DI/CT/UFES - 2ª. Prova de Sistemas Operacionais I
Período: 2009/2 - Profª. Roberta Lima Gomes
GABARITO

1. *Espera-se que a resposta do aluno cubra aspectos descritos na "resposta-padrão" abaixo*:

O que são *threads*? O que uma *thread* acrescenta ao modelo de processo?

Em Sistemas Operacionais tradicionais, cada processo tem um espaço de endereçamento e um único fluxo (thread) de execução. Threads são partes de código de um processo que podem ser executadas de forma independente e/ou paralela. O que a thread acrescenta ao modelo de processo é permitir que múltiplas execuções ocorram no mesmo ambiente do processo com um grau de independência uma da outra, além de compartilhar diversos recursos do processo, ganhando em eficiência computacional. As Threads são mais fáceis de se criar e destruir que os processos, pois não possuem recursos associados aos mesmos. Em sistemas operacionais com suporte a threads, o processo está associado à alocação de recursos enquanto a thread representa a unidade de escalonamento.

Quais os itens compartilhados por todas as *threads* de um processo?

Os itens compartilhados são o espaço de enderaçamento, as variáveis globais, os arquivos abertos, os processos filhos, os alarmes pendentes, os sinais e tratadores de sinais e as informações de contabilidade.

Diferencie *user-level threads* de *kernel-level threads*.

ULT:
- Implementadas através de bibliotecas de usuário; gerenciadas pela aplicação (inclusive escalonamento) e não enxergadas pelo kernel
  - Qdo uma ULT é bloqueada, todo o processo é bloqueado;
  - Não se beneficia de multiprocessamento;
- mais eficiente pois não exige passagem p/ kernel mode.

KLT:
- São suportadas pelo kernel;
- Operações sobre elas realizadas através de SVCs (ex: criar, sincronizar, destruir thread);
  - São mais "pesadas" , gerência exige passagem p/ *kernel mode*;
- Escalonamento definido pelo kernel
  - unidades de escalonamento independentes (se uma thread bloqueia, ñ são bloqueadas as demais threads do processo)
  - Pode usufruir de multiprocessamento;

2.

Semaphore C1=0, C2=0, W=2		

Processo Produtor
repeat
  P(W)
  P(W)
  coloca dados no buffer
  V(C1)
  V(C2)
until forever
	Processo Consumidor1
	repeat
	  P(C1)
	  retira dados do buffer
	  V(W)
	until forever
		Processo Consumidor2
		repeat
		  P(C2)
		  retira dados do buffer
		  V(W)
		until forever

3. *Nesta solução considera-se que o (M+1)ésimo canibal, ao encontrar a travessa vazia , acorda o cozinheiro. O cozinheiro só desbloqueia esse canibal (e os seguintes) após encher a travessa (e fazer "V(comida)" M vezes).*

```
semaphore cozinha = 0 ;
semaphore comida = M+1;
semaphore mutex = 1;
semaphore enchendo = 0;
int count = 0;
```

Canibal

```
While (1) {
        P(comida);
        P(mutex);
        count++;
        if (count > M)  {
                V(cozinha);
                P(enchendo);
                count=1;
        }
        V(mutex);
        come();
}
```

Cozinheiro

```
While (1) {
        P(cozinha);
        enche_travessa();
        for (int i=1; i ≤ M; i++)
                V(comida);
        V(enchendo);
}
```

4.

```
GLOBAL VARIABLES: mutex, wrt , rd: semaphores initialized to 1;
                  readcount  : integer initialized to 0;
READER: ......
        p(rd);
        p(mutex);
        readcount := readcount+1;
        if readcount=1 then p(wrt);
        v(mutex)
        v(rd);
        read
        p(mutex);
        readcount := readcount-1;
        if (readcount=0) then V(wrt);
        v(mutex);
        ....
WRITER:       .......
        p(rd);
        p(wrt);
        write
        v(wrt);
        v(rd);
        ....
```