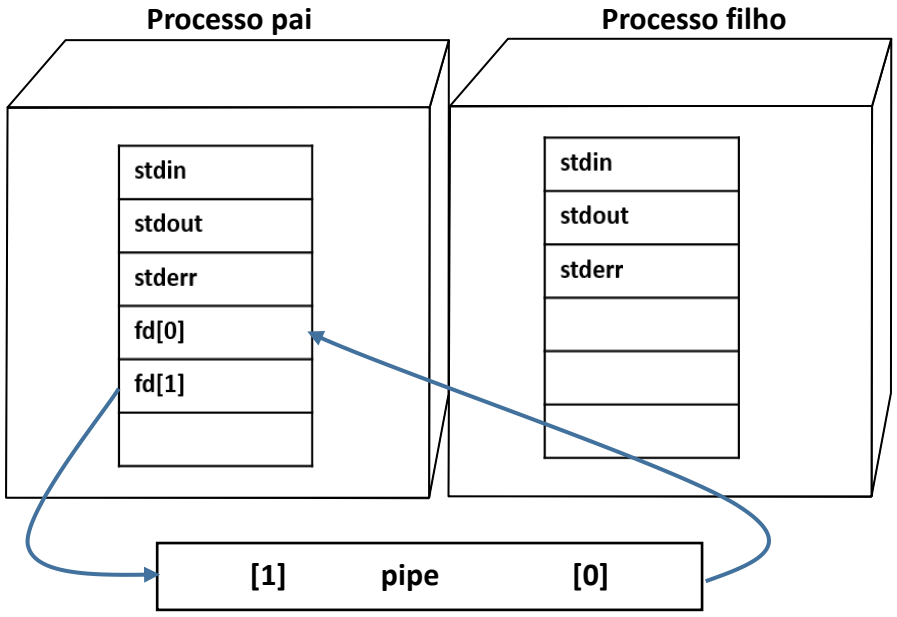
Número: Nome:

LEIC/LETI – 2019/20 – Repescagem do 1º Teste - 04 de fevereiro de 2020

- **Duração: 1h30m**
- **Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.**
- **Nas perguntas de escolha múltipla, uma resposta errada desconta 1/3 da cotação.**

Grupo I [7,3]



1) Considere a figura acima que esquematicamente representa dois processos, pai e filho, e um *pipe*.
   a) [0,6] Para que servem as tabelas presentes em cada um dos dois processos?

   b) [0,6] Que tipos de entidades podem ser referenciadas por estas tabelas? Seja tão exaustivo quanto possível na sua resposta.

c) [0,5] Qual das seguintes afirmações descreve corretamente as tabelas representadas na Figura 1?
   i) Ambas correspondem à mesma tabela, que é partilhada pelo processo pai e processo filho.
   ii) As tabelas do processo pai e do processo filho não têm qualquer relação.
   iii) O filho herda o *standard input, output* e *error* do processo pai mas não herda os mecanismos de comunicação criados pelo pai antes de efetuar *fork*.
   iv) A tabela do processo filho é copiada do processo pai, aquando da criação do processo filho.

2) *Pipes*
   a) [0,5] Qual das afirmações carateriza corretamente o objeto de comunicação *pipe*?
      i) Estabelece um canal bidirecional.
      ii) Permite comunicação entre processos independentes na mesma máquina.
      iii) Tem uma interface de sequência de caracteres - *byte stream*.
      iv) Para enviar mensagens, utiliza-se a função *sendto*.

   b) [0,6] Com base nos elementos da figura, programe a criação do *pipe* pelo processo pai com as respetivas estruturas de dados.

3) Considere agora a criação do processo pai e do processo filho.
   a) [1,0] Considere que o processo pai chamou a função *fork* depois de ter criado o *pipe*. Assim sendo, a Figura 1 está incompleta. Complemente a figura com:
      i) a informação em falta;
      ii) através de setas, descreva a relação da tabela com as extremidades de leitura/escrita do *pipe*.

   b) [0,5] Nesta situação, quem pode ler ou escrever no pipe?
      i) Processo pai pode ler de fd[1] e escrever em fd [0] (entre outros)
      ii) Processo filho pode ler de fd[0] e escrever em fd [1] (entre outros)
      iii) Processo pai pode ler e escrever em fd[0] (entre outros)
      iv) Processo filho pode ler e escrever em fd[0] (entre outros)

4) Considere agora que pretende efectuar a redireção da **saída (*stdout*) do processo pai para a entrada (*stdin*) do processo filho.**

a) [0,6] A chamada sistema fundamental na redirecção é ***dup***. Explique o que faz.

b) [0,5] Para implementar a redirecção descrita acima, apenas uma das operações abaixo faz sentido ser executada (entre outras). Escolha a correta.

i) Filho faz *dup* de fd[0]
ii) Pai fecha *stdin*
iii) Pai faz *dup* de fd[0]
iv) Filho fecha *stdout*

c) [1,1] Programe a redireção do lado do pai.

d) [0,8] No caso descrito nesta alínea o processo filho executa sem qualquer alteração o seu código original. Seria possível fazê-lo com um ficheiro de entrada? Explique claramente a sua resposta.

