Gustavo Laires Albuquerque Costa

Questões do Capítulo 2

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

Os PCB (*Process Control Block)* ou TCB (*Task Control Block)* são estruturas que armazenam as informações necessárias à gerência de uma tarefa. Os PCB contém:

- **Identificador da tarefa, estado da tarefa;**
- **Informações de contexto do processador;**
- **Lista de áreas de memória usadas pela tarefa;**
- **Lista de arquivos abertos, conexões de rede e outros recursos utilizados pela tarefa;**
- **Informações de gerência e contabilização.**

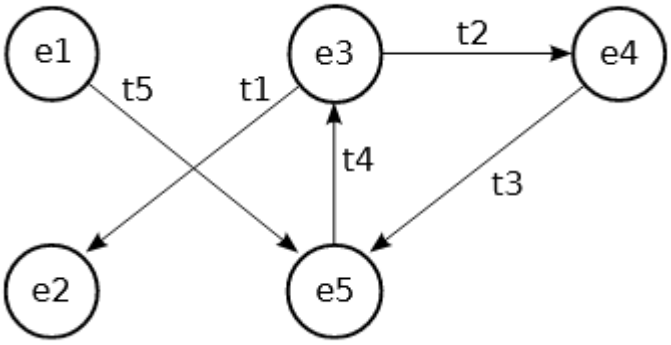
2. O que significa *time-sharing* e qual a sua importância em um sistema operacional?

O *time-sharing* (compartilhamento de tempo) é uma solução implementada nos sistemas operacionais para evitar problemas de longa duração na execução de tarefa (seja por questões de entrada e saída de dados ou por *loops* infinitos). Com essa solução a tarefa recebe um tempo determinado de processamento, após esse período o processamento é cedido a outra tarefa.

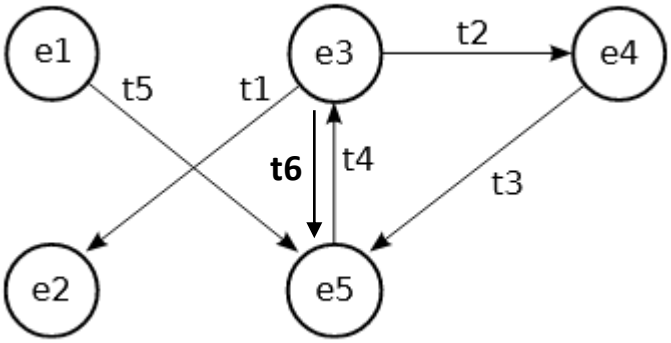
3. Como e com base em que critérios é escolhida a duração de um *quantum* de processamento?

A duração do *quantum* depende do sistema operacional, do tipo e da prioridade da tarefa.

4. Considerando o diagrama de estados dos processos apresentado na figura a seguir, complete o diagrama com a transição de estado que está faltando (t6) e apresente o significado de cada um dos estados e transições.



e1 = Nova;
e2 = Terminada;
e3 = Executando;
e4 = Suspensa;
e5 = Pronta;



t1 = Execução finalizada;
t2 = Aguardando dado externo ou outro evento;
t3 = Dado disponível ou evento ocorreu;
t4 = Recebe o processador;
t5 = Carregada em memória;
t6 = Fim do *Quantum;*

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→P

Sim, quando o *quantum* da tarefa chega ao fim.

- E→S

Sim, quando a tarefa depende de algum dado externo ou outro evento.

- S→E

Não.

- P→N

Não.

- S→T

Não.

- E→T

Sim, quando a execução da tarefa é finalizada.

- N→S

Não.

- P→S

Não.

