Capítulo 2

Gerência de atividades

1. **Explique o que é, para que serve e o que contém um PCB - *Process Control Block*.**

   São descritores presente em cada tarefas ou estrutura de dados que representa no núcleo. Num PCB, são armazenadas informações relativas ao seu contexto e outros dados relacionado a sua gerencia.
   São encontrados num TCB as seguintes tarefas: identificador da tarefa; estado da tarefa; informações de contexto do processador; lista de áreas de memória usadas pela tarefa; listas de arquivos abertos, conexões de rede e outros recursos usados pela tarefa e Informações de gerência e contabilização

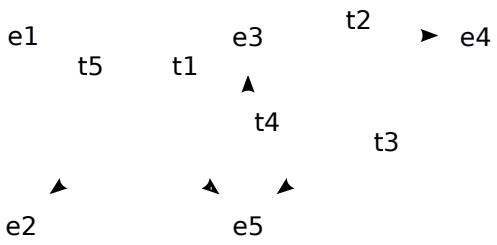
2. **O que significa *time sharing* e qual a sua importância em um sistema operacional?**

   Significa compartilhamento de tempo. Serve para evitar ociosidade do processador e gerenciar o tempo que cada tarefa fica em execução. O tempo em que cada tarefa fica em execução é denominado de quantum. Quando o quantum é esgotado, o processo volta para fila de tarefas prontas e aguarda ser executada novamente. Em um sistema operacional, isso torna importante. porque cada tarefas são executadas em um determinado e evita lentidões no próprio sistema.

3. **Como e com base em que critérios é escolhida a duração de um *quantum* de processamento?**

   O quantum de processamento é escolhido com base nas interrupções geradas pelo temporizador programável do hardware. Normalmente, o temporizador é programado para gerar interrupções em intervalos regulares (a cada milissegundos, por exemplo).

4. Considerando o diagrama de estados dos processos apresentado na figura a seguir, **complete o diagrama** com a transição de estado que está faltando (t_6) e **apresente o significado** de cada um dos estados e transições.



5. Indique se cada uma das transições de estado de tarefas a seguir definidas é possível ou não. Se a transição for possível, dê um exemplo de situação na qual ela ocorre **(*N*: Nova, *P*: pronta, *E*: executando, *S*: suspensa, *T*: terminada).**

   - $E \rightarrow P$
   - $E \rightarrow S$
   - $S \rightarrow E$
   - $P \rightarrow N$
   - $S \rightarrow T$
   - $E \rightarrow T$
   - $N \rightarrow S$
   - $P \rightarrow S$

6. Relacione as afirmações abaixo aos respectivos estados no ciclo de vida das tarefas (N: Nova, P: Pronta, E: Executando, S: Suspensa, T: Terminada):

   [N] O código da tarefa está sendo carregado.
   [P] A tarefas são ordenadas por prioridades.
   [E] A tarefa sai deste estado ao solicitar uma operação de entrada/saída.
   [T] Os recursos usados pela tarefa são devolvidos ao sistema.
   [P] A tarefa vai a este estado ao terminar seu *quantum*.
   [E] A tarefa só precisa do processador para poder executar.
   [S] O acesso a um semáforo em uso pode levar a tarefa a este estado.
   [E] A tarefa pode criar novas tarefas.
   [E] Há uma tarefa neste estado para cada processador do sistema.
   [S] A tarefa aguarda a ocorrência de um evento externo.

7. Desenhe o diagrama de tempo da execução do código a seguir, informe qual a saída do programa na tela (com os valores de *x*) e calcule a duração aproximada de sua execução.

```
int main()
{
    int x = 0 ;

    fork () ; x++ ; sleep (5) ; wait (0) ; fork ()
    ; wait (0) ; sleep (5) ; x++ ; printf ("Valor
    de x: %d\n", x) ;
}
```

8. Indique quantas letras “X” serão impressas na tela pelo programa abaixo quando for executado com a seguinte linha de comando:

    a.out 4 3 2 1

Observações:

- a.out é o arquivo executável resultante da compilação do programa.
- A chamada de sistema fork cria um processo filho, clone do processo que a executou, retornando o valor zero no processo filho e um valor diferente de zero no processo pai.

