Relatório da Atividade Prática 4: Programação Multithread (Parte 02: Coordenação de tarefas)

Laila Mota[1], **Matheus Hofstede**[1], **Alberto Neto**[1]

[1]Instituto de Matemática e Estatística – Departamento de Ciência da Computação
Universidade Federal da Bahia (UFBA)
Salvador – BA – Brasil

1. Considerações gerais

O objetivo deste projeto é o fortalecimento dos aspectos teóricos e práticos relacionados à programação paralela, a partir da implementação de programas envolvendo a coordenação de múltiplas threads, além disso, entender as primitivas básicas para coordenção de threads do padrão POSIX Threads que foi detalhado no relatório da atividade prática anterior.

Como os membros da equipe já são familiarizados com lógica de programação, com a linguagem C e com a utilização do git, optou-se por utilizar um repositório git para gestão, controle de versão e trabalho colaborativo. Não foram necessárias grandes revisões em outros aspectos, com a exceção da biblioteca pthreads.h.

A equipe realizou um estudo sobre a biblioteca e suas funções previamente para a correta implementação nos programas solicitados pela atividade.

O repositório supracitado está disponível através do link https://github.com/lailamt/so-ufba-lab-04-multi-thread2, onde é possível encontrar as orientações e comandos de execução no arquivo README.md e os relatórios desta atividade.

2. Problemas básicos de programação

O objetivo desta atividade é o ganho de experiência no uso de semáforos a partir de problemas selecionados para implementação.

2.1. Produtor/Consumidor no controle de buffers com capacidade limitada [Maziero, 2020]

O problema do Produtor/Consumidor consiste em coordenar o acesso a tarefas (processo ou threads) a um buffer compartilhado e de capacidade limitada a N itens onde são considerados dois tipos de processos com comportamentos cíclicos e simétricos.
O processo Produtor produz e deposita um item no buffer caso haja vaga, se não houver vaga o processo aguarda por uma e ao conseguir depositar um item, o produtor ”consome”uma vaga livre.
O processo Consumidor retira um item do buffer e o consome; caso o buffer esteja vazio, aguarda que item sejam depositados pelos produtores e ao consumir um item o produtor ”produz”uma vaga livre.
Para a atividade proposta, em questão, foram utilizados os semáforos, através da biblioteca semaphore.h da linguagem C, para limitar o número de número de threads ’simultâneas’, executando em paralelo, utilizando o princípio da exclusão mútua ou mutex, para evitar que dois processos ou threads tenham acesso simultaneamente a um recurso partilhado.
Em uma das soluções apresentamos uma abordagem com três semáforos. Um deles marca

as posições vazias do buffer, outro marca as posições cheias e um (mutex) limita o uso dos recursos para exclusão mútua evitando acesso concorrente à seção crítica.
Apresentamos duas soluções para o problema, com abordagens semelhantes que podem ser encontradas na pasta ”Outras”.

2.2. Produtor/Consumidor no caso do controle da plantação

Como este exercício apresenta um problema bastante semelhante ao anterior Produtor/-Consumidor, a implementação não difere significativamente. Neste caso temos a thread Joao executando de forma similar ao Produtor, o buffer então marca mais uma posição cheia. Na thread Pedro o buffer é marcado com menos uma posição vazia e, por fim, na thread Paulo o buffer é marcado com menos uma posição cheia, Paulo tem uma execução similar ao do consumidor.
O recurso compartilhado entre as threads Joao e Paulo é o representado pela pá.
Para este problema, também apresentamos duas soluções, com abordagens semelhantes que podem ser encontradas na pasta ”Outras”.

3. Problemas avançados de programação

O objetivo desta atividade é verificar a habilidade da equipe na coordenação de tarefas em problemas mais complexos.
De forma a abordar formas distintas de uso dos semáforos, para os problemas desta seção foram utilizadas uma abordagem pouco diferente da seção anterior.

3.1. Monitor Dorminhoco [Silberschatz, 2018]

Para solução deste problemas utilizamos mutex para coordenação da seção crítica e dados compartilhados. Mutex ’down’ (lock) acontece em um thread e mutex ’up’ (unlock) deve acontecer no mesmo thread posteriormente. Quando a thread adquire um mutex e está ocupado excluindo um nó, se outro thread tentar adquirir o mesmo mutex, ele será colocado no modo de espera até que o mutex seja liberado.

3.2. Barbearia [Stallings, 2017]

Neste problema, para melhor organização do código as funções de verificação da fila de barbeiros, fila de clientes e a gerência dos semáforos foram separadas em arquivos distintos, com códigos devidamente comentados. Além disso as mensagens que são exibidas pelo programa foram compiladas em mensagens.h.
Dessa forma o arquivo barbearia.c contem a rotina principal (main) e as chamadas para as rotinas dos arquivos complementares.

4. Considerações finais

Uma vez que a equipe pôde compreender a utilização dos semáforos nas formas apresentadas nas soluções dos problemas, as rotinas de coordenação de tarefas foram implementadas de forma semelhante, diferindo nas especificidades de cada problema em suas necessidades para solução.