Lista 1

1. Um computador tem: memória cache, memória principal e um disco usado para memória virtual. Se uma palavra procurada estiver na cache, serão necessários 20 ns para acessa-la. Se estiver na memória principal mas não na cache, serão precisos 60 ns para carrega-la na cache, seguidos de 20 ns para ler a palavra na cache. Se a palavra não estiver na memória principal, são precisos 12 ms para buscar a palavra no disco, seguidos por 60 ns para copia-la na cache, e depois 20 ns para ler a palavra. A razão de ocorrência na cache é de 0.9 e a razão de ocorrência na memória principal é de 0.6. Qual o tempo médio (em ns) para acesso a uma palavra procurada neste sistema?

2. O sistema operacional `Windows` roda em todos os computadores IBM-PC compatíveis (`arquitetura Wintel`). O sistema operacional `Mac` roda nos computadores `Macintosh` da `Apple Computer`, sendo o hardware e o software projetados pela `Apple`. Quais são as vantagens e desvantagens dessas duas diferentes políticas comerciais? Quais as possíveis consequências sobre os respectivos sistemas operacionais?

3. O MS-DOS não proporcionou meios para processamento concorrente. Discuta três importantes complicações que o processamento concorrente agrega a um sistema operacional.

4. Tente explicar por que uma máquina `UNIX` não é considerada útil para aplicações em tempo real devido ao fato de que um processo que roda no modo *kernel* não pode ser suspenso (preempted).

Lista 2

1. Considere o diagrama de transição entre estados do processo no caso de cinco estados (Novo, Pronto, Rodando, Bloqueado, Terminado). Por que não há uma transição do estado Bloqueado para o estado Rodando?

2. Considere o código em C abaixo sobre o uso da instrução `fork()`. Descreva o funcionamento do código. Rodando esse código, quantos processos são gerados?

```
int main()
{
int pid;
int pid2;
int i;
pid = getpid();
printf("PID = %4d \n", pid);
for (i=0; i<3; ++i)
```

```
  {
    pid = getpid();
    printf("i = %4d, PID = %4d \n", i, pid);
    pid2=fork();
    if (pid2 != 0) printf("I am the parent %4d, i = %4d \n", pid2, i);
    if (pid2 == 0) printf("I am a child %4d, i = %4d \n", pid2, i);
    sleep(5);
  }
}
```

3. Na Figura abaixo cada processo pertence a uma fila específica.

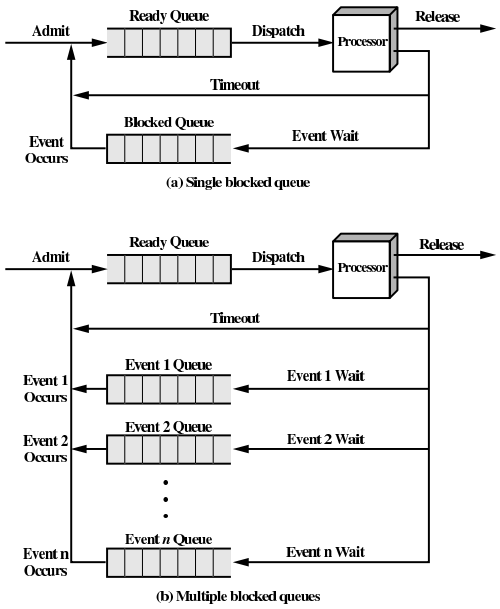

Figure 3.7 Queuing Model of Figure 3.5

(a) É possível que em alguma circunstância seja útil permitir ao mesmo processo pertencer a duas filas distintas? Apresente um exemplo.

(b) Nesse caso como se deveria modificar a estrutura na Figura para implementar essa possibilidade?

4. Por que quando uma thread, que é executada a nível de usuário, executa uma chamada de sistema todas as threads do mesmo processo são bloqueadas?