Nota: Este exemplo corresponde a um enunciado real. No enanto, o ***número total de questões***, a ***proporção de cotação dada*** a cada uma, e a ***proporção dada cada tipo de matéria*** pode variar. O ***formato*** (parte única *vs.* duas partes com intervalo) também pode variar. A ***matéria que sai*** depende do funcionamento de cada ano e não dos exemplos de enunciados.

1. 10% Considere a arquitetura de sistema *micro-kernel* (é a que o Windows utiliza). Indique se esta arquitetura é muito estável ou pouco estável e explique a razão de ser dessa estabilidade (ou falta dela).

2. O código abaixo representa um programa que procura e mostra números primos. A *thread* procura identifica os números e a *thread* mostra imprime esses números.
a) 10% Identifique eventuais partes menos boas no código tendo em atenção que a matéria envolvida é de SO e não de IP nem P.
b) 10% Resolva essas partes menos boas utilizando funções genéricas tal como nas aulas teóricas, ou usando as folhas de consulta autorizadas. Tem que manter a procura e a apresentação em *threads* diferentes.

```
/* vars globais */
int novo=0;
int primo=0;
```

```
/* Na thread procura */
for (int i=0; i<1000000; i++) {

   if (ePrimo(i)) {
      novo=1;
      primo=i;
   }
}
novo=1;
primo=-1;
```

```
/* Na thread mostra */
while (1) {
   while (novo==0);
   if (primo==-1)
      break;
   printf("%d", primo);
   novo=0;
}
```

3. 10% Considere uma máquina com 10 Kb de RAM de memória paginada (paginas de 2 Kb) em que o sistema simula a existência de mais memória à custa de um *page file*. O algoritmo de substituição de páginas é o NRU. A tabela descreve a totalidade da memória no instante t0. No instante t1 ocorrem a leitura de posições que se encontram nas páginas 0, 2 e 4. No instante t2 surge a necessidade de encaixar uma nova página em memória. Descreva o que acontece da sequência desse pedido.

	Base	Prot	Priv.	P	R	M
0	2048	r-x	3	1	0	1
1	0	rw-		1	0	0
2	4096	r-x	0	1	1	1
3	6144	r-x	3	1	0	1
4	8192	rw-		1	0	0

4.

(5%) Utilizando apenas uma linha de comandos, elimine todos os named pipes existentes na sua *homedir* (directoria pessoal). A resposta não deve depender da directoria em que se encontra actualmente.

5.

(5%) Utilizando apenas uma linha de comandos, coloque no ficheiro "resultado.txt" o nome de todos os ficheiros que existem na pasta "/tmp" que tenham extensão ".txt" e pertencentes ao utilizador com o *username* (*login)* "antonio". A lista de ficheiros deve estar organizada por ordem crescente do tamanho dos ficheiros.

6.

(25%) Pretende-se uma aplicação para pesquisar N palavras num ficheiro de texto (o ficheiro é grande e a pesquisa demorada). Para tornar o processo mais rápido, a sua aplicação deverá criar N threads, sendo cada uma responsável pela pesquisa de uma das N palavras. Assuma que dispõe da função (já feita) *pesquisa()* que recebe o nome do ficheiro e uma palavra a pesquisar, e devolve um valor inteiro com o número de ocorrências dessa palavra no ficheiro. Após determinar o número de ocorrências da palavra, cada *thread* deverá incrementar um contador geral, usado para guardar a soma de todas as ocorrências das N palavras. No final, após a conclusão de todas as pesquisas, a aplicação deve mostrar no terminal o valor do contador geral. O nome do ficheiro a pesquisar é especificado através da variável de ambiente FICHEIRO.

```
#define NPALAVRAS 3
char palavras[NPALAVRAS][10] = {"Lisboa", "Porto", "Coimbra"};
```

NOTA: Deve incluir todo código necessário, à exceção das diretivas #include.

7.

(25%) Pretende-se uma aplicação **cliente** para um **servidor já existente** que transmite ficheiros a pedido. O servidor recebe uma mensagem inicial no *named pipe* "serv_files" com o nome do ficheiro pretendido e, eventualmente, mais informação que entenda necessária (e que deve especificar). Como resposta, envia ao cliente o nome de um segundo e terceiro *named pipe* (que são criados e geridos pelo servidor) e dois inteiros (NumP e TamP). De seguida transmite o ficheiro sob a forma de NumP partes de tamanho TamP cada uma, escrevendo no segundo pipe NumP mensagens de TamP bytes cada. O cliente deverá ler essas mensagens, escrevendo as sucessivas partes recebidas num ficheiro local com o mesmo nome. Por cada parte lida o cliente deverá escrever no terceiro *named pipe* um de dois valores inteiros: 1 (que sinaliza ao servidor que deve enviar a próxima parte do ficheiro); 0 (que sinaliza ao servidor que alguma operação do cliente retornou erro e que o servidor deve proceder ao reenvio da respetiva parte).

Assuma que TamP nunca excede 1024 caracteres (podendo ser menos). Complete a especificação das mensagens trocadas, definindo as estruturas envolvidas, e implemente o **cliente**, mandando-o transferir o ficheiro "Ficheiro-versao.bin".

NOTA: Escreva todo o código do cliente, à exceção das diretivas #include. O servidor já existe.