Prova com consulta apenas da documentação fornecida **Exemplo de perguntas de Exame**
Duração: 1,5 horas **XX.Junho.2021**
Cotação máxima: XX pontos ; peso na nota final da disciplina: 50%
Estrutura da prova: escolha múltipla

Utilização: para cada pergunta só há uma resposta correcta; indique-a (com a letra correspondente) na folha de respostas, completando uma tabela semelhante à que se segue; se não souber a resposta correcta, nada preencha ou faça um traço nessa alínea.

Cotação: cada resposta certa vale 1 ponto; cada resposta errada vale – 0,5 ponto (note o sinal menos!); cada resposta ambígua, ininteligível ou não assinalada vale 0 ponto. O total é XX pontos, que irão equivaler a 20 valores.

Se desejar, *poderá* fazer um pequeno comentário em alguma pergunta que lhe pareça ambígua. No caso de a sua resposta ser considerada errada, o seu comentário *poderá* ajudar a perceber a sua ideia e *poderá* minorar a penalização da classificação.

1	2	3					4						
		3.1	3.2	3.3	3.4	3.5	4.1	4.2	4.3	4.4	4.5	4.6	4.7

---

1. Um sistema operativo é um "simulador de máquina virtual". Isso quer dizer que:

A) facilita a programação de aplicações.

B) facilita a utilização de aplicações.

C) facilita o teste de aplicações.

2. Considere um sistema em que a memória, numa dada ocasião, apresenta os seguintes espaços vazios (holes), por ordem crescente de posição: 10K, 4KB, 20KB, 18KB, 7KB e 9KB. Nessa situação, foram feitos 3 pedidos sequenciais de memória: 12KB, 10KB e 9 KB, após o que a lista ordenada de espaços vazios ficou: 4KB, 8 KB, 9KB 7 KB e 9 KB. Qual foi o método de alocação utilizado?

A) Primeiro a servir (*first fit*).

B) Melhor a servir (*best fit*).

C) Pior a servir (*worst fit*)

3. Pretende-se implementar 2 processos, P1 e P2, que executem as suas tarefas (`doWork1()` ou `doWork2()`) de forma alternada, sendo P1 o primeiro a executar. Estes processos dependem de outro processo, P0, que arranca primeiro e que controla uma variável booleana (goOn). A sincronização dos processos deve ser efectuada por semáforos e as funções disponíveis são `init(sem,value)`, `wait(sem)` e `signal(sem)`.
Os esqueletos de código seguintes representam a situação e contém locais assinalados por '*.n*', *(*por exemplo, .2) que devem ser preenchidos apropriadamente.

P0:

```
( .1 )
goOn = TRUE;
// P1 and P2 are launched
// ...after some time...
goOn = FALSE;
```

P1:

```
while (goOn) {
   ( .2 )
   doWork1();
   ( .3 )
}
```

P2:

```
while (goOn) {
   ( .4 )
   doWork2();
   ( .5 )
}
```

Escolha, de entre as opções a seguir mostradas para cada local *.n*, a mais apropriada de forma a que, no conjunto, se obtenha uma solução para o problema.
SUGESTÃO IMPORTANTE: tente completar o código de cada local antes de escolher de entre os extractos mostrados!

.1 :

A)
```
init(sem1,1);
init(sem2,0);
```

B)
```
init(sem1,0);
init(sem2,1);
```

C)
```
init(sem1,0);
init(sem2,0);
```

.2 :

A)
```
wait(sem2);
```

B)
```
wait(sem1);
```

C)
```
wait(sem1); signal(sem2);
```

.3 :

A)
```
signal(sem2);
signal(sem1);
```

B)
```
signal(sem1);
signal(sem2);
```

C)
```
signal(sem2);
```

.4 :

A)
```
wait(sem2);
```

B)
```
wait(sem2); wait(sem1);
```

C)
```
wait(sem2); wait(sem1); signal(sem2);
```

.5 :

A)
```
signal(sem1); signal(sem1);
```

B)
```
signal(sem1);
```

C)
```
signal(sem1); signal(sem1); wait(sem1);
```

4. Pretende-se escrever um programa *multithread* que, recebendo da linha de comando um conjunto de nomes de ficheiros de texto, revela o ficheiro que tem o maior número de palavras. O programa é invocado como:

```
$ fileMaxWord file1 file2 file3 ...
```

e apresenta o resultado no seguinte formato:

```
file2 has maximum number of words (352).
```

a) `fileMaxWord` invoca tantos *threads* quantos os ficheiros a analisar, passando o nome de um ficheiro a cada *thread*, de função `counter()`, como é demonstrado no esqueleto de código abaixo, em que cada local assinalado por '.*n*' (por exemplo, .2) deve ser preenchido com código apropriado.

```
typedef struct {
   char *filename;
   pthread_t tid;  // id of the thread that counts words in 'filename'
} File;

void *counter(void *filename);  // thread function

int main(int argc, char *argv[]) {
   File *pFile = ( .1 ) malloc( .2 );
   int i;
   for (i = 0; i < argc-1; i++) {
      pFile[i].filename = ( .3 );
      pthread_create ( .4 );
   }
   {...}  // waits for threads to finish
}
```

Escolha, de entre as opções a seguir mostradas para cada local *.n*, a mais apropriada de forma a que, no conjunto, se obtenha uma solução para o problema.
SUGESTÃO IMPORTANTE: tente completar o código de cada local antes de escolher de entre os extractos mostrados!

.1 :

A) `File*`

B) `File`

C) `int*`

.2 :

A) `File*(argc-1)`

B) `sizeof(File)*(argc-1)`

C) `sizeof(int)*(argc+1)`

.3 :

A) `argv[i-1]`

B) `argv[i]`

C) `argv[i+1]`

.4 :

A) `&pFile[i].tid, NULL, counter, (void*) pFile[i].filename`

B) `&pFile[i].tid, counter, (void*) pFile[i].filename`

C) `pFile[i].tid, NULL, counter(void*), pFile[i].filename`

b) Cada *thread* obtém o número de palavras presentes no ficheiro invocando a função pré-existente `numWords()`. No esqueleto de código abaixo, cada local assinalado por '*.n*' (por exemplo, .2, deve ser preenchido com código apropriado.

```
int numWords(int fd);      // returns number of words in file with valid descriptor fd

void *counter(void *filename) {  // computes number of words in file 'filename'
   int *pnum = (int *) malloc(sizeof(int));
   int fdesc = open( .5 );
   if ( .6 )
      *pnum = -1;
   else
      *pnum = numWords(fdesc);
   return ( .7 );
}
```

Escolha, de entre as opções mostradas para cada local *.n*, a mais apropriada de forma a que, no conjunto, se obtenha uma solução para o problema.
SUGESTÃO IMPORTANTE: tente completar o código de cada local antes de escolher de entre os extractos mostrados!

.5 :

A) `(void*) filename, O_RDWR`

B) `(char*) filename, O_RDONLY`

C) `(char) filename, O_RDONLY`

.6 :

A) `fdesc < -1`

B) `fdesc != -1`

C) `fdesc == -1`

.7 :

A) `(void*) pnum`

B) `void pnum`

C) `(int*) pnum`