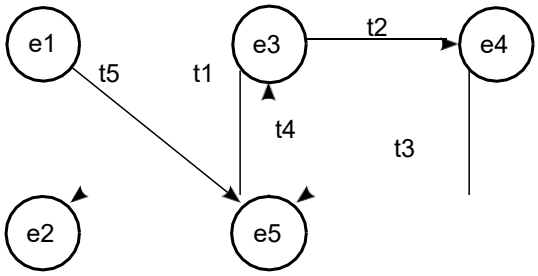
Aluno: Jackson Ricardo dos Santos da Silva
Turma: 2019.3.01404.1M
Respostas do capítulo 2 – Gerência de atividades

1. Explique o que é, para que serve e o que contém um PCB - *Process Control Block*.
   R: PCB é um a estrutura de dados que serve para armazenar as informações relativas ao contexto e os demais dados necessários à gerencia de uma tarefa presente no sistema. Ele serve também para que seja efetuada a Troca de Contexto o que é: interromper a execução de uma tarefa e retornar a ela mais tarde, sem corromper seu estado interno.

2. O que significa *time sharing* e qual a sua importância em um sistema operacional?
   R: Time sharing(compartilhamento de dados), foi um conceito introduzido nos anos 60 para resolver a inviabilização do sistema devido a uma tarefa em execução que nunca termina e nem solicita operações de entrada/saída, monopolizando o processador e impedindo a execução dos demais tarefas.Nessa solução, cada atividade que detém o processador recebe um limite de tempo de processamento, denomidado quantium. Esgotado seu quatium, a tarefa em execução perde o processador e volta para uma fila de tarefas "prontas", que estão na memória aguardando sua oportunida de executar.

3. Como e com base em que critérios é escolhida a duração de um *quantum* de processamento?
   R: A duração atual do quatum depende muito do tipo de sistema operacional. No Linux ela varia de 10 a 200 milissegundos, dependendo do tipo e prioridade da tarefa. Vários critérios podem ser definidos para a avaliação de escalonadores(quem define a duração dos quantums).

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 → P:* ***esta transição ocorre quando se esgota a fatia de tempo destinadaàtarefa(ouseja,ofimdoquantum);comonessemomentoatarefanãoprecisa de outros recursos além do processador, ela volta à fila de tarefas prontas, para esperar novamente o processador***
   - *E → S:* ***caso a tarefa em execução solicite acesso a um recurso não disponível, como dados externos ou alguma sincronização, ela abandona o processador e fica suspensa até o recurso ficar disponível***

- $S \rightarrow E$
- $P \rightarrow N$
- $S \rightarrow T$
- $E \rightarrow T$***: ocorrequandoatarefaencerrasuaexecuçãoouéabortada em consequência de algum erro (acesso inválido à memória, instrução ilegal, divisão por zero, etc.)***
- $N \rightarrow S$
- $P \rightarrow S$

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.

```
1  int main()
2  {
3     int x = 0 ;
4
5     fork () ;
6     x++ ;
7     sleep (5) ;
8     wait (0) ;
9     fork () ;
10    wait (0) ;
11    sleep (5) ;
12    x++ ;
13    printf ("Valor de x: %d\n", x)  ;
14 }
```

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.

```
1  #include <stdio.h>
   #include <sys/types.h>
2  #include <unistd.h>
   #include <stdlib.h>
3
4  int main(int argc, char   argv[])
5  {
       pid_t pid[10];
6      int i;                    *
7      int N = atoi(argv[argc-2]);
8
       for (i=0; i<N; i++)
9          pid[i] = fork();
       if (pid[0] != 0 && pid[N-1] != 0)
10         pid[N] = fork();
11     printf("X");
       return 0;
12 }
13
14
15
16
17
18
19
```

9. O que são *threads* e para que servem?
   R: De forma geral, cada fluxo de execução do sistema, seja associado a um processo ou no interior do núcleo, é denominado thread. Threads servem para se executar mais de um processo oa mesmo tempo.

