasse. Exemplifique explicando uma solução possível de acordo com a estratégia anteriormente adaptada.
        > Através de rollback
        Para isto é necessário gravar periodicamente os estados dos processos. Assim quando se chega a um deadlock, é só voltar atrás usando a informação guardada, até que seja possível arranjar recursos para esse processo.