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

int main(int argc, char *argv[])
{ pid_t pid[10]; int i; int N =
    atoi(argv[argc-2]);

    for (i=0; i<N; i++) pid[i] =
        fork();
    if (pid[0] != 0 && pid[N-1] != 0) pid[N] = fork();
    printf("X"); return 0;
}
```

9. **O que são *threads* e para que servem?**

   Threads são, cada fluxo de execução de um sistema associado a um processo. Serve para dividir um processo em duas ou mais tarefas com objetivo de dividir cada processo executado pelo Sistema Operacional e executa-las de forma concorrentes.

10. **Quais as principais vantagens e desvantagens de *threads* em relação a processos?**

    Os threads são compostos de vantagens e desvantagens e umas dessas vantagens é a facilidade de desenvolvimento, uma vez que possibilita desenvolver o programa em modulos e executa-los de forma isolada. Uma outra vantagem é que não deixam processos parados, portanto quando um deles estiver aguardando um dispositivo de I/O, outro recurso do sistema, um outro thread poderá está em execução.

    Todavia, uma das desvantagens é a complexidades de gerenciamentos de vários threads. Outra desvangem é quando o kernel atribuir um processo apena a uma CPU. Quando isso acontece, não é possível executar dois threads numa arquitetura multiprocessamento.

11. **Forneça dois exemplos de problemas cuja implementação *multi-thread* não tem desempenho melhor que a respectiva implementação sequencial.**

apesar de ser adequados a maiorias das aplicações, os serviços multi-threads são pouco escaláveis. Quando há um numero significativo de threads no kernel o sistema passa a perder desempenho.

12. **Associe as afirmações a seguir aos seguintes modelos de *threads*: a) *many-to-one* (N:1); b) *one-to-one* (1:1); c) *many-to-many* (N:M):**

[A] Tem a implementação mais simples, leve e eficiente.
[B] Multiplexa os *threads* de usuário em um pool de *threads* de núcleo.
[B] Pode impor uma carga muito pesada ao núcleo.
[A] Não permite explorar a presença de várias CPUs pelo mesmo processo.
[C] Permite uma maior concorrência sem impor muita carga ao núcleo.
[A] Geralmente implementado por bibliotecas.
[B] É o modelo implementado no Windows NT e seus sucessores.
[A] Se um *thread* bloquear, todos os demais têm de esperar por ele.
[C] Cada *thread* no nível do usuário tem sua correspondente dentro do núcleo.
[C] É o modelo com implementação mais complexa.

13. Considerando as implementações de *threads* N:1 e 1:1 para o trecho de código a seguir, a) desenhe os diagramas de execução, b) informe as durações aproximadas de execução e c) indique a saída do programa na tela. Considere a operação sleep() como uma chamada de sistema (*syscall*).

    Significado das operações:

- thread_create: cria uma nova *thread*, pronta para executar.
- thread_join: espera o encerramento da *thread* informada como parâmetro.
- thread_exit: encerra a *thread* corrente.

```
int y = 0 ;

void threadBody
{
    int x = 0 ;
    sleep (10) ; printf ("x: %d, y:%d\n", ++x, ++y)
    ; thread_exit();
}

main ()
{ thread_create (&tA, threadBody, ...) ;
    thread_create (&tB, threadBody, ...) ; sleep (1) ;
    thread_join (&tA) ; thread_join (&tB) ; sleep
    (1) ; thread_create (&tC, threadBody, ...) ;
    thread_join (&tC) ;
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

14. **Explique o que é escalonamento *round-robin*, dando um exemplo.**

Escalonamento round-robin, é um tipo de escalonamento mais simples em agendamentos de processos de sistemas operacionais . Porém distribui melhor as

tarefas no que se refere ao uso do processador. O escalonamento round-robin pode propiciar tempos de resposta maiores em aplicações interativas.

15. Considere um sistema de tempo compartilhado com valor de quantum t_q e duração da troca de contexto t_{tc}. Considere tarefas de entrada/saída que usam em média p% de quantum de tempo cada vez que recebem o processador. Defina a eficiência E do sistema como uma função dos parâmetros t_q, t_{tc} e p.

