Universidade do Estado de Santa Catarina – Bacharelado em Ciência da Computação
Sistemas Operacionais – Prof. Rafael R. Obelheiro – Semestre: 2020.1

Trabalho 1 – Processos e Threads

Sobre o trabalho

- O trabalho é individual.
- Exercícios de implementação devem ser realizados na linguagem C, e serão testados no sistema operacional Linux. As submissões estarão sujeitas a controle de plágio usando ferramentas de detecção de similaridade.
- Exercícios teóricos devem ser submetidos em formato PDF. Para os exercícios de implementação deve ser submetido o código fonte, em formato compilável (ou seja, os arquivos `*.c`, e não listagens em PDF). Submeta um único arquivo ZIP contendo todas as suas respostas (teóricas e de implementação).
- O trabalho deverá ser entregue até **SEGUNDA-FEIRA, 20 DE JULHO**. O Moodle aceitará submissões até 23h59min, sendo automaticamente bloqueado após a data limite. É **RESPONSABILIDADE DOS ALUNOS** garantir que o trabalho seja entregue no prazo.
- Em caso de dificuldades ou dúvidas na interpretação do trabalho, entre em contato com o professor (`rafael.obelheiro@udesc.br`).

Exercícios

1. Considere um programa concorrente com três *threads*, `A`, `B` e `C`, mostradas no pseudocódigo abaixo. Deseja-se que a sequência de execução seja tal que o programa imprima para toda a eternidade (ou enquanto o programa executar) a mensagem "`Kit Kat\n`". Mostre como garantir isso usando semáforos. A solução não deve alterar a estrutura do pseudocódigo existente (por exemplo, trocar um `while` ou `for` por outra coisa, ou mudar sua condição), apenas incluir instruções para garantir a sequência de execução desejada. Você pode inicializar os semáforos como variáveis regulares (`semaphore x=1`) e manipulá-los com as primitivas `down()` e `up()` ou `sem_wait()` e `sem_post()`.

   NOTA: este exercício é teórico; trabalhe com pseudocódigo, sem se preocupar em produzir uma implementação completamente funcional.

```
A1  void A() {
A2    while (1) {
A3      printf("i");
A4      printf("a");
A5    }
A6  }
```

```
B1  void B() {
B2    int i;
B3    while (1) {
B4      for (i=0; i<2; i++) {
B5        printf("t");
B6        if (i == 0)
B7          printf(" ");
B8      }
B9    }
B10 }
```

```
C1  void C() {
C2    int j;
C3    while (1) {
C4      for (j=0; j<2; j++) {
C5        printf("K");
C6      }
C7      printf("\n");
C8    }
C9  }
```

2. Escreva um programa em C no Linux usando a biblioteca Pthreads e que atenda aos seguintes requisitos:

   R1. O programa recebe dois parâmetros m e n na linha de comando; ou seja, o programa deve ser invocado como

   ```
   $ ./prog m n
   ```

   R2. O programa deve alocar dinamicamente uma matriz de $m \times n$ números inteiros, onde m é o número de linhas e n é o número de colunas, e preenchê-la com valores aleatórios (dica: use a função `random()`).

   R3. O programa deve criar m *threads*, todas elas executando a mesma função.

   R4. Cada *thread* deve receber como parâmetros, pelo menos, o seu número de identificação (de 1 a m) e um ponteiro para a m-ésima linha da matriz (podem ser passados quaisquer parâmetros adicionais, conforme a necessidade).

   R5. Cada *thread* deve contar quantos números ímpares existem na sua linha da matriz.

R6. A contagem em cada *thread* só pode ser iniciada depois que a última *thread* tiver sido criada.

R7. Ao encerrar a contagem, uma *thread* deve inserir o seu número de identificação e a quantidade de ímpares em sua linha da matriz em uma lista que segue a ordem de encerramento (i.e., o *k*-ésimo elemento da lista será a *k*-ésima *thread* a encerrar a contagem). Essa lista pode ser estática ou dinâmica.

R8. Depois de atualizar a lista, uma *thread* deve bloquear até que a última *thread* termine o seu processamento. Em outras palavras, uma *thread* só pode encerrar sua execução quando todas as *threads* tiverm concluído a contagem.

R9. O programa principal (`main()`) deve esperar que todas as *threads* terminem, e a seguir mostrar a lista com a ordem de encerramento das *threads* e o total de números ímpares na matriz.

R10. O programa deve tratar condições de disputa no código.

O exemplo a seguir ilustra a saída esperada do programa para a matriz $M = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 6 & 8 & 10 & 9 \\ 11 & 21 & 31 & 43 \end{pmatrix}$ quando as *threads* terminam na ordem 1, 3, 2:

```
$ ./prog 3 4
  1: thread   1 =>      2 impares
  2: thread   3 =>      4 impares
  3: thread   2 =>      1 impares
---------------------------------
Total=7 impares
```

A *thread* 1 processa a linha 1, que tem dois ímpares (3 e 5), a *thread* 2 processa a linha 2, onde 9 é o único ímpar, e a *thread* 3 processa a linha 3, onde todos os números são ímpares.

3. Três programas devem ser executados em um computador monoprocessado. Todos os programas são compostos por dois ciclos de processador e um ciclo de E/S. A entrada e saída de todos os programas é feita sobre a mesma unidade de disco. Os tempos para cada ciclo de cada programa são mostrados abaixo:

Programa	CPU	Disco	CPU	instante de chegada
A	2	4	5	3
B	4	4	2	1
C	9	4	4	0

Observe que nem todos os processos chegam no mesmo instante. O algoritmo de escalonamento de CPU usado no sistema é o de múltiplas filas com realimentação, onde as filas são escalonadas por prioridade e os processos em cada fila por *round-robin*, de acordo com o seguinte esquema:

- fila de alta prioridade: $quantum = 2$
- fila de baixa prioridade: $quantum = 5$

Os processos iniciam um ciclo de CPU na fila de alta prioridade, e mudam de fila caso não tenham concluído seu ciclo de CPU ao término do *quantum*. A chegada de um processo na fila de alta prioridade preempta um processo de baixa prioridade que esteja em execução; quando isso ocorre, o processo preemptado permanece na mesma posição da fila de baixa prioridade, e ao retomar a CPU ele executa pelo tempo remanescente do *quantum*.

(a) Construa um diagrama de tempo mostrando qual programa está ocupando o processador e o disco a cada momento, até que todos os programas terminem.

(b) Determine o tempo médio de espera para o conjunto de processos.

(c) Determine o tempo médio de retorno para o conjunto de processos.

(d) Determine a vazão para essa escala, sabendo que cada unidade de tempo equivale a 0,1 s.

(e) Esse algoritmo está sujeito a inanição? Justifique.