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.
[**P**] 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 fluxos de execução do sistema associados a um processo ou ao interior do núcleo. A separação da execução em *Threads* permite a execução simultânea de cada *Thread* por um processador ou a sensação de execução simultânea de *Threads* diferentes (situação de processador de núcleo único com modelo de *Threads* N:1)**

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

Vantagens:

- **Permite executar múltiplas tarefas dentro de um processo ao mesmo tempo;**
- **Melhor aproveitamento de processadores de múltiplos núcleos;**

Desvantagens:

- **Pode gerar maior sobrecarga no núcleo;**
- **Pode reduzir o tempo de execução de cada *thread;***
- **Pode aumentar a complexidade de implementação;**

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

A implementação *multi-thread* não tem desempenho melhor em problemas de resolução sequencial, onde há dificuldades de dividir a resolução do problema em etapas que podem ser executadas de maneira concorrente, e problemas que exigem intenso e repetido de operações bloqueantes, o que pode gerar competição pelos recursos disponíveis.

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.
[**C**] 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.
[**B**] 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 ;

voidthreadBody
{
    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);
}
```

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

O escalonamento Round-Robin (ou escalonamento por revezamento) é um algoritmo de escalonamento de tarefas no qual as tarefas são atendidas segundo sua entrada na fila de tarefas prontas. Entretanto, nesse algoritmo as tarefas possuem um tempo de dedicação dos recursos de hardware (*Quantum*), passado esse intervalo de tempo e a tarefa ainda não foi finalizada, a tarefa retorna a fila de tarefas prontas e os recursos de hardware são disponibilizados a próxima tarefa da fila.

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

Solução:

$$E = \frac{\frac{p \times tq}{100}}{\frac{p \times tq}{100} + ttc} = \frac{\frac{p \times tq}{100}}{\frac{p \times tq + 100ttc}{100}} = \frac{p \times tq}{p \times tq + 100ttc}$$

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

O *task aging* (envelhecimento) é uma forma de garantir que tarefas que estão aguardando o processador tenham suas prioridades aumentas, permitindo que quando seja realizado o escalonamento de tarefas por prioridades as tarefas de baixa prioridade não fiquem em inanição (nunca recebam a

dedicação dos recursos de hardware). A técnica funciona incrementando a prioridade das tarefas por uma fator de envelhecimento ao longo do tempo. Com isso, a cada turno o escalonador escolhe como a próxima tarefa a ser executada aquela que possuir maior prioridade.

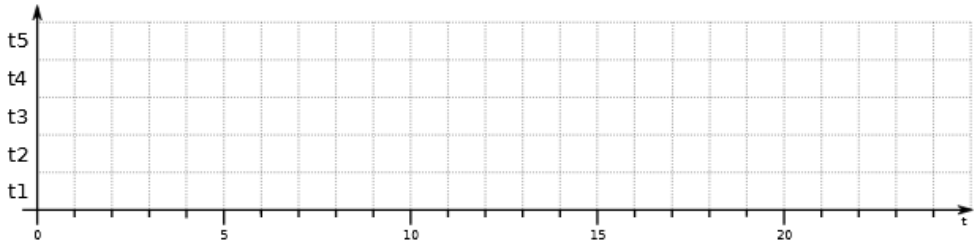
17. A tabela a seguir representa um conjunto de tarefas prontas para utilizar um processador:

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 sequência 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 cooperativa10
(b) SJF cooperativa
(c) SJF preemptiva (SRTF)
(d) PRIO cooperativa
(e) PRIO preemptiva
(f) RR comtq=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 ti 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 inversão de prioridade ocorre quando uma tarefa de baixa prioridade mantém um recurso alocado e impedido de uso por outras tarefas após perder o processador para tarefas de prioridade maior. Como o recurso solicitado está alocado pela tarefa de baixa prioridade, as tarefas de alta prioridade solicitantes precisam aguardar a liberação do recurso. Isso define a ocorrência da inversão de prioridade.

Como solução para o problema da inversão de prioridade existe a herança de prioridade. Através da herança de prioridade, a tarefa de baixa prioridade, que tem um recurso alocado e requisitado por outra tarefa de prioridade mais elevada, herda a prioridade mais alta para que volte a ser processado em menor tempo, liberando assim o recurso mais rapidamente.

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 tg e tm 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 tg 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.