U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO | 2º ANO
EIC0027| SISTEMAS OPERATIVOS | 2016/2017 – 2º SEMESTRE

EXAME - ÉPOCA NORMAL | 2017-06-05

Com consulta. Duração: 2h15m

Nome do estudante: _____ **Código:** _____

1.

[1.5] Muitos dos "mecanismos" dos sistemas operativos modernos não poderiam ser implementados sem suporte adequado do *hardware*. Explique numa frase a importância de cada um dos seguintes "mecanismos" de *hardware*:

• DMA	
• instrução máquina Test&Set	
• *bit* de página válida/inválida	

2.

a) [1.5] Pretende-se implementar uma "barreira de 3 processos", isto é, garantir que cada processo espera, em determinado ponto da sua execução, que os outros 2 processos atinjam um certo ponto da respetiva execução, como ilustrado ao lado. Recorrendo a semáforos, indique como implementaria a parte do código de **P1**, **P2** e **P3** (a inserir em **A**, **B** e **C**, respetivamente) que garante esta forma de sincronização; indique também como e onde deve ser feita a inicialização dos semáforos. Considere que dispõe das seguintes funções que operam sobre semáforos: `init(sem,value)`, `wait(sem)` e `signal(sem)`.

P1
...
...
A: // espera que P2 chegue a B e que P3 chegue a C
...
...

P2
...
B: // espera que P1 chegue a A e que P3 chegue a C
...
...
...

P3
...
...
...
C: // espera que P1 chegue a A e que P2 chegue a B
...

b) [0.8] Mostre que a solução que propôs na alínea anterior não dá origem a *deadlocks*.

3.

[0.7] Comente a seguinte afirmação: "Um escalonamento do processador do tipo preemptivo favorece os processos *CPU-bound*.

Nome do estudante: _____ **Código:** _____

4.

a) [0.8] Num sistema de gestão de memória baseado em paginação, são usados endereços de 16 bits e páginas de 512 bytes. Dado o endereço lógico 0000010001111101, determine o número da página e o *offset* respetivo. Justifique brevemente.

Sabendo que a página foi carregada no quadro (*frame*) número 15, qual será o endereço físico correspondente?

b) [1.2] Diga como pode ser detetada a ocorrência de *thrashing* num sistema de computação.

Como se pode parar o *thrashing*?

É possível evitar a sua ocorrência? Em caso afirmativo, como? Caso contrário, por que não é possível?

5.

a) [0.8] Vários processos independentes pretendem registar, num ficheiro de texto, dados sobre as operações que vão executando. Explique por que é que o ficheiro deve ser aberto em "modo de apensamento" (O_APPEND) em vez de em "modo de escrita" (O_WRONLY).

b) [0.7] Explique o que seria necessário fazer se o "modo de apensamento" não estivesse disponível.

Nome do estudante: _____ **Código:** _____

Nota: nas questões seguintes apenas é necessário fazer tratamento de erros nos casos indicados explicitamente

6.

a) [1.2] Pretende-se gerar uma sequência pseudo-aleatória de *bits* recorrendo aos programas **gen_bit** e **gen_seq**. O programa **gen_bit** não recebe argumentos da linha de comandos e o seu *exit status* (0 ou 1) é um dos *bits* da sequência a gerar. O programa **gen_seq** (código ao lado) recebe como argumento da linha de comandos o número de *bits* que devem ser gerados, cria um processo **gen_bit** por cada *bit* a gerar e escreve em **stdout** a sequência de *bits* gerados.

Exemplo de uma possível invocação do programa:

```
>gen_seq 8
```

Complete os espaços livres no código ao lado e indique onde acrescentaria a instrução **printf()** que mostra a sequência gerada. A chamada **execl()** deve lançar em execução o programa **gen_bit** que está no diretório **/home/sope**.

```
1  // CÓDIGO DE gen_seq
2  int main(int argc, char *argv[]) {
3    int numbits = _____;
4    // aloca memória e inicializa-a a zero
5    char *bits = calloc(numbits + 1, sizeof(char));
6
7    for (int i = 1; i <= numbits; ++i) {
8      if (fork() == 0)
9        execl(_____);
10     else {
11       int ret;
12       wait(&ret);
13       strcat(bits, ret == 0 ? "0" : "1");
14     }
15   }
16
17   return 0;
18 }
```

Após a linha _____ acrescentar a instrução:

```
printf(_____);
```

b) [0.8] Considere que:
- o código de **gen_bit** é o que se apresenta ao lado;
- os processos **gen_bit**, criados por **gen_seq**, têm PIDs consecutivos.