10. Quais as principais vantagens e desvantagens de *threads* em relação a processos?
R: Vantagens: Leve e fácil implementação. Como o núcleo somente considera uma thread, a carga de gerência imposta ao núcleo é pequena e não depende do núcleo número de threads dentro da aplicação.

Desvantagens: Como essas operações são intermediadas pelo núcleo, se um thread de usuário solicitar uma operação de E/S (recepção de um pacote de rede) o thread de núcleo correspondente será suspenso até a conclusão da operação, fazendo com todos os threads de usuário associados ao processo parem de executar enquanto a operação não for concluída.

11. Forneça dois exemplos de problemas cuja implementação *multi-thread* não tem desempenho melhor que a respectiva implementação sequencial.
R: O modelo de threads 1:1(multi-thread) é adequado para a maioria das 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 uma carga significativa ao núcleo do sistema, inviabilizando aplicações com muitas tarefas(como grande servidores Web e simulações de grande porte).

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.

[**b**]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. [ ]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.

```
1  int y = 0 ;
2
3  void threadBody
4  {
5     int x = 0 ;
6     sleep (10) ;
7  printf ("x: %d, y:%d\n", ++x, ++y) ;
8     thread_exit();
9  }
10
11 main ()
12 {
13    thread_create (&tA, threadBody,  ;
                                    …)
14    thread_create (&tB, threadBody,  ;
                                    …)
15    sleep (1) ;
16    thread_join (&tA) ;
17    thread_join (&tB) ;
18    sleep (1) ;
19    thread_create (&tC, threadBody,  ;
                                    …)
20    thread_join (&tC) ;
21 }
```

14. Explique o que é escalonamento *round-robin*, dando um exemplo.
R: Round-robin (RR) é um dos algoritmos mais simples de agendamento de processos em um sistema operacional, que atribui frações de tempo para cada processo em partes iguais e de forma circular, manipulando todos os processos sem prioridades. Escalonamento Round-Robin é simples e fácil de implementar.

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 seu 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.

R: E = tq / tq + ttc

16. Explique o que é, para que serve e como funciona a técnica de *aging*.
R: Para evitar a inanição(quando um processo nunca assegura um recurso) e garatntir a proporcionalidade expressa através das prioridades estáticas, um fator interno denominado envelhecimento(task aging) deve ser definido. O envelhecimento indica há quanto tempo uma tarefa está aguardando o processadador e aumenta sua prioridade

proporcionalemente. Dessa forma, o envelhecimento evita a inanição dos processos de baixa prioridade, permitindo a eles obter o processador periodicamente.

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)

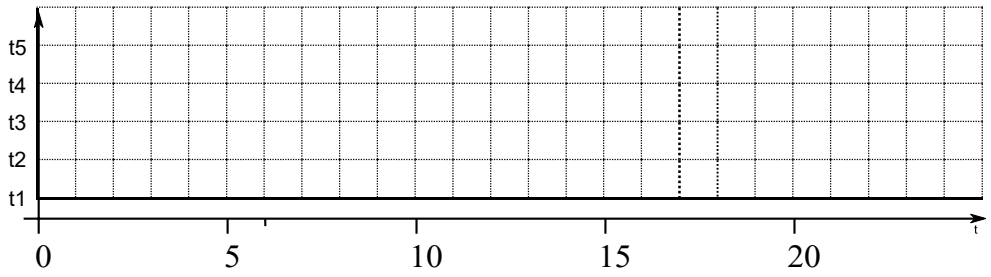
(d)PRIO cooperativa

(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, fila de para uma tarefa em espera o prontos e para uma tarefa que ainda n 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*.
R: A inversão de prioridades consiste em processo de alta prioridade serem impedidos de executar, por causa de um processo de baixa prioridade. A herança de prioridade é um método para eliminar a inversão de prioridades não ligada, a herança de prioridade consiste em aumentar a prioridade do processo.

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.