Fundamentos de Sistemas de Operação

FCT UNL 2017/2018

Teste 2, versão A – 16/12/2017

1h45 *Sem consulta. Nas perguntas com múltiplas respostas, assinale a correta com X do lado esquerdo. As respostas erradas descontam 1/4 da cotação da pergunta.*

*Núm:*_____ *Nome:* _____

Q1-(0,6) A implementação de bibliotecas partilhadas dinâmicas (*shared libs*) baseia-se no mecanismo de mapeamento de ficheiros em memória. Escolha a frase verdadeira que indica uma vantagem face às bibliotecas estáticas.

- As bibliotecas dinâmicas crescem à medida das necessidades do programador poupando memória.
- Os ficheiros executáveis não necessitam de conter as bibliotecas e, o seu mapeamento durante a execução, permite a sua partilha entre processos que usem as mesmas bibliotecas.
- As bibliotecas, por serem dinâmicas, podem ser muito mais pequenas e poupar memória central.
- Os executáveis são mais fáceis de compilar e ligar às bibliotecas, evitando-se problemas de ligação (*linking*) em execução e permitindo alterar o código do programa durante a execução.

Q2-(0,6) Pretende-se que um programa crie (e abra para escrita) um ficheiro de texto que só pode ser alterado pelo utilizador que executa o programa e que impeça a futura alteração do ficheiro por quaisquer outros utilizadores. No entanto, deve permitir a leitura do ficheiro por utilizadores do mesmo grupo do utilizador que executa o programa. Indique qual seria o código utilizado nesse programa:

- `F = creat(nomeFich, S_IRWXU | S_IRGRP | S_IROTH);`
- `F = open(nomeFich, O_RDONLY|O_CREAT, 0777);`
- `F = creat(nomeFich, S_IRUSR | S_IWUSR | S_IRGRP);`
- `F = open(nomeFich, O_RDONLY|O_WRONLY|O_CREAT, S_IRUSR | S_IWUSR | S_IRGRP);`

Q3-(0,6) Considere um programa que pede vários recursos ao sistema de operação, o que inclui canais via *open* e reserva de memória via *malloc* (ou outra interface). Este programa nunca liberta esses recursos (não usa o *close*, nem o *free*). Indique o que acontece quando um processo que executa este programa termina e volta depois a ser executado, por diferentes processos.

- Quando o processo termina, esses recursos continuam ocupados no sistema. Em caso de novas execuções, o SO atribui ao novo processo novos recursos, podendo estes vir a esgotar-se.
- Quando o processo termina, esses recursos continuam ocupados no sistema. Em caso de novas execuções, o SO atribui ao novo processo os recursos que ficaram do processo anterior, incluindo o conteúdo da memória, para reutilização.
- Quando o processo termina, esses recursos são considerados pelo SO como livres, independentemente do processo fazer close e free ou não. No entanto, novas execuções podem ainda vir a obter os recursos anteriores, incluindo o conteúdo da memória, se ainda estiverem disponíveis.
- Quando o processo termina, esses recursos são considerados pelo SO como livres, independentemente do processo fazer close e free, ou não. Novas execuções terão novos recursos atribuídos.

Q4-(0,6) O escalonamento em sistemas multiprogramados baseia-se muitas vezes em variantes do Multi-Level Feedback Queue (MLFQ). Indique qual das frases seguintes define uma característica do MLFQ:

- Permite *starvation* dos processos CPU bound.
- Garante ausência de *starvation* porque aumenta periodicamente a prioridade dos processos IO *bound*.
- Implementa um sistema de prioridades para melhorar a resposta mas sem *starvation* de processos.
- Implementa um sistema de prioridades para melhorar a resposta mas com *starvation* de processos CPU *bound*.

Q5-(0,6) O escalonamento de processos, para usar o CPU, em Round-Robin significa o quê?

- Significa que os processos vão sendo executados sequencialmente, voltando-se ao primeiro após executar o último.
- Significa que se usa uma roleta para escolher o próximo processo a executar.
- Significa que se escolhe alternadamente entre os que têm menos tempo acumulado de execução.
- Significa que entre todos se escolhe o processo que se espera vir a demorar menos tempo (shortest).

Q6-(0,6) Os SO, no seu sistema de IO / Sistema de Ficheiros fazem cache dos blocos lidos dos discos para evitar repetir essas operação, que são demoradas. No entanto tal pode ter alguns problemas. Assinale qual dos seguintes problemas pode ocorrer.

- A informação num bloco em memória pode ficar desatualizada porque um processo escreveu no disco nesse bloco.
- No caso de um crash (p.ex. falta de energia), o conteúdo do disco pode não estar atualizado e levar a inconsistências no disco.
- A informação num bloco em memória pode ser alterada por um processo que não o devia fazer, levando a inconsistências no disco.
- No caso de crash (p.ex. "segmentation fault") de um processo que estava a usar um bloco de um ficheiro, este ficará sempre a ocupar espaço em memória.

