Universidade: Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Unidade: Faculdade de Informática (FACIN)
Curso: Bacharelado em Ciência da Computação
Disciplina: Sistemas Operacionais
Professor: Carlos R. Moratelli
Turma: 128
Alunos: Emanoel Vianna, Gabriell Araujo

Relatório do trabalho 1

1. Jantar para *N* filósofos

Na nossa implementação existe um ponteiro do tipo *pthread_t*, que corresponde aos filósofos, e um ponteiro do tipo *pthread_mutex_t*, que corresponde aos garfos. O garfo da direita de um filósofo, é o próprio id do filósofo (ponteiro_garfos[id_filosofo]), enquanto que o garfo da esquerda é o id do filósofo decrementado em 1(ponteiro_garfos[id_filosofo-1]).

O problema todo é resolvido na função filósofo, que consiste em um laço *while* que é executado por um certa quantidade de tempo (a quantidade de tempo é passada como parâmetro no main). A primeira coisa feita no laço, é um *try_lock* no garfo da esquerda do filósofo. Se o *try_lock* é bem sucedido, então esse mutex é trancado, e a função efetua um *try_lock* no garfo da direita. Se esse *try_lock* é bem sucedido, então é chamada a função *comer*, e após é feito *unlock* nos garfos, e então é chamada a função *pensar* (por 5 segundos), e depois volta para o início do laço. Se a função não obtém sucesso no *try_lock* para o garfo da esquerda, é chamada a função *pensar* (por entre 0 e 3 segundos). Se função obtém sucesso no *try_lock* da esquerda (com isso, é efetuado lock no garfo da esquerda), e após não obtém sucesso no *try_lock* do garfo da direita, é feito *unlock* no garfo da esquerda, é executada a função *pensar* (por entre 0 e 3 segundos), e então, a função volta para o início do laço.

Para contabilizar quantas vezes um filósofo comeu, pensou, e não conseguiu pegar os gárfos, na função das threads é passado como parâmetro uma *struct*. Cada *thread* computa os dados na sua *struct* recebida por parâmetro, e no final da execução é imprido na tela os resultados que cada *thread* obteve.

Para contabilizar o tempo, usamos como auxílio a função *gettimeofday*, a função *get_time* que usamos é baseada no que conseguimos encontrar de materiais sobre programação em C.

2. Comunicação entre processos com memória compartilhada

Para resolver este problema, no código-fonte servidor.c, criamos um segmento de memória compartilhada para um ponteiro do tipo *char*, e dois ponteiros do tipo *sem_t*, e após, os segmentos são anexados. O semáforo do cliente é inicializado com 1, e o semáforo do servidor é inicializado com 0. O algoritmo do servidor é basicamente

um laço *while*, cujo dentro possui um *sem_wait*, que espera a mensagem de um cliente. Logo que um cliente manda uma mensagem, o servidor sai desse estado de espera, imprime a mensagem escrita pelo cliente, limpa a região de memória correspondente ao ponteiro de *char*, executa um *sem_post*, incrementando o semáforo correspondente aos clientes (ou seja, permite que um próximo cliente possa escrever no segmento de memória), e volta para o início do laço, para o *sem_wait*.

No código-fonte do cliente, são localizados os segmentos de memória do ponteiro de *char* e dos semáforos, e após é feito o anexo dos segmentos. A seguir, é executado um *sem_wait*, que espera uma mensagem de liberação oriunda do servidor. Quando é recebido o sinal, o programa escreve a mensagem (que é passada por parâmetro no main) no ponteiro de char do segmento de memória compartilhada, e após executa um *sem_post*, incrementando o semáforo do servidor. Após isso, o programa é encerrado.

Ressaltamos que o semáforo do servidor é inicializado com 0 e o semáforo dos clientes é inicializado com 1, isto, para que ao servidor começar a ser executado, ele imprima uma mensagem somente após o primeiro cliente escrevê-la no segmento de memória compartilhada.

3. Dificuldades encontradas

Para resolver o problema do jantar para *n* filósofos, não tivemos qualquer dificuldade, o motivo é que já vimos uma modelagem do problema na disciplina de *Métodos Formais,* ministrada pelo professor Júlio Machado, e além disso, já estamos familiarizados com programação em C, bem como programação Multitrheading.

Para resolver o problema da comunição entre processos, demoramos um pouco mais de tempo, o motivo é que não sabíamos como iríamos compartilhar semáforos entre processos (e esta foi a dificuldade), e passamos um bom tempo pesquisando tutoriais e exemplos de como fazer isso. Pensamos que para usar semáforos no segmento de memória compartilhada, seriam necessárias funções especiais e tratamento específico, mas após algum tempo de estudo, percebemos que os semáforos poderiam ser tratados como qualquer outro tipo de dado no segmento de memória compartilhada, e então conseguimos resolver o problema.