GRUPO II [6,3]

Anailise o seguinte extrato de um programa que implementa um Produtor/Consumidor:

```
1.  typedef struct {
2.      int buf[BUFF_SIZE];   /* shared var */
3.      int in;               /* buf[in%BUFF_SIZE] is the first empty slot */
4.      int out;              /* buf[out%BUFF_SIZE] is the first full slot */
5.      sem_t slots;
6.      sem_t msgs;
7.      sem_t sc;
8.  } sbuf_t;
9.
10. sbuf_t shared;
11.
12. void *Producer(void *arg) {
13.     int item, index;
14.
15.     for (;;){
16.         /* Produce item */
17.         item = ProduzItem();
18.
19.         sem_wait(&shared.slots);
20.         sem_wait(&shared.sc);
21.         shared.buf[shared.in] = item;
22.         shared.in = (shared.in+1)%BUFF_SIZE;
23.         sem_post(&shared.sc);
24.         sem_post(&shared.msgs);
25.     }
26. }
```

1) [1] Explique como funciona um semáforo escrevendo o pseudo-código da função ***post*** (assinalar). Considere que a exclusão mútua já está assegurada.

2) [0,5] Para que serve o semáforo ***slots***?
   a) Permite garantir a exclusão mútua no acesso a *buf.*
   b) Permite assegurar que os produtores não destroem itens em *buf* que não tenham sido ainda lidos.
   c) Permite assegurar que os consumidores só lêem itens quando estes existem em *buf*.
   d) Permite alternar a execução dos produtores e consumidores.

3) [0,6] Programe a inicialização dos semáforos utilizados.

4) Em vez do semáforo ***sc*** poderia usar um outro objecto cuja função é totalmente adequada à que este semáforo desempenha.
   a) [0,4] Qual?

   b) [0,6] Qual a diferença para um semáforo?

   c) [0,6] Acha que essa diferença pode ter algum impacto no desempenho das funções sistema que o manipulam? Justifique.

5) [1,5] Proponha um programa para os consumidores, admitindo que podem ser múltiplos.

6) [1,1] Se a programação inicial das linhas 19 e 20 fosse

```
sem_wait(&shared.sc);
sem_wait(&shared.slots);
```

o programa funcionaria da mesma forma? Justifique.

Grupo III [3,0]

Considere o seguinte problema que utiliza signals com a semântica BSD. Os sinais SIGINT e SIGQUIT estão associados a sequências de teclas CNTL-C e CNTL-\, respetivamente.

```
1.  #include <signal.h>
2.
3.  static void
4.  sigHandler(int sig)
5.  {
6.     static int count = 0;
7.     if (sig == SIGINT) {
8.       count++;
9.       return;
10.    }
11.    printf("Result %d \n Caught SIGQUIT \n", count); /* unsafe*/
12.    exit(EXIT_SUCCESS);
13. }
14.
15. int
16. main(int argc, char *argv[])
17. {
18.    if (signal(SIGINT, sigHandler) == SIG_ERR) errExit("signal");
19.    if (signal(SIGQUIT, sigHandler) == SIG_ERR) errExit("signal");
20.    for (;;)
21.      pause();
22. }
```

1) [0,6] Explique para que são usados os *signals* em Unix.

2) [0,8] No programa, a função usada como *handler* tem um parâmetro de entrada. Explique para que serve utilizando um exemplo do programa.

3) [0,6] Qual a funcionalidade da chamada da linha 21 ***pause*****?**

4) [1,0] Considere que, com este programa a executar-se, se efectuava a seguinte input:
   CNTL-C; CNTL-C; CNTL-C¸ CNTL-\
   Descreva o que seria o comportamento do programa.

Grupo IV [3,4]

Considere o seguinte excerto de programa:

```
1.  for (;;){
2.     cfd = accept(lfd, NULL, NULL);
3.     if (cfd == -1) {
4.              syslog(LOG_ERR, "Failure in accept(): %s", strerror(errno));
5.              exit(EXIT_FAILURE);
6.     }
7.
8.     switch (fork()) {
9.     case -1:
10.        syslog(LOG_ERR, "Can't create child (%s)", strerror(errno));
11.        close(cfd);
12.        break;
13.
14.     case 0:
15.         close(lfd);
16.         handleRequest(cfd);
17.         exit(EXIT_SUCCESS);
18.
19.     default:
20.          close(cfd);
21.          break;                    /
22.     }
    }
```

1) [0,4] O *socket* é usado em modelo com ligação ou sem ligação? Justifique.

2) [0,4] O código é do cliente ou do servidor? Justifique.

3) [0,6] O programa tem um *fork.* Qual a justificação para a criação deste processo?

4) [0,8] O que procura distinguir a função *switch* em relação ao resultado do *fork*?

5) Em cada ramo da instrução s*witch* são fechados descritores de ficheiros.
   a) [0,6] Explique a razão de, na cláusula case 0, ser efectuado *close* (lfd) e se tem alguma implicação na funcionalidade do processo .

   b) [0,6] Explique a razão de, na cláusula *default,* ser efectuado *close* (cfd) e se tem alguma implicação na funcionalidade do processo .