Explique por que é que, nestas condições, praticamente não haverá aleatoriedade das sequências geradas.
Sugestão: considere as situações em que o primeiro processo **gen_bit**, criado por **gen_seq**, tem um PID par ou um PID ímpar.

```
// CÓDIGO DE gen_bit
int main() {
  return (getpid() % 2);
}
```

c) [1.0] Considerando as condições enunciadas em b), como seria possível aumentar a aleatoriedade das sequências geradas, modificando apenas o código de **gen_seq**? Para localizar as alterações, use a numeração das linhas do código fornecido em a). Explique por que é que, apesar de aumentada, a aleatoriedade continua a ser limitada.

d) [1.0] Considere que o código de **gen_bit** é o seguinte:

```
// CÓDIGO DE gen_bit
int main() {
  pid_t ppid = getppid();
  int r = generateRandomBit();
  if (r == 0) kill(ppid, SIGUSR1);
  else kill(ppid, SIGUSR2);
  return 0;
}
```

Indique as alterações a introduzir no código de **gen_seq** para lidar com a nova forma de comunicação do *bit* gerado por **gen_bit**; para localizar as alterações, use a numeração das linhas do código em a).

Nome do estudante: _____ **Código:** _____

7.

Considere o conceito, operação e programação de canais de comunicação simples (*pipes* e FIFOs).

a) [1.0] Que tipo de canal recomendaria para a comunicação bidirecional de *threads* ligados a processos distintos, eventualmente pertencentes a utilizadores distintos? Esquematize a arquitetura comunicacional.

b) [1.0] Escreva o código que um dos *threads* (à sua escolha) teria de executar para preparar os meios que indicou em a), até ao ponto de poder trocar dados com o outro *thread*. Faça o tratamento de erros.

c) [0.5] Escreva o código através do qual o *thread* cujo código escreveu em b) envia para o outro *thread* uma mensagem contendo o PID (*process identifier*) e o TID (*thread identifier*). Faça o tratamento de erros.

d) [0.5] Como poderia um dos *threads* aperceber-se que o outro *thread* decidiu terminar a comunicação?

e) [1.0] Escreva o código que um dos *threads* (à sua escolha) teria de executar antes de terminar, eliminando do sistema os vestígios das comunicações. Faça o tratamento de erros.

Nome do estudante: _____ **Código:** _____

8.

Pretende-se simular o funcionamento de um serviço que gere um concurso em que os concorrentes fazem chamadas telefónicas, sendo premiados os concorrentes cujas chamadas tiverem o número de ordem, 1000, 2000, 3000, etc. O serviço tem 10 funcionários que atendem as chamadas e um funcionário que telefona aos concorrentes premiados. A simulação deve ser feita recorrendo a um programa *multithreaded*, sendo os funcionários que atendem as chamadas simulados por uma mesma função (**receiver**) do programa e o funcionário que faz as chamadas para os concorrentes premiados por uma outra função (**caller**).

a) [1.0] Escreva o código que cria os 10 *threads* **receiver** e o *thread* **caller**. Cada *thread* **receiver** deve receber como parâmetro o seu número de ordem (um número sequencial, entre 1 e 10); o *thread* **caller** não tem parâmetros. O código deve recolher os TIDs de todos os *threads* criados.

b) [1.0] Apresenta-se a seguir o núcleo da função **receiver()**:

```
1  // MAX_NUM_PHONE_CALLS: constante global que indica o nº máx. total de chamadas
2  while (numPhoneCalls < MAX_NUM_PHONE_CALLS) {
3    // espera até que chegue uma chamada telefónica e "anota" o número de telefone
4    int phoneNumber = receiveCall();
5    numPhoneCalls++;
6    if (numPhoneCalls % 1000 == 0) {
7      // o concorrente cujo número de telefone é phoneNumber é um dos vencedores
8      // informa o thread caller que deve fazer uma chamada para phoneNumber
9    }
10 }
```

Explique que mecanismos de sincronização usaria para garantir que a contagem do número de chamadas recebidas (**numPhoneCalls**) é feita corretamente e que o *thread* **caller** é informado quando tem de fazer uma chamada para um concorrente premiado. Por simplificação, considere que o tempo necessário para receber 1000 chamadas é muito superior ao tempo necessário para fazer uma chamada para um dos vencedores. Não escreva código.

c) [1.0] Faça as declarações de variáveis/tipos e as inicializações necessárias para usar os mecanismos de sincronização que escolheu, indicando em que parte(s) do programa as colocaria.

d) [1.0] Complete o código da função **receiver()**, apresentado em b), usando os mecanismos de sincronização que escolheu. Indique a localização do código adicional usando a numeração das linhas.

FIM