Q7-(0,6) O SO, numa arquitetura que suporta paginação, pode melhorar o desempenho da criação de processos (p.e. fork) e poupar na utilização da memória real, partilhando o espaço de endereçamento entre processos idênticos. Indique o mecanismo que permite lidar com as alterações que cada processo fará no seu espaço (escritas) para manter o conceito base de memória privada para cada processo.

- As páginas são carregadas dinamicamente e, a pedido, quando escritas.
- As páginas podem ser *swapped out* e *swapped in*, a quando da sua escrita por cada processo.
- As páginas podem ser apenas para leitura até à escrita e, serem então copiadas (*copy-on-write*).
- As páginas são mantidas em diferentes TLB do CPU para cada processo.
- As páginas são logo copiadas pelo SO no *fork*, antes do novo processo começar a executar.

Q8-(0,6) Comparando dois volumes de discos, um RAID 1 (*mirror*) e outro RAID 5, com o mesmo número de discos, indique a afirmação verdadeira:

- O espaço no RAID 5 é idêntico ao no RAID 1, assim como o nível de redundância.
- O espaço no RAID 5 é inferior ao no RAID 1 mas com maior nível de redundância.
- O espaço no RAID 5 é superior ao no RAID 1 com idêntico nível de redundância.
- O espaço no RAID 5 é idêntico ao no RAID 1 mas o RAID 1 não tem qualquer redundância.

Q9-(0,6) Um SO oferece a noção de Memória Virtual, independente da memória realmente instalada no computador, suportada por espaço em disco (memória secundária de *swap*). O sistema pode entrar em *thrashing* quando:

- Existem poucos processos e o CPU fica constantemente desocupado durante as transferências de/para *swap*.
- Existem muitos processos que ocupam a memória real existente, mantendo o CPU sempre ocupado.
- Existem muitos processos *IO bound* e o CPU é pouco usado.
- Existem muitos processos ocupando toda a memória real existente obrigando a constantes transferências de/para *swap.*

Q10-(0,6) A chamada *open* de um ficheiro obriga o SO a consultar diversa informação, a qual pode estar em disco. Assinale os possíveis acessos ao disco num único *open*.
(nesta pergunta assinale **todas** as opções que ache que são verdadeiras)

- Lê o bloco contendo informação sobre os blocos ocupados no disco (p.e. bitmap).
- Lê o bloco contendo o inode da raiz do sistema de ficheiros.
- Lê o bloco contendo o inode do ficheiro que se quer abrir.
- Lê o bloco que tem todas as permissões dadas a todos os utilizadores.
- Lê o bloco que tem o código do SO para aceder ao sistema de ficheiros no disco.

Q11-(1,2) Considere o código ao lado.
a) Indique o segmento do espaço de endereçamento onde se encontra cada variável (.data, .text, .stack, .bss, heap):

`value` - _____

`pid` - _____

`p` - _____

`*p` (apontada por p e criada por *malloc*) – _____

b) Indique, das instruções seguintes, quais podem ser usadas para libertar a memória de variáveis que já não são necessárias:

- `free( pid );`
- `free( value );`
- `free( p );`
- `free( *p );`

```
#include (...)

int value;

int main(int argc, char *argv[]) {
  int pid;
  int *p;

  if (argc != 2) {
    fprintf(stderr, "use: <val>\n");
    exit(1);
  }
  p = (int*)malloc(sizeof(int));
  assert(p != NULL);
  (...)
  return 0;
}
```

Q12-(1,2) No trabalho prático 4 (sistema de ficheiros), usava-se um *inode* com 20 índices diretos para blocos e dois índices para tabelas simplesmente indiretas. Sabendo que se usaram blocos de 4 KBytes e que cada índice (número de bloco) ocupa 2 bytes, qual o tamanho máximo que um ficheiro pode ter neste sistema de ficheiros (em Kbytes)? (indique como calcula o número máximo de blocos num ficheiro e apresente os cálculos efetuados)

Núm. Blocos =

Tam. Máximo =

Q13-(1) Considere um sistema de escalonamento dos pedidos feitos a um disco que usa o algoritmo SCAN (ou do elevador). Indique a ordem de atendimento dos seguintes pedidos (em nº da pista) e o movimento total da cabeça do disco, em pistas percorridas, para satisfazer todos estes pedidos. Considere que, de início, a cabeça do disco está na pista 100.

45, 120, 3, 50, 278, 125, 445, 89, 689

ordem de atendimento =

número de pistas percorridas =

Q14-(1,3) Explique porque é necessária uma chamada ao sistema para mudar o nome a uma diretoria, em vez de se usar as chamadas *unlink* e *link* para obter esse mesmo efeito.

