UFES - DEPARTAMENTO DE INFORMÁTICA

2ª. Prova de Sistemas Operacionais - Período: 2012/2 - Profª. Roberta Lima Gomes

1) **(2,0)** No primeiro trabalho prático da disciplina, foi solicitado que na ocorrência de Ctrl-C o processo mysh deveria ignorar o SIGINT, já seus filhos em foreground fariam o tratamento default (*i.e.* finalizar). Explique como o seu grupo implementou esse comportamento e explique se seria possível (e como) fazer com que os filhos de mysh em foreground também ignorassem o SIGINT.

2) **(2,0)** Assumindo que todas as operações fork() retornarão com sucesso, quantos processos serão criados a partir do momento que o usuário solicita ao S.O a execução do seguinte programa? Explique o que será impresso utilizando um grafo de precedência para definir a ordem das strings impressas.

```
main() {
    int ret_val;
    for (int i=0; i<4;i++)
        if ((ret_val = fork()) > 0) {
            printf("teste: %d. \n", i);
            i = i + 1;
            execlp("ls","ls",null);
        }
}
```

3) **(2,0)** Explique como o mecanismo de troca de mensagens pode ser utilizado para garantir exclusão mútua. Qual a principal vantagem que esse mecanismo tem se comparado com semáforos ou monitores.

4) **(2,5)** Considere o algoritmo dos leitores/escritores abaixo. Modifique o programa de forma que a solução considere os seguintes requisitos suplementares:

- Apenas podem estar 5 leitores simultaneamente acessando o banco de dados.
- Quando houver 2 escritores à espera não podem entrar mais leitores.

```
//número de leitores ativos
int rc = 0;

//protege o acesso à variável rc
Semaphore mutex = 1;

//Indica a um escritor se este
//pode ter acesso aos dados Semaphore db
= 1;
```

```
//Escritor
while (TRUE){
  down(db);
  ...
  //writing is
  //performed
  ...
  up(db);
}
```

```
//Leitor
while (TRUE){
  down(mutex);
  rc++;
  if (rc == 1)  down(db);
  up(mutex);
  ...
  //reading is performed
  ...
  down(mutex);
  rc--;
  if (rc == 0)    up(db);
  up(mutex); }
```

5) **(1,5)** Boa parte dos sistemas operacionais atuais (incluindo Linux) apresenta uma associação fixa (*binding*) entre uma ULT (User Level Thread) e uma KLT (Kernel Level Thread), seguindo o modelo "um para um". Já algumas versões mais antigas de UNIX permitem ao processo ter uma quantidade ***x*** de ULTs e ***y*** de KLTs, sendo que, dinamicamente, qualquer uma das ULTs pode ser *binded* a qualquer uma das KLTs desse processo. Quais são as vantagens e disvantagens desse modelo, se comparado com o modelo "um para um".

Boa Prova !!!