    $E = t_q/t_q+t_{tc}$

16. **Explique o que é, para que serve e como funciona a técnica de *aging*.**

a técnica aging serve para indicar quanto tempo uma tarefa esta aguardando o processador. Com isso, o nível de prioridade aumenta de forma proporcional. A técnica aging evita a inanição, garantindo proporcionalidade por meio de prioridade.
O funcionamento da *task aging*, ocorre quando a cada turno, o escalonador escolhe uma próxima tarefa (t_p) com maior prioridade dinâmica (p_{dp}), na qual a prioridade dinâmica dessa tarefa é igual a sua prioridade estática (pd p <> pe p) e então ela recebe o processador e a prioridade dinâmica das tarefas são aumentadas de α, ou seja, elas envelhecem e no próximo turno terao mais chances de ser escolhidas.

17. A tabela a seguir representa um conjunto de tarefas prontas para utilizar um processador:

Tarefa	t_1	t_2	t_3	t_4	t_5
ingresso	0	0	3	5	7
duração	5	4	5	6	4
prioridade	2	3	5	9	6

Indique a seqüência de execução das tarefas, o tempo médio de vida (*tournaround time*) e o tempo médio de espera (*waiting time*), para as políticas de escalonamento a seguir:

(a) **FCFS cooperativa**
(b) SJF cooperativa
(c) SJF preemptiva (SRTF)
(d) PRIO cooperativa
(e) PRIO preemptiva
(f) RR com $t_q = 2$, sem envelhecimento

Considerações: todas as tarefas são orientadas a processamento; as trocas de contexto têm duração nula; em eventuais empates (idade, prioridade, duração, etc), a tarefa t_i com menor i prevalece; valores maiores de prioridade indicam maior prioridade.

Para representar a sequência de execução das tarefas use o diagrama a seguir. Use × para indicar uma tarefa usando o processador, – para uma tarefa em espera na fila de prontos e para uma tarefa que ainda não iniciou ou já concluiu sua execução.

18. Idem, para as tarefas da tabela a seguir:

Tarefa	t_1	t_2	t_3	t_4	t_5
ingresso	0	0	1	7	11
duração	5	6	2	6	4
prioridade	2	3	4	7	9

19. **Explique os conceitos de *inversão* e *herança de prioridade*.**

A inversãode prioridades consiste em processos de alta prioridade serem impedidos deexecutar por causa de um processo de baixa prioridade. Um exemplo de como pode ocorrer uma inversão de prioridades:
1.Em um dado momento, o processador está livre e é alocado a um processo de baixa prioridade pb;
2. durante seu processamento, pb obtém o acesso exclusivo a umrecurso R e começa a usá-lo;
3. pb perde o processador, pois um processo com prioridade maior que a dele (pm) foi acordadodevido a uma interrupção;
4.pb volta ao final da fila de tarefas prontas, aguardando o processador; enquanto ele não voltara executar, o recurso R permanecerá
alocado a ele e ninguém poderá usá-lo;
5.Um processo de alta prioridade para recebe o processador e solicita acesso ao recurso R; como orecurso está alocado ao processo pb, pa
é suspenso até que o processo de baixa prioridade pblibere o recurso. Neste momento, o processo de alta prioridade pa não pode continuar sua execução, porque o recurso de que necessita está nas mãos do processo de baixa prioridade pb. Dessa forma, pa deve esperar que pb execute e libere R, o que justifica o nome inversão de prioridades.

20. Você deve analisar o software da sonda *Mars Pathfinder* discutido no livro-texto. O sistema usa escalonamento por prioridades preemptivo, sem envelhecimento e sem compartilhamento de tempo. Suponha que as tarefas t_g e t_m detêm a área de transferência de dados durante todo o período em que executam. Os dados de um trecho de execução das tarefas são indicados na tabela a seguir (observe que t_g executa mais de uma vez).

Tarefa	t_g	t_m	t_c
ingresso	0, 5, 10	2	3

duração	1	2	10
prioridade	alta	baixa	média

Desenhe o diagrama de tempo da execução sem e com o protocolo de herança de prioridades e discuta sobre as diferenças observadas entre as duas execuções.