Lista Prática 2 – Ambiente Linux

1) Antes de começarmos, vamos preparar o ambiente para a prática. Siga os passos abaixo:
   - Abra o terminal do Ubuntu (CTRL + ALT +T)
   - Digite o comando “ls” (sem aspas), sua função é listar o conteúdo de um diretório.
   - Crie uma pasta para você salvar os arquivos da prática. Para isso utilize o comando “mkdir Lab02”, este comando vai criar uma pasta chamada Lab02.
   - Digite “ls” e veja se você consegue visualizar a nova pasta criada.
   - Após criar a pasta, digite “cd Lab02” para acessar o conteúdo da pasta.
   - Acesse o moodle e baixe o arquivo “codes2.zip”. Salve-o na pasta “Lab02”
   - Digite o comando “unzip codes2.zip” para descompactar os arquivos.
   - Caso tenha alguma dúvida sobre os parâmetros de um comando, digite “man” e depois o nome do comando para ver a documentação.

2) Comandos Linux – Execute os comandos relacionados a execução de processos em background e foreground. Responda as questões a seguir. Antes de iniciar a resolução, abra o terminal do Ubuntu (CTRL+ALT+T).

   a) Inicie um processo em background que não faça nada durante 5 minutos: “sleep 5m &”. Utilize o comando “Jobs”. O que apareceu na tela?

   b) Inicie mais dois processos em background: “sleep 10m &” e “sleep 15m &”. Utilize novamente o comando jobs e analise o resultado.

   c) Com base no item b, traga o segundo processo para "foreground" com “fg %2”. O que aconteceu?

   d) Suspenda o processo do item c com CTRL-Z. Experimente novamente o comando “jobs”. O que vê?

   e) Coloque o processo novamente em "background" com o comando “bg %2”.

   f) Use o comando “kill %2” e em seguida o comando “jobs”. O que aconteceu?

   g) Use o comando “ps –lA” para ver os processos que estão em execução e acabe com os sleeps recorrendo ao seu PID. Utilize o comando “kill PID”. Execute novamente o comando “ps –lA” para verificar o que aconteceu.

3) Threads - Definição: Acesse o diretório onde você salvou os arquivos de “codes2.zip”.

- Procedimento 1:
  1. Abra o terminal do Ubuntu (CTRL + ALT +T)
  2. Acesse o diretório em que você salvou os arquivos
  3. Digite o comando nano thread1.c
  4. Analise o código-fonte. Digite (Ctrl+X) para sair do editor
  5. Execute o código-fonte
  6. Para compilar digite: gcc -pthread thread1.c – o thread1
  7. Execute o comando: ./thread1

Responda as questões a seguir:

a) O que aconteceu ao executar o código-fonte? Quantos threads foram criados?

b) Tente executar o passo 7 algumas vezes. É possível determinar uma ordem específica para a execução dos threads de 1 a 3? Quem determina a ordem que esses threads serão executados?

- Procedimento 2:
  1. Abra o terminal do Ubuntu (CTRL + ALT +T)
  2. Acesse o diretório em que você salvou os arquivos
  3. Digite o comando nano thread2.c
  4. Analise o código-fonte. Digite (Ctrl+X) para sair do editor
  5. Execute o código-fonte
  6. Para compilar digite: gcc -pthread thread2.c – o thread2
  7. Execute o comando: ./thread2

Responda as questões a seguir:

c) Ao analisar o código-fonte, percebe-se que foi declarado e inicializado um vetor de inteiros, de forma que, o Thread 1 deveria atribuir os valores 10, 20 e 30 para as posições 0, 1 e 2, respectivamente. Já o Thread 2, deveria atribuir os valores 40 50 e 60 para as posições 3, 4 e 5. Ao final da execução, percebe-se que o vetor foi corretamente preenchido pelos Threads. Por que isso foi possível? Qual é uma das principais características de um Thread?

d) Por que o comportamento foi diferente em relação a atividade 4 da prática 1. Qual a principal diferença entre um processo e um thread?

4) Threads – Exemplo de Sincronização: Acesse o diretório onde você salvou os arquivos de “codes2.zip”.
    - Procedimento 1:
        1. Abra o terminal do Ubuntu (CTRL + ALT +T)
        2. Acesse o diretório em que você salvou os arquivos
        3. Digite o comando nano thread3.c
        4. Analise o código-fonte. Digite (Ctrl+X) para sair do editor
        5. Execute o código-fonte
        6. Para compilar gcc -pthread thread3.c – o thread3
        7. Execute o comando: ./thread3

Responda as questões a seguir:

a) O código-fonte apresenta dois threads, um que incrementa a variável global “n” em uma unidade e outro thread que decrementa o valor de “n” em uma unidade. Ambos são executados o mesmo número de vezes. É possível garantir que o valor de “n” será sempre igual a zero após a execução (execute algumas vezes o programa)? Porquê?

b) O que são as condições de corrida (race conditions)? Como podemos evitá-las?

- Procedimento 2:
    1. Abra o terminal do Ubuntu (CTRL + ALT +T)
    2. Acesse o diretório em que você salvou os arquivos
    3. Digite o comando nano thread4.c
    4. Analise o código-fonte. Digite (Ctrl+X) para sair do editor
    5. Execute o código-fonte
    6. Para compilar digite: gcc -pthread thread4.c – o thread4
    7. Execute o comando: ./thread4

Responda as questões a seguir:

e) O código thread4.c é semelhante ao código thread3.c, porém utilizando o conceito de semáforos. O que ocorreu com o valor da variável global “n” ao final da execução do programa? Qual foi seu valor final?

f) Por que o resultado final foi diferente no procedimento 2?

g) Podemos garantir a exclusão mútua utilizando semáforos?