

Exame Época Normal

1ª Parte

Parte Teórica

- 1) O PCB (Process Control Block) contém, entre outras informações:
 - ☒ apontador para a tabela de páginas do processo, apontador para ficheiros abertos pelo processo, apontador para a pilha de execução do processo
 - ☐ apontador para ficheiros abertos pelo processo, apontador para a pilha de execução do processo, código do manipulador de interrupções
 - ☐ apontador para a tabela de páginas do processo, apontador para ficheiros abertos pelo processo, apontador para o controlador de interrupções
- 2) Ao implementar sincronização entre processos concorrentes, a utilização do mecanismo de “busy waiting” (espera ocupada) traz vantagens em relação à utilização de semáforos baseados em fila quando:
 - ☒ o tempo para executar a secção crítica é menor que o tempo de troca de contexto do processo
 - ☐ o tempo para executar a secção crítica é maior do que o tempo de espera pelo semáforo
 - ☐ o tempo para executar a secção crítica é maior que o tempo de troca de contexto do processo
- 3) Que comando é utilizado para terminar (de forma normal) um processo, sabendo-se o PID do mesmo?
 - ☒ kill -1 PID
 - ☐ terminate PID
 - ☐ killprocess PID
 - ☐ kill -2 PID
- 4) Qual das afirmativas abaixo melhor define os conceitos de multi-programação e multi-tarefa?
 - ☐ multi-programação permite que processos sejam interrompidos por quantum de tempo; multi-tarefa permite que vários processos possam utilizar a CPU ao mesmo tempo
 - ☒ multi-programação permite que vários programas estejam no sistema ao mesmo tempo; multi-tarefa permite que processos sejam interrompidos por quantum de tempo
 - ☐ multi-programação permite que vários programas estejam no sistema ao mesmo tempo; multi-tarefa permite que vários processos possam utilizar a CPU ao mesmo tempo
- 5) Análise do código abaixo. A solução para o problema da exclusão mútua entre dois processos está correcta?

```
shared int c1 = 1; int c2 = 1;

void p0() {
    for(;;) {
        c1 := 0 ;
        while (c2 == 0) {}
        "secção crítica"
        c2 := 1;
    }
}

void p1(){
    for(;;) {
        c2 := 0 ;
        while (c1 == 0) {}
        "secção crítica"
        c2 := 1;
    }
}

main () {
    p0();
    p1();
}
```

 - ☐ sim
 - ☒ não
 - ☐ não há elementos suficientes para responder a esta questão pois a solução depende do código da secção crítica

Parte Prática

- 1) Dado o código abaixo, qual das seguintes afirmações é a mais adequada?

```
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMSZ 27

main() {
    char c, *shm, *s;
    int chave= 5678, shmid;

    shmid= shmget(chave, SHMSZ, (IPC_CREAT|0666));
    shm= (char *)shmat(shmid, NULL, 0);

    s= shm;
    for (c='a'; c<='z'; c++)
        *s++= c;
    *s= '\0';

    while (*shm != '*')
        sleep(1);
    shmdt(shmid);
    exit(0);
}
```

- ☐ o processo associado a este código entra em ciclo
☒ o processo associado a este código depende de outro processo para terminar
☐ o processo associado a este código termina normalmente

- 2) Considere o programa abaixo:

```
#include <stdio.h>
#include <unistd.h>

#define Read 0
#define Write 1
main(int argc, char *argv[])
{
    int fd[2];

    pipe(fd);
    if (fork() == 0) {
        (1)
        execlp(argv[2], argv[2], NULL);
        (2)
    }
    else {
        (3)
        execlp(argv[1], argv[1], NULL);
        (4)
    }
}
```

Se o valor de argv[1] é "who" e o valor de argv[2] é "wc", qual é a sequência de código que deve preencher nas posições (1), (2), (3) e (4)?

