Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Sistemas Operativos, Verão de 2020/2021

Teste Parcial #1 / Teste Parcial #2 / Exame

Parte I, Duração – 1h 15m

ATENÇÃO: Responda às questões assinaladas com **«A»** separadamente das marcadas com **«B»**.

1) **«A»** [5] Considere uma unidade de gestão de memória com tabelas de tradução em três níveis, cada uma com um máximo de 2048 entradas e cada entrada contendo um *page frame number* de 34 bits e 11 bits de controlo. Cada tabela ocupa uma página de memória, embora a tabela do primeiro nível de tradução só possa utilizar os primeiros 128 *bytes*.

   Responda às seguintes questões, apresentando os cálculos e as deduções que suportam as respostas:

   a) [0.5] Quantos *bytes* ocupa uma *page table entry* (PTE), sabendo que esse número tem de ser uma potência inteira de 2?

   b) [0.5] Qual o tamanho de cada página de memória?

   c) [1.5] Quantos bits tem um endereço virtual?

   d) [1.5] Sabendo que, para cumprir uma única instrução de leitura de um valor da memória, foi necessário consultar os endereços físicos 0xC43B8018, 0x612AF3E30, 0xA5A5A8 e 0xA64CB35042C, qual era o endereço virtual utilizado na instrução?

   e) [1.0] Após o acesso à memória descrito na alínea d), foi necessário cumprir uma instrução de escrita no mesmo endereço virtual. Dessa vez, houve apenas um acesso de escrita ao endereço físico 0xA64CB35042C. Porquê?

2) **«B»** [3] Considere uma execução do seguinte programa e que não há outros processos do mesmo executável:

```
#define DTSIZE (1024*1024*2)

uint8_t data[DTSIZE];

void fill(uint8_t *vals, size_t sz,
          uint8_t v) {
  for(int i=0; i < sz; ++i)
    vals[i]= v;
}

uint32_t sum(uint8_t *vals, size_t sz) {
  uint32_t s = 0;
  for(int i=0; i < sz; ++i)
    s += vals[i];
  return s;
}
```

```
1.  int main() {
2.    fill(data, DTSIZE, 1);
3.    uint8_t *table = mmap(
          NULL, DTSIZE, PROT_WRITE,
          MAP_SHARED | MAP_ANONYMOUS, -1, 0);
4.    fill(table, DTSIZE/2, 2);
5.    if (fork() == 0) {
6.       fill(data, DTSIZE, 1);
7.       fill(table, DTSIZE/2, 4);
8.       return(0);
9.    }
10.   wait(NULL);
11.   fill(table + DTSIZE/2, DTSIZE/2, 2);
12.   printf("%d\n", sum(table, DTSIZE));
13.   return 0;
14. }
```

   Responda às questões, justificando as suas respostas:

   a) [2] Qual a linha na execução do programa onde a soma dos *resident set* privados dos processos é máxima? Qual o valor dos *resident set* partilhado e privado de cada processo nesse ponto?

   b) [1] Qual o *output* produzido pela execução do programa?

3) **«A»** [2] O *kernel* Linux inclui um mecanismo para libertar automaticamente as páginas de memória física (*page frames*) que não sejam acedidas durante algum tempo.

   a) [1] Considerando processadores da família Intel/AMD x86, como consegue o *kernel* distinguir as *page frames* acedidas das não-acedidas durante algum tempo?

   b) [1] O *kernel* Linux utiliza um esquema de gestão das *page frames* ocupadas baseado em duas listas. Explique sucintamente o papel destas listas e as transferências de *page frames* entre elas.

4) **«B»** [6] O código seguinte, apresenta a implementação do programa `pargrep`, que é equivalente à execução de `grep -H` sobre vários ficheiros, mas com as pesquisas nos vários ficheiros a decorrerem em paralelo em processo diferentes.

```
// FUNÇÃO A IMPLEMENTAR
void npgrep(const char * expr, const char * filenames[], size_t len);

// Compilado para o executável pargrep
// Exemplo de invocação:
//    pargrep simples texto1.txt texto2.txt texto3.txt
int main(int argc, char * argv[]) {
   npgrep(argv[1], argv + 2, argc - 2);
   return 0;
}
```

   a) [4] Escreva a função `npgrep`, que recebe como primeiro argumento uma expressão a pesquisar (`expr`) seguida de um *array* de nomes de ficheiros (`filenames`) e do comprimento desse *array* (`len`). Para cada nome de ficheiro (`filenames[i]`) é lançado um processo que deve executar o comando:

      `grep -H` *`expr filenames[i]`*

   Estes comandos executam-se em paralelo, com a função `npgrep` a esperar que todos tenham terminado. Para simplificar a implementação, pode-se ignorar a ocorrência de erros durante a execução.

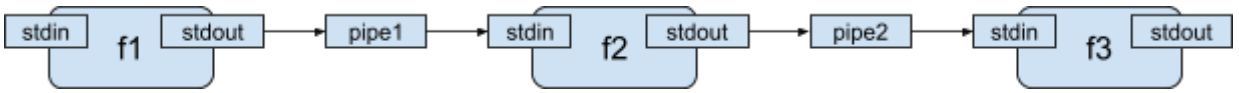
   b) [2] Sabendo que:
   - `grep -H :` prefixa cada resultado encontrado com o nome do ficheiro (por exemplo: se `grep x texto1.txt` produz como resultado uma linha com `x`, então a linha produzida por `grep -H x texto1.txt` será `texto1.txt:x`)
   - `tee` *`filename`* : transfere o *standard input* para o ficheiro *`filename`* e para o *standard output*
   - `wc -l` : afixa no *standard output* o número que resulta de contar as linhas do *standard input*

   Indique, justificando, o conteúdo do ficheiro `res.txt` e do *standard output* após a execução de:

```
pargrep beta texto1 texto2 | tee res.txt | wc -l
```

texto1	texto2
alpha	ana
beta	beta
gamma	carla
beta	diana
alpha	eva

5) **«A»** [4] Escreva a função `pipeline3`, que recebe como argumento três ponteiros para função (`f1`, `f2` e `f3`) e executa cada uma das funções num processo diferente, com os *standard input* e *output* dos três processos ligados em *pipeline*:




```
void pipeline3(void (*f1)(), void (*f2)(), void (*f3)());
```

Os três processos executam-se em paralelo e a função `pipeline3` espera que todos tenham terminado.