UFES – Departamento de Informática

2ª. Prova de Sistemas Operacionais – 2019/1 – Profª. Roberta L. Gomes

Aluno:_____

1) **(2,0)** Explique a razão pela qual um tratador de sinais pode encontrar estruturas de dados inconsistentes. De que forma o programador pode evitar essas situação NO UNIX?

2) **(2,5)** O processo P1 envia números inteiros ao processo P2 utilizando uma variável inteira que encontra-se em uma região de memória compartilhada. P2 só deve ler a memória compartilhada uma vez que P1 tiver colocado o inteiro. Implemente (em pseudocódigo) um monitor com os seguintes procedimentos de entrada :

PUT(int data) – utilizado por P1; não retorna nenhum valor.

GET() – utilizado por P2 ; retorna o inteiro que é copiado da região de memória compartilhada.

Considere que o Monitor segue a proposta por Hoare (*Signal and Wait*), em que o processo sinalizador fica bloqueado até que o processo sinalizado saia do monitor ou execute outro wait.

3) **(1,0)** (POSCOMP 2010) Embora ambos tenham seu escalonamento feito pelo gerenciamento de processos, threads e processos são estruturalmente distintos. Qual é a principal diferença entre eles? **[RESPONDER DA FOLHA DE PROVA]**

a. Apenas threads podem ser executadas em paralelo.
b. Processos executam mais rapidamente.
c. Processos apenas ocorrem em sistemas de grande porte.
d. Threads possuem contexto simplificado.
e. Threads apenas ocorrem em processadores multicore.

Justifique sua resposta: _____

_____

_____

_____

4) **(2,0)** Sobre o código a seguir, responda: **[RESPONDER DA FOLHA DE PROVA]**

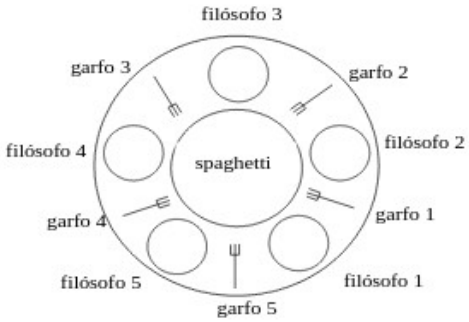
```
#define SIZE 6
#define READ 0
#define WRITE 1
main()
{
    pid_t pid1, pid2, pid3, pid4; int status;
    int fd[2]; char buffer[SIZE+1];
    struct rusage usage;
    pipe(fd);
    if ((pid1=fork())==0) {         // child 1
      while(1) {
        read(fd[READ], buffer, SIZE);
        buffer[SIZE]='\0';
        write(fd[WRITE], "tomato", SIZE);
     }
    } else if ((pid2=fork())==0) {  // child 2
      while(1) {
        read(fd[READ], buffer, 6);
        buffer[6]='\0';
        write(fd[WRITE], "turnip", 6);
     }
    } else {  // parent
      write(fd[WRITE],"potato",6);
      fprintf(stderr,"Parent: I wrote a potato!\n", buffer);
      sleep(3);
      read(fd[READ], buffer, 6);  buffer[6]='\0';
      fprintf(stderr,"Parent: I got back a %s!\n", buffer);
      kill(pid1, SIGINT); pid3 = wait(&status, 0, &usage);
      kill(pid2, SIGINT); pid4 = wait(&status, 0, &usage);
    }
}
```

a) O que este código faz? É possível prever qual será a saída do programa?

b) O que acontece se omitirmos o comando write do pai?

c) O que acontece se omitirmos o comando write do um dos filhos?

5) **(2,0)** Dado o problema dos Filósofos Glutões, complete o código a seguir com chamadas a semáforos. **[RESPONDER DA FOLHA DE PROVA]**



Dados Compartilhados

```
#define N 5
#define LEFT (i+N-1)%N
#define RIGHT (i+1)%N

#define THINKING 0
#define HUNGRY 1
#define EATING 2
int state[N];

typedef int semaphore;

semaphore mutex = 1;
semaphore philo[N]=
  {0,0,...,0};
```

```
void philosopher(int i)
    { while (TRUE) {
        think();
        take_forks(i);
        eat();
        put_forks(i); }}


void take_forks(int i)
 { _____________;
   state[i] = HUNGRY;
   test(i);
   _____________;
   _____________; }

void put_forks(i)
 { _____________;
   state[i] = THINKING;
   test(LEFT);
   test(RIGHT);
   _____________; }

void test(i)
{ if (state[i] == HUNGRY &&
      state[LEFT] != EATING &&
      state[RIGHT] != EATING) {
        state[i] = EATING;
        _____________;
  }
}
```

Boa prova!!!