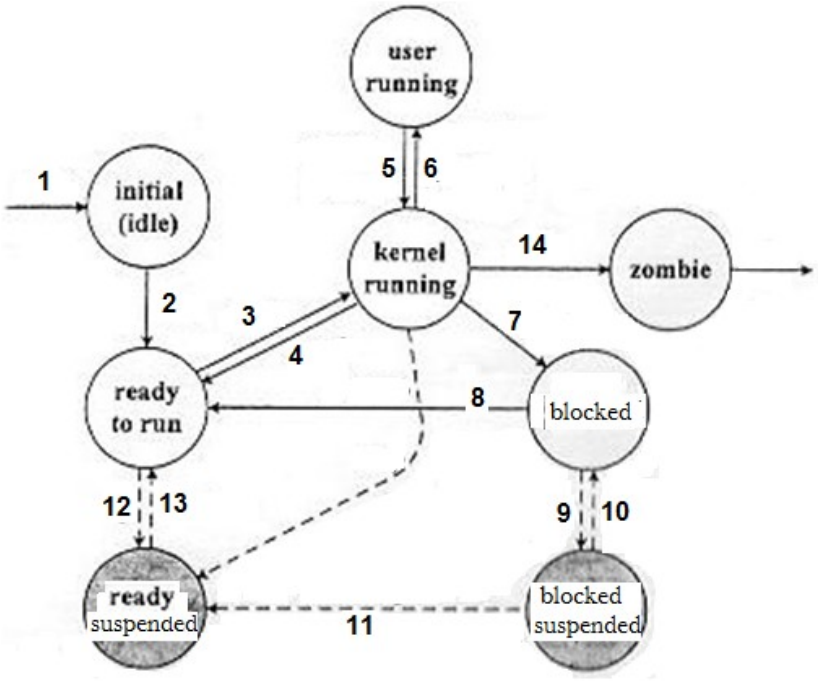
UFES – Departamento de Informática

1ª. Prova de Sistemas Operacionais – 2014/2 – Profª. Roberta L. Gomes

1) (2,0) Dado o diagrama de estados do Unix ilustrado abaixo, escreva e explique os eventos que provocam cada uma das transições entre estados.



2) (1,0) Em algumas implementações do UNIX, o kernel é não-preemptivo. O que isto significa? Quais as vantagens e desvantagens desta abordagem?

3) (1,5) Quatro programas devem ser executados em um computador. Todos os programas são compostos por {ciclo de CPU, ciclo de E/S, ciclo de CPU, ciclo de E/S}. A entrada e saída de todos os programas é feita sobre a mesma unidade de disco. Os tempos para cada ciclo de cada programa são mostrados abaixo:

Programa	CPU	Disco	CPU	Disco
P1	3	10	3	12
P2	4	12	6	8
P3	7	8	8	10
P4	6	14	2	10

Construa um diagrama de Gantt (diagrama de tempo) mostrando qual programa está ocupando o processador e o disco a cada momento, até que os 4 programas terminem. Suponha que o algoritmo de escalonamento utilizado na CPU seja *round-robin*, com fatias de 4 unidades, e que a fila de protos encontra-se ordenada na ordem listada (P1 é o primeiro, e P4 é o último processo). Suponha que as solicitações de acesso ao disco sejam atendidas na ordem de chegada (FCFS).

4) (2,0) Cinco processos, de A até E, chegam ao computador ao mesmo tempo. Eles têm seus tempos de processamento estimados em 10, 6, 2, 4 e 8 minutos respectivamente. Suas prioridades (atribuídas externamente) são 3, 5, 2, 1 e 4, respectivamente, sendo 5 o representante da prioridade mais alta. Nenhum dos processos faz I/O. Para cada um dos algoritmos de escalonamento abaixo, determine o tempo médio de turnaround dos processos. Ignore o overhead causado pela troca de contexto.

(a) Escalonamento com prioridade
(b) FIFO (ordem de execução: A, B, C, D, E)
(c) SRTF (Shortest Remaining Time First / SJF preemptivo)

5) (2,0) A execução do programa a seguir geraria qual saída? Considere que todos os fork()'s e exec()'s são bem sucedidas.

```
int main()
{    int f1;
     char *argv[2];
     argv[0] = "prog1";  argv[1] = '\0';
     if (fork() == 0){
          printf("fork 1 ok!\n");
          execvp("prog1", argv);
          printf("exec 1 ok!\n");
          if ((f1=fork()) == 0){
               printf("fork 2 ok!\n");
          }
          else
               wait(NULL);
     }
     else {
          printf("again!\n");
          if ((f1=fork()) == 0)
               printf("fork 3 ok!\n");
          else {
               execvp("prog1", argv);
               printf("exec and wait ok!\n");
          }
          printf("bye!\n");
     }
     return(0);

  }
```

… dado que `prog1` é um arquivo executável gerado a partir do seguinte programa:

```
int main()
{    printf("test\n");
     wait(NULL);
     return(0);
 }
```

6) (1,5) Para cada questão a seguir, escolha a alternativa conforme solicitado e explique/justifique sua escolha.

A) Considere um sistema com um escalonador *round-robin* e processos em sua maioria interativos. Suponha que, na média, um processo seja executado por 10 ms até gerar uma solicitação de E/S com bloqueio. Suponha que 90% dos ciclos de processador tem uma duração entre 8 ms e 15 ms. Do ponto de vista do escalonamento, está ERRADO afirmar que:

(I) Uma fatia de tempo de 1ms tem a vantagem de fornecer uma boa ilusão de paralelismo na execução dos processos.

(II) Uma fatia de tempo de 200 ms vai gerar baixo *overhead* por chaveamento de contexto.

(III) Acima de um certo valor, a fatia de tempo transforma o algoritmo de escalonamento em FCFS.

(IV) Quanto menor a fatia de tempo mais rápida a execução do ciclo de processador de cada processo.

B) Qual o impacto de uma interrupção gerada pelo controlador de disco sobre o estado dos processos no sistema?

(I) Processo executando pode ficar bloqueado.

(II) Sinaliza o término da fatia de tempo do processo executando.

(III) Processo executando pode ser substituído por um processo “pronto” de mais alta prioridade.

(IV) Processo executando deve ser abortado por violação do espaço em disco.

C) Com respeito a usar a própria pilha do processo para salvar seu contexto de execução, pode-se afirmar que:

(I) É preciso ter certeza de que existe espaço na pilha do processo para salvar os registradores.

(II) Durante a sua execução o processo pode destruir o contexto de execução que foi salvo em sua pilha.

(III) Um processo poderia destruir o contexto de execução de outro processo que foi salvo na pilha.

Boa Prova!