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

Trabalho Prático — *Publish-Subscribe*

1 Objetivo

O objetivo deste trabalho é projetar e desenvolver uma aplicação concorrente que implemente um mecanismo de comunicação do tipo *publish-subscribe*.

2 Descrição da Aplicação

Em um sistema de comunicação *publish-subscribe* há duas classes de componentes: os *publishers*, que disseminam dados, e os *subscribers*, que recebem esses dados. Um tipo de sistema *publish-subscribe* é o baseado em tópicos: cada item de dados disseminado por um *publisher* tem um tópico associado, e os *subscribers* recebem apenas os itens de dados referentes aos tópicos de seu interesse, de acordo com um registro efetuado previamente.

O trabalho consiste em uma implementação de *publish-subscribe* baseado em tópicos, em que múltiplas *threads* desempenham o papel de *subscribers*, enquanto uma outra *thread*, que pode ser a própria função `main()`, desempenha o papel de *publisher*, disseminando itens de dados lidos de um arquivo.

O programa recebe dois parâmetros na linha de comando, de acordo com o seguinte formato de invocação:

```
$ pubsub nthr nomearq
```

Os parâmetros são os seguintes:

- `nthr`: número de *threads* representando *subscribers* (≥ 1);
- `nomearq`: nome do arquivo contendo os itens de dados a serem publicados.

O nome de arquivo `nomearq` também é usado para compor o nome do arquivo com os tópicos de interesse de cada *thread*, no formato `nomearq-idThread`, onde *idThread* é um número inteiro de 1 a `nthr`, que representa o identificador da *thread*. Por exemplo, se o programa for invocado com

```
$ pubsub 4 abc
```

o arquivo `abc` conterá os dados que serão disseminados, e os arquivos `abc-1`, `abc-2`, `abc-3` e `abc-4` conterão os tópicos de interesse das *threads* de 1 a 4.

O arquivo contendo os tópicos de interesse de cada *thread* tem um formato bastante simples, em que a primeira palavra de cada linha representa um tópico. Os tópicos são sensíveis a caso: `xyz`, `xYz` e `XYZ` são tópicos distintos. No processamento desse arquivo, é necessário prever a possibilidade de erros de formato, como linhas em branco, linhas com múltiplas palavras, linhas que começam por espaços em branco, e tópicos repetidos; a aplicação deve aproveitar o que for possível (usar a primeira palavra da linha e ignorar as restantes, descartar espaços em branco iniciais) e desconsiderar o que não fizer sentido (linhas em branco, tópicos repetidos).

O arquivo contendo os itens a serem publicados tem o seguinte formato (os nomes das colunas não estão contidos no arquivo):

Tópico	*Conteúdo*
`abc`	`lorem ipsum dolor`
`foo`	`the quick brown fox jumps over the lazy dog`
`book`	`the book is on the table`

A primeira palavra de uma linha corresponde ao tópico, e o restante ao conteúdo. Tópico e conteúdo podem ser separados por espaços ou tabulações. O caractere de fim de linha (`\n`) deve ser descartado, e cada linha pode ter até 255 caracteres.

O programa principal deve criar `nthr` *threads*, e esperar que elas tenham registrado seus tópicos de interesse e estejam prontas para receber dados. Na sequência, o *publisher* deve ler cada item de dados do arquivo de entrada, e repassar esse item apenas para os *subscribers* interessados no tópico correspondente. As linhas de entrada que não respeitarem o formato definido para o arquivo devem ser desconsideradas. O pseudocódigo para o programa principal é o seguinte:

```
cria_threads();
espera_threads_prontas();
libera_publisher();
espera_fim_threads();
imprime_estatisticas();
```