Q15-(1,3) Em arquiteturas com endereços virtuais paginados, o tamanho das tabelas de páginas pode ser um problema, ao ocuparem muito espaço em memória. Explique uma alternativa na implementação da tabela de páginas, que permita reduzir o espaço necessário.

Q16-(4) Pretende-se implementar um programa "***newls***" que lista o conteúdo de uma diretoria indicada na linha de comando, afixando apenas os ficheiros (ignora todos os outros nomes) e, para cada um, escreve o seu nome, tamanho, e o número de links (nomes) existentes para esse ficheiro.
Complete, com o seu código, o esqueleto do programa seguinte:

```
#include <stdio.h>
(...)
#define MAXC 2048

void syserror(char *err) {
  perror( err );
  exit(1);
}

void newls(char * dir);

int main(int argc, char * argv[] ) {
       char *dir = ".";

       if ( argc != 1 ) dir = argv[1];
       newls(dir);

       return 0;
}
```

```
void newls(char * dir)
{









 }
```

Q17-(4) Pretende-se implementar um programa para trocar entre si, de modo eficiente, dois bytes de um ficheiro, assumindo que a sua dimensão pode ser, p. ex., 1 G bytes. O esqueleto de código que se segue inclui já a implementação da abertura do ficheiro, inspeção da sua dimensão, bem como a invocação de uma função de nome "mswitch" que deve implementar essa troca. A função "mswitch" recebe como argumentos o descritor do ficheiro aberto, a sua dimensão em bytes, bem como o offset dos dois bytes no ficheiro cujo conteúdo deve ser trocado. O programa principal usa essa função para trocar o primeiro com último byte do ficheiro.

Complete a implementação da função **mswitch** de modo a cumprir os requisitos indicados em cima.

```
#include <stdlib.h>
(...)
void fatal_error(char *str){
  perror(str);
  exit(1);
}

/* mswitch -- given a file with a sequence of bytes, this function
   swaps the bytes located at offsets i and j
 */
void mswitch(int fd, int size, int i, int j) {
    if( i >= size || j >= size )
       fatal_error("Invalid offsets");
    unsigned char *bufp; /* ptr to the file's bytes */
    bufp =
}

int main(int argc, char *argv[])  {
    /* Check the required command line argument */
    if (argc != 2) {
       printf("usage: %s <filename>\n", argv[0]);
       exit(1);
    }
    int fd = open(argv[1], O_RDWR);   // aberto em modo de leitura e escrita
    if( fd < 0 ) fatal_error("open");

    struct stat stat;
    if( fstat(fd, &stat) != 0) fatal_error("fstat");

    // This invocation swaps the file's first and last bytes
    mswitch(fd, stat.st_size, 0, stat.st_size-1);

    close(fd);
    return 0;
}
```

ANEXO
funções e tipos úteis:

```
void *mmap(void *addr, int len, int prot, int flags, int fd, int offset)
int msync(void *addr, int len, int flags)
int munmap(void *addr, int len)
int open( char *fname, int flags,... /*int mode*/ )
int creat( char *fname, int mode )
int close( int fd )
int read( int fd, void *buff, int size )
int write( int fd, void *buff, int size )
int lseek(int fildes, int offset, int whence)
int stat(char *path, struct stat *buf)
int fstat(int fd, struct stat *buf)
DIR *opendir(const char *filename)
struct dirent *readdir(DIR *dirp)

struct dirent { inot_t d_ino;
                char d_name[NAME_MAX+1];
}

struct stat { dev_t st_dev;
              ino_t st_ino;
              mode_t st_mode;
              int st_nlink;
              uid_t st_uid;
              off_t st_size;
              time_t st_mtime;
          ... };
```

constantes e flags úteis:

```
NULL

O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_TRUNC

PROT_READ, PROT_WRITE,
MAP_SHARED, MAP_PRIVATE, MAP_ANON

S_ISREG(m)  /* regular file */
S_ISDIR(m)  /* directory */
S_ISFIFO(m) /* named pipe (fifo) */
S_ISLNK(m)  /* symbolic link */

#define S_IRWXU 0000700    /* RWX mask for owner */
#define S_IRUSR 0000400    /* R for owner */
#define S_IWUSR 0000200    /* W for owner */
#define S_IXUSR 0000100    /* X for owner */
#define S_IRWXG 0000070    /* RWX mask for group */
#define S_IRGRP 0000040    /* R for group */
#define S_IWGRP 0000020    /* W for group */
#define S_IXGRP 0000010    /* X for group */
#define S_IRWXO 0000007    /* RWX mask for other */
#define S_IROTH 0000004    /* R for other */
#define S_IWOTH 0000002    /* W for other */
#define S_IXOTH 0000001    /* X for other */
```