UFES - DEPARTAMENTO DE INFORMÁTICA

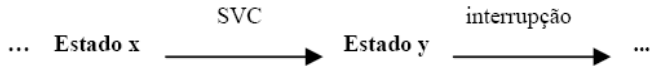
1ª. Prova de Sistemas Operacionais / Sistemas de Programação II

Período: 2011/2 – Data: 13/09/2010 - Profa. Roberta Lima Gomes

1) A partir do diagrama completo de transição de estados para processos em UNIX, apresente uma possível seqüência de estados referente ao seguinte histórico de um processo:
"*O processo foi criado e iniciou a execução de instruções comuns (toda instrução que não corresponde a uma chamada de sistema), sem deixar a CPU até a ocorrência de uma requisição de E/S, a qual demanda um tempo 'longo' para ser atendida. Durante este tempo, o processo sofreu a ação do escalonador de médio prazo (swapper). Uma vez atendida a requisição de E/S, o processo voltou imediatamente a executar instruções comuns, até liberar a CPU para um outro processo. Ao ganhar a CPU novamente, o processo prosseguiu executando instruções comuns até o seu término*" . **(1,5)**

Nota1: Estados do diagrama completo UNIX são: *Initial*, *Ready* (pronto), *Stopped* (pronto suspenso), *Asleep* (bloqueado), *Stopped+asleep* (bloqueado suspenso), *Kernel Running*, *User Running*, *Zombie.*

Nota 2: É necessário associar as transições presentes na seqüência de estados a cada evento listado no histórico. Um exemplo fictício de resposta é mostrado abaixo :



2) Compare e contraste a manipulação de chamadas de sistema (ou chamadas ao supervisor - SVCs) e interrupções em um sistema operacional. **(1,0)**

3) Suponha que os processos seguintes fiquem prontos para execução nos tempos indicados. Cada processo será executado pela CPU pelo de acordo com a duração de seu CPU Burst. Na resposta às questões a seguir, tome todas as decisões com base na informação disponível no instante em que a decisão deve ser feita.

Processo	Duração do CPU Burst	Tempo de chegada
P1	8	0 (passadas 2 unidades de tempo)
P2	5	2
P3	4	4
P4	1	4

a) Desenhe diagramas de Gantt ilustrando a execução desses processos usando os algoritmos de alocação SJF (*Shortest Job First*) e SRTF (*Shortest Remaining Time First*). **(1,0)**
b) Qual o tempo de espera médio (*waiting time*) e o tempo de processamento médio (*turnaround*) para esses processos com o SJF e o SRTF? **(1,0)**

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

5) Suponha um S.O. com escalonador de filas multiníveis com realimentação na qual há quatro níveis. O quantum do primeiro nível é 0,5 segundos. Cada nível mais baixo tem um quantum de tamanho duas vezes maior que o quantum do nível anterior. Um processo novo entra na fila do primeiro nível, que é servida segundo a estratégia RR. Quando ele ganha a CPU ele recebe 0,5 s. Se não liberar a CPU em 0,5 s, o processo é movido para a fila do nível imediatamente abaixo. Um processo não pode sofrer

preempção até o seu quantum terminar. O sistema executa processos em lote (*CPU bound*) e interativos (*I/O bound*).

a) Que tipo de processo este escalonador irá favorecer? Por quê? **(0,5)**
b) Por que esse sistema é deficiente? Quais mudanças mínimas você proporia para tornar o esquema mais aceitável para o mix de processos que pretende? **(0,5)**

6) Assumindo que todas as operações *fork()* retornarão com sucesso, desenhe um diagrama representando a hierarquia de processos criados? Explique (justifique) o que será impresso? (Suponha que a sequência de atribuições de PIDs aos processos seja 100, 101, 102, ...) **(2,0)**

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main (int argc, char *argv[]) {
   pid_t childpid = 0;
   int i, n = 5;
   for (i = 1; i < n; i++)
      if ((childpid = fork()) <= 0)
         break;
   for( ; ; ) {
      childpid = wait(NULL);
      if ((childpid == -1) && (errno != EINTR))
        break;
   }
   fprintf(stderr, "I am process %ld, my parent is %ld\n",
                   (long)getpid(), (long)getppid());
   return 0;
}
```

7) Assumindo que todas as operações *fork()* e *execlp()* são executadas com sucesso, apresente uma saída possível para o seguinte programa? Explique (justifique) o que será impresso? (Suponha que a sequência de atribuições de PIDs aos processos seja 100, 101, 102, ...) **(1,5)**

```
...
int main(){
   printf("Alo do pai\n");
   if ((f1 = fork( ))==0){
       execlp(“ls”,“ls”,NULL);
       fprintf(stderr,”ls (1) concluido \n”);
   }
   execlp(“ls”,“ls”,NULL);
   fprintf(stderr,”ls (2) concluido \n”);
   printf("Filho 1 criado\n");
   waitpid( f1,null,0);
   printf("Filho 1 morreu\n");
   if ((f2 = fork( ))==0){
       execlp(“ls”,“ls”,NULL);
       fprintf(stderr,”ls (3) concluido \n”);
   }
   printf("Filho 2 criado\n");
   waitpid(f2,null,0);
   printf("Filho 2 morreu\n");
}
```

Boa Prova !!!