1. **Diga o que entende por Sistema Operativo, quais os seus objectivos, como pode ser classificado, e em que partes pode ser decomposto.**

É o software que gere os recursos do computador e serve de base para as restantes aplicações. Apresenta a complexidade do Hardware com uma interface simples de entender e de programar. É conhecida como máquina virtual. O sistema operativo é aquela porção de software que corre em modo Kernel.

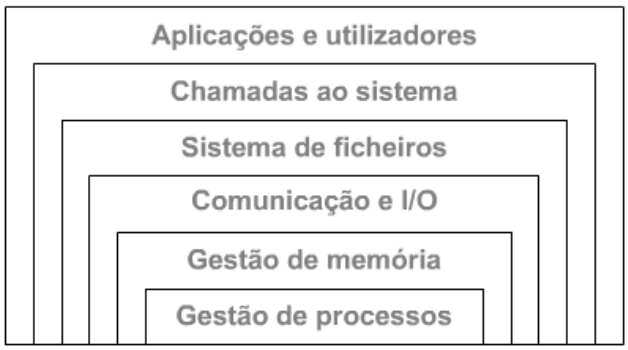
Os seus objectivos são:

- Executar programas do utilizador e tornar mais fácil a resolução de problemas.
- Tornar fácil o uso da máquina.
- Utilizar o hardware do computador duma forma eficiente

Podem ser classificados em:

- Multi-utilizador: o tempo de processamento do cpu de um computador pode ser partilhado por mais do que um utilizador de forma interactiva.
- Mono-utilizador: O CPU só pode estar dedicado de forma interactiva a um conjunto de processos do mesmo utilizador.
- Multi-programação: Capacidade de correr vários programas em simultâneo.
- Mono-programação: um programa a correr de cada vez.
- Dedicado: sistema operativo projectado para aplicações específicas.
- Uso geral: projectados para uma fácil utilização, permitem a execução de uma grande variedade de programas e reconhecem uma grande diversidade de periféricos.
- Centralizado: O SO cria uma máquina virtual sobre o único computador;
- Distribuído: o SO corre sobre um conjunto de computadores, dado a ilusão de que este conjunto é uma entidade única.

Pode ser decomposto em: gestão de processos, gestão de memória, comunicação e I/O, sistema de ficheiros, chamadas ao sistema, aplicações e utilizadores.



2. **Enuncie as duas principais diferenças entre "Processo" e "Thread" no que diz respeito à gestão de processos e threads dum Sistema Operativo e no que diz respeito à gestão de memória.**

Cada processo tem o seu próprio program counter, stack, register set e espaço de endereçamento os processos não têm nada a haver uns com os outros.

Em muitos aspectos as threads são como mini processos, cada thread tem o seu próprio programa counter e stack para saber o que fazer. Threads partilham o CPU exactamente como os processos o fazem, só num multiprocessador é que eles correm em paralelo. Threads podem criar threads filho e bloquear à espera de um system call. Mas as threads não são tão independentes umas das outras como os processos são, todas as threads têm o mesmo espaço de endereçamento, o que significa que partilham as mesmas variáveis globais e podem limpar a stack de uma outra thread. Não existe protecção entre threads porque é impossível e não é necessária. Mas basicamente uma thread funciona exactamente como um processo.

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 Leitor-Escritor, 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.

Leitores e Escritores (Resumo): Pode haver mais que um processo a ler da base de dados, mas se um processo estiver a escrever, mais nenhum pode estar a ler e/ou a escrever.

Solução: O primeiro leitor a ter acesso À BD executa um Down no semáforo db. Os leitores seguintes só incrementam um contador RC, antes de aceder À DB. Cada leitor que deixar a bd decrementa o RC, e o último leitor a sair executa um UP na BD (vendo que a variável db é igual a 0), permitindo que um escritor bloqueado (se houver algum), tenha acesso à BD.

Os leitores têm prioridade sobre os escritores. Se um escritor aparecer quando houverem leitores, o escritor espera.

```
typedef int semaphore;                  /* use your imagination */
semaphore mutex = 1;                    /* controls access to 'rc' */
semaphore db = 1;                       /* controls access to the database */
int rc = 0;                             /* # of processes reading or wanting to */

void reader(void)
{
     while (TRUE) {                     /* repeat forever */
          down(&mutex);                 /* get exclusive access to 'rc' */
          rc = rc + 1;                  /* one reader more now */
          if (rc == 1) down(&db);       /* if this is the first reader ... */
          up(&mutex);                   /* release exclusive access to 'rc' */
          read_data_base( );            /* access the data */
          down(&mutex);                 /* get exclusive access to 'rc' */
          rc = rc – 1;                  /* one reader fewer now */
          if (rc == 0) up(&db);         /* if this is the last reader ... */
          up(&mutex);                   /* release exclusive access to 'rc' */
          use_data_read( );             /* noncritical region */
     }
}


void writer(void)
{
     while (TRUE) {                     /* repeat forever */
          think_up_data( );             /* noncritical region */
          down(&db);                    /* get exclusive access */
          write_data_base( );           /* update the data */
          up(&db);                      /* release exclusive access */
     }
}
```

4.

a. **Defina deadlock ou impasse (interblocagem).**
b. **Considerando as várias estratégias possíveis para solucionar um impasse num SO, qual seria a estratégia a adoptar se tivesse o objectivo de reduzir o custo do dano causado pelo impasse. Exemplifique, explicando uma solução possível de acordo com a estratégia anteriormente adoptada.**

a)

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.
Surge de recursos non-preemptable.

b) Considerando as várias estratégias possíveis para solucionar um impasse num SO, qual seria a estratégia a adoptar se tivesse o objectivo de reduzir o custo do dano causado pelo impasse. Exemplifique, explicando uma solução possível de acordo com a estratégia anteriormente adoptada.

> 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.