ISEP – LEEC - Sistemas Computacionais 2022/2023

Ficha 5 – *Semáforos e mutex POSIX*

1 - Analise o seguinte extrato de um programa:

```
int main() {
  int *v = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
                MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  sem_t *psem = sem_open("/sem1", O_CREAT | O_RDWR, 0600, 0);

  v[0] = 0;
  int r, n = 0;
  r = fork();
  ++n;
  ++v[0];
  if(r == 0) {
    sem_wait(psem);
    printf("%d: *v = %d, n = %d\n", getpid(), v[0], n);
    return(0);
  }
  sleep(5);
  printf("%d: *v = %d, n = %d\n", getpid(), v[0], n);
  sem_post(psem);
  return(0);
}
```

Apresente a sequência de impressões produzidas por este programa. Assuma que não existem interferências de outros processos no sistema, que o identificador do processo inicial é 2000 e que o(s) novo(s) processo(s) toma(m) o(s) valor(es) seguinte(s). Assuma que que as instruções `++n` e `++v[0]` são atómicas. Apresente um diagrama temporal representativo da execução do programa e justifique sucintamente. A sequência de impressões deve ser apresentada de forma destacada.

Note que recorrendo à *flag* `MAP_ANONYMOUS`, a função `mmap` permite criar um bloco de memória partilhada sem ser necessário recorrer à função `shm_open`.

2 - Extraia o conteúdo do ficheiro `ficha-sinc-ficheiros.zip` para o seu diretório de trabalho. Analise o programa contido em `ex2.c`. A função `myprint` é a mesma função descrita no exercício 1 da ficha anterior. Pode utilizar o comando `make` para criar o executável **`ex2`**.

2.1 - Utilize o mecanismo de semáforos com nome para garantir que a impressão de cada processo é enviada para o ecrã sem ser interrompida pelas impressões do outro processo.
Observação: tenha atenção ao facto que o semáforo com nome mantém o seu estado mesmo após a terminação dos processos que o utilizam. Para repor o valor inicial do semáforo, terá que o apagar (`sem_unlink`).

2.2 – Apresente um diagrama temporal da execução deste programa desde o seu arranque até ao momento em cada processo já executou pelo menos uma vez a função `myprint`.

2.3 – Ao contrário do que se observou no exercício 1 da ficha anterior, basta agora fazer uma chamada à função `malloc` (nomeadamente, `buf = malloc(256)`) para que cada "tarefa" armazene e imprima a sua própria *string*. Porquê?

2.4 (Memória partilhada sem nome) - Altere a linha

```
char *buf = malloc(256);
```

para

```
char *buf = (char *) mmap(NULL, 256,PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);
```

e verifique que ambos os processos passam a imprimir a mesma mensagem.

2.5 – Repita a alínea 2.1 usando um semáforo sem nome.

3 – Utilizando o mecanismo *mutex,* altere o programa do exercício 1 da ficha 4 (*threads*) de forma a garantir que a impressão de cada *thread* (função `myprint`) é enviada para o ecrã sem ser interrompida pelas impressões da outra *thread*. Não deverá alterar a função `myprint`.

4 - Considere os seguintes programas:

```
//ex4-reader.c
#include "common.h"

int main(int argc, char** argv) {
  int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
  if (shm_fd == -1) handle_error("shm_open");

  int r = ftruncate(shm_fd, SHM_SIZE);
  if (r == -1) handle_error("ftruncate");

  char* shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
  if (shm_ptr == MAP_FAILED) handle_error("mmap");

  strcpy(shm_ptr, "");

  while (1) {
    printf("Dados: %s\n", shm_ptr);
    sleep(2);
  }

  return 0;
}
```

```
//ex4-writer.c
#include "common.h"

int main(int argc, char** argv) {
  char buffer[SHM_SIZE];

  int shm_fd = shm_open(SHM_NAME, O_RDWR, 0);
  if (shm_fd == -1) handle_error("shm_open");

  char* shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
  if (shm_ptr == MAP_FAILED) handle_error("mmap");

  printf(": ");
  fgets(buffer, SHM_SIZE, stdin);
  strcpy(shm_ptr, buffer);

  return 0;
}
```

O programa em `ex4-reader.c` imprime periodicamente o texto armazenado no bloco de memória partilhada. Recorrendo ao mecanismo de semáforos, altere ambos os programas de modo que o programa em `ex4-reader.c` apenas faça impressões quando o programa em `ex4-writer.c` escreve uma nova linha de texto no bloco de memória partilhada. Adicionalmente, garanta que o acesso à memória partilhada é mutuamente exclusivo, seja entre `ex4-reader` e `ex4-writer` ou entre múltiplas execuções de `ex4-writer`.

5 - Analise o seguinte extrato de um programa:

```
int main() {
  int *v = mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
                MAP_SHARED | MAP_ANONYMOUS, -1, 0);

  sem_t *psem = sem_open("/sem1", O_CREAT | O_RDWR, 0600, 1);

  v[0] = 0;
  int r, n = 0;
  for(int i=0;i<2;++i) {
    sleep(1);
    r = fork();
    ++n;
    if(r == 0) {
      r = fork();
      ++v[0];
      if(r == 0) {
        sem_wait(psem);
        sleep(3);
        sem_post(psem);
        printf("n = %d, v = %d, pid = %d, ppid = %d.\n", n, v[0], getpid(), getppid());
        return(0);
      }
      sleep(1);
      printf("%d a terminar; *v = %d.\n", getpid(), v[0]);
      exit(0);
    }
    waitpid(r, NULL, 0);
    printf("%d terminado; *v = %d, n = %d.\n", r, v[0], n);
  }
  return(0);
}
```

Apresente a sequência de impressões produzidas por este programa. Assuma que não existem interferências de outros processos no sistema, que o identificador do processo inicial é 2000 e que o(s) novo(s) processo(s) toma(m) o(s) valor(es) seguinte(s). Assuma que que a instrução `++v[0]` é atómica. Apresente um diagrama temporal representativo da execução do programa e justifique sucintamente. A sequência de impressões deve ser apresentada de forma destacada.