Cada *thread* primeiramente registra seus tópicos de interesse a partir de seu arquivo de entrada, e espera até que todas as demais *threads* estejam prontas. Na sequência, a *thread* deve processar novos itens de dados à medida em que eles vão sendo disseminados. O processamento dos itens consiste em gerar um arquivo de saída `nomearq-`*`idThread`*`.out` com esses itens, no mesmo formato do arquivo de entrada do *publisher*. Quando não houver mais itens a serem processados, a *thread* deve encerrar, retornando o número de itens processados com sucesso e a quantidade de bytes de conteúdo recebidos. O pseudocódigo para uma *thread* é o seguinte:

```
itens_proc = 0; nbytes = 0;
registra_topicos();
sinaliza_thread_pronta_e_espera_demais();
enquanto (houver itens não processados) {
   item = novo_item();
   itens_proc++;
   nbytes += bytes(item.conteudo);
   gera_saida(item.topico, item.conteudo);
}
retorna(itens_proc, nbytes)
```

Não há limites predefinidos para o número de *threads*, de tópicos por *subscriber*, ou de itens que serão publicados. As listas na solução devem ser implementadas preferencialmente usando listas encadeadas ou outras estruturas alocadas dinamicamente. Soluções ineficientes usando vetores e `realloc()` serão penalizadas na nota.

A solução implementada deve evitar condições de disputa entre as *threads*. As *threads* devem executar com o máximo de concorrência possível; por exemplo, *subscribers* devem executar junto com o *publisher*, não sendo aceitável uma solução em que todos os itens são publicados para que então os *subscribers* comecem a processá-los. Um detalhe importante é que as *threads subscribers* devem processar os tópicos ainda não analisados, mesmo que o *publisher* já tenha chegado ao final. A aplicação não deve estar sujeita a *deadlock* por conta do controle de concorrência.

Ao final da execução, o programa principal deverá exibir as estatísticas retornadas pelas *threads* (número de itens e total de bytes de conteúdo), bem como a média de bytes por item, no seguinte formato:

```
Thread 1 - Itens processados: 2 - Total de bytes: 60 - Média: 30.00 bytes/item
Thread 2 - Itens processados: 2 - Total de bytes: 41 - Média: 20.50 bytes/item
Thread 3 - Itens processados: 3 - Total de bytes: 109 - Média: 36.33 bytes/item
Thread 4 - Itens processados: 4 - Total de bytes: 126 - Média: 31.50 bytes/item
```

Exemplos de arquivos de entrada e saída estão disponíveis no Moodle; seu programa deverá produzir a mesma saída para os arquivos de entrada fornecidos.

3 Apresentação e Avaliação

- O trabalho pode ser realizado individualmente ou em dupla.
- Um aspecto importante do *software* desenvolvido é o tratamento de erros, especialmente erros do usuário (como valores inválidos). Suas aplicações devem prever a possibilidade de erros, e tratá-los adequadamente. Por tratamento adequado, se entende que os erros devem ser detectados e contornados pela aplicação.
- O trabalho deve ser implementado em C, sob o sistema operacional Linux, usando as APIs para programação concorrente (Pthreads, memória compartilhada, semáforos POSIX) vistas em aula. Não é permitido usar outras bibliotecas que implementem porções significativas do trabalho. Em caso de dúvida, verifique **antes** com o professor.
- Deverá ser entregue impresso um relatório com até três páginas usando o formato de artigo da SBC (ponteiro disponível na página da disciplina). O relatório deve descrever de que forma foi implementado o controle de concorrência na aplicação, e qual a API utilizada. O código fonte dos programas, devidamente documentado, deve ser entregue **em formato ZIP** via Moodle.
  Obs.: é necessário respeitar apenas a formatação do modelo da SBC (tamanho de fonte, margens, etc.), não a estrutura do documento.
- O trabalho deverá ser entregue até **QUINTA-FEIRA, 19 DE SETEMBRO**. 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.
- A critério do professor, alguns alunos/grupos poderão ser solicitados a realizar uma apresentação do trabalho. Essa apresentação incluirá uma demonstração do funcionamento do *software* desenvolvido e uma discussão do fonte.
- A nota do trabalho corresponde a 20% da média final. Alunos que não entregarem o trabalho, não cumprirem o prazo ou não participarem da apresentação (se houver) terão atribuída nota zero.
- Em caso de cópia de trabalhos, **todos** os alunos envolvidos terão atribuída nota zero. **É sua responsabilidade garantir que o seu trabalho não seja copiado indevidamente.**

Em caso de dúvidas ou dificuldades, entre em contato com o professor.