Atividade 1:

2.

Sim, o log de execução foi sempre de acordo com o esperado, como pode ser visto em **hellobyte.log** no repositório.
A única surpresa seria talvez a predominância da thread B entrado no “if” com x = 0 em 28 dos 30 testes. Entrou com a condição x = 1 uma vez e pulou uma vez também (x = 2).

2b)

A condição lógica é de x>=2 para poder prosseguir com o B, que pode ser observado em todos os casos.

3.

A thread B espera eternamente a condição x==2, sinalizada pelos outros fluxos da função A(). Cada thread que executa a função A() aumenta em o x em 1 unidade. Com apenas uma thread, x = 1 para sempre, sem que o sinal de desbloqueio seja lançado.

```
:~/Documents/a.silvana/lab4$ ./hellobye.out
A: Comecei
B: Comecei
B: x= 0, vai se bloquear...
HELLO
^C
:~/Documents/a.silvana/lab4$ ./hellobye.out
A: Comecei
B: Comecei
B: x= 0, vai se bloquear...
HELLO
^C
:~/Documents/a.silvana/lab4$ ./hellobye.out
A: Comecei
B: Comecei
B: x= 0, vai se bloquear...
HELLO
^C
```

5.

Duas pequenas modificações no código, além da adição da variável:

Mudança do signal para broadcast para transmitir para todas as threads

```
x++;
if (x==2) {
    printf("A:  x = %d, vai sinalizar a condicao \n", x);
    pthread_cond_broadcast(&x_cond);
}
pthread_mutex_unlock(&x_mutex);
```

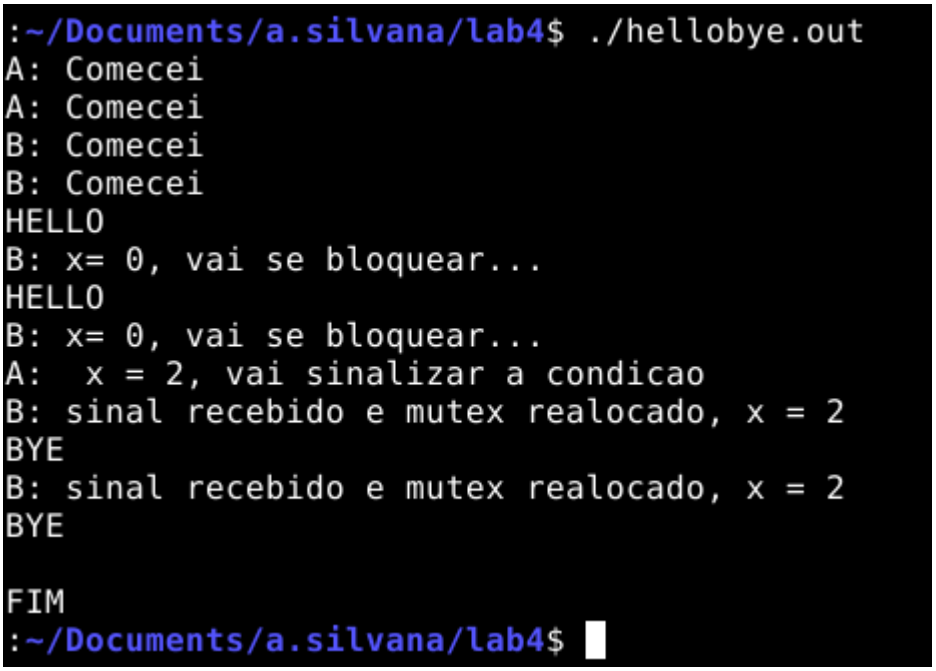
NTHREADS 3 para NTHREADS 4

```
#define NTHREADS  4
```

Adição da variável:

```
/* Cria as threads */
pthread_create(&threads[3], NULL, A, NULL);
pthread_create(&threads[2], NULL, A, NULL);
pthread_create(&threads[1], NULL, B, NULL);
pthread_create(&threads[0], NULL, B, NULL);
```

Tela do terminal:



Ativadade 2:

2.

Sim, como pode ser visto no **byehello.log** no repositório. O programa estava de acordo com o que foi previsto na aula.

2b)

A condição lógica (x>0) para rodar as threads “bye” sempre foi satisfeita em todos os testes. Nenhuma thread ficou permanentemente parada.

Atividade 3

2.

Sim. Como sempre, o programa teve comportamento igual ao mostrado em aula. A condição de x%10=0 também é mantida na impressão.
Tela é mostrada no printX.log.

3.

Não. Como pode ser vista no output, o valor de x quando o sinal é enviado não necessariamente se mantém até que a thread continue a ser executada,i.e.: o valor de x pode não ser múltiplo de 10 quando for impresso. Dessa forma, o while, para checar de novo, seguido pelo grande tempo gasto na função A gera um tempo melhor para o x ser corretamente impresso.
Ao tirarmos esse while, os valores quebrados poderiam ser impressos.

4.

O x passou a ter valores além dos múltiplos de 10, como era de se esperar. Todo o output foi dentro do previsto.
Como descrito, a condição lógica não esteve correta em diversos casos.