[x]

- (1) close(fd[Write]);
dup2(fd[Read], 0);
close(fd[Read]);
- (2) perror("ligação não sucedida");
- (3) close(fd[Read]);
dup2(fd[Write], 1);
close(fd[Write]);
- (4) perror("ligação não sucedida");

[]

```
(1) close(fd[Read]);  
    dup2(fd[Write], 1);  
    close(fd[Write]);
```

```
(2) exit(0);
```

```
(3) close(fd[Write]);  
    dup2(fd[Read], 0);  
    close(fd[Read]);
```

```
(4) exit(0);
```

[]

```
(1) close(fd[Write]);  
    dup2(fd[Read], 0);  
    close(fd[Read]);
```

```
(2) perror("ligação não sucedida");
```

```
(3) close(fd[Read]);  
    dup2(fd[Write], 0);  
    close(fd[Write]);
```

```
(4) wait(NULL);
```

[]

```
(1) close(fd[Write]);  
    dup2(fd[Read], 0);  
    close(fd[Read]);
```

```
(2) exit(0);
```

```
(3) close(fd[Read]);  
    dup2(fd[Write], 0);  
    close(fd[Write]);
```

```
(4) exit(0);
```

- 3) Considere o excerto de programa que se segue como a implementação da etapa 'matmult' do enunciado do Trabalho I.

```
1.  main(int argc, char *argv[]) {
2.      int i, j, len, L;
3.      char *line = NULL;
4.      FILE *stream = fopen(argv[1], "r");
5.      fscanf(stream, "%d", &L);
6.      getline(&line, &len, stream);
7.      int in[2], out[L][2];
8.      for (i = 0; i < L; i++) {
9.          pipe(in);
10.         pipe(out[i]);
11.         if (fork() != 0) {
12.             close(out[i][WRITE]);
13.             getline(&line, &len, stream);
14.             // escrita na pipe de stdin do filho
15.         } else {
16.             for (j = 0; j <= i; j++)
17.                 close(out[j][READ]);
18.             // redirecionamento do stdout do filho
19.             // redirecionamento do stdin do filho
20.             // execução do comando vecmult
21.         }
22.     }
23.     for (i = 0; i < L; i++) {
24.         char buf[100];
25.         // leitura ordenada do output dos filhos
26.     }
27. }
```

A linha 25, que corresponde à leitura ordenada do output dos filhos, pode ser implementada pela sequência de instruções:

```
[ ]  fscanf(out[i][READ], "%s", buf);
      printf("%s", buf);

[ ]  while ((j = fscanf(out[i][READ], "%s", buf)) != 0) {
      buf[j] = 0;
      printf("%s", buf);
  }


[x]  while ((j = read(out[i][READ], buf, 100)) != 0) {
      buf[j] = 0;
      printf("%s", buf);
  }


[ ]  j = read(out[i][READ], buf, 100);
      buf[j] = 0;
      printf("%s", buf);
```


- 4) A representação gráfica que melhor retrata a implementação abaixo é (assuma que P é um processo e que cada barra vertical representa uma descendência):


```
main()
{
    int i, n = 4, ID;

    for (i = 1; i < n; i++)
        if ((ID = fork()) == 0)
            break;
    fprintf(stderr, "proc(\\%ld) com pai(\\%ld)\\n", (long)getpid(), (long)getppid());
}
```

[] 

[] 

[x] 

[] 

- 5) Seja o seguinte programa:

```
#include <stdio.h>

main()
{
    int i;

    if (fork() == 0)
        for (i=0; i<5; i++) {
            printf("Proc1: %d\\n", i);
            sleep(2);
        }
    else
        for (i=0; i<5; i++) {
            printf("Proc2: %d\\n", i);
            sleep(3);
        }
}
```

- [] a ordem de escrita dos processos é determinada e o processo pai sempre escreve antes do processo filho
 [] a ordem de escrita dos processos é determinada e o processo filho intercala as escritas com o processo pai
 [] a ordem de escrita dos processos é determinada e o processo filho sempre escreve antes do processo pai
 [x] a ordem de escrita dos processos é indeterminada