SISTEMAS OPERACIONAIS – 2013/2 – Prof. José Gonçalves

Gabarito da Prova 2

Questão 1 (1,5 pontos) – Threads (modelo M:N comparado ao modelo 1:1)
Critério: 0,5 ponto ppor cada uma das respostas sublinhadas abaixo, limitado a 1,5 pontos.

- Permite que muitas threads no nível do usuário sejam associadas a muitas threads no nível do kernel (diferentes ULT de um processo são mapeadas em KLT distintas).
- Oferece um grau de paralelismo menor do que o proporcionado pelo modelo 1:1 (a aplicação não tira vantagem máxima do multiprocessamento)
- Consome menos recursos do kernel do que o modelo 1:1
- Mais complexo de gerenciar do que o modelo 1:1
- Info adicionais: sobre o tema da questão:
  - O modelo M:N oferece ao usuário dois níveis de escalonamento: nível usuário e nível sistema
  - O modelo 1:1 é adequado para a maioria da situações e atende bem às necessidades das aplicações interativas e servidores de rede. No entanto, é pouco escalável: a criação de um grande número de *threads* impõe um carga significativa ao núcleo do sistema, prejudicando ou eventualmente inviabilizando aplicações com muitas tarefas, como grandes servidores Web e simulações de grande porte.

QUESTÃO 2 (2,0 pontos)

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

int main() {
    int i=0;

        while (i<10) {
                i = i + 2;
                if (fork() != 0) {
                        printf ("I’m %d\n", i);
                        i = i - 1;
                        exit(0);
                }
                sleep(5);
        }
    return 0;
}
```

RESULTADO:
a) (valendo 1,5) Será impresso:
I'm 2
I'm 4
I'm 6
I'm 8
I'm 10

b) (valendo 0,5) Não há a formação de processos *zombies*. Os processos criados viram órfãos pois os seus pais terminam antes deles. Eles são adotados pelo processo *init*.

QUESTÃO 3 (1,5 pontos)

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void main() {
        pid_t pid;
        int i;

        pid = fork();
        pid = fork();
        for(i=0; i<5; i++)
        {
                pid = fork();
                execlp("exame", "exame", 0);          /* exemplo: execlp(ls", "ls", NULL); */
                if (pid == 0)
                        break;
        }
        execlp("alunos", "alunos", "SO", 0);          /* exemplo: execlp("df", "df", NULL); */
}
```

Resposta: Antes do comando execlp() já existem 8 processos criados, resultantes dos comandos fork(). O programa "exame" é executado, portanto, 8 vezes. Como há a substituição do código na execução do comando execlp(), o programa "alunos" nunca é executado.

QUESTÃO 4 (valor: 1,5 pontos)

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void funcao() {
    execlp("ls", "ls", NULL);  /* execlp("prog", "prog", 0); */
}

void main() {
   pid_t pid;
   int i;
   pid = fork();
   if (pid == 0)
      for (i=0; i<3; i++)
      {
         kill (getppid(), SIGUSR1);
         sleep(5);
      }
   else
```

```
        if (pid >0)
        {
            signal (SIGUSR1,funcao);
            for (; ;)
               pause();
        }
}
```

RESPOSTA:
O programa "prog" (ls) é executado apenas uma vez. No recebimento do sinal USR1 é executada a rotina de tratamento do sinal, denominada “funcao()”. Esta rotina substitui o código do processo pai pelo código do programa "prog". O processo filho, por sua vez, continua em loop devido ao comando "for" (seu código não foi substituído). Em cada laço do "for" ele envia o sinal USR1 ao seu processo pai. Observe que, neste momento, ele já foi (terá sido) adotado pelo "init", já que o seu pai original "morreu".

Questão 5 (2,0 pontos): Anulada
O enunciado na folha de prova continha um erro: ao invés de ***lê***_*dados_do_buffer*, estava escrito ***retira***_*dados_do_buffer*. A solução que atende aos requisitos do enunciado correto é dada abaixo. Observe os valores de inicialização dos semáforos.

```
semaphore s0 = 2;
semaphore s1 = 0;
semaphore s2 = 0;
```

```
Produtor                          Consumidor_1                    Consumidor2
...                               ...                             ...
repeat                            repeat                          repeat
   P(s0); P(s0);                     P(s1);                          P(s2);
   coloca_dados_no_buffer();         lê_dados_do_buffer();           lê_dados_do_buffer();
   V(s1); V(s2);                     V(s0);                          V(s0);
until forever                     until forever                   until forever
```

Questão 6 (1,5 pontos) –
Problemas com a solução que desabilita as interupções (DI/EI):
Critério: 1,0 ponto se abordo algum item sublinhado na resposta abaixo e 1,5 caso tenha abordado dois ou mais itens. “Besteira” = -0,5ponto.

- Esta solução não funciona em ambientes com vários processadores. Em uma máquina multiprocessada ou multicore, duas tarefas concorrentes podem executar simultaneamente em processadores separados, acessando a seção crítica ao mesmo tempo.
- Inibir interrupções por um longo período de tempo provoca a perda da sincronização com os dispositivos periféricos. Enquanto as interrupções estão desativadas, os dispositivos de E/S deixam de ser atendidos pelo núcleo, o que pode causar perdas de dados ou outros problemas. Por exemplo, uma placa de rede pode perder novos pacotes se seus buffers estiverem cheios e não forem tratados pelo núcleo em tempo hábil. A tarefa que está na seção crítica não pode realizar operações de E/S, pois os dispositivos não irão responder.
- Ao inibir as interrupções, a preempção por tempo deixa de funcionar. Caso a tarefa entre em um laço infinito dentro da seção crítica, o sistema inteiro será bloqueado. Uma tarefa mal-intencionada pode forçar essa situação e travar o sistema.
- É desaconselhável dar aos processos de usuário o poder de desabilitar interrupções.