Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Técnicas de Virtualização de Sistemas, Inverno de 2022/2023

Época de Recurso — Teste Parcial #1 — Teste Global, parte 1

ATENÇÃO: Responda às questões **1** e **2** num conjunto de folhas e às questões **3**, **4** e **5** noutro conjunto.

1. [6] Considere a função `int launch_for_output(const char *prog, char *args[])`, que coloca em execução o ficheiro executável `prog`, com os argumentos `args`, e retorna um descritor que permite recolher o *output* produzido durante a execução do comando. O chamador tem a responsabilidade de fechar esse descritor após a sua utilização. Em caso de erro no lançamento do comando a função deve retornar -1. Note que o comando lançado fica em execução em paralelo com o código do chamador.

   a. [4.5] Implemente a função `launch_for_output`

   b. [1.5] Em sistemas onde esta função é usada verifica-se o aparecimento de processos *zombie*. Indique como é que esta função promove esse problema e o que seria preciso fazer para o resolver.

2. [4] Durante a tradução do endereço virtual 0x0000038401C06B47 por um processador *x86-64* são consultadas quatro tabelas de páginas e a tradução é concluída com sucesso.

   a. [2] Indique o índice de cada *page table entry* acedido em cada um dos níveis da tradução. Apresente os cálculos.

   b. [2] Apesar da tradução ter sucesso, o acesso à memória pode resultar num *page fault*. Apresente duas causas possíveis para o *page fault* indicando, para ambos os casos, que dado concreto presente na *page table entry* leva o processador a lançar a exceção.

3. [3] Considere o seguinte código fonte de uma biblioteca (ficheiro com extensão `.so` ou *shared object*):

```
int lifo[16*1024] = { 99, 12, 71 };
int nitems = 3;
int lifo_push(int item) {
   const int NEW_IDX = nitems++;
   lifo[NEW_IDX] = item;
   return NEW_IDX + 1;
}
int lifo_pop() { return lifo[--nitems]; }
```

   a. [1.5] Tendo em conta apenas o código apresentado, que alterações se pode prever que terá a organização do espaço de endereçamento de um processo ao carregar esta biblioteca com `dlopen`? Justifique cada uma das alterações previstas, relacionando-as com os elementos de código.

   b. [1.5] Sendo esta uma biblioteca destinada a ser partilhada entre processos (*shared object*), caso um segundo processo carregue esta mesma biblioteca, ambos os processos ficarão a partilhar o mesmo espaço de memória que suporta o *array* `lifo`?
      - ○ Se sim, indique se o endereço virtual do array é igual em ambos os processos.
      - ○ Se não, indique o que é partilhado neste *shared object*.
      - ○ Se depender de condições, indique as condições que permitem a partilha.

(continua)

4. [2.5] Um programa de utilizador, escrito em C, executa-se num sistema Linux com processador *x86-64*. Quando é invocada a operação de sistema `int pipe(int pipefd[2])`, o *kernel* cria um objeto que permite comunicação entre processos. No entanto, o código do programa que invoca `pipe` está a correr em nível de privilégio 3, mas o código de *kernel* que implementa o comportamento da função de sistema `pipe` está numa zona de memória que exige nível de privilégio 0 para ser executado. Indique como se processa a chamada de sistema para o processador indicado, incluindo a passagem de parâmetros, o retorno do inteiro resultante e as transições entre níveis de privilégio.

5. [4.5] Considere o seguinte programa:

```
#define SIZE1 (1024*1024)
#define SIZE2 (SIZE1*2)

int main() {
  char* ptr1 = (char*) mmap(NULL, SIZE1, PROT_READ|PROT_WRITE,
                                  MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);      // 1
  for(int p=0; p < SIZE1; ++p) ptr1[p] = 'A';                               // 2
  pid_t pid = fork();
  if (pid == 0) {
    char * ptr2 = (char*) mmap(NULL, SIZE2, PROT_READ|PROT_WRITE,
                                    MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    for(int p=0; p < SIZE2; p += PAGE_SIZE) ptr2[p] = ptr1[p/2];            // 3
    munmap(ptr2, SIZE2);                                                    // 4
  } else {
    waitpid(pid, NULL, 0);
    munmap(ptr1, SIZE1);                                                    // 5
  }
  return 0;
}
```

O código fonte do programa inclui instruções para a criação de novas regiões de memória válidas. Pretende-se que acompanhe o estado apenas das regiões com origem nessas instruções ao longo dos pontos assinalados de 1 a 5. Para cada um desses pontos, no final de cada linha assinalada, e para cada região existente nesse momento, indique em KBytes:

- a dimensão da região de memória;
- o *resident set size* (Rss) dessa região;
- o *proportional set size* (Pss) dessa região;
- a dimensão das partes classificadas como *private clean*, *shared clean*, *private dirty* e *shared dirty*

NOTA: Todos os pontos precisam de ser devidamente justificados.

Duração: 1 hora e 15 minutos

ISEL, 2 de fevereiro de 2023